Universidad de Oviedo
Universidá d'Uviéu
University of Oviedo

Escuela de Ingeniería Informática
Universidad de Oviedo

VALIDACIÓN DE FLUJOS DE DATOS RDF MEDIANTE SHAPES

VALIDATING RDF STREAMS USING SHAPES

MÁSTER EN INGENIERÍA WEB

TRABAJO DE FIN DE MÁSTER

AUTOR

Eduardo Ulibarri Toledo

TUTOR

José Emilio Labra Gayo

Julio 2022

***Este documento ha sido creado basándose en la plantilla elaborada por JOSÉ MANUEL REDONDO LÓPEZ* [1].**

Agradecimientos

En primer lugar, quisiera agradecer a José Emilio Labra Gayo y el resto de los miembros del grupo de investigación WESO por darme la oportunidad de realizar este proyecto y aprender de ellos.

También he de dar las gracias a mis padres, así como a Guille, Sonia, Alejandro, Marcial, Claudia, Javi, Raúl, Sergio y Fer por escucharme y ayudarme a creer en mi proyecto.

Finalmente, dar las gracias a Irene por más cosas de las que entrarían en las páginas de esta memoria.

Resumen

Desde la perspectiva del usuario promedio, internet funciona como debería. Sin embargo, hay una tendencia históricamente creciente de formatos incompatibles que nos impiden sacarle el máximo provecho. Hoy en idea, reconstruir un internet libre de problemas desde cero es impráctico e impensable, de ahí los esfuerzos del *World Wide Web Consortium* (W3C), liderado por Tim Berners-Lee, para crear y velar por un conjunto de estándares que impulsen la web actual hacia una web semántica.

La web semántica es una web con significado, construida sobre estándares, formatos y herramientas universales, de forma que sus contenidos puedan ser expresados en lenguaje natural dirigido a humanos al tiempo que son complementados con (meta)datos que permiten a los programas informáticos comprender, procesar y relacionar la información, resolviendo problemas cotidianos como las búsquedas en internet. Por el momento, existen varios mecanismos universales para representar datos enlazados operables por máquinas a modo de grafos de conocimiento, siendo el más relevante el *Resource Description Framework* (RDF). Pese a esto, los casos de usos reales necesitan de mecanismos de validación que aseguren que los datos son coherentes e íntegros. Actualmente, los mecanismos de validación de vanguardia son *Shape Expressions* (*ShEx*) y *Shapes Constraint Language* (*SHACL*).

La meta de este proyecto es desarrollar un producto capaz de recibir, interpretar y validar flujos (*streams*) de datos RDF mediante *ShEx*, en contraposición a cómo se suele operar con grandes porciones individuales de RDF. El software resultante produce un flujo de salida con los resultados de la validación (más metadatos), de forma de puedan ser empleados por terceras partes en sus propios procesos. Cabe destacar que el producto final es extensible de cara a soportar cualquier flujo de entrada y mecanismo de validación, si bien se proporciona una implementación que soporta *Apache Kafka* y *Shape Expressions* como base para futuros desarrollos. Además, se ha desarrollado una aplicación web a modo de demo con fines educativos.

El proyecto ha sido desarrollado en colaboración con el grupo de investigación WESO de la Universidad de Oviedo, el cual ha proporcionado la base de los módulos de la aplicación web, así como la infraestructura de despliegue y los sujetos de las pruebas de usabilidad.

Palabras clave

World Wide Web, Web semántica, RDF, Shape Expressions, validación, flujos, biblioteca, API REST, aplicación web, Apache Kafka.

Abstract

Through the lens of the average user, the internet is working simply fine. Nonetheless, there is a historically increasing trend of incompatible document formats that prevents us from reaching its peak potential. Nowadays, the idea of re-building a problem-free world wide web from scratch is unfathomable, thus the efforts of the World Wide Web Consortium (W3C), led by Tim Berners-Lee, to supply and safeguard a set of standards to harmonize and shove the current web into the semantic web.

The semantic web is the web given meaning, built on universally standardized languages, formats and tooling, so that its contents may be expressed using natural, human-oriented language while being complemented with (meta)data that allows software to understand, process and relate the information, solving day-to-day issues such as web searches. At the moment, there exist several universal mechanisms to represent machine-operable linked data into knowledge graphs, the most relevant being the Resource Description Framework (RDF). Despite this, real use cases require validation mechanisms to ensure we operate on meaningful, legitimate data. Currently, the state-of-the-art validation mechanisms are Shape Expressions (ShEx) and Shapes Constraint Language (SHACL).

It is the aim of this project to develop a system capable of receiving, parsing, and validating incoming streams of RDF data using ShEx, in sheer contrast with how RDF data is usually operated in large individual chunks. The resulting software produces an output stream holding the validation results (plus metadata), meant to be further processed by third parties during their workflows. Notably, the final solution is extendable to support any incoming stream source and validation mechanism, yet an implementation supporting Apache Kafka and Shape Expressions is already provided as a foundation for further development. Furthermore, a fully-fledged web application has been created to serve as a demo and educational playground.

This project has taken place in partnership with University of Oviedo's WESO research group, which provided the basis for the web application modules, as well as the deployment infrastructure and usability-tests subjects.

Keywords

World Wide Web, Semantic web, RDF, Shape Expressions, validation, streams, library, REST API, web application, Apache Kafka.

Índice de contenido

Estructura del documento **25**
Métrica v.3 25
Resumen de los capítulos del proyecto 25
Capítulo 1 Planificación del Sistema de Información **29**
1.1 PSI 1: Inicio del Plan de Sistemas de Información 30
1.1.1 PSI 1.1: Análisis de la Necesidad del PSI 30
1.1.2 PSI 1.2: Identificación del Alcance del PSI 32
1.1.3 PSI 1.3: Determinación de responsables 33
1.1.4 PSI 1.4: Solución Propuesta 34
1.2 PSI 2: Definición y Organización del PSI 36
1.2.1 PSI 2.1: Especificación del Ámbito y Alcance 36
1.2.2 PSI 2.2: Organización del PSI 40
1.3 PSI 3: Estudio de la Información Relevante 43
1.3.1 PSI 3.1: Selección y Análisis de Antecedentes 43
Capítulo 2 Definición de la Arquitectura Tecnológica **49**
2.1 PSI 7.1: Identificación de las Necesidades de Infraestructura Tecnológica 50
2.1.1 Biblioteca de validación de flujos RDF 50
2.1.2 Refactorización y ampliación RDFShape 53
2.1.3 Publicación del sistema 56
2.1.4 Integración continua 60
2.1.5 Bases de datos 61
2.2 PSI 7.2: Selección de la Arquitectura Tecnológica 62
2.2.1 Biblioteca de validación de flujos RDF 62
2.2.2 Refactorización y ampliación RDFShape 65
2.2.3 Publicación del sistema 67
2.2.4 Integración continua: GitHub Actions 71
2.2.5 Bases de datos: MongoDB (Atlas) 71
Capítulo 3 Estudio de Viabilidad del Sistema **73**

3.1 EVS 4, 5 y 6: Estudio y Valoración de Alternativas de Solución y Selección de Alternativa Final 74
3.1.1 Alternativas Disponibles 74
3.1.2 Selección Final 77
Capítulo 4 Análisis del Sistema de Información 78
4.1 ASI 1: Definición del Sistema 79
4.1.1 Determinación del Alcance del Sistema 79
4.2 ASI 2: Establecimiento de Requisitos 87
4.2.1 Obtención de los Requisitos del Sistema 87
4.2.2 Identificación de Actores del Sistema 108
4.2.3 Especificación de Casos de Uso 111
4.2.4 Diagramas de Estado y Flujo 121
4.3 Planificación y Gestión del TFM 125
4.3.1 Planificación del Proyecto 125
4.3.2 Ejecución del Proyecto 155
4.3.3 Cierre del proyecto 160
4.4 ASI 3: Identificación de Subsistemas de Análisis 172
4.4.1 Descripción de los Subsistemas 172
4.4.2 Descripción de los Interfaces entre Subsistemas 174
4.5 ASI 4: Análisis de los Casos de Uso 176
4.5.1 Usuario Biblioteca 176
4.5.2 Usuario API 182
4.5.3 Administrador del Sistema 193
4.5.4 Servidor Remoto 196
4.6 ASI 5: Análisis de Clases 198
4.6.1 Módulo Validador (Comet) 199
4.6.2 Módulo API 208
4.6.3 Módulo Cliente 214
4.7 ASI 8: Definición de Interfaces de Usuario 221
4.7.1 Descripción de la Interfaz 221
4.7.2 Definición del Aspecto de la Interfaz 223
4.7.3 Descripción del Comportamiento de la Interfaz 234

4.7.4 Diagrama de Navegabilidad 235
4.8 ASI 10: Especificación del Plan de Pruebas 236
4.8.1 Pruebas Unitarias 237
4.8.2 Pruebas de Integración 240
4.8.3 Pruebas de Usabilidad 247
4.8.4 Pruebas de Accesibilidad 247
4.8.5 Pruebas de Rendimiento 248
Capítulo 5 Diseño del Sistema de Información 249
5.1 DSI 3: Diseño de Casos de Uso Reales 250
5.1.1 Usuario Biblioteca 251
5.1.2 Usuario API 253
5.2 DSI 4: Diseño de Clases 257
5.2.1 Módulo Validador (Comet) 257
5.2.2 Módulo API 260
5.2.3 Módulo Cliente 263
5.3 DSI 5: Diseño de la Arquitectura de Módulos del Sistema 266
5.3.1 DSI 5.1 Diseño de Módulos del Sistema 266
5.3.2 DSI 5.2 Diseño de Comunicaciones entre Módulos 278
5.3.3 DSI 5.3 Revisión de la Interfaz de Usuario 280
5.4 DSI 6: Diseño Físico de Datos 282
5.4.1 Descripción del SGBD Usado 282
5.4.2 Integración del SGBD en Nuestro Sistema 283
5.4.3 Diagrama E-R 283
5.5 DSI 10: Especificación Técnica del Plan de Pruebas 284
5.5.1 Pruebas Unitarias 284
5.5.2 Pruebas de Integración 289
5.5.3 Pruebas de Usabilidad 308
5.5.4 Pruebas de Accesibilidad 311
5.5.5 Pruebas de Rendimiento 311
Capítulo 6 Construcción del Sistema de Información 315
6.1 CSI 1: Preparación del Entorno de Generación y Construcción 316
6.1.1 Estándares y normas seguidos 316

6.1.2 Lenguajes de programación 317
6.1.3 Herramientas y programas usados para el desarrollo 320
6.1.4 Aspectos de seguridad 333
6.2 CSI 2: Generación del Código de los Componentes y Procedimientos 336
6.3 CSI 3: Ejecución de las Pruebas Unitarias 337
6.4 CSI 4: Ejecución de las Pruebas de Integración 341
6.5 CSI 5: Ejecución de las Pruebas del Sistema 349
6.5.1 Prueba de Usabilidad 349
6.5.2 Pruebas de Accesibilidad 353
6.5.3 Pruebas de rendimiento 359
6.5.4 Análisis de los resultados 360
6.6 CSI 6: Elaboración de los Manuales de Usuario 362
6.6.1 Manual de Instalación 362
6.6.2 Manual de Ejecución 368
6.6.3 Manual de Usuario 373
6.6.4 Manual del Programador 398
Capítulo 7 Implantación y Aceptación del Sistema 413
7.1 IAS 1: Establecimiento del Plan de Implantación 414
7.1.1 Plan de implantación del sistema 414
7.2 IAS 7: Preparación del Mantenimiento del Sistema 417
7.2.1 Manuales de despliegue y mantenimiento 417
7.3 IAS 9-10: Presentación y Aprobación del Sistema y Paso a Producción 420
Capítulo 8 Conclusión y Ampliaciones 421
8.1 Conclusiones 422
8.1.1 Conclusiones técnicas 422
8.1.2 Conclusiones personales 423
8.2 Ampliaciones 424
Anexos 425
A1. Plan de gestión de riesgos 426
A1.1 Introducción 426
A1.2 Metodología 427
A1.3 Herramientas y Tecnologías 429

A1.4 Roles y Responsabilidades....430
A1.5 Presupuesto de Riesgos....431
A1.6 Calendario de Riesgos....431
A1.7 Categorías de Riesgo....432
A1.8 Definiciones de Probabilidad....433
A1.9 Definiciones de Impacto por Objetivos....433
A1.10 Matriz de Probabilidad-Impacto....435
A1.11 Niveles de Tolerancia....435
A1.12 Planes de Contingencia....436
A1.13 Formatos de la Documentación....436
A2. Hojas de riesgos....437
A2.1 Hoja de riesgos I....439
A2.2 Hoja de riesgos II....445
A2.3 Hoja de riesgos III....451
A2.4 Hoja de riesgos IV....455
A2.5 Hoja de riesgos V....461
A3. Informe “Post Mortem” RDFShape....465
A3.1 Descripción....465
A3.2 Contenidos....465
A4. Contenido entregado....469
A4.1 Contenidos....469
A5. Referencias bibliográficas....471

Índice de Figuras

ILUSTRACIÓN 1. EJEMPLO DE DOS PERSONAS Y SUS RELACIONES - RDF (TURTLE) 44
ILUSTRACIÓN 2. EJEMPLO DE DOS PERSONAS Y SUS RELACIONES - RDF (IMAGEN) 44
ILUSTRACIÓN 3. VALIDACIÓN DE DOS PERSONAS Y SUS RELACIONES REPRESENTADOS MEDIANTE SHEX – ESQUEMA (SHEXC) 46
ILUSTRACIÓN 4. VALIDACIÓN DE DOS PERSONAS Y SUS RELACIONES REPRESENTADOS MEDIANTE SHEX - SHAPEMAP .. 46
ILUSTRACIÓN 5. VALIDACIÓN DE DOS PERSONAS Y SUS RELACIONES REPRESENTADOS MEDIANTE SHEX - RESULTADOS (RDFSHAPE) 46
ILUSTRACIÓN 6. LOGO PROVISIONAL DE LA BIBLIOTECA DE VALIDACIÓN 62
ILUSTRACIÓN 7. LOGO DE CATS EFFECT 62
ILUSTRACIÓN 8. LOGO DE APACHE KAFKA 63
ILUSTRACIÓN 9. LOGO DE FS2-KAFKA 64
ILUSTRACIÓN 10. LOGO DE FS2 64
ILUSTRACIÓN 11. LOGO DE HTTP4S 65
ILUSTRACIÓN 12. LOGO DE BOOTSTRAP 67
ILUSTRACIÓN 13. LOGO DE REACT 67
ILUSTRACIÓN 14. LOGO DE DOCKER 68
ILUSTRACIÓN 15. LOGO DE DOCKER COMPOSE 68
ILUSTRACIÓN 16. LOGO DE NGINX 69
ILUSTRACIÓN 17. LOGO DE MDOC 70
ILUSTRACIÓN 18. LOGO DE DOCUSAURUS 70
ILUSTRACIÓN 19. LOGO DE MONGODB 72
ILUSTRACIÓN 20. ESPECIFICACIÓN DE CASOS DE USO - USUARIO BIBLIOTECA 111
ILUSTRACIÓN 21. ESPECIFICACIÓN DE CASOS DE USO - USUARIO API 114
ILUSTRACIÓN 22. ESPECIFICACIÓN DE CASOS DE USO - ADMINISTRADOR DEL SISTEMA 118
ILUSTRACIÓN 23. ESPECIFICACIÓN DE CASOS DE USO - SERVIDOR REMOTO 120
ILUSTRACIÓN 24. DIAGRAMA DE FLUJO DE UNA VALIDACIÓN CON COMET 122
ILUSTRACIÓN 25. DIAGRAMA DE FLUJO - PROCESAMIENTO DE UNA VALIDACIÓN EN EL SERVIDOR 123
ILUSTRACIÓN 26. DIAGRAMA DE FLUJO - PROCESAMIENTO DE UNA VALIDACIÓN EN EL CLIENTE 124
ILUSTRACIÓN 27. PBS DEL PROYECTO - INICIO 129
ILUSTRACIÓN 28. PBS DEL PROYECTO - EJECUCIÓN 130
ILUSTRACIÓN 29. PBS DEL PROYECTO - GESTIÓN Y CIERRE 131
ILUSTRACIÓN 30. PBS DEL PROYECTO – DESARROLLO SOFTWARE 132
ILUSTRACIÓN 31. DIAGRAMA DE GANTT - PLANIFICACIÓN INICIAL I 135
ILUSTRACIÓN 32. DIAGRAMA DE GANTT - PLANIFICACIÓN INICIAL II 136
ILUSTRACIÓN 33. DIAGRAMA DE GANTT - PLANIFICACIÓN INICIAL III 137
ILUSTRACIÓN 34. DIAGRAMA DE GANTT - PLANIFICACIÓN INICIAL IV 138
ILUSTRACIÓN 35. ESTADÍSTICAS DE LA LÍNEA BASE INICIAL 155
ILUSTRACIÓN 36. ESTADÍSTICAS DE LA LÍNEA BASE INTERMEDIA 156
ILUSTRACIÓN 37. ESTADÍSTICAS DE LA LÍNEA BASE FINAL 156
ILUSTRACIÓN 38. DIAGRAMA DE GANTT - PLANIFICACIÓN FINAL I 161
ILUSTRACIÓN 39. DIAGRAMA DE GANTT - PLANIFICACIÓN FINAL II 162
ILUSTRACIÓN 40. DIAGRAMA DE GANTT -PLANIFICACIÓN FINAL III 163
ILUSTRACIÓN 41. DIAGRAMA DE GANTT - PLANIFICACIÓN FINAL IV 164
ILUSTRACIÓN 42. INTERFACES ENTRE SUBSISTEMAS 174
ILUSTRACIÓN 43. DIAGRAMA DE CLASES DE ANÁLISIS - COMET 199
ILUSTRACIÓN 44. DIAGRAMA DE CLASES DE ANÁLISIS - API 208

ILUSTRACIÓN 45. DIAGRAMA DE CLASES DE ANÁLISIS - CLIENTE 214
ILUSTRACIÓN 46. INTERFAZ DE USUARIO - DESCRIPCIÓN 221
ILUSTRACIÓN 47. INTERFAZ DE USUARIO - MENSAJES EMERGENTES 222
ILUSTRACIÓN 48. INTERFAZ DE USUARIO - INTERFAZ EXISTENTE EN RDFSHAPE 223
ILUSTRACIÓN 49. INTERFAZ DE USUARIO - ENTRADA DE DATOS 225
ILUSTRACIÓN 50. INTERFAZ DE USUARIO - ENVÍO DE DATOS 227
ILUSTRACIÓN 51. INTERFAZ DE USUARIO - CONTROLES Y RESULTADOS (EN CURSO) 229
ILUSTRACIÓN 52. INTERFAZ DE USUARIO - CONTROLES Y RESULTADOS (PAUSA) 231
ILUSTRACIÓN 53. INTERFAZ DE USUARIO - ERROR EN EL PROCESO 233
ILUSTRACIÓN 54. DIAGRAMA DE NAVEGABILIDAD DEL CLIENTE WEB 235
ILUSTRACIÓN 55. DIAGRAMA DE SECUENCIA - INVOCAR UNA VALIDACIÓN DE FLUJOS (BIBLIOTECA) 251
ILUSTRACIÓN 56. DIAGRAMA DE SECUENCIA - INVOCAR UNA VALIDACIÓN DE FLUJOS (API) 253
ILUSTRACIÓN 57. DIAGRAMA DE SECUENCIA - GENERAR PERMALINKS (CLIENTE) 255
ILUSTRACIÓN 58. DIAGRAMA DE SECUENCIA - CONSUMIR PERMALINKS (CLIENTE) 256
ILUSTRACIÓN 59. DIAGRAMA DE CLASES DE DISEÑO - COMET 258
ILUSTRACIÓN 60. DIAGRAMA DE CLASES DE DISEÑO - API 261
ILUSTRACIÓN 61. DIAGRAMA DE CLASES DE DISEÑO - CLIENTE 264
ILUSTRACIÓN 62. PATRONES DE DISEÑO - STRATEGY (COMET) 267
ILUSTRACIÓN 63. PATRONES DE DISEÑO - ADAPTER (COMET) 268
ILUSTRACIÓN 64. DIAGRAMA DE PAQUETES - COMET 269
ILUSTRACIÓN 65. DIAGRAMA DE DESPLIEGUE - COMET 271
ILUSTRACIÓN 66. DIAGRAMA DE PAQUETES - API 273
ILUSTRACIÓN 67. DIAGRAMA DE DESPLIEGUE - API 275
ILUSTRACIÓN 68. DIAGRAMA DE PAQUETES - CLIENTE 276
ILUSTRACIÓN 69. DIAGRAMA DE DESPLIEGUE - CLIENTE 277
ILUSTRACIÓN 70. DIAGRAMA DE DESPLIEGUE - SISTEMA 279
ILUSTRACIÓN 71. DISEÑO DE MENSAJES DEL SISTEMA - ERROR DE VALIDACIÓN 281
ILUSTRACIÓN 72. DISEÑO DE MENSAJES DEL SISTEMA - NOTIFICACIONES “TOAST” 281
ILUSTRACIÓN 73. DIAGRAMA ENTIDAD-RELACIÓN 283
ILUSTRACIÓN 74. PLANTILLA RDF PARA PRUEBAS UNITARIAS 284
ILUSTRACIÓN 75. PLANTILLA SHEX PARA PRUEBAS UNITARIAS 285
ILUSTRACIÓN 76. LOGO DE SCALA 318
ILUSTRACIÓN 77. LOGO DE HTML 5 318
ILUSTRACIÓN 78. LOGO DE CSS 319
ILUSTRACIÓN 79. LOGO DE JAVASCRIPT 319
ILUSTRACIÓN 80. LOGO DE GIT 320
ILUSTRACIÓN 81. LOGO DE INTELLIJ IDEA 322
ILUSTRACIÓN 82. LOGO DE VSCODE 323
ILUSTRACIÓN 83. LOGO DE SBT 323
ILUSTRACIÓN 84. LOGO DE POSTMAN 324
ILUSTRACIÓN 85. LOGO DE CIRCE 326
ILUSTRACIÓN 86. LOGO DE UMLET 330
ILUSTRACIÓN 87. LOGO DE DIAGRAMS.NET 330
ILUSTRACIÓN 88. LOGO DE BALSAMIQ WIREFRAMES 331
ILUSTRACIÓN 89. LOGO DE GIMP 332
ILUSTRACIÓN 90. PRUEBAS UNITARIAS - RESULTADOS FINALES 340
ILUSTRACIÓN 91. PRUEBAS DE ACCESIBILIDAD - PANTALLA DE INICIO 354
ILUSTRACIÓN 92. PANTALLA DE INICIO DEL SISTEMA 355
ILUSTRACIÓN 93. PRUEBAS DE ACCESIBILIDAD - PANTALLA DE CONSULTAS SPARQL 355

ILUSTRACIÓN 94. PANTALLA DE EJECUCIÓN DE CONSULTAS SPARQL 356
ILUSTRACIÓN 95. PRUEBAS DE ACCESIBILIDAD - PANTALLA DE ANÁLISIS RDF 356
ILUSTRACIÓN 96. PANTALLA DE ANÁLISIS Y VISUALIZACIÓN DE RDF 357
ILUSTRACIÓN 97. PRUEBAS DE ACCESIBILIDAD - PANTALLA DE ANÁLISIS SHEX 357
ILUSTRACIÓN 98. PANTALLA DE ANÁLISIS Y VISUALIZACIÓN DE SHEX 358
ILUSTRACIÓN 99. PRUEBAS DE ACCESIBILIDAD - PANTALLA DE VALIDACIÓN SHEX 358
ILUSTRACIÓN 100. PANTALLA DE VALIDACIÓN DE RDF MEDIANTE SHEX 359
ILUSTRACIÓN 101. RESULTADOS DE LAS PRUEBAS DE RENDIMIENTO EN INTELLIJ IDEA (10000 ÍTEMS, TIEMPO Nº3) ... 360
ILUSTRACIÓN 102. CLIENTE WEB - PANTALLA DE INICIO 387
ILUSTRACIÓN 103. CLIENTE WEB - BARRA DE NAVEGACIÓN 387
ILUSTRACIÓN 104. CLIENTE - LEYENDA DE LA INTERFAZ DE USUARIO 389
ILUSTRACIÓN 105. CLIENTE - NOTIFICACIÓN PERMALINK ADQUIRIDO 390
ILUSTRACIÓN 106. CLIENTE - BOTÓN PERMALINK (PROCESANDO) 390
ILUSTRACIÓN 107. CLIENTE - BOTÓN PERMALINK 390
ILUSTRACIÓN 108. CLIENTE - MENSAJE DE CARGA DE PERMALINKS 391
ILUSTRACIÓN 109. CLIENTE - PANTALLA DE ERROR DE PERMALINKS 391
ILUSTRACIÓN 110. CLIENTE - FORMULARIO DE ENTRADA DE DATOS MEDIANTE STREAM 392
ILUSTRACIÓN 111. CLIENTE - RESULTADOS DE UNA VALIDACIÓN EXITOSA 393
ILUSTRACIÓN 112. CLIENTE - TABLA DE RESULTADOS Y SALIDAS DE UNA VALIDACIÓN 395
ILUSTRACIÓN 113. CLIENTE - ALERTA DE INICIO DE VALIDACIÓN DE FLUJOS 396
ILUSTRACIÓN 114. CLIENTE - RESULTADOS Y OPCIONES DE UNA VALIDACIÓN DE FLUJOS EN CURSO 397

Índice de Tablas

TABLA 1. EQUIPOS DE TRABAJO DEL PROYECTO 41
TABLA 2. CALENDARIO DE ENTREGAS PRINCIPALES DEL PROYECTO 42
TABLA 3. ESPECIFICACIÓN CASO DE USO "INTEGRAR LA BIBLIOTECA DE VALIDACIÓN EN EL PROYECTO" (USUARIO BIBLIOTECA) 111
TABLA 4. ESPECIFICACIÓN CASO DE USO "CONFIGURAR FLUJO DE ENTRADA" (USUARIO BIBLIOTECA) 112
TABLA 5. ESPECIFICACIÓN CASO DE USO "CONFIGURAR EL VALIDADOR DE LOS ÍTEMS DEL FLUJO" (USUARIO BIBLIOTECA) 112
TABLA 6. ESPECIFICACIÓN CASO DE USO "INVOCAR UNA VALIDACIÓN DE FLUJOS" (USUARIO BIBLIOTECA) 112
TABLA 7. ESPECIFICACIÓN CASO DE USO "IMPLEMENTAR NUEVOS FLUJOS DE ENTRADA" (USUARIO BIBLIOTECA) 112
TABLA 8. ESPECIFICACIÓN CASO DE USO "CONSULTAR DOCUMENTACIÓN DE LA BIBLIOTECA" (USUARIO BIBLIOTECA) 113
TABLA 9. ESPECIFICACIÓN CASO DE USO "CONFIGURAR LA VALIDACIÓN DE LOS ÍTEMS DEL FLUJO" (USUARIO API - BACKEND) 114
TABLA 10. ESPECIFICACIÓN CASO DE USO "INVOCAR UNA VALIDACIÓN DE FLUJOS" (USUARIO API - BACKEND) 115
TABLA 11. ESPECIFICACIÓN CASO DE USO "DETENER UNA VALIDACIÓN DE FLUJOS" (USUARIO API - BACKEND) 115
TABLA 12. ESPECIFICACIÓN CASO DE USO "CONSULTAR DOCUMENTACIÓN DEL API" (USUARIO API - BACKEND) 115
TABLA 13. ESPECIFICACIÓN CASO DE USO "PAUSAR UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 115
TABLA 14. ESPECIFICACIÓN CASO DE USO "REANUDAR UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 116
TABLA 15. ESPECIFICACIÓN CASO DE USO "ORDENAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 116
TABLA 16. ESPECIFICACIÓN CASO DE USO "DESCARTAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 116
TABLA 17. ESPECIFICACIÓN CASO DE USO "BUSCAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 116
TABLA 18. ESPECIFICACIÓN CASO DE USO "EXPORTAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 117
TABLA 19. ESPECIFICACIÓN CASO DE USO "GENERAR PERMALINKS" (USUARIO API - CLIENTE) 117
TABLA 20. ESPECIFICACIÓN CASO DE USO "CONSUMIR PERMALINKS" (USUARIO API - CLIENTE) 117
TABLA 21. ESPECIFICACIÓN CASO DE USO "CONSULTAR AYUDA DEL CLIENTE" (USUARIO API - CLIENTE) 117
TABLA 22. ESPECIFICACIÓN CASO DE USO "DESPLEGAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA) 118
TABLA 23. ESPECIFICACIÓN CASO DE USO "MONITORIZAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA) 118
TABLA 24. ESPECIFICACIÓN CASO DE USO "REINICIAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA) 119
TABLA 25. ESPECIFICACIÓN CASO DE USO "CONFIGURAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA) 119
TABLA 26. ESPECIFICACIÓN CASO DE USO "AUTENTICAR CONSUMIDOR" (SERVIDOR REMOTO) 120
TABLA 27. ESPECIFICACIÓN CASO DE USO "ENVIAR FLUJO DE DATOS" (SERVIDOR REMOTO) 120
TABLA 28. ROLES DEL PROYECTO 126
TABLA 29. OBS DEL PROYECTO (RESUMEN) 127
TABLA 30. CUADRO RESUMEN DE LA PLANIFICACIÓN INICIAL 141
TABLA 31. LISTADO DE RIESGOS 142
TABLA 32. PRESUPUESTO DE COSTES (INICIAL) 152
TABLA 33. COSTES INDIRECTOS (INICIAL) 153
TABLA 34. PRESUPUESTO DE CLIENTE DETALLADO (INICIAL) 154
TABLA 35. PRESUPUESTO DE CLIENTE RESUMIDO (INICIAL) 154
TABLA 36. TAREAS DE SEGUIMIENTO DE LA PLANIFICACIÓN 155
TABLA 37. RESUMEN TEMPORAL DE LAS INCIDENCIAS DEL PROYECTO 159
TABLA 38. CUADRO RESUMEN DE LA PLANIFICACIÓN FINAL 167

TABLA 39. PRESUPUESTO DE RIESGOS (FINAL) 167
TABLA 40. CALENDARIO DE RIESGOS (FINAL) 168
TABLA 41. PRESUPUESTO DE COSTES (FINAL) 170
TABLA 42. PRESUPUESTO DE CLIENTE RESUMIDO (FINAL) 170
TABLA 43. ANÁLISIS CASO DE USO "INTEGRAR LA BIBLIOTECA DE VALIDACIÓN EN EL PROYECTO" (USUARIO BIBLIOTECA) 177
TABLA 44. ANÁLISIS CASO DE USO "CONFIGURAR EL FLUJO DE ENTRADA" (USUARIO BIBLIOTECA) 178
TABLA 45. ANÁLISIS CASO DE USO "CONFIGURAR EL VALIDADOR DE LOS ÍTEMS DEL FLUJO" (USUARIO BIBLIOTECA) .. 179
TABLA 46. ANÁLISIS CASO DE USO "INVOCAR UNA VALIDACIÓN DE FLUJOS" (USUARIO BIBLIOTECA) 180
TABLA 47. ANÁLISIS CASO DE USO "IMPLEMENTAR NUEVOS FLUJOS DE ENTRADA" (USUARIO BIBLIOTECA) 181
TABLA 48. ANÁLISIS CASO DE USO "CONSULTAR DOCUMENTACIÓN DE LA BIBLIOTECA" (USUARIO BIBLIOTECA)..... 182
TABLA 49. ANÁLISIS CASO DE USO "CONFIGURAR LA VALIDACIÓN DE LOS ÍTEMS DEL FLUJO" (USUARIO API - BACKEND) 184
TABLA 50. ANÁLISIS CASO DE USO "INVOCAR UNA VALIDACIÓN DE FLUJOS" (USUARIO API - BACKEND) 185
TABLA 51. ANÁLISIS CASO DE USO "DETENER UNA VALIDACIÓN DE FLUJOS" (USUARIO API - BACKEND) 185
TABLA 52. ANÁLISIS CASO DE USO "CONSULTAR DOCUMENTACIÓN DEL API" (USUARIO API - BACKEND) 186
TABLA 53. ANÁLISIS CASO DE USO "PAUSAR UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 187
TABLA 54. ANÁLISIS CASO DE USO "REANUDAR UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 188
TABLA 55. ANÁLISIS CASO DE USO "ORDENAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 188
TABLA 56. ANÁLISIS CASO DE USO "DESCARTAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 189
TABLA 57. ANÁLISIS CASO DE USO "BUSCAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE)190
TABLA 58. ANÁLISIS CASO DE USO "EXPORTAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS" (USUARIO API - CLIENTE) 190
TABLA 59. ANÁLISIS CASO DE USO "GENERAR PERMALINKS" (USUARIO API - CLIENTE) 191
TABLA 60. ANÁLISIS CASO DE USO "CONSUMIR PERMALINKS" (USUARIO API - CLIENTE)..... 192
TABLA 61 . ANÁLISIS CASO DE USO "CONSULTAR LA AYUDA DEL CLIENTE" (USUARIO API - CLIENTE)..... 193
TABLA 62. ANÁLISIS CASO DE USO "DESPLEGAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA)..... 193
TABLA 63. ANÁLISIS CASO DE USO "MONITORIZAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA)..... 194
TABLA 64. ANÁLISIS CASO DE USO "REINICIAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA) 195
TABLA 65. ANÁLISIS CASO DE USO "CONFIGURAR EL SISTEMA" (ADMINISTRADOR DEL SISTEMA)..... 195
TABLA 66. ANÁLISIS CASO DE USO "AUTENTICAR CONSUMIDOR" (SERVIDOR REMOTO) 196
TABLA 67. ANÁLISIS CASO DE USO "ENVIAR FLUJO DE DATOS" (SERVIDOR REMOTO)..... 197
TABLA 68. PRUEBAS DE VALIDACIÓN – COMET (ANÁLISIS) 237
TABLA 69. PRUEBAS DE EXTRACCIÓN – COMET (ANÁLISIS) 238
TABLA 70. PRUEBAS DE PARADA – COMET (ANÁLISIS)..... 239
TABLA 71. PRUEBAS DE INTEGRACIÓN – COMET (ANÁLISIS)..... 241
TABLA 72. PRUEBAS DE INTEGRACIÓN – API (ANÁLISIS) 244
TABLA 73. PRUEBAS DE INTEGRACIÓN – CLIENTE (ANÁLISIS) 247
TABLA 74. PRUEBAS DE USABILIDAD - CLIENTE..... 247
TABLA 75. PRUEBAS DE ACCESIBILIDAD – CLIENTE (ANÁLISIS)..... 248
TABLA 76. PRUEBAS DE RENDIMIENTO – COMET (ANÁLISIS)..... 248
TABLA 77. PRUEBAS DE VALIDACIÓN – COMET (DISEÑO) 286
TABLA 78. PRUEBAS DE EXTRACCIÓN – COMET (DISEÑO) 287
TABLA 79. PRUEBAS DE PARADA – COMET (DISEÑO)..... 289
TABLA 80. PRUEBAS DE INTEGRACIÓN - COMET (DISEÑO) 291
TABLA 81. PRUEBAS DE INTEGRACIÓN – API (DISEÑO) 301
TABLA 82. PRUEBAS DE INTEGRACIÓN – CLIENTE (DISEÑO) 308

TABLA 83. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS GENERALES 309
TABLA 84. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS DE FUNCIONALIDAD 309
TABLA 85. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS DE USABILIDAD 310
TABLA 86. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS DE ACCESIBILIDAD 310
TABLA 87. CUESTIONARIO DE EVALUACIÓN - OBSERVACIONES 310
TABLA 88. CUESTIONARIO PARA EL RESPONSABLE DE LAS PRUEBAS DE USABILIDAD 311
TABLA 89. PRUEBAS DE ACCESIBILIDAD – CLIENTE (DISEÑO) 311
TABLA 90. PRUEBAS DE RENDIMIENTO – COMET (DISEÑO) 313
TABLA 91. MEDIDAS DE SEGURIDAD TOMADAS EN EL CÓDIGO DEL SISTEMA 335
TABLA 92. PRUEBAS DE VALIDACIÓN – COMET (RESULTADOS) 337
TABLA 93. PRUEBAS DE EXTRACCIÓN – COMET (RESULTADOS) 338
TABLA 94. PRUEBAS DE PARADA – COMET (RESULTADOS) 339
340
TABLA 95. PRUEBAS DE INTEGRACIÓN - COMET (RESULTADOS) 342
TABLA 96. PRUEBAS DE INTEGRACIÓN – API (RESULTADOS) 345
TABLA 97. PRUEBAS DE INTEGRACIÓN – CLIENTE (RESULTADOS) 348
TABLA 98. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS GENERALES (RESPUESTAS) 350
TABLA 99. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS DE FUNCIONALIDAD (RESPUESTAS) 350
TABLA 100. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS DE USABILIDAD (RESPUESTAS) 351
TABLA 101. CUESTIONARIO DE EVALUACIÓN - PREGUNTAS DE ACCESIBILIDAD (RESPUESTAS) 352
TABLA 102. CUESTIONARIO DE EVALUACIÓN - OBSERVACIONES (RESPUESTAS) 352
TABLA 103. CUESTIONARIO PARA EL RESPONSABLE DE LAS PRUEBAS DE USABILIDAD (RESPUESTAS) 353
TABLA 104. PRUEBAS DE RENDIMIENTO - RESULTADOS 360
PRESUPUESTO DE RIESGOS 431
CALENDARIO DE RIESGOS 431
DEFINICIONES DE PROBABILIDAD DE RIESGOS 433
DEFINICIONES DE IMPACTO POR OBJETIVOS – AMENAZAS 434
DEFINICIONES DE IMPACTO POR OBJETIVOS – OPORTUNIDADES 435
MATRIZ DE PROBABILIDAD-IMPACTO 435
TABLA 105. ESTRUCTURA GENERAL DE LOS ANEXOS ENTREGADOS 469
TABLA 106. ESTRUCTURA DE LA CARPETA "DOCUMENTACION" DE LOS ANEXOS 470

Índice de Código

CÓDIGO 1. ADICIÓN DE COMET COMO DEPENDENCIA 363
CÓDIGO 2. ADICIÓN DE SONATYPE AL GESTOR DE DEPENDENCIAS 363
CÓDIGO 3. DESCARGA DE LA IMAGEN DOCKER DEL MÓDULO API 365
CÓDIGO 4. CONSTRUCCIÓN DE LA IMAGEN DOCKER DEL MÓDULO API 365
CÓDIGO 5. DESCARGA DE LA IMAGEN DOCKER DEL MÓDULO CLIENTE 367
CÓDIGO 6. CONSTRUCCIÓN DE LA IMAGEN DOCKER DEL MÓDULO CLIENTE 367
CÓDIGO 7. COMANDO DE EJEMPLO DE EJECUCIÓN DEL MÓDULO API DESDE SBT 368
CÓDIGO 8. COMANDO DE EJEMPLO DE EJECUCIÓN DEL MÓDULO API DESDE EL EJECUTABLE 368
CÓDIGO 9. EJECUCIÓN DEL MÓDULO API A TRAVÉS DE DOCKER (LÍNEA DE COMANDOS) 370
CÓDIGO 10. DETENCIÓN DEL MÓDULO API A TRAVÉS DE DOCKER (LÍNEA DE COMANDOS) 371
CÓDIGO 11. RELANZAMIENTO DEL MÓDULO API A TRAVÉS DE DOCKER (LÍNEA DE COMANDOS) 371
CÓDIGO 12. COMANDO DE EJECUCIÓN DEL MÓDULO CLIENTE DESDE NPM 371
CÓDIGO 13. GENERACIÓN DE LA APLICACIÓN WEB REACT DESDE NPM 371
CÓDIGO 14. EJECUCIÓN DEL MÓDULO CLIENTE A TRAVÉS DE DOCKER (LÍNEA DE COMANDOS) 372
CÓDIGO 15. DETENCIÓN DEL MÓDULO CLIENTE A TRAVÉS DE DOCKER (LÍNEA DE COMANDOS) 372
CÓDIGO 16. RELANZAMIENTO DEL MÓDULO CLIENTE A TRAVÉS DE DOCKER (LÍNEA DE COMANDOS) 372
CÓDIGO 17. EJEMPLO DE USO DE COMET (I) 374
CÓDIGO 18. EJEMPLO DE USO DE COMET (II) 375
CÓDIGO 19. ESTRUCTURA DE LA CLASE VALIDATOR 376
CÓDIGO 20. ESTRUCTURA DE LA CLASE STREAMEXTRACTOR 377
CÓDIGO 21. FIRMA DEL MÉTODO "TORDFELEMENT" 378
CÓDIGO 22. EJEMPLO DE PETICIÓN DE VALIDACIÓN DE FLUJOS 383
CÓDIGO 23. EJEMPLO SIMPLIFICADO DE RESPUESTA DEL SERVIDOR PARA UN RESULTADO 384
CÓDIGO 24. EJEMPLO DE RESPUESTA DEL SERVIDOR ANTES DE CERRAR LA CONEXIÓN 386
CÓDIGO 25. HABILITANDO EL USO DE WEBSOCKETS EN HTTPS4 401
CÓDIGO 26. CADA MENSAJE WS DEL CLIENTE SE ENCOLA PARA SER PROCESADO EN UN NUEVO FLUJO DE VALIDACIONES SALIENTE 402
CÓDIGO 27. VALIDACIÓN DE FLUJOS EN EL SERVIDOR - PASO 0 402
CÓDIGO 28. VALIDACIÓN DE FLUJOS EN EL SERVIDOR - PASO 1 402
CÓDIGO 29. VALIDACIÓN DE FLUJOS EN EL SERVIDOR - PASO 2 403
CÓDIGO 30. VALIDACIÓN DE FLUJOS EN EL SERVIDOR - PASO 3 403
CÓDIGO 31. VALIDACIÓN DE FLUJOS EN EL SERVIDOR - PASO 4 403
CÓDIGO 32. VALIDACIÓN DE FLUJOS EN EL SERVIDOR - PASO 5 404
CÓDIGO 33. DEFINICIÓN DE LA RUTA DE VALIDACIÓN DE DATOS RDF CON EL DSL DE RHO 404
CÓDIGO 34. DEFINICIÓN DE LA DOCUMENTACIÓN A GENERAR PARA PARÁMETROS DE ENTRADA DE LA CLASE URL 405
CÓDIGO 35. CONFIGURACIÓN DE LOGBACK PARA EL APPENDER DE CONSOLA 406
CÓDIGO 36. CONFIGURACIÓN DE LOGBACK PARA EL APPENDER DE FICHEROS 407
CÓDIGO 37. ARCHIVO DE CONFIGURACIÓN DE NGINX COMPATIBLE CON LAS RUTAS DE REACT 409
CÓDIGO 38. INSTANCIA Y CONFIGURACIÓN GLOBALES DE AXIOS 410
CÓDIGO 39. CÓDIGO DE INICIALIZACIÓN DEL CONTEXTO 411
CÓDIGO 40. MONTAJE DEL APPLICATIONPROVIDER EN EL ÁRBOL DE COMPONENTES 411
CÓDIGO 41. INICIALIZACIÓN CON LOS DATOS DEL CONTEXTO EN EL COMPONENTE SHEXVALIDATE 412
CÓDIGO 42. ACTUALIZACIÓN MANUAL DE UNA IMAGEN DOCKER DEL SISTEMA 419

El presente documento guía al lector a través de las diferentes fases que componen la metodología de desarrollo Métrica v.3 **[2]**, que se ha empleado para completar este proyecto.

Métrica v.3

Métrica es una metodología para la planificación, desarrollo y mantenimiento de sistemas de información. El Consejo Superior de Informática (CSI) de España creó la metodología con el fin de tener un modelo común a seguir en los proyectos informáticos para la administración pública.

La tercera versión de Métrica fue publicada en el año 2001 y su documentación se encuentra disponible públicamente a cualquier tercero que quiera aplicarla. Esta memoria parte de la base establecida por Métrica v.3, si bien se ha adaptado al contexto particular de este proyecto.

Resumen de los capítulos del proyecto

A continuación, se resume brevemente el contenido de cada capítulo de la memoria, basado en los manuales de Métrica v.3.

Capítulo 1: Planificación del Sistema de Información (PSI)

La PSI comprende el contexto completo del proyecto, incluyendo los antecedentes, responsables y la solución propuesta.

Se pone al lector del documento en situación y se explica, a alto nivel, qué módulos se han desarrollado así como sus funcionalidades y el rol del cliente final del proyecto.

Capítulo 2: Definición de la Arquitectura Tecnológica

Este apartado está destinado a concretar la solución tecnológica que se va a emplear a lo largo del resto del documento para el desarrollo del sistema. La definición de la arquitectura es originalmente parte de la PSI, pero se emancipa a otro apartado para mayor claridad, puesto que el proyecto hace uso de una gran cantidad de herramientas.

Antes de adentrarse en el desarrollo del sistema, se hace una comparativa de alternativas tecnológicas para verificar que la arquitectura elegida es la mejor disponible.

Capítulo 3: Estudio de Viabilidad del Sistema (EVS)

El EVS parte de la información adquirida en el PSI y contrasta las alternativas disponibles para el caso de uso principal del proyecto, la validación de RDF.

Capítulo 4: Análisis del Sistema de Información (ASI)

El ASI se acerca con mucho más nivel de detalle en el sistema informático. Se define el sistema final y se formalizan sus requisitos. Así mismo, se crean modelos base para las clases, interfaces o el plan de pruebas del sistema.

Se ha incluido en este capítulo la sección de planificación y gestión del TFM, que incluye la información sobre el cronograma, presupuesto y riesgos del proyecto.

Capítulo 5: Diseño del Sistema de Información (DSI)

En el DSI se concretan los modelos definidos en el ASI para definir los ítems (clases, pruebas, módulos, etc.) que formarán parte del sistema final.

Capítulo 6: Construcción del Sistema de Información (CSI)

El CSI comprende la codificación del sistema informático final, incluyendo la definición del entorno de desarrollo empleado, los estándares seguidos, los resultados del plan de pruebas y los manuales de usuario.

Capítulo 7: Implantación y Aceptación del Sistema (IAS)

La IAS se incluye puesto que el presente proyecto se elabora para un cliente real e incluye los planes seguidos para desplegar y mantener el sistema en la infraestructura del cliente, así como datos sobre la recepción del sistema.

Capítulo 8: Conclusión y Ampliaciones

Capítulo final donde se discuten, a modo de resumen, los resultados obtenidos y posibles líneas de desarrollo e investigación a desarrollar a partir del proyecto actual.

Anexos

Compendio de documentación específica excluida del documento general para mantener la legibilidad. Incluye el plan de gestión y las hojas de riesgos, así como otros informes, la bibliografía y un resumen de los entregables del proyecto.

Capítulo 1 Planificación del Sistema de Información

Fase de Planificación

PSI

1.1 PSI 1: INICIO DEL PLAN DE SISTEMAS DE INFORMACIÓN

1.1.1 PSI 1.1: Análisis de la Necesidad del PSI

1.1.1.1 *Definición inicial del proyecto*

Este proyecto consiste en desarrollar una biblioteca de código capaz recibir datos RDF **[3]** que llegan a través de un flujo o *stream* local o externo. Los datos son, a continuación, validados mediante el mecanismo *Shape Expressions* **[4]** y como resultado se emite un nuevo flujo que contiene los resultados de cada validación. Los resultados pueden, a continuación, seguir siendo procesados por el programa que invocó la biblioteca.

El concepto flujo de datos es abstracto, pero se puede entender como cualquier conjunto de elementos con un inicio definido pero no necesariamente un final **[5]**. Es decir, un grupo de ítems potencialmente infinitos cuyos contenidos se van haciendo disponibles a lo largo del tiempo: según nos lleguen, según se vayan generando, etc. Estos ítems sufren diversas mutaciones según pasan por filtros *(filters)* hasta que son consumidos o descartados en un sumidero *(sink).* Un listado de números en memoria o un grupo de ficheros pueden ser un flujo en tanto que son grupos que datos disponibles localmente que se van consumiendo, por otra parte, una conexión en vivo a través de *WebSockets* **[6]**, *Apache Kafka* **[7]** u otras soluciones *AMQP* **[8]** son ejemplos de flujos con un final indeterminado. La solución desarrollada debe aportar una interfaz común para la recepción y validación de todo tipo de flujos, priorizando aquellos que:

a) Se encuentren en una ubicación remota, accesible a través de la red.
b) Se vayan emitiendo a lo largo del tiempo, con posibilidad de validar datos emitidos en tiempo real.

En síntesis, la biblioteca desarrollada debe actuar como un gran filtro capaz de mutar un flujo remoto de datos RDF en un flujo de resultados, dada una *Shape Expression* contra la que validar los datos del flujo de entrada. El flujo de salida contendrá los resultados de la validación, así como otros metadatos de utilidad para que terceras partes puedan seguir transformando o consumiendo los resultados para, por ejemplo, archivar los resultados obtenidos, lanzar alertas en casos de fallos de validación, etc.

Además, se partirá del trabajo realizado previamente en WESO **[9]** para ampliar y documentar el proyecto *RDFShape* **[10]**: un API que ofrece varias herramientas para operar, transformar, consultar y validar datos RDF. El API cuenta a su vez con un cliente web **[11]** que implementa varias visualizaciones e interfaces de usuario. La biblioteca desarrollada se integrará dentro del servidor de *RDFShape* de forma que se puedan invocar validaciones de *streams* a través del API. Finalmente, se harán los cambios pertinentes en el cliente para poder solicitar y visualizar las validaciones de flujos en una interfaz accesible y amigable para su uso en cursos y charlas de la Escuela de Ingeniería Informática así como para la promoción de las tecnologías semánticas y del propio grupo WESO.

1.1.1.2 *Justificación del proyecto*

El proyecto, pese a ser de desarrollo, se ha realizado con el grupo WESO y, por tanto, se mueve en un ámbito cercano a la investigación. El campo de validación de datos RDF ha visto grandes avances con la aparición de *Shape Expressions* (ShEx) y *Shapes Constraint Language* (SHACL) **[12]**, pero las soluciones existentes que apliquen estas tecnologías en casos de uso reales son todavía escasas e inmaduras. Uno de estos casos de uso es el procesamiento con posibilidad de validación de flujos de datos RDF.

Los flujos de datos son una parte importante de la investigación y comunidad en torno a la web semántica. Por ejemplo, Wikipedia, la fuente de conocimiento colaborativa de mayor relevancia, y basada en la infraestructura semántica de Wikidata **[13]**, publica sus últimos cambios en forma de un *stream* consumible programáticamente desde su API **[14]**. Sin embargo, pese a que el procesamiento de flujos es una necesidad común y que las tecnologías para validar RDF comienzan a ser lo suficientemente maduras, no hay una única solución fiable, extensible y de código abierto para la validación de flujos de datos.

Para el desarrollo de una herramienta como la necesitada, uno podría pensar en emplear las herramientas de validación existentes, junto con de bucles y otras estructuras de control no secuenciales, para implementar una validación de varios conjuntos de datos. Sin embargo, este enfoque no se sostiene cuando tratamos con flujos potencialmente interminables con datos que no están en memoria y es necesario aproximarse al problema con la perspectiva de flujos desde el principio.

1.1.2 PSI 1.2: Identificación del Alcance del PSI

El objetivo principal es la construcción de una herramienta que permita validar flujos de datos RDF y que cumpla los siguientes objetivos estratégicos:

1. Debe ser desarrollada a alto nivel, comprometiéndose con un nivel de abstracción elevado que permita:
    a. Implementaciones personalizadas para procesar cualquier flujo de entrada de cualquier fuente (listas, ficheros, *Kafka*, soluciones *AMQP*, *WebSockets*, etc.).
    b. Implementaciones personalizadas para validar RDF con cualquier mecanismo de validación.

    El sistema final contará con implementaciones para varios tipos de flujos de entrada pactados con el cliente pero centrándose en *Apache Kafka*. Así mismo, se dará prioridad y proporcionará una implementación para la validación mediante *Shape Expressions*, la tecnología de validación diseñada y promovida desde el grupo WESO.

2. Debe estar ideada para obtener el mayor alcance y difusión posible, con el fin de mejorar la calidad de vida de los investigadores del campo y dar promoción al grupo WESO. Se opta por construir una biblioteca de código en lugar de un programa concreto y restringido, de manera que cualquier programador pueda usar la biblioteca o modificarla para su caso de uso particular. Para fomentar el uso y difusión de la biblioteca:
    a. El código fuente se publicará en las cuentas de WESO y la solución compilada estará disponible para sistemas de gestión de dependencias.
    b. Se trabajará en una documentación completa, inteligible y accesible online.
    c. Se priorizarán las tecnologías, de entre las disponibles y aceptadas por el cliente, que permitan el uso de la biblioteca a un número máximo de usuarios.

3. Se integrará dentro de la herramienta *RDFShape* del grupo WESO, a modo de demostración del producto.
    a. Se podrá invocar la herramienta de validación desde *RDFShape*, bien programáticamente a través del API o a través de la interfaz web.
    b. Se realizarán las mejoras pertinentes, además de las consideradas por el cliente, en la herramienta *RDFShape*, para facilitar su despliegue y uso por terceras partes.

1.1.3 PSI 1.3: Determinación de responsables

Se definen los siguientes responsables generales del proyecto, junto con sus responsabilidades:

- **Director del proyecto**
  - Definir y corregir desviaciones en los requisitos del proyecto.
  - Validar la consecución de objetivos intermedios a lo largo del desarrollo del proyecto.
- **Grupo de investigación Web Semántica Oviedo (WESO) y Universidad de Oviedo**
  - Colaborar en las labores del director como supervisor del proyecto.
  - Ayudar en la formación en las tecnologías del proyecto.
  - Proporcionar los módulos iniciales de la aplicación web *RDFShape*.
  - Proporcionar y mantener la infraestructura de pruebas y de producción de la demo del sistema.
  - Suministrar una serie de usuarios, escogidos entre alumnos e investigadores, para las pruebas de usabilidad del sistema.
- **Alumno**
  - Definir el contexto del sistema y la solución propuesta.
  - Formalizar los requisitos del proyecto.
  - Seleccionar las alternativas tecnológicas del sistema final.
  - Analizar, desarrollar, construir, desplegar y elaborar los planes de mantenimiento del sistema.
  - Realizar las tareas de gestión del proyecto.
  - Elaborar y revisar la memoria, así como el resto de documentación asociada.

1.1.4 PSI 1.4: Solución Propuesta

La solución propuesta debe responder a la necesidad del proyecto y llegar a las metas de alcance previamente definidas.

Se opta por una solución multimodular, constituida por tres subsistemas con identidad propia. El primero de los módulos estará desarrollado exclusivamente para el proyecto de validación de flujos, mientras que los otros dos partirán de la infraestructura disponible de *RDFShape* para elaborar una aplicación web que sirva tanto de prueba de concepto como de escaparate publicitario.

1. Módulo Validador

Herramienta capaz de validar flujos de datos RDF. Ha de tener las siguientes características:

- Debe distribuirse como una biblioteca de código abierto y disponible a través de los sistemas de gestión de dependencias del lenguaje objetivo. De esta manera, podrá ser utilizada y modificada de forma independiente tanto en el proyecto *RDFShape* como por cualquier tercera parte para su caso de uso particular.
- Debe proporcionar una base de código mutable y extensible a cualquier tipo de flujo de RDF y mecanismo de validación.
    - Debe incorporar, como mínimo, una implementación capaz de recibir y validar flujos RDF remotos en tiempo real, transmitidos a través de *Apache Kafka* y otras tecnologías de elección del cliente.
    - Debe incorporar, como mínimo, una implementación del validador capaz de validar los datos RDF mediante *Shape Expressions*.
- Debe ser directamente compatible con la infraestructura tecnológica del *backend* de la aplicación *RDFShape* del cliente.
    - Se usará como lenguaje de programación aquel con mejores tecnologías disponibles para el caso de uso a tratar, de entre los directamente compatibles con la *Java Virtual Machine* (*JVM*) y, por tanto, con *RDFShape*.
- Debe contar con una documentación completa, tanto dentro del propio código como documentación online en un sitio web propio o proporcionado por WESO.

2. Módulo API

Programa ejecutado en la infraestructura de WESO que expone el API utilizado en el *backend* de *RDFShape*. Además de incorporar el módulo de validación, es refactorizado para incluir varias mejoras de funcionalidad y despliegue. Ha de tener las siguientes características:

- Debe incorporar el código desarrollado para el módulo validador y exponerlo a través del API. Para transmitir los resultados de validaciones de flujos, se considerarán alternativas como *WebSockets* o *Server Sent Events* **[15]**.

- Debe contar con una documentación completa, tanto dentro del propio código como documentación online en un sitio web propio o proporcionado por WESO.
- El API debe contar con una documentación pública, construida sobre la *OpenAPI Specification* **[16]** y disponible online a través de la cuenta de WESO en *SwaggerHub* **[17][18]**.
- Debe contar con un sistema de registros o *logs* que almacene información sobre las invocaciones del API por parte de terceros.
- Debe contar con un mecanismo de comunicación con un SGBD para almacenar enlaces permanentes o *permalinks* generados por usuarios.
- Debe ser refactorizado para facilitar su despliegue mediante tecnología de contenedores **[19]**.

3. Módulo Cliente

Portal web desplegado en la infraestructura de WESO, ofrece una interfaz web para interactuar con el API de *RDFShape* de forma amigable y con visualizaciones. Ha de tener las siguientes características:

- Debe refactorizar o incorporar nuevas pantallas para invocar la validación de flujos a través de la interfaz web y visualizar los resultados obtenidos.
- Debe contar con un sistema de enlaces permanentes o *permalinks*, que proporcione un mecanismo para que los usuarios puedan acceder a sus operaciones previas mediante enlaces cortos.
- Debe ser refactorizado para facilitar la reusabilidad de los componentes en la construcción del sitio web.
- Debe ser refactorizado para mejorar la usabilidad del sitio web en base a los requisitos del cliente y el *feedback* obtenido del uso de la herramienta en charlas y cursos.
- Debe ser refactorizado para poseer un nivel de accesibilidad *AA*, en base a los estándares del *WCAG* **[20]**.
- Debe ser refactorizado para facilitar su despliegue mediante tecnología de contenedores.

1.2 PSI 2: DEFINICIÓN Y ORGANIZACIÓN DEL PSI

1.2.1 PSI 2.1: Especificación del Ámbito y Alcance

En base a los objetivos estratégicos vistos previamente en *1.1-PSI 1: Inicio del Plan de Sistemas de Información*, el proyecto se divide en fases con objetivos concretos. Dichas fases, listadas a continuación, se corresponden a grandes rasgos con las planificadas en *4.3.1.3-Planificación Inicial. WBS*, basadas a su vez en Métrica v.3.

Fase 1: Estudios iniciales

Independientemente de los requisitos finales, la naturaleza del proyecto y la necesidad de integrarlo y desplegarlo en proyectos e infraestructuras ya existentes hacen que sea necesaria una etapa inicial que posibilite las siguientes etapas del proyecto. Los objetivos principales de esta fase son:

- **Adquisición de conocimientos sobre el dominio de la web semántica**
  Procesamiento y sintaxis de RDF, procesamiento y sintaxis *Shape Expressions*, validación de RDF y herramientas existentes.
- **Adquisición de conocimientos sobre manipulación de flujos y el envío y recepción de datos en remoto y tiempo real**
  Acopio de protocolos, estándares, tecnologías y herramientas aplicables a la comunicación de flujos de datos RDF de cara a las nuevas funcionalidades de *RDFShape*.
- **Formación en el entorno de trabajo del cliente**
  Conocer las tecnologías sobre las que está construido y desplegado *RDFShape* para su posterior mejora.

Fase 2: Preparación del sistema

Concentra las etapas de Métrica v.3 previas al desarrollo técnico del sistema. Los objetivos principales de esta fase son:

- **Planificar del sistema**
  Describir de forma ordenada la necesidad, justificación y contexto general del proyecto, que llevan a la solución propuesta.
- **Definir la arquitectura del sistema**
  Identificar las necesidades y restricciones tecnológicas del sistema con el fin de seleccionar la arquitectura y tecnologías del proyecto.
- **Realizar el estudio de viabilidad del sistema**
  Contrastar las tecnologías disponibles frente a las seleccionadas, marcando las ventajas e inconvenientes y razonando las decisiones finales.

Fase 3: Análisis y diseño del sistema

Comprende las etapas homónimas de Métrica v.3, donde se prepara y refina la información necesaria para el desarrollo final. Los objetivos principales de esta fase son:

- **Establecer los requisitos funcionales y no funcionales del sistema**
  Identificar y categorizar los requisitos funcionales y no funcionales del sistema
- **Identificar los actores del sistema y definir los casos de uso**
- **Describir los módulos del sistema**
  Identificar y definir los módulos y subsistemas finales, así como las interfaces entre ellos. Esto incluye diagramas de paquete, despliegue y bocetos de las interfaces de usuario.
- **Diseñar las clases del sistema**
- **Establecer el diseño final de los datos en el SGBD**
- **Establecer un plan de pruebas**
  Pruebas unitarias de cada uno de los módulos, así como pruebas de integración y de accesibilidad y usabilidad del módulo cliente.

Fase 4: Construcción del sistema

Consiste en el desarrollo software de cada uno de los módulos del sistema, materializando lo definido en las fases de análisis y diseño. Los objetivos principales de esta fase son:

- **Construcción del módulo validador**
  - Biblioteca de validación de flujos
    - Construir y publicar la biblioteca de validación de flujos.
    - Permitir a terceras partes visualizar el código y acceder a la documentación del módulo a través de la página del proyecto y las redes públicas de WESO.
    - Permitir a cualquier programador que quiera hacer uso de la biblioteca incluirla en su fichero de dependencias para descargarla automáticamente.
    - Permitir a los usuarios de la biblioteca hacer uso de las implementaciones ya disponibles y ampliarlas para nuevas tecnologías de flujos o de validación.
  - Stream de pruebas
    - Construir y publicar un flujo de datos en Apache Kafka que permita realizar las pruebas de integración del módulo validador contra un flujo de datos externo.
    - Automatizar el despliegue del flujo de datos.
    - Permitir configurar el contenido y la periodicidad de los datos emitidos.
    - El stream publicará datos RDF de ejemplo usando la tecnología escogida por el cliente e implementada en el módulo validador.
- **Refactorización y ampliación de *RDFShape***
  - Módulo API
    - Refactorizar y documentar el código existente en el servidor.

        - Permitir a terceras partes solicitar, a través del API, la validación de flujos de datos publicados en internet y personalizar el comportamiento del sistema en función de los resultados. Los resultados de la validación serán recibidos en tiempo real según sean generados y enviados.
        - Permitir a terceras partes visualizar el código, así como su documentación y la del API a través de la página del proyecto y las redes públicas de WESO.
        - Permitir a terceras partes solicitar, a través del API, enlaces acortados que redirijan a operaciones disponibles en el API.
        - Permitir a los administradores del sistema consultar los registros de uso del API, incluyendo la ruta solicitada, el momento de la solicitud y el origen de las peticiones.
        - Permitir a los administradores del sistema desplegar el API mediante tecnología de contenedores.

    - Módulo Cliente
        - Refactorizar y documentar el código existente en el cliente, facilitando la reutilización de componentes para construir las interfaces.
        - Permitir a los usuarios mantener los datos que están operando según navegan entre pantallas, junto con otras mejoras de usabilidad.
        - Permitir a los usuarios solicitar, a través de una interfaz gráfica, la validación de flujos de datos publicados en internet y personalizar el comportamiento del sistema en función de los resultados. Los resultados de la validación serán visualizados en tiempo real según sean generados y enviados.
        - Permitir a los usuarios solicitar, a través de una interfaz gráfica, enlaces acortados que redirijan a operaciones disponibles en el API.
        - Permitir a los usuarios navegar por una interfaz que cumple los criterios de accesibilidad *AA* del *WCAG*.
        - Permitir a los administradores del sistema desplegar el cliente mediante tecnología de contenedores.
    - Permitir al tutor del proyecto aplicar las mejoras en la herramienta *RDFShape* a las clases de Web Semántica de la asignatura del Máster en Ingeniería Web.

- **Ejecutar el plan de pruebas**
  Ejecutar las pruebas y documentar los resultados, corrigiendo las desviaciones encontradas.
- **Redactar los manuales del sistema**
  Se contará con manuales de instalación y ejecución, así como de usuario y de programador. El contenido de los manuales se apoyará en la documentación del código de los módulos y los contenidos disponibles en línea.
- **Desplegar el sistema final en los servidores de pruebas**
  Según se tengan construidos los módulos y una vez se automatice el despliegue con contenedores, se harán despliegues periódicos de prueba en los servidores del cliente para detectar errores y facilitar las pruebas de integración y sistema.

Fase 5: Implantación y aceptación del sistema

Comprende el despliegue final y verificación del sistema en la infraestructura del cliente. Los objetivos principales de esta fase son:

- **Desplegar el sistema final en los servidores de producción**
  - Permitir a los usuarios acceder a la aplicación web de forma segura a través de HTTPS **[21]**.
    - Permitir a los administradores del sistema obtener y renovar los certificados SSL de manera automática.
  - Permitir a los administradores del sistema desplegar todos los módulos de la aplicación web de forma automatizada mediante contenedores.
    - Permitir a los administradores del sistema monitorizar el estado de la aplicación web.
    - Permitir a los administradores del sistema reiniciar los módulos que se encuentren caídos de manera automática.
- **Promocionar el proyecto, el grupo de investigación WESO y la Escuela de Ingeniería Informática**
  - Permitir al tutor del proyecto emplear el sistema en las clases de Web Semántica de la asignatura del Máster en Ingeniería Web, así como en otras conferencias y círculos investigadores o docentes.
  - Permitir a cualquier usuario, alumno, docente o investigador acceder libremente a la aplicación web de *RDFShape* a modo de demo del módulo validador y el resto de las herramientas refactorizadas.

Fase 6: Gestión de proyecto y seguimiento con el cliente

Esta fase está presente a lo largo de todo el desarrollo del proyecto, desde la reunión inicial hasta el visto bueno final con el tutor. Comprende las tareas inherentes a la gestión del proyecto, así como la supervisión y seguimiento del tutor e interesados. Los principales objetivos de esta fase son:

- **Llevar a cabo la gestión del proyecto**
  Identificar las necesidades del proyecto y elaborar el cronograma, presupuesto y registro de riesgos, junto con los planes de seguimiento de estos.
  - Se elaborarán hasta tres versiones de la planificación: inicial, intermedia y final.
  - Se elaborarán hasta dos versiones del presupuesto: inicial y final.
  - Se registrarán hasta quince riesgos y se hará un seguimiento pormenorizado de cinco de ellos, documentado a través de hojas de riesgos.
  - Se corregirán las desviaciones encontradas a lo largo del desarrollo con las reservas de tiempo destinadas a ello.
- **Llevar a cabo el seguimiento del proyecto con el tutor**

Permitir al tutor ser partícipe del proyecto contando con él para la adquisición de requisitos y organizando reuniones periódicas de control y resolución de dudas.

1.2.2 PSI 2.2: Organización del PSI

1.2.2.1 *Equipos de trabajo*

La siguiente tabla especifica los equipos de trabajo partícipes del proyecto, junto con sus funciones principales:

USUARIO	PERFIL	FUNCIÓN
EQUIPO DE SUPERVISIÓN		
José Emilio Labra Gayo	Investigador principal (IP) del grupo WESO de la Universidad de Oviedo	- Supervisión de todas las fases del proyecto. - Ayuda en el establecimiento de requisitos iniciales. - Selección de la tecnología de trasmisión de flujos prioritaria. - Selección de la tecnología de validación RDF prioritaria.
EQUIPO DE DESARROLLO		
Eduardo Ulibarri Toledo (alumno)	Ingeniero informático	- Seguimiento del proyecto con el equipo de supervisión. - Selección y formación en las tecnologías del sistema. - Análisis y diseño del sistema. - Codificación de los módulos del proyecto. - Redacción de los manuales del sistema.
EQUIPO DE PRUEBAS		
Eduardo Ulibarri Toledo (alumno)	Ingeniero informático	- Elaboración del plan de pruebas del sistema. - Codificación y ejecución del plan de pruebas.
Otro personal de la Universidad / usuarios seleccionados	Informático	- Pruebas accesibilidad y usabilidad del sistema.
EQUIPO DE SISTEMAS		
Eduardo Ulibarri Toledo (alumno)	Ingeniero informático	- Pruebas de integración del sistema. - Adaptación del sistema a tecnología de contenedores.

		- Plan de implantación y despliegues del sistema de los entornos de pruebas y producción.
EQUIPO DE GESTIÓN		
Eduardo Ulibarri Toledo (alumno)	Ingeniero informático	- Gestión inicial del proyecto: cronograma, presupuesto y riesgos. - Seguimiento del proyecto: corrección de las desviaciones de cronograma, presupuesto y riesgos.
EQUIPO DE DISEÑO		
Eduardo Ulibarri Toledo (alumno)	Ingeniero informático	- Bocetos y diseños finales de las interfaces de usuario. - Navegabilidad de la aplicación web. - Ayuda en las pruebas de usabilidad.
EQUIPO DE REDACCIÓN		
Eduardo Ulibarri Toledo (alumno)	Ingeniero informático	- Redacción de la memoria y documentación asociada. - Revisión de formato y contenidos de la memoria y documentación asociada.

Tabla 1. Equipos de trabajo del proyecto

1.2.2.2 *Material*

El material identificado como necesario para llevar a cabo el proyecto es el siguiente:

1. Una máquina para la codificación y la ejecución de las pruebas del sistema. Este equipo será aportado por el propio alumno.
2. Un servidor que permita el despliegue de la aplicación web *RDFShape*, tanto en entornos de pruebas del sistema como en el despliegue de producción final. Este servidor debe ser aportado por la propia escuela, junto con una dirección IP pública y nombre DNS que permitan acceder a él.

1.2.2.3 *Entregables*

A continuación, se listan los principales entregables del proyecto junto con las fechas inicialmente establecidas para ellos (ver *4.3.1.3-Planificación Inicial. WBS*). Los hitos del proyecto, muchos de los cuales se corresponden con estas entregas, pueden consultarse en el documento de *Microsoft Project* entregado junto a la memoria.

Fecha	Entregable
16/09/2021	Cronograma del proyecto
14/02/2022	Módulo validador: biblioteca de validación de flujos RDF
17/02/2022	Stream parametrizable para pruebas del módulo validador
09/03/2022	Módulo API: backend de la plataforma RDFShape refactorizado y ampliado
24/03/2022	Módulo cliente: frontend de la plataforma RDFShape refactorizado y ampliado
26/04/2022	Manuales del sistema
02/05/2022	Sistema final desplegado en la infraestructura del cliente
10/05/2022	Memoria del proyecto y documentación asociada

Tabla 2. Calendario de entregas principales del proyecto

1.3 PSI 3: ESTUDIO DE LA INFORMACIÓN RELEVANTE

1.3.1 PSI 3.1: Selección y Análisis de Antecedentes

1.3.1.1 *Web semántica y estándares asociados*

Es el objetivo de este proyecto apoyarse en estándares y tecnologías de la web semántica existentes y crear nuevas herramientas que faciliten trabajar con ellos y los promocionen. Desde su creación de la mano de Tim Berners-Lee en 1989 durante su trabajo en el CERN **[22]**, internet (o la *World Wide Web*) se ha expandido y generalizado hasta convertirse en un pilar de cómo las sociedades modernas entendemos los fenómenos sociales, laborales, ociosos, informativos, deportivos, etc.

Si bien desde la perspectiva de un usuario no técnico, internet funciona "correctamente", la realidad es que construir una red global en común ha sido una ardua tarea y muchas decisiones han sido fruto del contexto geopolítico y tecnológico del momento. En la actualidad, surge el concepto de la web semántica: un internet cuyo contenido haya sido homogeneizado mediante el uso de estándares que permitan que los datos sean anotados para su lectura y procesamiento por parte de sistemas informáticos, no sólo personas. Los esfuerzos para estandarizar los contenidos y materializar esta visión de internet están liderados por el propio Tim Berners-Lee **[23]**, el *World Wide Web Consortium* (W3C), del que es director, y la *World Wide Web Foundation* **[24]**, de la que es cofundador.

La web semántica es una web con significado, construida sobre estándares, formatos y herramientas universales, de forma que sus contenidos puedan ser expresados en lenguaje natural dirigido a humanos al tiempo que son complementados con (meta)datos que permiten a los programas informáticos comprender, procesar y relacionar la información. Los pilares en que se apoya la web semántica son *Resource Description Framework* (RDF) y *Web Ontology Language* (OWL) **[25]**, capaces de representar entidades y las relaciones entre ellas.

Los datos RDF emplean una sintaxis en común junto con un sistema de prefijos y nombres que permiten identificar de forma inequívoca cada entidad y relación representada en los datos enlazados. Hasta la fecha, se cuenta con varias sintaxis o formatos para serializar y representar datos RDF de forma legible **[26]**, las principales son: *Turtle*, *N-triples*, *N-Quads*, *JSON-LD*, *RDF-XML* y *RDF-JSON*. Además de RDF, existen otra serie de tecnologías como microdatos y *RDFa* **[27]**, pensados para incrustar metadatos semánticos en *HTML*.

El siguiente ejemplo, disponible en la herramienta *RDFShape*, muestra a dos entidades, Alice y Bob, sus datos asociados y relaciones entre sí. A través de la sintaxis *Turtle* y de una serie de prefijos comunes y conocidos, es posible universalizar la forma en la que expresamos que Alice y Bob son personas, se conocen y tienen una fecha de nacimiento determinada.

```
@prefix :       <http://example.org/> .
@prefix schema: <http://schema.org/> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .

:alice  a                   foaf:Person ;
        schema:birthDate    "1995-02-01"^^xsd:date .

:bob    a                   foaf:Person ;
        schema:birthDate    "1990-07-04"^^xsd:date ;
        foaf:knows          :alice .
```

Ilustración 1. Ejemplo de dos personas y sus relaciones - RDF (Turtle)

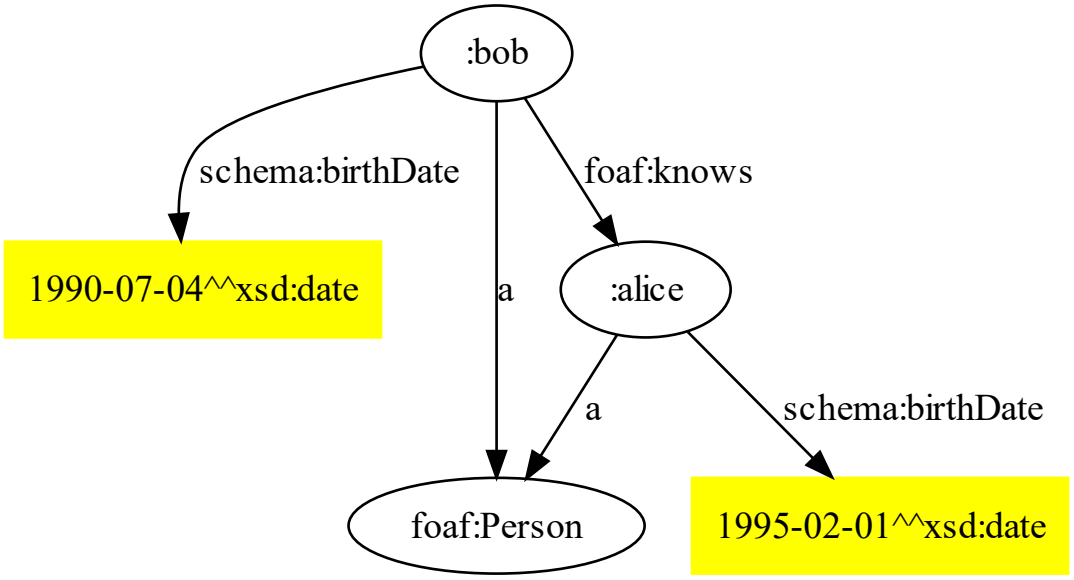


Ilustración 2. Ejemplo de dos personas y sus relaciones - RDF (imagen)

1.3.1.2 *Validación de datos RDF*

Hoy en día se cuentan con estándares inteligibles por máquinas para decorar datos enlazados pero los casos de uso reales necesitan de mecanismos para validar y comprobar que estos datos son correctos si queremos usarlos como soporte primario de nuestra información. Cuando validamos RDF contra una serie de condiciones, se dice que validamos RDF contra un esquema. A pesar de los avances del W3C en materia de RDF y datos enlazados, en forma de estándares y recomendaciones, los esfuerzos en validación de datos aún pertenecen al ámbito de la investigación.

Las especificaciones más punteras en validación de datos RDF son *Shape Expressions* (*ShEx*) y *Shapes Constraint Language* (*SHACL*), impulsadas por el *ShEx Community Group* **[28]** y *SHACL Community Group* **[29]**, respectivamente. En el año 2017, *SHACL* obtiene el estándar de recomendación del W3C mientras que actualmente *ShEx* sigue siendo un esfuerzo de la comunidad. Ambas herramientas cuentan con una sintaxis amplia, sujeta a cambios y revisiones frecuentes, si bien el mayor inconveniente para llevarlas a la práctica es la escasez de implementaciones disponibles. Es decir, aún faltan herramientas lo suficientemente maduras que implementen la sintaxis de *ShEx* o *SHACL* y la utilicen para la validación de RDF.

Durante el desarrollo del proyecto, nos centraremos en el uso de *Shape Expressions* como mecanismo preferente para la validación de RDF. *Shape Expressions*, cuya especificación se encuentra públicamente disponible **[30]** en los repositorios del *ShEx Community Group*, es una tecnología en la que el grupo de investigación WESO de la Universidad de Oviedo colabora activamente mediante el desarrollo de implementaciones de *ShEx* y la participación dentro del *ShEx Community Group*.

Actualmente, estas son las implementaciones más relevantes de *Shape Expressions* **[31]**:

- *shex-s* **[32]** / *SHaclEX* **[33]**: Implementaciones en *Scala* de *ShEx*, así como de *ShEx* y *SHACL*, respectivamente. Desarrolladas y mantenidas por el grupo WESO de la Universidad de Oviedo. A lo largo del estudio tecnológico, se priorizará *SHaclEX* por su familiaridad con el alumno y tutor, así como por estar ya en uso en la herramienta *RDFShape*.
- *shex.js* **[34]**: Paquete *JavaScript* para la validación y visualización de *ShEx*, desarrollado y mantenido por Eric Gordon Prud'hommeaux, empleado del W3C.
- *shex-java* **[35]**: Implementación en *Java* de *ShEx*, desarrollado y mantenido por personal de la Universidad de Lille.
- *PyShex* **[36]**: Intérprete de *ShEx* en *Python*, desarrollado y mantenido personal de la Universidad Johns Hopkins, en Maryland.
- *Apache Jena* **[37]**: Implementación de *ShEx* bajo el paraguas de la *Apache Software Foundation* **[38]**.

Al igual que ocurría con RDF, *ShEx* posee varias sintaxis para serializar esquemas: compacta (*ShExC*) y JSON (*ShExJ*), que comparten varios elementos en común con las sintaxis RDF. Así mismo, se sirven de los mismos mecanismos de prefijos e identificadores para definir los recursos y las relaciones a las que hacen referencia. Finalmente, una estructura clave-valor, conocida como *ShapeMap*, indica al programa qué entidades se quieren validar contra qué condiciones.

El siguiente ejemplo, disponible en la herramienta *RDFShape*, verifica que Alice y Bob, del ejemplo anterior, tienen una fecha de nacimiento válida (tipo *xsd:date*) y que las entidades a las que conocen son, a su vez, personas.

```
PREFIX :       <http://example.org/>
PREFIX schema: <http://schema.org/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema>
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>

:User {
  schema:birthDate     xsd:date  ;
  foaf:knows          IRI @:User*
}
```

Ilustración 3. Validación de dos personas y sus relaciones representados mediante ShEx – Esquema (ShExC)

```
:alice@:User,:bob@:User
```

Ilustración 4. Validación de dos personas y sus relaciones representados mediante ShEx - ShapeMap

–	Node↑	Shape↑	Status↑
–	*:alice*	*:User*	*Valid*
• "1995-02-01"^^<http://www.w3.org/2001/XMLSchemadate> has datatype <http://www.w3.org/2001/XMLSchemadate> • "1995-02-01"^^<http://www.w3.org/2001/XMLSchemadate> has datatype <http://www.w3.org/2001/XMLSchemadate>			
–	*:bob*	*:User*	*Valid*
• <http://example.org/alice> is an IRI • "1990-07-04"^^<http://www.w3.org/2001/XMLSchemadate> has datatype <http://www.w3.org/2001/XMLSchemadate>			

Ilustración 5. Validación de dos personas y sus relaciones representados mediante ShEx - Resultados (RDFShape)

1.3.1.3 *Flujos de datos*

1.3.1.3.1 Comunicación mediante flujos de datos

A diferencia de la web semántica y los sistemas en que se apoya, la problemática de comunicación a través de flujos de datos está extensamente documentada y madura en la actualidad dada la amplia cantidad de casos de uso a las que es aplicable (trasmisión de datos, sistemas de alertas y monitorización, envío de mensajes, etc.). Algunas de las tecnologías más relevantes en la comunicación de *streams* son:

- *Apache Kafka*: Desarrollado por LinkedIn, fue donado y migrado en 2014 a la empresa *Confluent*, amparada por la *Apache Software Foundation*. Es una plataforma distribuida de alto rendimiento para el almacenamiento y procesamiento de flujos de datos.
- *Protocolo AMQP (Advanced Message Queueing Protocol)*: Protocolo abierto y seguro para manejar y enrutar colas de mensajes entre varios nodos. A diferencia de Apache Kafka, que es una tecnología y protocolo propio, *AMQP* es un estándar cuyas implementaciones más relevantes de código abierto son *RabbitMQ* **[39]** y *ActiveMQ* **[40]**.
- *WebSockets (WS) / Server Sent Events (SSE)*: En los entornos de desarrollo web, existen estos dos protocolos similares para la transmisión de mensajes a través de conexiones TCP. Ambos mecanismos cuentan con un API estándar en *JavaScript* y diversas implementaciones en los principales *frameworks* de desarrollo permiten interactuar con estos protocolos desde el servidor. La principal limitación de *SSE* es que, pese a ser más ligero, sólo permite el envío de mensajes del servidor al cliente y no al revés.

El proyecto abrirá la puerta al uso de varias tecnologías de *streaming* junto con mecanismos para el control de fallos de conexión y validación. A lo largo del estudio tecnológico, se priorizará *Kafka* por su familiaridad con el alumno y tutor. Por otra parte, se valorarán alternativas como *WebSockets* y *Server Sent Events* para comunicar los resultados desde la aplicación web *RDFShape* hasta sus clientes.

1.3.1.3.2 Manipulación y validación de flujos de datos RDF

El trabajo previo en tanto a la aplicación de flujos de datos a tecnologías semánticas es escaso, más aún si nos enfocamos en la validación de datos RDF en particular. Los principales antecedentes al respecto son:

- Labor de otros profesionales y grupos de investigación:
  - *Jean-Paul Calbimonte* **[41]**: Ex-investigador del *Ontology Engineering Group* (Madrid) y profesor asociado en el *Institute of Information Systems* (Suiza). Su investigación está centrada en la manipulación de flujos RDF.
- Proyectos similares realizados en el dominio de estudio:
  - *Triplewave* **[42]**: Una plataforma de código abierto centrada en la publicación en línea de flujos de datos en formato RDF.

- *Turbine* **[43]**: Prueba de concepto del grupo de investigación WESO centrado en la validación de flujos de datos RDF. Pese a ser la alternativa más cercana al objetivo del proyecto y grupo de trabajo del alumno, actualmente está incompleta y carece de documentación y mantenimiento.

Capítulo 2 Definición de la Arquitectura Tecnológica

Fase de Planificación

PSI

2.1 PSI 7.1: IDENTIFICACIÓN DE LAS NECESIDADES DE INFRAESTRUCTURA TECNOLÓGICA

En el presente apartado se analizan las tecnologías concretas que son necesarias para llevar a cabo el proyecto, cumpliendo tanto los objetivos generales como los requisitos específicos del cliente. Para ello, se define la infraestructura tecnológica que se va a emplear, junto a las alternativas tecnológicas viables para construirla.

El principal requisito del sistema es que cuente con tres módulos diferenciados: una biblioteca encargada de procesar y validar flujos de datos y una aplicación web, subdividida a su vez en API y cliente, capaz de ofrecer las funcionalidades de la biblioteca a usuarios externos tanto programáticamente como gráficamente. Además, se requiere que la aplicación web, *RDFShape*, sea fácilmente desplegable y monitorizable mediante contenedores, así como accesible de forma segura a través de cualquier equipo con conectividad a internet.

Cabe destacar que, para los módulos API y Cliente de *RDFShape*, se parte de un sistema ya desarrollado al que se desea integrar la funcionalidad de la biblioteca así como otros añadidos (sistema de *logs*, documentación en línea, contenedores, etc.) de cara a la demo pública. Es por esto por lo que se hace mención pero no se estudian alternativas para los *frameworks* sobre los que se construyeron estos sistemas, pues se consideran restricciones tecnológicas del cliente.

2.1.1 Biblioteca de validación de flujos RDF

2.1.1.1 *Entorno de programación funcional*

La biblioteca de validación de flujos debe integrarse con facilidad con el *backend* de *RDFShape*. Es importante subrayar que *RDFShape* ha sido construido en el lenguaje de programación *Scala* siguiendo el paradigma de programación funcional y utilizando un ecosistema de dependencias enfocadas al mismo. En consecuencia y en búsqueda de la máxima compatibilidad con *RDFShape*, se emplearán *Scala* y el mencionado paradigma de programación funcional en la construcción de la biblioteca de validación.

Se manejan los siguientes marcos de trabajo para programación funcional en *Scala*:

2.1.1.1.1 Cats Effect

Cats Effect **[44]** es un *framework* que se enmarca en el ecosistema de *Typelevel*, una serie de herramientas y bibliotecas que complementan las utilidades estándar de *Scala*. *Cats Effect* proporciona una serie de interfaces y abstracciones para la programación funcional y está pensado para aplicaciones en producción que precisan de alto rendimiento y paralelismo.

El núcleo del mecanismo de *Cats Effect* es la mónada *IO*, que permite capturar y controlar las acciones que el código va a realizar dentro de un contexto "puro" y seguro, sin efectos secundarios y que puede ser combinado y compuesto con otras acciones, denominadas efectos.

La principal meta de *Cats Effect*, así como del resto de alternativas, es suministrar herramientas al compilador y programador para llevar una cabo una programación totalmente funcional, que permita controlar, encadenar, paralelizar y manipular en general los efectos secundarios que produce el código (aquellos cambios que van más allá del valor devuelto, como operar con archivos o bases de datos).

2.1.1.1.2 ZIO

ZIO **[45]** es una biblioteca pensada para la programación asíncrona y con concurrencia en contextos puramente funcionales. Al igual que *Cats Effect*, *ZIO* proporciona una implementación propia de la mónada *IO* para operar con el código en forma de efectos encadenables en un contexto seguro. La principal diferencia con *Cats Effect* es que, aunque ofrece más posibilidades, tiene un ecosistema menos maduro y la curva de dificultad es mayor al ser las APIs de *ZIO* más diferentes de las APIs básicas de *Scala* para tratar con operaciones asíncronas y manejo de errores.

2.1.1.1.3 Monix

Monix **[46]** es una biblioteca más para la creación de aplicaciones basadas en la gestión de eventos, en un marco asíncrono y funcional. A grandes rasgos, ofrece funcionalidades similares a *Cats Effect* y *ZIO*, enfocadas al caso de uso de manejo de eventos. Además, posee un API más completa y con mejor integración con la librería estándar de *Scala*, aunque los mecanismos de manejo y rastreo de errores están menos desarrollados que en *Cats Effect*.

2.1.1.2 *Flujo de datos implementado*

Si bien la biblioteca debe proveer interfaces para validar cualquier tipo de flujo, también debe suministrar implementaciones para al menos una tecnología de comunicación de flujos remotos a través de internet. Se debe priorizar aquella con mayor popularidad además de familiaridad con el equipo de desarrollo.

2.1.1.2.1 Apache Kafka

Apache Kafka es una plataforma de código abierto para el procesamiento de flujos de eventos en entornos de alto rendimiento y en tiempo real. Actualmente se encuentra desarrollado y financiado por la empresa *Confluent* así como la *Apache Software Foundation*.

Kafka ofrece un modelo de creación y consumo de eventos basado en productores (*producers*) consumidores (*consumers*), que van colocando y recogiendo mensajes de un tema (*topic*) al que han sido asignados, como si cada tema se tratase de una cadena de montaje en la que hay varios actores depositando y recogiendo ítems. Consta de un sistema distribuido donde uno o más servidores almacenan los mensajes recibidos y procesados y reparten la carga de producción y consumo entre

los agentes conectados según la configuración del sistema, siguiendo un modelo *publish-subscribe*. Es importante destacar que, para sus funciones, el servidor *Kafka* necesita ser ejecutado junto a una instancia de *Apache Zookeeper*, lo que añade una capa de complejidad adicional al despliegue y configuración. En este momento, se está trabajando en eliminar la dependencia en *Zookeeper*, sin embargo la situación no ha cambiado a fecha de elaboración del proyecto.

Kafka se define como una plataforma puesto que pone a disposición de los usuarios un conjunto de binarios y ejecutables necesarios para iniciar un servidor de *Kafka*, así como bibliotecas para publicar y consumir datos en el servidor para los principales lenguajes de programación.

2.1.1.2.2 RabbitMQ

RabbitMQ es una reconocida implementación de código abierto del protocolo *AMQP*. Al igual que *Kafka*, es capaz de actuar como nodo central de un sistema distribuido de procesamiento de mensajes y eventos y está más orientado al enrutamiento de mensajes en sistemas distribuidos complejos.

Soporta varios protocolos de comunicación de mensajes, encolamiento de mensajes, distribución de trabajo, verificación de la recepción de mensajes, entre otros. También tiene binarios y bibliotecas disponibles para por principales lenguajes de programación, habilitando el desarrollo de aplicaciones de mensajería independientemente del lenguaje seleccionado.

A diferencia de *Kafka*, carece de limitaciones en el tamaño de los mensajes, pero ofrece un rendimiento menor al estar pensado para escenarios más complejos.

2.1.1.2.3 *Stream* HTTP

Otra posibilidad sería agregar a la biblioteca la posibilidad de consumir datos provenientes de un flujo emitido por un servidor *HTTP*. El mecanismo planteado sería que la biblioteca actuase como cliente *HTTP*, lanzando una petición a un servidor que está emitiendo un flujo de datos a través de *Server Sent Events (SSE)*.

SSE ofrece un sistema versátil y fácil de implementar para emitir flujos de datos a través de internet, pues sólo es necesario contar con un servidor *HTTP* configurado correctamente. A diferencia de las otras tecnologías especializadas, *SSE* no incorpora ningún sistema por defecto para configurar la identidad o frecuencia de consumo de los consumidores de mensajes, además de que los datos emitidos están limitados a contenidos textuales (*string*) codificados en *UTF-8*, que luego deben ser deserializados por los clientes.

2.1.1.3 *Manipulación de flujos de datos*

Al margen del procesamiento o validación que apliquemos sobre los datos, es necesario contar con un marco de trabajo orientado a flujos, abstraíble a cualquier tipo de dato y enmarcado en el ecosistema funcional.

2.1.1.3.1 Fs2

Fs2 **[47]** ofrece un sistema de código abierto para crear, componer y transformar flujos de datos a partir de cualquier tipo de ítem, desde listados en memoria, ficheros, hasta datos entrantes a través de internet. La biblioteca *Fs2* para *Scala* está pensada para trabajar en un marco de programación funcional, junto con cualquier mónada definida por el *framework* que se desee y facilita la paralelización y segmentación del procesamiento de flujos. También permite el manejo de errores a través de las APIs por defecto de *Scala*.

Si bien las utilidades ofrecidas por *Fs2* son abstracciones que no se comprometen con ningún tipo de datos ni mecanismo de flujos concreto, tiene una gran comunidad detrás que ha creado bibliotecas complementarias para trabajar fácilmente con la mayoría de los sistemas de flujos y eventos disponibles, tales como *Apache Kafka* (*fs2-kakfa*) y *RabbitMQ* (*fs2-rabbit*). Además, el *framework* que opera el API de *RDFShape* emplea *Fs2* para manejar las peticiones y las funcionalidades de servidor *WebSockets*, por lo que es un sistema altamente compatible con la infraestructura ya existente.

2.1.1.3.2 Akka

Akka **[48]** ofrece un sistema completo de bibliotecas enfocadas a la programación funcional, concurrente y basada en el manejo de eventos en Scala, entre otros. Dentro de su ecosistema, *Akka* propone un sistema de actores que ofrece posibilidades de manejos de flujos similares a las de *Fs2*, si bien exige un mayor conocimiento y compromiso con el resto de las bibliotecas del entorno *Akka* y una curva de aprendizaje más elevada.

2.1.2 Refactorización y ampliación RDFShape

Para la codificación de las nuevas funcionalidades de *RDFShape*, se parte de un sistema ya existente. Se enumeran las tecnologías base empleadas en la construcción de cada módulo junto con las alternativas necesarias para las tareas de ampliación.

2.1.2.1 *API*

2.1.2.1.1 Construcción del API

2.1.2.1.1.1 http4s

http4s **[49]** es un *framework* para el desarrollo de servidores y clientes web que soporta los protocolos *HTTP* y *WebSockets*, tanto en su versión estándar como securizada con certificados SSL. Está enfocado a la construcción de aplicaciones en un ecosistema funcional y emplea la biblioteca *Fs2* para procesar las peticiones recibidas como un flujo de eventos.

Incorpora un *DSL* propio para la definición de APIs de forma idiomática desde el código de la aplicación y hace un gran uso de las variables y parámetros implícitos de Scala para (de)serializar JSON y otro tipo de datos recibidos de forma transparente al programador con ayuda de la biblioteca

Circe **[50]**. Además, incorpora varios *middlewares* para casos de uso como redirecciones o para registrar las peticiones recibidas en el sistema de *logs* pertinente.

Es el *framework* sobre el que está construida el API de la aplicación *RDFShape* y el que se utilizará a lo largo del desarrollo del módulo correspondiente.

2.1.2.1.2 Documentación del API

Se necesita documentar el API REST de *RDFShape* siguiendo la *Open API Specification* para su posterior publicación en *Swagger*.

2.1.2.1.2.1 Rho (ρ)

Rho **[51]** es una biblioteca construida sobre *http4s* y pensada para complementar al *framework* en cuestión. Sus principales funcionalidades son extender el *DSL* de *http4s* con más y mejores mecanismos para definir las rutas del sistema, además de la capacidad de generar, automáticamente, la documentación del API en las sintaxis JSON y YAML y en el formato especificado por la *Open API Specification*.

Aunque carece de una documentación extensa, sí cuenta con varios ejemplos y está activamente mantenida por la comunidad. Sus fortalezas son la capacidad de integrarse fácilmente en el código de la aplicación gracias a su *DSL*, así como las facilidades para definir documentación personalizada para aquellas rutas en que la generada por defecto sea insuficiente.

2.1.2.1.2.2 Documentación a mano

Una alternativa a la generación automática de documentación es la generación, a mano, de ficheros de documentación del API que cumplan con la especificación. Esta alternativa requeriría de un gran esfuerzo inicial, pero una vez realizado los esfuerzos siguientes serían pequeños e incrementales.

Las ventajas serían el tiempo ahorrado en aprender tecnologías para anotar el API, si bien el proceso manual es más propenso a errores y tedioso de mantener, además de no automatizable.

2.1.2.1.3 Sistema de logs

Es preciso incorporar un sistema que registre el uso que los usuarios hacen de la aplicación, de la manera más completa y transparente posible al programador.

2.1.2.1.3.1 Logback

Logback **[52]** es un sistema de *logs* para lenguajes de la *Java Virtual Machine* (*JVM*) empleado inicialmente en *RDFShape* y en otros proyectos de WESO, si bien inicialmente sólo contiene la configuración por defecto y el cliente nos permite escoger otro sistema si es preciso. *Logback* está mantenido por la comunidad y ofrece la mayoría de las configuraciones esperables de un sistema de *logs* maduro: varios niveles de mensajes, posibilidad de mostrar mensajes en consola o guardarlos en ficheros, rotación de *logs*, filtros de mensajes, etc.

Logback se configura a través de un fichero de configuración, que puede estar redactado en *XML* o en el lenguaje *Groovy*, y cuenta con una extensa documentación online con ejemplos.

2.1.2.1.3.2 Log4J

Log4J **[53]** es otro popular *framework* para manipular registros en aplicaciones de la *JVM* que ofrece una gran comunidad y conjunto de funcionalidades equiparables a las de *Logback*.

Cabe destacar que ambas bibliotecas de trabajo con *logs* son soportadas por el sistema *scala-logging* **[54]**, un frontal que permite operar y publicar mensajes desde el código de manera independiente al *framework* empleado por debajo.

2.1.2.2 *Cliente web*

2.1.2.2.1 Construcción del Cliente

2.1.2.2.1.1 React

React **[55]** es un *framework* de código abierto para la creación de interfaces de usuario en entornos web. Su filosofía es la creación y posterior reutilización de componentes y sus *props* asociados, junto con un enfoque declarativo, para construir las distintas pantallas de la web con comportamientos predecibles y reusables. Así mismo, incorpora diversas utilidades para realizar pruebas de renderizados de la web y comprobar, el tiempo de ejecución, que los tipos de los *props* recibidos por cada componente son los correctos.

Es el *framework* con mayor popularidad en el momento de desarrollo del proyecto y sobre el que está construido el cliente de la aplicación *RDFShape*, por lo que se utilizará a lo largo del desarrollo del módulo correspondiente.

Cabe destacar que el desarrollo inicial se ha realizado en *JavaScript* y no se contempla una migración a *TypeScript* por restricciones de tiempo y alcance.

2.1.2.2.1.2 Bootstrap

Bootstrap es uno de los *frameworks* más populares para el desarrollo de interfaces de usuario en la web a través de varias clases y hojas de estilo, así como *scripts* predefinidos. Su contraparte en *React*, el módulo NPM *React Bootstrap* **[56]**, replica las funcionalidades del citado *framework* adaptándolas a *React*, de modo que ofrece varios componentes personalizables que permiten construir interfaces de forma sencilla y con un lenguaje de diseño en común.

Es el *framework CSS* sobre el que está construido el cliente de la aplicación *RDFShape*, por lo que se utilizará a lo largo del desarrollo del módulo correspondiente, en conjunto con otros módulos de la comunidad que extienden *Bootstrap* con nuevos componentes.

2.1.2.3 *Comunicaciones entre API y Clientes*

Una vez el *backend* de *RDFShape* esté integrado con la biblioteca de validación de flujos, debe ser capaz de ofrecer a sus clientes la posibilidad de invocar las funcionalidades de la biblioteca y de recibir los resultados. La recepción de resultados implica el envío una cantidad potencialmente infinita de mensajes desde el servidor, que son generados y enviados en tiempo real según se procesan los flujos de datos.

Se plantean las siguientes alternativas para ofrecer los resultados a los clientes.

2.1.2.3.1 WebSockets

WebSockets (*WS*), como se ha descrito anteriormente en el documento, es un protocolo de comunicación que permite a dos sistemas establecer una comunicación bidireccional a través de una conexión TCP. Una vez establecida la conexión, ambas partes podrán intercambiar mensajes binarios o de texto en tiempo real, sin límites de tamaño definidos y con un buen rendimiento.

Sus principales ventajas son su fácil integración con las tecnologías ya existentes en el *backend* de *RDFShape*, así como la capacidad de crear un canal de comunicación completamente bidireccional, que permite a los clientes interactuar activamente con el servidor.

2.1.2.3.2 Server Sent Events

Server Sent Events (*SSE*) funciona de manera similar a *WebSockets*, tal que incluso el API de navegadores para procesar *SSE* es muy similar a la de *WS*. Ofrece un mayor rendimiento y velocidad que *WS* a cambio de ser unidireccional (sólo se permiten mensajes del servidor a los clientes) y estar limitado al envío de textos *UTF-8*.

2.1.3 Publicación del sistema

Uno de los principales objetivos estratégicos del proyecto es promocionar la universidad y el grupo de investigación WESO, de forma que el sistema desarrollado debe ser fácilmente accesible y desplegable, además de atractivo y documentado de cara a potenciales usuarios y desarrolladores.

2.1.3.1 *Publicación de la biblioteca*

Además de publicar el código fuente de la biblioteca de validación, se subirán los binarios correspondientes a un servidor que permita descargarlos a través de un sistema gestor de dependencias.

2.1.3.1.1 Sonatype

Sonatype es una compañía que ofrece el *Sonatype Open Source Software Repository Hosting* (*OSSRH*) **[57]** que permite alojar los binarios de proyectos de código abierto. Simplemente se requiere tener el proyecto disponible en un repositorio público y verificar nuestra identidad para poder subirlo a sus

servidores libre de precio. El repositorio de *Sonatype* usa el formato *Maven* y permite a los desarrolladores del entorno *Java* (y derivados) acceder a las bibliotecas subidas desde su gestor de dependencias favorito, como si se encontrasen subidas al repositorio central de *Maven* **[58]**.

Este sistema es el empleado en el entorno *RDFShape* y otros proyectos de WESO, que cuentan con procesos de integración continua para actualizar las bibliotecas del repositorio con cada revisión.

2.1.3.1.2 GitHub Packages

GitHub Packages **[59]** es un entorno en la nube gestionado por *GitHub* donde se pueden subir todo tipo de archivos, aunque está especialmente pensado para alojar binarios y bibliotecas. Destaca por tener un plan gratuito de hasta 500MB de almacenamiento. Su principal ventaja es poder incorporar automáticamente el sistema de permisos de cada usuario u organización a los paquetes que se encuentren publicados por este, tal que se puede restringir el acceso al paquete a los usuarios, grupos o *bots* que se requiera.

2.1.3.2 *Despliegue de la aplicación web*

Se desea simplificar el despliegue de la aplicación web *RDFShape* una vez finalizada. El entorno de despliegue actual, basado en un sistema Ubuntu Linux y un conjunto de *scripts*, debe ser reemplazado por un entorno de contenedores. Además de la tecnología que se va a utilizar para crear y lanzar los contenedores, se debe decidir un registro o repositorio al que subir las imágenes públicamente.

2.1.3.2.1 Tecnologías de contenedores

2.1.3.2.1.1 Docker

Docker **[60]** es un proyecto de código abierto que permite automatizar el despliegue de todo tipo de aplicaciones en forma de contenedores portables, ligeros y auto suficientes. Está actualmente mantenido por la compañía del mismo nombre y es la tecnología de contenedores más popular del mercado. Cabe destacar que *Docker* es fundador y colaborador de la *Open Container Iniciative* (*OCI*) **[61]**, una asociación creada para estandarizar el uso de tecnologías de contenedores.

Sus principales ventajas son su amplia adopción dentro del grupo WESO y la experiencia previa del alumno con la tecnología y sus derivados *docker-compose* y *docker-swarm*.

2.1.3.2.1.2 Podman

Podman **[62]** es una de las alternativas más populares a *Docker*. De código abierto y desarrollado por *RedHat*, *Podman* ofrece posibilidades similares a *Docker* aunque en un entorno más seguro dado que no depende de un *daemon* persistente para funcionar y permite administrar y entrar a los contenedores sin privilegios de usuario *root* o administrador. Además, también cumple con los estándares de la *OCI*.

2.1.3.2.2 Registro de imágenes

2.1.3.2.2.1 Docker Hub

Docker Hub **[63]** es el repositorio de imágenes *Docker* de uso más extendido. Permite a los usuarios y organizaciones subir imágenes con un plan gratuito para imágenes públicas. También tiene un potente cliente web que permite a los usuarios administrar la descripción y metadatos de las imágenes, así como navegar por las diferentes etiquetas o versiones de estas.

Sus principales ventajas son su extendida popularidad y el hecho de ser el repositorio por defecto del que *Docker* intentará bajar las imágenes que no encuentre localmente.

2.1.3.2.2.2 GitHub Container Registry

GitHub Container Registry (*GHCR*) **[64]** es una rama del entorno *GitHub Packages* mentado anteriormente que se especializa en alojar imágenes de contenedores *OCI* para su uso y distribución privada o pública.

Hereda las ventajas de *GitHub Packages*, es decir, permite establecer permisos de acceso de forma sencilla en base a los permisos ya existentes en *GitHub*. Además cuenta con un extenso plan gratuito para repositorios públicos y privados y una buena integración con en el entorno de integración continua de *GitHub Actions*. Requiere que los usuarios agreguen el prefijo "*ghcr.io*" a las imágenes que se quieran buscar en el repositorio.

2.1.3.3 *Generación de documentación web*

Al margen de la propia documentación anexa al código del sistema, se desea tener una documentación pública y accesible a través de internet. Se favorecerán las alternativas que permitan automatizar la creación y publicación de la documentación, a la par de ser atractivas visualmente.

2.1.3.3.1 Wikis

GitHub, la plataforma en la que se alojará el código del proyecto como parte de la organización de WESO, ofrece un sistema de *wikis* **[65]** que permiten trabajar de forma colaborativa en la documentación del proyecto anexo a la *wiki* en cuestión.

Sus principales ventajas son su sencillez, el mínimo tiempo necesario para su publicación y la facilidad de edición en formato *Markdown*, así como los sistemas de permisos personalizables y la capacidad de automatizar el despliegue de la *wiki* con nuevos cambios, ya que internamente *GitHub* trata las *wikis* como repositorios de primera clase.

Algunos inconvenientes serían la limitada estructura y formato de contenido que permite una *wiki*, que podría hacerlo menos atractivo a algunos usuarios en comparación con un sitio web con su identidad propia. Además, no sería posible desplegar la documentación *Scaladoc* dentro de la *wiki* del proyecto, teniendo que recurrir a un servicio de hosting adicional.

2.1.3.3.2 mdoc + Docusaurus

2.1.3.3.2.1 mdoc

mdoc **[66]** es un motor de plantillas enfocado a generar documentación personalizada de proyectos Scala de manera simple. Permite a los usuarios crear una jerarquía de documentos en formato *Markdown*, definir su ubicación y otras variables y actualizarlos automáticamente al construir el proyecto.

Las principales ventajas del uso de *mdoc* son que mantiene la sencillez de redactar en *Markdown*, su facilidad de instalación mediante *plugins*, la detección de enlaces rotos entre documentos y su integración con la configuración del proyecto. Además, *mdoc* permite definir variables personalizadas que inyectar en las plantillas de documentación, de forma que cambiando un valor (como el nombre o versión de la aplicación) en un único punto se vea reflejado automáticamente en nuevas versiones de la documentación.

2.1.3.3.2.2 Docusaurus

mdoc integra un *plugin* para hacer uso de *Docusaurus* **[67]**, un *framework* de código abierto mantenido por *Facebook* empleado para construir sitios web personalizables para proyectos software. *Docusaurus* permite hacer uso de varias plantillas y componentes base para generar una web con los estilos y contenidos definidos por el usuario.

Destaca por ser sencillo de configurar pero altamente personalizable para quien lo desee. Además, las webs creadas son de alto rendimiento, con modo claro/oscuro y cuentan con un árbol de documentación en base a los contenidos generados por *mdoc*, que dan una apariencia profesional al resultado.

Finalmente, al estar tanto *mdoc* como *Docusaurus* integrados a través de *plugins* de *SBT* **[68]** (el mecanismo de configuración y construcción de proyectos *Scala*), es sencillo crear cadenas de tareas de manera que con un solo comando se genere el *Scaladoc*, se cree la documentación de *mdoc*, se construya el sitio web *Docusaurus* y, finalmente, se publique todo en un entorno previamente configurado, como *GitHub Pages*. Gracias a la interfaz de consola de *SBT* y otros *plugins* secundarios, es posible integrar todo dentro de los procesos de integración continua.

La principal ventaja de esta alternativa es que es altamente mantenible y automatizable, aparte de completa y profesional. Sin embargo, implica una alta curva de aprendizaje en el uso y coordinación de las tecnologías implicadas.

2.1.3.3.3 Jekyll

Jekyll **[69]** es un generador de sitios web estáticos con soporte integrado en *GitHub Pages* **[70]**, el entorno de despliegue objetivo para las páginas de los módulos del proyecto. Permite generar

páginas web enfocadas a blogs y documentación de proyectos mediante un mecanismo de plantillas basado en *Markdown* y *Liquid*.

Jekyll presenta un compromiso en comparación con las alternativas anteriores en lo que respecta a profesionalidad del resultado y facilidad de uso.

2.1.4 Integración continua

A lo largo del desarrollo de los módulos, se requiere de mecanismos de integración y despliegue continuos, encargados principalmente de:

- Construir y ejecutar los tests de los módulos del proyecto para cada cambio.
- Construir y desplegar la documentación web de los módulos del proyecto para cada cambio.
- Publicar, para cada nueva versión del módulo validador, los binarios de la biblioteca en el repositorio seleccionado, de forma que la actualización quede disponible a los usuarios.

2.1.4.1 *Travis CI*

Travis CI **[71]** es un servicio de integración continua que ofrece varios servidores en los que construir, probar o ejecutar todo tipo de acciones en respuesta a nuevos cambios en el código del proyecto. Cuenta con un cliente web que permite consultar el histórico de operaciones realizadas, registros de cada operación, administrar usuarios y permisos, etc. En entorno en la nube en que queremos que se ejecuten las acciones es configurable a través de ficheros en formato *YAML*.

Tiene un plan gratuito destinado a proyectos de código abierto y es una de las plataformas más longevas y maduras del sector, por lo que es sencillo de integrar con *GitHub* y otros sistemas de alojamiento de código

2.1.4.2 *GitHub Actions*

GitHub Actions **[72]** es otro servicio de integración continua creado por el propio *GitHub* y lanzado en 2018. Ofrece funcionalidades similares a las de *Travis*, con la ventaja adicional de estar completamente integrado con *GitHub*, tanto con su cliente web como respecto a los sistemas de permisos, facilidad de despliegues, *GitHub Packages*, etc.

También cuenta con un plan gratuito para proyectos públicos y privados. Además, es el mecanismo empleado en la mayoría de proyecto de WESO y con el que el alumno tiene más familiaridad.

2.1.5 Bases de datos

Una de las necesidades del cliente es la creación de un sistema enlaces permanentes o *permalink*s para *RDFShape*. Para poder gestionar dichos enlaces y de cara a futuras necesidades de persistencia del proyecto, se opta por emplear un SGBD.

2.1.5.1 *MongoDB*

MongoDB **[73]** es un SGBD no relacional (no *SQL*) multiplataforma. *MongoDB* se basa en el almacenamiento de documentos en varias tablas o índices, que luego pueden ser consultadas mediante el su propio lenguaje de consultas *MongoDB Query Language*.

Las principales características de *MongoDB* son la definición y almacenamiento de documentos de estructura y contenido flexible en formato *BSON*, una estructura clave-valor similar a *JSON*. Sus inconvenientes son que carece de la robustez, rendimiento y predictibilidad de los SGBD relacionales. Como contraparte, permite desarrollos rápidos y es muy flexible de cara a proyectos con requisitos cambiantes. Además, es una tecnología madura con una aceptación y herramientas de desarrollo comparables a las de los SGBD relacionales.

Adicionalmente, cuenta con un servicio de alojamiento de bases de datos en la nube: *MongoDB Atlas* **[74]**, que ofrece un potente cliente web y un plan gratuito enfocado a proyecto pequeños.

2.1.5.2 *MySQL*

MySQL **[75]** es un SGBD relacional, robusto, de alto rendimiento y de código abierto. Al igual que cualquier SGBD relacional, organizada sus entradas en un sistema de índices y tablas, consultables mediante el lenguaje *Structured Query Language* (*SQL*).

Su principal ventaja es su gran adopción y madurez, puesto que es uno de los SGBD más populares y lleva en producción desde el año 1995, además de la reducida curva de dificultad al tratarse de un sistema relacional tradicional.

2.2 PSI 7.2: SELECCIÓN DE LA ARQUITECTURA TECNOLÓGICA

Fruto de las necesidades y alternativas tecnológicas identificadas anteriormente, se seleccionan las siguientes alternativas para conformar la arquitectura del sistema.

2.2.1 Biblioteca de validación de flujos RDF

Tras el estudio previo y las reuniones con el tutor, se ha decidido escoger las alternativas tecnológicas más continuistas respecto a las del material sobre el que se está trabajando. Además, por decisión de alumno y por facilitar la publicación y "*marketing*" del módulo, se decide llamarlo por el nombre de ***Comet***.

Ilustración 6. Logo provisional de la biblioteca de validación

2.2.1.1 *Entorno de programación funcional: Cats-Effect*

En línea con el *backend* del API de *RDFShape*, se opta por usar *Cats Effect* como marco de trabajo funcional. Además, si bien cualquiera de las alternativas sería compatible con *RDFShape* y el resto de las opciones tecnológicas, *Cats Effect* ofrece una amplia documentación a la que se suman los propios ejemplos de uso que puedan extraerse de lo ya desarrollado para *RDFShape*.

Por otra parte, el alumno ya cuenta con una reducida experiencia previa en este *framework* gracias a sus trabajos anteriores dentro de WESO.

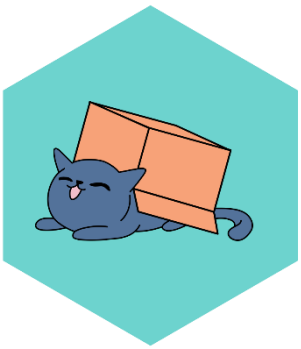


Ilustración 7. Logo de Cats Effect

2.2.1.2 *Flujo de datos implementado: Apache Kafka + otros*

Se opta por *Apache Kafka* como el principal flujo de datos con soporte de serie en la biblioteca de validación. La decisión viene motivada por varios motivos, principalmente:

- Interés del tutor y el resto de los interesados en la tecnología.
- Alta popularidad y disponibilidad de documentación tutoriales y herramientas para interactuar con la tecnología.
- Experiencia previa del alumno con la tecnología.

Adicionalmente, se implementará el procesamiento de flujos de datos provenientes de otras fuentes, con el doble propósito de proveer de ejemplos a los potenciales usuarios y desarrolladores futuros, así como facilitar la realización de pruebas del sistema mientras se construye un *stream* de pruebas basado en *Kafka*.

Por tanto, la biblioteca soportará, de serie, flujos de datos provenientes de las siguientes fuentes:

- ***Apache Kafka***: dado un flujo de datos emitidos a través de *Kafka*, la biblioteca deserializará cada mensaje recibido a fin de interpretarlo como datos RDF y procesarlo como tal.
- ***Listas***: dada una lista de elementos en memoria, la biblioteca interpretará cada elemento de la lista como un ítem de un flujo de datos RDF y lo procesará como tal.
- ***Ficheros***: dado un conjunto de ficheros accesibles por el sistema operativo, la biblioteca leerá e interpretará cada fichero como un ítem de un flujo de datos RDF y lo procesará como tal.

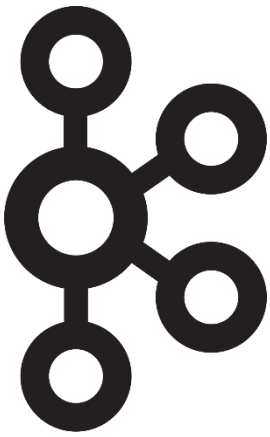


Ilustración 8. Logo de Apache Kafka

2.2.1.3 *Manipulación de flujos de datos: Fs2*

Para la manipulación de flujos de datos se decide emplear la biblioteca *Fs2*. Pese a que no se posee experiencia previa ni hay restricciones de cliente, la biblioteca Fs2 cuenta con varios puntos a su favor:

- Es una herramienta independiente de cualquier ecosistema, a diferencia de las alternativas.
- Posee una documentación clara, con ejemplos iniciales concisos.

- Es utilizado por *http4s*, el *framework* detrás de *RDFShape* para manipular las conexiones *WebSockets* como si fueran flujos. De esta manera, la experiencia adquirida desarrollando la biblioteca con *Fs2* es trasladable a otras etapas del proyecto.
- Tiene un ecosistema de conectores activamente mantenidos que facilitan trabajar con flujos de datos populares, lo que permite:
    - Trabajar de forma sencilla con *Apache Kafka*, mediante la biblioteca complementaria *fs2-kafka* **[76]**.
    - Crear nuevas implementaciones para trabajar con otros flujos partiendo de los conectores ya existentes, como se va a hacer con *Kafka*.

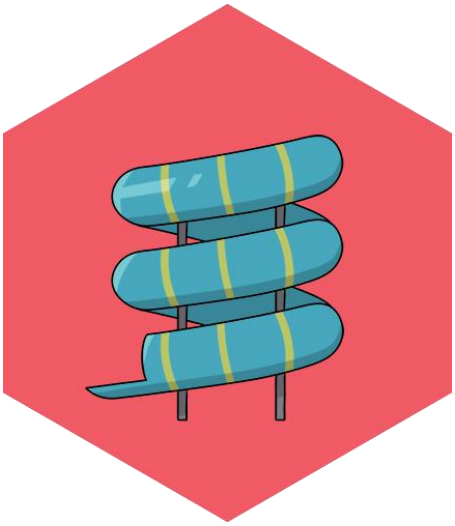


Ilustración 10. Logo de Fs2

Ilustración 9. Logo de fs2-kafka

2.2.2 Refactorización y ampliación RDFShape

2.2.2.1 *API*

2.2.2.1.1 Construcción del API: Http4s

Ilustración 11. Logo de http4s

Para el *backend* de la aplicación web *RDFShape*, se parte y expande el desarrollo previo basado en *http4s*, tanto por ser una restricción del cliente como por ser la opción más viable dado el contexto del proyecto.

2.2.2.1.2 Documentación del API: Rho (ρ)

Finalmente, se opta por emplear *Rho* como herramienta para documentar el API del proyecto, en detrimento de generar la documentación totalmente a mano. Para tomar esta decisión se ha priorizado la automatización del proceso de generar la documentación, de forma que aunque sea a coste de un trabajo inicial, las futuras versiones del API partan de una documentación base fácil de ampliar y generada sin que haya nadie del personal pendiente.

La documentación de *Rho* se encuentra bastante incompleta en el momento de realización del proyecto, dado el nicho tecnológico al que pertenece. Sin embargo, sí cuenta con varios ejemplos y un canal de ayuda activo en *Gitter* en el que se encuentran el tutor y el alumno.

Otros grandes incentivos para esta decisión han sido la sinergia de *Rho* con el presente *framework http4s*, así como la generación de la documentación tanto en JSON como en YAML.

2.2.2.1.3 Sistema de logs: Logback + scala-logging

Se opta por emplear *Logback*, frente a *Log4J*, como principal sistema de registros para el servidor de la aplicación web. Los principales motivos de la decisión son:

- Presencia de una instalación y configuración básica de *Logback* en *RDFShape*.

- Facilidades para comprimir y rotar los registros cuando son guardados como ficheros dentro del servidor. Se priorizan estas configuraciones puesto que se pretende ahorrar almacenamiento en la infraestructura del cliente.
- Gran cantidad de documentación con ejemplos.

Se hará uso, a su vez, de la biblioteca *scala-logging*, que permite interactuar con *Logback* a través del código de forma sencilla extendiendo una serie de clases predefinidas.

2.2.2.2 *Cliente web*

2.2.2.2.1 Construcción del Cliente: React + Bootstrap

Para el *frontend* proporcionado por WESO para *RDFShape*, se construye a partir del desarrollo previo basado en *React* y *Bootstrap* (y módulos derivados). La decisión viene motivada por las restricciones del cliente, además de por la experiencia previa del alumno con las tecnologías.

Además del módulo base de *Bootstrap*, cabe destacar los siguientes módulos que se prevé aprovechar o añadir para el desarrollo y ampliación del proyecto:

- ***axios*** **[77]**: Cliente *HTTP* empleado para realizar peticiones al API y procesar las respuestas recibidas.
- ***prop-types*** **[78]**: Herramienta oficial complementaria a *React* para la comprobación de los tipos de los datos contenidos en los componentes de la aplicación.
- ***viz.js*** **[79]*, cytoscape.js*** **[80]**: Bibliotecas para visualizaciones de gráficos y redes en el entorno del navegador. Se emplean para representar datos y esquemas de forma gráfica, normalmente con fines didácticos.
- ***CodeMirror*** **[81]*, Yate*** **[82]*, Yashe*** **[83]**: Bibliotecas y extensiones desarrolladas por WESO para la manipulación de contenidos RDF y *Shape Expressions* en el navegador. Se emplean para permitir a los usuarios redactar textos desde la aplicación web recibiendo errores de sintaxis o sugerencias de autocompletado.
- ***react-use-websocket*** **[84]**: Adaptación del API por defecto de navegadores para trabajar con *WebSockets* / *Server Sent Events* en el marco de los *hooks* de *React*. Se empleará para conectar con el API una vez sea capaz de invocar validaciones de flujos.

- ***react-bootstrap-table-next*** **[85]**: Componentes adicionales para *Bootstrap* destinados a la representación de datos en forma de tablas paginables, ordenables y descargables. Se empleará para mostrar los resultados de las validaciones.

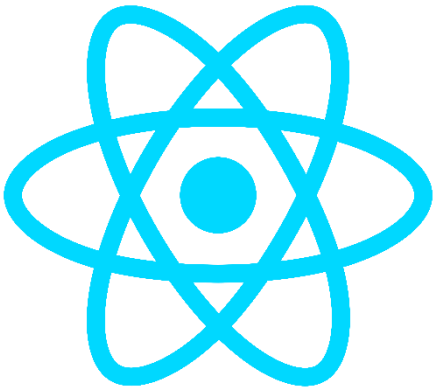


Ilustración 13. Logo de React

Ilustración 12. Logo de Bootstrap

2.2.2.3 *Comunicaciones entre API y Clientes: WebSockets*

La opción seleccionada para comunicar el API y el cliente cuando se invoquen validaciones de flujos es el uso de *WebSockets*. El motivo de la elección es la necesidad de un protocolo bidireccional en el que el cliente también pueda enviar mensajes al servidor para configurar o pausar la validación, además de la capacidad de *http4s* para actuar como servidor *WebSockets*, integrada en el propio *framework*.

2.2.3 Publicación del sistema

2.2.3.1 *Publicación de la biblioteca: Sonatype*

Se decide que la biblioteca de validación de flujos, *Comet*, será publicada en el repositorio que ofrece *Sonatype* para proyectos de código abierto. Los motivos principales de la elección son:

- No impone límites de almacenamiento.
- Uno de los objetivos estratégicos de WESO y del proyecto con conseguir una mayor publicidad y difusión, por lo que estar restringidos a proyecto de código abierto no es un inconveniente.
- Emplea el formato *Maven* y posee un portal web donde los desarrolladores pueden simplemente buscar la dependencia que necesiten y copiar las líneas que deben incluir en su fichero de dependencias.
- Existen precedentes de otros proyectos de WESO desplegados en la plataforma, por lo que se conoce de primera mano que funciona correctamente y cómo automatizar la publicación de nuevas versiones de software cuando haya cambios en el sistema gestor de versiones.

2.2.3.2 *Despliegue de la aplicación web*

2.2.3.2.1 Tecnologías de contenedores: Docker (Compose)

Para la adaptación del despliegue de la aplicación web de un despliegue manual a uno automatizado con contenedores, se ha optado por emplear *Docker*, dadas las siguientes circunstancias:

- Interés del tutor y el resto de los interesados en la tecnología.
- Alta popularidad y disponibilidad de documentación tutoriales y herramientas para interactuar con la tecnología.
- Experiencia previa del alumno con la tecnología.

Además, se decide utilizar *Docker Compose* **[86]**, una herramienta para definir y desplegar aplicaciones que precisan de varios contenedores, para coordinar los despliegues de *RDFShape*. *Docker Compose* permite configurar cómo se va a realizar el despliegue de la aplicación mediante ficheros de configuración YAML, incluyendo las opciones más comunes que normalmente se manejarían mediante argumentos en la *CLI* de *Docker*:

- Las imágenes *Docker* a desplegar y el orden de despliegue en base a las dependencias de cada uno de los contenedores respecto al resto.
- El mapeo de puertos entre los contenedores y la máquina anfitriona.
- La política de reinicio de los contenedores.
- Los volúmenes de los contenedores.
- Las variables de entorno de los contenedores.
- La política de chequeos de estado o *healthchecks*, comandos personalizados para comprobar que cada contenedor sigue funcionando correctamente.

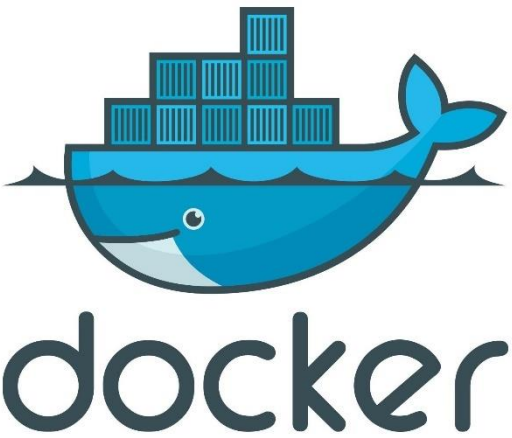


Ilustración 14. Logo de Docker

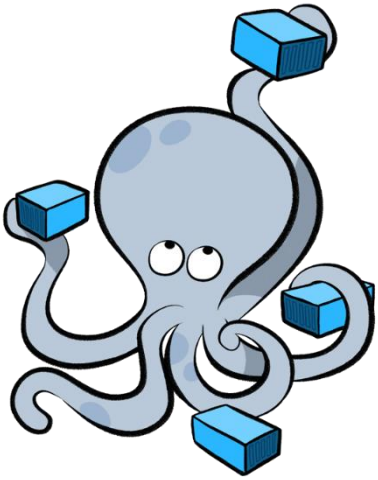


Ilustración 15. Logo de Docker Compose

2.2.3.2.1.1 Arquitectura final del despliegue

Una vez se tengan imágenes *Docker* de cada módulo de *RDFShape*, se creará una configuración de *Docker Compose* que permita lanzar toda la aplicación web al mismo tiempo. Para el despliegue final, se cuentan con las siguientes restricciones de infraestructura:

1. API y Cliente serán ejecutados en la misma máquina y dentro del mismo (sub)dominio "*rdfshape.weso.es*", cuyo nombre DNS viene dado por el grupo WESO.
2. Se evitará exponer el API y el cliente al exterior con el mismo nombre de dominio pero con distintos puertos, a fin de facilitar el acceso al producto y transmitir más profesionalidad. Es decir, se expondrá el API en el subdominio "*api.rdfshape.weso.es*" en lugar de "*rdfshape.weso.es:8080*" como se hacía hasta el momento. Por su parte, el cliente seguirá publicado en "*rdfshape.weso.es*".
3. Se contará con un sistema capaz de monitorizar y relanzar automáticamente los contenedores que no estén funcionando correctamente.

Identificados los requisitos arquitectónicos del despliegue, se opta por añadir dos contenedores adicionales al despliegue final, quedando la siguiente estructura de contenedores:

1. ***rdfshape-api***: Contenedor que ejecuta el *backend* de *RDFShape*, exponiendo el API y la validación de flujos.
2. ***rdfshape-client***: Contenedor con el servidor web que sirve el *frontend* diseñado para *RDFShape*.
3. ***nginx-proxy***: Contenedor que se encarga de procesar todas las peticiones *HTTP*(S) recibidas por el servidor, exponiendo los puertos 80 y 443 al exterior. En su interior, ejecuta una instancia del servidor *NGINX* que actúa como proxy inverso **[87]**, es decir, lee a qué dominio va dirigida cada petición y la redirige al API o cliente según corresponda. Este mecanismo permite ejecutar exponer las dos aplicaciones al exterior de forma transparente y ordenada, cada una con su dominio. Se emplea la imagen *Docker jwilder/nginx-proxy* **[88]**.
4. ***autoheal***: *Docker (Compose)* es capaz de implementar y ejecutar chequeos de estado de los contenedores, pero no tiene la capacidad de reiniciar los contenedores cuando la lectura es fallida. Se usa la imagen *Docker willfarrel/autoheal* **[89]** para comprobar que no haya contenedores caídos y reiniciarlos cuando ese sea el caso.

Ilustración 16. Logo de NGINX

2.2.3.2.2 Registro de imágenes: GitHub Container Registry

Se opta por usar *GHCR* como repositorio central de imágenes del proyecto. Los motivos de la decisión son:

- Mantener el sistema de roles y permisos existente en la organización de WESO y el proyecto *RDFShape* para subir, editar o eliminar imágenes *Docker*, en lugar de crear e imitar el sistema existente pero en *Docker Hub*.
- Generoso plan gratuito que se extiende a repositorios tanto públicos como privados.
- Evitar duplicar las explicaciones sobre el proyecto e instrucciones de instalación en *Docker Hub*, pudiendo tener una única fuente de código, manuales e imágenes *Docker* presente en *GitHub* o enlazada a través de ahí.

Además, se integrará la creación de imágenes *Docker* dentro de la línea de integración continua, de forma que cada nuevo cambio en el repositorio da lugar a una nueva imagen *Docker* con una etiqueta única. Se mantiene la posibilidad de que los administradores del proyecto creen y suban imágenes *Docker* con el identificador que deseen y se reserva la etiqueta "*dev*" para las imágenes testadas para su uso en producción.

2.2.3.3 *Generación de documentación web: mdoc + Docusaurus*

Se descarta el uso de *wikis* por las limitaciones de formato y por la imposibilidad de publicar el *Scaladoc* allí, así como el uso de *Jekyll* en favor de emplear *mdoc* en combinación con *Docusaurus*. Los principales motivos son:

- La capacidad de *mdoc* de definir variables dentro de la documentación.
- Las plantillas de sitio web por defecto de *Docusaurus* se adaptan perfectamente al caso de uso a tratar, al incluir las suficientes herramientas de personalización junto con otras funcionalidades base que permiten crear un sitio web profesional específico para cada proyecto.
- Al integrarse en el proyecto como *plugins* de *SBT*, el funcionamiento de *mdoc* y *Docusaurus* se puede personalizar completamente, encadenándolo con otras tareas como generar y publicar el *Scaladoc*. Además, incluye comandos de serie para publicar en *GitHub Pages*.

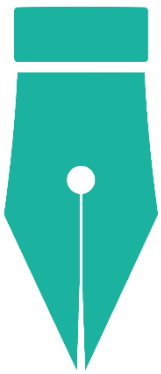


Ilustración 17. Logo de mdoc

Ilustración 18. Logo de Docusaurus

2.2.4 Integración continua: GitHub Actions

Se decide construir la línea de integración (y despliegue) continua de todos los módulos con *GitHub Actions* ya que:

- Está integrado en la propia interfaz web de *GitHub*.
- Es el sistema empleado en *RDFShape* en el momento de inicio del proyecto.
- El alumno posee mayor experiencia previa en esta alternativa.

2.2.5 Bases de datos: MongoDB (Atlas)

La opción final para la base de datos del proyecto es *MongoDB*. Los motivos de la elección son varios:

- Cuenta con bibliotecas oficiales, documentación y ejemplos para su uso con *Scala*.
- Permite una mayor flexibilidad y tolerancia a cambios en el desarrollo que las alternativas relacionales.
- Los requisitos de persistencia del proyecto son tan simples (sólo almacenar enlaces) que utilizar *MongoDB* hace que el esfuerzo de instalación inicial sea menor que con un SGBD relacional, además de dejar la puerta abierta a posibles modificaciones o ampliaciones en el sistema, que se estima seguirá evolucionando tras la finalización del proyecto.

Así mismo, para evitar saturar la infraestructura limitada del cliente y dada la escasez de datos a tratar, se decide que la mejor opción es usar el plan gratuito de *MongoDB Atlas* con el fin de:

- Externalizar la complejidad de administrar la base de datos.
- Dado que los enlaces almacenados persisten en la nube y no en la infraestructura del cliente, siempre van a estar disponibles aún si el proyecto migra de ubicación, se ejecuta en local o en servidores de terceras partes, etc. Por ejemplo, un enlace generado en un entorno de desarrollo local es almacenado en la nube y puede ser reutilizado en el sistema en producción para probar la misma operación en ambos entornos.
- Ofrecer un servicio público de *permalinks*.

Finalmente, y para prevenir un uso abusivo o malintencionado de la base de datos del proyecto, se tomarán las siguientes medidas:

- Se creará un rol específico en la base de datos, con los mínimos permisos de lectura y escritura para crear y leer *permalinks*. Dicho rol estará disponible públicamente y será el utilizado por defecto en *RDFShape*.
- Se creará una tarea o *trigger* en la base de datos que periódicamente eliminará aquellos enlaces que no hayan sido accedidos recientemente, a fin de contener el tamaño de la base de datos.

- Se permitirá que los usuarios puedan escoger una instancia de *MongoDB* de su elección mediante variables de entorno.

Ilustración 19. Logo de MongoDB

Capítulo 3 Estudio de Viabilidad del Sistema

Fase de Desarrollo

EVS

3.1 EVS 4, 5 y 6: Estudio y Valoración de Alternativas de Solución y Selección de Alternativa Final

La presente sección examina las alternativas públicas existentes para la validación de flujos de datos RDF mediante *Shape Expressions*.

Para un análisis pormenorizado del resto de necesidades y opciones tecnológicas para el desarrollo, despliegue y publicación del proyecto, ver el *Capítulo 2-Definición de la Arquitectura Tecnológica*.

3.1.1 Alternativas Disponibles

En el momento de realización del proyecto, no existe ninguna solución completa para validar flujos de datos RDF, de ahí que la construcción de una sea el objetivo principal. Como contraparte, sí existen varias herramientas para validar datos RDF mediante *ShEx*. Serán estas últimas herramientas las que se tengan en cuenta en el presente estudio, con la intención de integrarlas dentro de un contexto de flujos.

Para el listado y selección de alternativas, se han tomado como referencia las halladas durante el análisis de antecedentes (ver *1.3.1.2-Validación de datos RDF*), filtrando aquellas escritas en los lenguajes de la *JVM* y, en consecuencia, directamente compatibles con *RDFShape*.

3.1.1.1 *SHaclEX*

3.1.1.1.1 Descripción y Sitio Web

SHaclEX es una biblioteca desarrollada en *Scala* que implementa tanto *ShEx* como *SHACL* como mecanismos de validación de RDF. El código fuente, el Scaladoc y una *wiki* del proyecto se encuentran disponibles públicamente para apoyar a los desarrolladores con documentación y ejemplos de uso. Aunque es posible utilizar *SHaclEX* desde una interfaz de línea de comandos, los binarios están disponibles en *Maven* para la integración de la biblioteca en proyectos de terceros.

SHaclEX fue inicialmente desarrollada por José Emilio Labra Gayo **[90]**, investigador principal del grupo WESO de la Universidad de Oviedo, y actualmente es mantenida por WESO junto con otros colaboradores.

El sitio web del proyecto, junto con su código fuente, puede encontrarse aquí:

- https://www.weso.es/shaclex/

3.1.1.1.2 Ventajas

- Desarrollada y mantenida por la propia organización cliente, lo que permite acelerar la resolución de dudas y estar en contacto directo con el creador original de la herramienta.

- Capacidad de validar RDF utilizando tanto *ShEx* como *SHACL*, facilitando futuras ampliaciones del proyecto.
- Comparte el mismo ecosistema de desarrollo (*Scala* y *SBT*, programación funcional, etc.) que *RDFShape*, lo que facilitaría la integración final en la aplicación web y la reutilización del conocimiento adquirido en varios módulos del proyecto.
- Es la herramienta de validación de RDF actualmente en uso en el *backend* de *RDFShape*.

3.1.1.1.3 Inconvenientes

- En el momento de realización del estudio, carece de suficientes pruebas unitarias, lo que puede implicar contratiempos o errores inesperados durante el desarrollo. Este inconveniente es algo recurrente dado el nicho de mercado e investigación tan específico del proyecto.
- El ecosistema de desarrollo basado en Scala y programación funcional, junto con el amplio alcance de la herramienta, pueden incrementar la curva de dificultad sobremanera.
- Puede contagiar el estilo de programación funcional al resto de módulos que la utilicen, o bien forzar la inclusión de adaptadores (*adapters*) en estos.

3.1.1.2 *ShEx Java*

3.1.1.2.1 Descripción y Sitio Web

ShEx Java es una implementación de *ShEx* para el lenguaje de programación *Java* desarrollada y mantenida por personal de la Universidad de Lille. El código fuente del proyecto y el *Javadoc* se encuentran disponibles públicamente. Así mismo, es posible utilizar *ShEx Java* tanto desde una línea de comandos como a modo de biblioteca independiente disponible a través de *Maven*.

El código fuente del proyecto puede encontrarse aquí:

- https://github.com/iovka/shex-java

3.1.1.2.2 Ventajas

- Desarrollada en el lenguaje *Java*, que goza de gran popularidad y es altamente familiar al alumno.
- Cuenta con un amplio rango de pruebas unitarias.

3.1.1.2.3 Inconvenientes

- Limitada a la validación mediante *ShEx*, en contraposición con otras alternativas.
- La cantidad de documentación y ejemplos de uso es muy limitada. Los autores enlazan, a modo de ejemplo, a varios proyectos pequeños que utilizan esta implementación.

3.1.1.3 *Apache Jena*

3.1.1.3.1 Descripción y Sitio Web

El nombre de *Apache Jena* hace referencia a un *framework* completo escrito en *Java* para construir aplicaciones para la web semántica. Comprende varios APIs de desarrollo para manipular RDF, tratar con varias ontologías e inferencias o hacer consultas *SPARQL* sobre datos enlazados. De hecho, el resto de las alternativas del estudio hacen uso de *Jena* en mayor o menor medida para operar sobre RDF.

Jena incluye los módulos *jena-shex* y *jena-shacl* para operar con y validar con esquemas *ShEx* y *SHACL* tanto desde la línea de comandados como programáticamente mediante el uso de sus bibliotecas de código abierto.

El sitio web del proyecto, junto con su código fuente, puede encontrarse aquí:

- https://jena.apache.org/documentation/shex/

3.1.1.3.2 Ventajas

- Es el proyecto con mayor popularidad y publicidad de los estudiados, al estar amparado por la *Apache Software Foundation*.
- Extensa documentación, formada por la documentación del sitio web oficial así como el *Javadoc* automatizado.
- Amplio rango de pruebas unitarias.
- Comprende un ecosistema de módulos que permiten operar tanto con *ShEx* como con *SHACL*.

3.1.1.3.3 Inconvenientes

- Curva de aprendizaje elevada, el resto de las alternativas permiten usar *Jena* pero implementan varias utilidades por encima para manipular y validar RDF.

3.1.2 Selección Final

Finalmente, se ha decidido utilizar *SHaclEX* como biblioteca de validación de datos RDF. El módulo validador deberá por tanto ser capaz de recibir e interpretar RDF en flujos de datos externos y, a partir de allí, validarlo mediante *SHaclEX* para emitir el nuevo flujo de resultados.

Las causas que motivan la selección final son:

- **Relación directa con el grupo creador de la herramienta**
  Al contar con la ayuda de WESO para el proyecto, se tiene acceso privilegiado a los creadores de *SHaclEX*, mitigando la posibilidad de que aparezcan cuestiones y errores durante el uso de la biblioteca.
- **Soporte simultáneo para *ShEx* y *SHACL***
  Se valora muy positivamente que la herramienta no esté limitada a validar RDF mediante *Shape Expressions*, de manera que el soporte para *SHACL* pueda integrarse sin esfuerzo en el futuro.
- **Integración y similitudes con *RDFShape***
  Si bien el entorno de desarrollo de *SHaclEX* es el que porta una curva de dificultad más elevada, es el mismo que el empleado para *RDFShape*, la aplicación web del cliente en la que el módulo validador ha de integrarse. Los conocimientos adquiridos al emplear *SHaclEX* para construir la biblioteca de validación de flujos pueden trasladarse y aplicarse en la refactorización del API de *RDFShape*.
- **Visibilidad del grupo WESO**
  SHaclEX es una herramienta desarrollada y promocionada desde el grupo WESO, al igual que este proyecto. Emplear *SHaclEX* en la construcción del nuevo sistema de validación de flujos puede ampliar la repercusión de la biblioteca.
- **Acceso a *Apache Jena***
  SHaclEX hace uso de *Apache Jena* como dependencia para escanear y validar los contenidos de los datos RDF sobre los que opera, de forma que las herramientas base ofrecidas por *Jena* están disponibles cuando se emplea *SHaclEX*.

Capítulo 4 Análisis del Sistema de Información

Fase de Desarrollo

ASI

4.1 ASI 1: DEFINICIÓN DEL SISTEMA

4.1.1 Determinación del Alcance del Sistema

A continuación, se resumen y completan las principales metas de alcance para la solución propuesta del proyecto en base a lo descrito a lo largo del *Capítulo 1-Planificación del Sistema de Información*.

Como se ha comentado en secciones anteriores, el sistema final estará formado por tres módulos principales, construidos en torno a arquitectura tecnológica definida en *Capítulo 2-Definición de la Arquitectura Tecnológica* y *Capítulo 3-Estudio de Viabilidad del Sistema*:

1. La biblioteca de validación de flujos, desarrollada desde cero con el nombre de *Comet*.
2. *Backend* de la aplicación web *RDFShape*, consistente en un servidor HTTP que aloja un API REST (ampliado en base al código previo del cliente) y un servidor *WebSockets* (desarrollado para el proyecto).
3. Cliente web de la aplicación web *RDFShape*, ampliado en base al código previo del cliente.

La biblioteca de validación debe integrarse dentro de *RDFShape*, que expondrá sus funcionalidades a través de internet. Finalmente, el despliegue de *RDFShape* será automatizado, monitorizado y securizado mediante tecnología de contenedores.

4.1.1.1 *Biblioteca de Validación de Flujos (Comet)*

Se desarrollará una biblioteca capaz de leer flujos de datos en busca de datos RDF y emitir un nuevo flujo de resultados tras validar los datos RDF contra un esquema *Shape Expressions* determinado. Desde el exterior, se puede entender la funcionalidad de este módulo como una caja negra en la que entra un flujo de datos RDF y sale un flujo de datos con los resultados de la validación de cada ítem entrante, en una relación uno a uno.

Deberá implementar las siguientes funcionalidades:

1. **Recepción y lectura de flujos de datos**
    a. Interfaz común extensible para el procesamiento de todo tipo de flujos de datos bajo el marco de trabajo de *Fs2*
        i. Implementaciones disponibles para demostrar el funcionamiento con flujos de datos provenientes de distintas fuentes:
            1. *Apache Kafka*
            2. Listas en memoria
            3. Ficheros en disco.
    b. Capacidad para deserializar los contenidos del flujo de datos entrante a RDF
        i. El contenido recibido a través del flujo puede estar serializado en cualquier formato (texto en bruto o encriptado, instancias de clases…), la aplicación permitirá a los usuarios especificar cómo quieren deserializar los datos recibidos a RDF.
        ii. Implementaciones disponibles para deserializar flujos de texto a RDF.
    c. Se permitirá a los usuarios definir un intervalo temporal por el que el módulo debe esperar a que lleguen nuevos ítems del flujo de entrada antes de cerrar la conexión.

2. **Validación de flujos de RDF**
    a. Capacidad de validar cada ítem RDF mediante *Shape Expressions* a través de la biblioteca *SHaclEX*.
        i. Se permitirá a los usuarios especificar el esquema *ShEx* y el *ShapeMap* contra el que validar los datos RDF entrantes.
        ii. Se permitirá a los usuarios personalizar el comportamiento del sistema ante ítems RDF malformados, así como ante aquellos casos en que no pasen la validación.

3. **Emisión de resultados de la validación de flujos RDF**
    a. Los resultados de la validación se emitirán como un flujo de datos pensado para seguir siendo transformado o consumido por terceras partes.
    b. Cada elemento contenido en el flujo de datos saliente contendrá los resultados de la validación:
        i. El resultado completo de validación emitido por *SHaclEX*.

ii. Metadatos adicionales:
    1. Estado final de la validación: exitosa, no exitosa o detenida por errores.
    2. Instante temporal de la validación.

c. El flujo de datos resultante permitirá a las terceras partes que usen la biblioteca personalizar el manejo de errores producidos en la validación a través de un sistema de excepciones personalizadas.

4. Facilidad de uso y presencia en las redes

a. Los binarios de la biblioteca serán públicos y accesibles a los gestores de dependencias.
b. Sitio web del proyecto con identidad propia y accesible a través de internet
    i. Documentación *Scaladoc* completa y publicada.
    ii. Documentación redactada manualmente para la instalación y ejecución de la biblioteca.

4.1.1.2 *RDFShape*

La biblioteca de validación es el corazón del proyecto, pero es otro objetivo importante del proyecto integrarla en la plataforma *RDFShape* del grupo WESO para darle una utilidad real, así como demostrar su utilidad y publicitarla. Se realizan varias mejoras en la aplicación web para el proceso de integración de la biblioteca.

4.1.1.2.1 Backend

El *backend* de RDFShape seguirá exponiendo un API REST al exterior, con diversas herramientas para leer, convertir, consultar o validar datos RDF mediante *ShEx* y *SHACL*. Además, se refactorizarán, ampliarán y añadirán las siguientes funcionalidades:

1. **Documentación del módulo**
    a. Se completará la documentación del código del módulo a través de *Scaladoc*.
    b. Se completará la documentación del API expuesta por el módulo:
        i. El API se documentará de forma automatizada mediante *Rho*.
        ii. La documentación del API seguirá la *Open API Specification*.
2. **Sistema de registros o *logs***
    a. Se implementará un sistema de *logs* mediante *Logback*
    b. Se registrarán, anónimamente, las peticiones recibidas por el API, incluyendo la ruta solicitada, el origen de la solicitud y la fecha de la petición.
    c. Los registros se comprimirán y almacenarán en el sistema de archivos del servidor que ejecuta la aplicación y rotarán en base a un máximo tamaño permitido.
    d. Los administradores del sistema contarán con un fichero de configuración para alterar el sistema de registros según lo precisen.
3. **Sistema de enlaces permanentes acortados o *permalinks***
    a. El servidor será capaz de leer y escribir de una base de datos externa para almacenar enlaces a operaciones realizadas desde el cliente web.
    b. El API contará con nuevas rutas para exponer la funcionalidad de crear o recuperar *permalinks* a usuarios externos.
    c. Los enlaces deben contar con:
        i. Cuerpo del enlace, con la información necesaria para la operación que representa.
        ii. Fecha de creación del enlace, actualizada cada vez que se utilice para diferenciar los enlaces que estén en desuso de los que no.
        iii. Identificador único del enlace, utilizado para recuperar el resto de información de este.
4. **Despliegue mediante contenedores**
    a. Se crearán imágenes *Docker* que permitan a los administradores del sistema desplegar el módulo con mínimo esfuerzo.
5. **Nuevas rutas y servidor *WebSockets* para invocar la validación de flujos**

a. El módulo actuará como un servidor *WebSockets* para transmitir los mensajes de la validación a los clientes en tiempo real.
b. Los clientes del servidor podrán iniciar una nueva validación de un flujo de datos *Apache Kafka* y definir su comportamiento:
    i. Podrán definir la dirección y puerto de origen del flujo de datos RDF.
    ii. Podrán definir el esquema y *ShapeMap* empleados en la validación.
    iii. Podrán abortar la validación ante la aparición de errores o elementos inválidos.
c. Los administradores del sistema podrán definir el tiempo máximo que se mantendrá una conexión abierta esperando que lleguen nuevos datos del flujo antes de abortar la validación para ahorrar recursos.

6. **Facilidad de uso y presencia en las redes**
    a. Sitio web del proyecto con identidad propia y accesible a través de internet
        i. Documentación *Scaladoc* completa y publicada.
        ii. Documentación redactada manualmente para la instalación y ejecución del módulo.
    b. Documentación del API disponible a en *SwaggerHub*.

4.1.1.2.2 Frontend (Cliente Web)

El frontal para *RDFShape* proporcionado por WESO seguirá comunicándose con el servidor de la aplicación para permitir el uso del API desde una interfaz gráfica. Con el fin de integrar y consumir las validaciones de flujos desde el cliente web, se refactorizarán y añadirán las siguientes funcionalidades:

1. **Despliegue mediante contenedores**
    a. Se crearán imágenes *Docker* que permitan a los administradores del sistema desplegar el módulo con mínimo esfuerzo.
2. **Nuevas pantallas y cliente *WebSockets* para consumir la validación de flujos**
    a. El módulo actuará como un cliente *WebSockets* para solicitar al servidor la validación de flujos y recibir los resultados según se vayan generando.
    b. Los usuarios podrán iniciar una nueva validación de un flujo de datos *Apache Kafka* y definir su comportamiento desde una interfaz gráfica.
        i. Podrán introducir el origen del flujo y comportamiento antes errores del flujo mediante un formulario web.
        ii. Podrán introducir el esquema y *ShapeMap* de validación en campos de texto con detección de sintaxis y autocompletado.
        iii. Podrán pausar y continuar la validación en cualquier momento, así como abortarla completamente.
    c. Los usuarios podrán navegar por los resultados de la validación de forma visual e interactiva:
        i. Podrán ordenar los resultados recibidos por fecha de recepción.
        ii. Podrán realizar búsquedas textuales entre los resultados recibidos.

iii. Podrán exportar los resultados recibidos en formato CSV.

3. **Facilidad de uso y presencia en las redes**
   a. Los usuarios podrán navegar entre las distintas pantallas de la interfaz sin tener que reintroducir los datos y esquemas con los que estaban operando cada vez.
   b. Los usuarios podrán hacer uso del sistema de enlaces permanentes implementado en el API.
   c. Se incluirá un *favicon* y una previsualización de la web para las previsualizaciones de enlaces.

4.1.1.3 *Despliegue y Mantenimiento*

Se requieren las siguientes funcionalidades del despliegue del sistema final:

1. **Despliegue coordinado mediante contenedores**
    a. El administrador del sistema debe ser capaz de lanzar todos los componentes del sistema de forma automatizada y coordinada mediante *Docker Compose*.
    b. El administrador del sistema podrá configurar la ubicación a la que el cliente intentará encontrar el API.
2. **Gestión transparente de los dominios de la infraestructura del cliente**
    a. El API y el cliente serán desplegados bajo el domino "*rdfshape.weso.es*".
    b. Se utilizará un proxy inverso *Nginx*, operando en un contenedor aparte, para redirigir las peticiones al módulo API o al cliente web en función del subdominio solicitado, de forma que ambos módulos puedan convivir en los puertos 80 y/o 443.
3. **Monitorización y reinicio en caso de fallos**
    a. El API y el cliente contarán con chequeos periódicos para comprobar que siguen operando correctamente.
    b. Los administradores del sistema podrán configurar el despliegue para relanzar automáticamente los contenedores que no superen los chequeos mediante *Docker autoheal*, que operará en un contenedor aparte.
4. **Facilidad de uso y presencia en las redes**
    a. Tanto API como cliente serán accesibles de forma segura a través de HTTPS.
        i. Se instalarán certificados SSL en la máquina de despliegue.

4.1.1.4 *Observaciones y Limitaciones*

De forma complementaria a las metas anteriores, se concretan con el tutor las siguientes limitaciones y restricciones al alcance del proyecto:

- El núcleo y parte principal del proyecto es la biblioteca de validación de flujos y su publicación a través del API de *RDFShape*.
    - La filosofía detrás del módulo de validación es abrir la puerta a futuros trabajos de validación de flujos lo que mejoren o empleen. Al marco de trabajo creado para trabajar fácilmente con flujos y RDF se le añaden implementaciones para *Apache Kafka* y otros, a modo de demostración.
- Las funcionalidades adicionales del cliente web de *RDFShape* son sólo un prototipo no definitivo para:
    - Mostrar de forma amigables las posibilidades de la validación de flujos.
    - Emplearse de forma didáctica en cursos y charlas.
    - Atraer la atención de posibles usuarios y desarrolladores sobre la tecnología desarrollada.

Por tanto, quedan fuera del alcance del presente proyecto:

- Refactorizar los módulos del cliente, más allá de los requeridos para la implementación de la demo.
- Migrar el cliente total o parcialmente de *JavaScript* a *TypeScript*.
- Mejorar el cliente web en materia de usabilidad (gestión de usuarios, nuevas herramientas, etc.) y accesibilidad (fuentes, colores, contraste, etc.), más allá de los requisitos mínimos y pantallas necesarias para la demo.

4.2 ASI 2: ESTABLECIMIENTO DE REQUISITOS

4.2.1 Obtención de los Requisitos del Sistema

Ya establecido el alcance del proyecto, en esta sección se especifican formalmente los requisitos funcionales y no funcionales del sistema a desarrollar. Los requisitos se organizan por módulos, con una sección dedicada así mismo al despliegue del sistema.

4.2.1.1 *Requisitos Funcionales*

4.2.1.1.1 Biblioteca de Validación de Flujos (Comet)

RF-1. El sistema debe ser capaz de recibir e interpretar flujos de datos externos:

RF-1.1. El sistema permitirá a los usuarios configurar el proceso de lectura de ítems del flujo entrante antes de comenzar la validación:

RF-1.1.1. El sistema debe ser capaz de interpretar los contenidos del flujo de datos entrante como RDF:

RF-1.1.1.1. El sistema permitir a los usuarios especificar el tipo de datos que se esperan recibir del flujo de entrada.

RF-1.1.1.2. El sistema permitirá a los usuarios especificar una función para deserializar los datos entrantes a datos RDF.

RF-1.1.1.2.1. La función suministrada se aplicará a cada elemento del flujo entrante.

RF-1.1.1.2.1.1 La función recibirá como entrada valores del tipo especificado en *RF-1.1.1.1*.

RF-1.1.1.2.1.2 La función producirá como salida instancias de datos RDF procesables por la aplicación.

RF-1.1.1.2.2. El sistema implementará algunas funciones para la deserialización de los datos entrantes.

RF-1.1.1.2.2.1 Deserialización de cadenas de texto (*strings*) a RDF.

RF-1.1.1.3. Configuración del formato en que se espera recibir los datos RDF:

RF-1.1.1.3.1. El sistema soportará la recepción e interpretación de los siguientes formatos de RDF:

RF-1.1.1.3.1.1 Turtle.

RF-1.1.1.3.1.2 RDF/XML.

RF-1.1.1.3.1.3 RDF/JSON.

RF-1.1.1.3.1.4 JSON-LD.

RF-1.1.1.3.1.5 N-Triples.

RF-1.1.1.3.1.6 Tri-G.

RF-1.1.1.3.2. El sistema no permitirá iniciar la validación si no se especifica este valor.

RF-1.1.2. Configuración de la inferencia que se desea aplicar sobre los datos RDF recibidos:

RF-1.1.2.1. El sistema soportará no aplicar ninguna inferencia sobre los datos.

RF-1.1.2.2. El sistema soportará los siguientes motores de inferencia:

RF-1.1.2.2.1. RDFS.

RF-1.1.2.2.2. OWL.

RF-1.1.2.3. Por defecto, el sistema no aplicará ninguna inferencia sobre los datos.

RF-1.1.3. Configuración del número máximo de elementos que se permite al sistema extraer y deserializar a RDF en paralelo:

RF-1.1.3.1. Al recibir un valor de 1, el sistema extraerá y evaluará los elementos recibidos de manera secuencial.

RF-1.1.3.2. El valor por defecto será de 10 elementos.

RF-1.1.3.3. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-1.1.3.3.1. El valor debe ser un número entero.

RF-1.1.3.3.2. El valor debe ser mayor que 0.

RF-1.1.3.4. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-1.1.4. Configuración del período de tiempo máximo que el sistema debe esperar sin recibir nuevos elementos del flujo antes de cancelar la validación:

RF-1.1.4.1. El comportamiento por defecto del sistema será no cancelar la validación en ningún momento.

RF-1.1.4.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-1.1.4.2.1. Es una cantidad de tiempo positiva.

RF-1.1.4.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-1.1.4.4. El sistema interrumpirá la validación si transcurre un período de tiempo igual o superior al suministrado sin recibir nuevos elementos del flujo:

RF-1.1.4.4.1. El sistema reiniciará el período de espera cada vez que reciba un nuevo elemento de parte del flujo.

RF-1.1.4.4.2. El sistema emitirá un mensaje de error que incluirá el período de tiempo en el que se esperaba recibir nuevos elementos.

RF-2. El sistema debe poseer una interfaz común para el procesamiento de todo tipo de flujos de datos:

RF-2.1. La interfaz debe ser extensible por los desarrolladores del sistema para soportar nuevos flujos de datos.

RF-2.2. El sistema debe suministrar implementaciones de la interfaz funcionales para la validación de flujos provenientes de:

RF-2.2.1. *Apache Kafka*

RF-2.2.1.1. El sistema interpretará cada mensaje recibido de *Kafka* como un ítem independiente del flujo de datos de entrada.

RF-2.2.1.2. El sistema permitirá a los usuarios configurar el proceso de lectura de ítems del flujo de datos de *Kafka*:

RF-2.2.1.2.1. Configuración del *topic* de *Kafka* del que se desean leer mensajes:

RF-2.2.1.2.1.1 El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.1.1.1. Debe ser una cadena de caracteres no vacía.

RF-2.2.1.2.1.2 El sistema no permitirá iniciar la validación si no se especifica este valor.

RF-2.2.1.2.2. Configuración de la ubicación (origen) del flujo de datos de *Kafka*:

RF-2.2.1.2.2.1 Configuración del servidor origen del flujo:

RF-2.2.1.2.2.1.1. El sistema utilizará el dominio local "*localhost*" por defecto cuando no se especifique ningún valor.

RF-2.2.1.2.2.1.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.2.1.2.1. Debe ser una cadena de caracteres no vacía.

RF-2.2.1.2.2.1.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.2.2 Configuración del puerto origen del flujo:

RF-2.2.1.2.2.2.1. El sistema utilizará el valor *9092* por defecto cuando no se especifique ningún valor.

RF-2.2.1.2.2.2.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.2.2.2.1. Debe ser un número entero.

RF-2.2.1.2.2.2.2.2. Debe ser mayor que *1*.

RF-2.2.1.2.2.2.2.3. Debe ser menor que *65535*.

RF-2.2.1.2.2.2.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.3. Configuración del grupo de consumidores bajo el que debe operar el sistema:

RF-2.2.1.2.3.1 El sistema utilizará un valor por defecto cuando no se especifique, generado en base al nombre y versión de la aplicación:

RF-2.2.1.2.3.1.1. El valor contendrá el nombre de la aplicación dado en la configuración del proyecto.

RF-2.2.1.2.3.1.2. El valor contendrá la versión de la aplicación dada en la configuración del proyecto.

RF-2.2.1.2.3.1.3. El valor concatenará el nombre y versión de la aplicación.

RF-2.2.1.2.3.2 El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.3.2.1. Debe ser una cadena de caracteres no vacía.

RF-2.2.1.2.3.3 El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.4. Configuración del comportamiento del sistema al sincronizar con *Kafka* que elementos del flujo han sido procesados:

RF-2.2.1.2.4.1 Estrategia que seguir cuando no se posee información del último mensaje procesado:

RF-2.2.1.2.4.1.1. El sistema soportará las siguientes estrategias:

RF-2.2.1.2.4.1.1.1. Consumir desde el inicio: se consumen todos los mensajes disponibles desde el inicio del flujo.

RF-2.2.1.2.4.1.1.2. Consumir desde el final: se ignoran los mensajes previos y se consumen a partir del momento de conexión al flujo.

RF-2.2.1.2.4.1.1.3. No consumir: el sistema interrumpirá la validación y emitirá un mensaje de error si no hay información de los mensajes procesados.

RF-2.2.1.2.4.1.2. El sistema, por defecto, comenzará a procesar mensajes a partir del último recibido (ver *RF-2.2.1.2.4.1.1.2*).

RF-2.2.1.2.4.2 Estrategia que seguir a la hora de indicar a *Kafka* qué mensajes del flujo han sido procesados:

RF-2.2.1.2.4.2.1. El sistema, por defecto, marcará los mensajes como procesados automáticamente según se consuman.

RF-2.2.1.2.4.2.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.4.2.2.1. Debe ser verdadero o falso.

RF-2.2.1.2.4.2.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.4.3 Estrategia que seguir cuando falla el proceso de marcado de mensajes como procesados:

RF-2.2.1.2.4.3.1. El sistema soportará las siguientes estrategias:

RF-2.2.1.2.4.3.1.1. Reintentar: reintentar contactar con el servidor y actualizar los mensajes leídos hasta 15 veces.

RF-2.2.1.2.4.3.1.2. Ninguna: ignorar los fallos.

RF-2.2.1.2.4.3.2. El sistema, por defecto, reintentará el marcado de los mensajes (ver *RF-2.2.1.2.4.3.1.1*).

RF-2.2.1.2.4.3.2.1. El sistema interrumpirá la validación y emitirá un mensaje de error si fallan todos los reintentos.

RF-2.2.1.2.4.4 Intervalo máximo de tiempo permitido para indicar a *Kafka* qué mensajes han sido procesados:

RF-2.2.1.2.4.4.1. El sistema utilizará un valor por defecto de *15* segundos.

RF-2.2.1.2.4.4.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.4.4.2.1. Es una cantidad de tiempo mayor que *10* milisegundos.

RF-2.2.1.2.4.4.2.2. Es una cantidad de tiempo menor que *5* minutos

RF-2.2.1.2.4.4.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.4.5 Intervalo de tiempo máximo entre sincronizaciones del con *Kafka* sobre el estado de procesado de los mensajes del flujo:

RF-2.2.1.2.4.5.1. El sistema utilizará un valor por defecto de *15* segundos.

RF-2.2.1.2.4.5.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.4.5.2.1. Es una cantidad de tiempo mayor que *10* milisegundos.

RF-2.2.1.2.4.5.2.2. Es una cantidad de tiempo menor que *5* minutos

RF-2.2.1.2.4.5.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.5. Configuración del intervalo temporal que el sistema intentará conectar con el servidor de *Kafka* inicialmente o ante interrupciones de conexiones existentes:

RF-2.2.1.2.5.1 El sistema utilizará un valor por defecto de *10* segundos.

RF-2.2.1.2.5.2 El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.5.2.1. Es una cantidad de tiempo positiva.

RF-2.2.1.2.5.3 El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.6. Configuración del comportamiento del sistema para extraer nuevos elementos del flujo de *Kafka*:

RF-2.2.1.2.6.1 Intervalo máximo de tiempo para cada operación de recuperación de nuevos elementos del flujo:

RF-2.2.1.2.6.1.1. El sistema utilizará un valor por defecto de *50* milisegundos

RF-2.2.1.2.6.1.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.6.1.2.1. Es una cantidad de tiempo mayor que *10* milisegundos.

RF-2.2.1.2.6.1.2.2. Es una cantidad de tiempo menor que *5* minutos.

RF-2.2.1.2.6.1.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.1.2.6.2 Intervalo de tiempo máximo entre solicitudes de nuevos elementos del flujo:

RF-2.2.1.2.6.2.1. El sistema utilizará un valor por defecto de *50* milisegundos

RF-2.2.1.2.6.2.2. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-2.2.1.2.6.2.2.1. Es una cantidad de tiempo mayor que *10* milisegundos.

RF-2.2.1.2.6.2.2.2. Es una cantidad de tiempo menor que *5* minutos.

RF-2.2.1.2.6.2.3. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-2.2.2. Listas de objetos en memoria:

RF-2.2.2.1. El sistema interpretará cada elemento de la lista como un ítem independiente del flujo de datos de entrada.

RF-2.2.2.2. El sistema permitirá a los usuarios configurar el proceso de lectura de ítems del flujo de datos de listas:

RF-2.2.2.2.1. Configuración del listado de objetos que se deben validar:

RF-2.2.2.2.1.1 El sistema no permitirá iniciar sesión si no se especifica este valor.

RF-2.2.3. Conjuntos de ficheros en disco:

RF-2.2.3.1. El sistema interpretará los contenidos de cada fichero como un ítem independiente del del flujo de datos de entrada.

RF-2.2.3.2. El sistema permitirá a los usuarios configurar el proceso de lectura de ítems del flujo de datos de ficheros:

RF-2.2.3.2.1. Configuración del conjunto de ficheros que se deben validar:

RF-2.2.3.2.1.1 El sistema no permitirá iniciar sesión si no se especifica este valor.

RF-2.2.3.2.2. Configuración del juego de caracteres empleado en los ficheros que se deben validar:

RF-2.2.3.2.2.1 El sistema empleará por defecto el juego de caracteres *UTF-8* para la lectura de los contenidos de los ficheros.

RF-2.2.3.2.2.2 El sistema soportará los siguientes juegos de caracteres:

RF-2.2.3.2.2.2.1. UTF-8.

RF-2.2.3.2.2.2.2. UTF-16.

RF-2.2.3.2.2.2.3. UTF-32.

RF-2.2.3.2.2.2.4. ASCII.

RF-2.2.3.2.2.2.5. ISO-8859-1.

RF-3. El sistema debe ser capaz de validar elementos RDF extraídos del flujo de datos de entrada:

RF-3.1. El sistema validará cada elemento del flujo mediante el mecanismo *Shape Expressions*:

RF-3.1.1. El sistema permitirá a los usuarios configurar el esquema *ShEx* contra el que se validarán los datos:

RF-3.1.1.1. El sistema soportará los siguientes formatos de *Shape Expressions*:

RF-3.1.1.1.1. ShExC.

RF-3.1.1.1.2. ShExJ.

RF-3.1.1.2. El sistema no iniciará la validación y emitirá un mensaje de error si la *Shape Expression* introducida no es válida.

RF-3.1.2. El sistema permitirá a los usuarios configurar el *ShapeMap* contra el que se validarán los datos:

RF-3.1.2.1. El sistema soportará los siguientes formatos de *ShapeMap*:

RF-3.1.2.1.1. Compacto.

RF-3.1.2.1.2. JSON.

RF-3.1.2.2. El sistema no iniciará la validación y emitirá un mensaje de error si el *ShapeMap* introducido no es válido.

RF-3.2. El sistema permitirá a los usuarios configurar el comportamiento de la validación antes de que comience:

RF-3.2.1. Comportamiento del sistema ante excepciones durante el proceso de validación:

RF-3.2.1.1. El sistema soportará las siguientes estrategias:

RF-3.2.1.1.1. Detener:

RF-3.2.1.1.1.1 El sistema interrumpirá la validación del flujo de datos.

RF-3.2.1.1.1.2 El sistema emitirá un mensaje con la información del error.

RF-3.2.1.1.2. Ignorar:

RF-3.2.1.1.2.1 Para el elemento fallido, el sistema emitirá un resultado que incluirá el motivo del error en lugar de resultados de validación.

RF-3.2.1.1.2.2 El sistema continuará procesando los ítems del flujo con normalidad.

RF-3.2.1.2. El sistema, por defecto, ignorará los errores producidos (ver *RF-3.2.1.1.2*).

RF-3.2.2. Comportamiento del sistema ante ítems que no superen la validación:

RF-3.2.2.1. El sistema soportará las siguientes estrategias:

RF-3.2.2.1.1. Detener:

RF-3.2.2.1.1.1 El sistema emitirá un último elemento conteniendo el resultado de la validación del ítem inválido.

RF-3.2.2.1.1.2 El sistema interrumpirá la validación del flujo de datos y emitirá un mensaje de error.

RF-3.2.2.1.2. Ignorar:

RF-3.2.2.1.2.1 El sistema emitirá un nuevo elemento conteniendo el resultado de la validación del ítem inválido.

RF-3.2.2.1.2.2 El sistema continuará procesando los ítems del flujo con normalidad.

RF-3.2.2.1.3. El sistema, por defecto, ignorará la presencia de datos RDF inválidos (ver *RF-3.2.2.1.2*).

RF-3.2.3. Número máximo de elementos que se permite al sistema validar en paralelo:

RF-3.2.3.1. Al recibir un valor de 1, el sistema extraerá y evaluará los elementos recibidos de manera secuencial.

RF-3.2.3.2. El valor por defecto será de 10 elementos.

RF-3.2.3.3. El sistema validará que el valor suministrado cumpla las siguientes condiciones:

RF-3.2.3.3.1. El valor debe ser un número entero.

RF-3.2.3.3.2. El valor debe ser mayor que 0.

RF-3.2.3.4. El sistema no permitirá el inicio de la validación y emitirá un mensaje de error si se suministra un valor que no cumple las condiciones.

RF-4. El sistema debe emitir los resultados de la validación:

RF-4.1. El sistema emitirá los resultados de la validación como un flujo de datos.

RF-4.1.1. El flujo de datos seguirá la misma interfaz que la empleada por la biblioteca para manipular los flujos de datos.

RF-4.2. Los usuarios del sistema deberán poder manipular los resultados de la validación:

RF-4.2.1. Los resultados de la validación serán flujos manipulables desde el código de terceros que empleen la biblioteca.

RF-4.2.1.1. Cada resultado será un ítem del flujo de salida.

RF-4.2.1.2. Cada resultado incluirá:

RF-4.2.1.2.1. El estado final de la validación del ítem.

RF-4.2.1.2.1.1 Válido: validado sin errores.

RF-4.2.1.2.1.2 Inválidos: validado con errores.

RF-4.2.1.2.1.3 Erróneo: no se ha podido validar pues se han producido errores.

RF-4.2.1.2.2. Los resultados detallados de la validación, en caso de que no haya sido errónea.

RF-4.2.1.2.3. Datos complementarios:

RF-4.2.1.2.3.1 Instante de finalización de la validación

RF-4.2.2. El sistema permitirá que los usuarios finales del sistema definan el comportamiento del sistema para el manejo de errores.

RF-4.2.2.1. El sistema incluirá un conjunto de clases de error para cada problema específico del sistema.

RF-4.2.2.1.1. Se incluyen los siguientes errores relacionados con el manejo de flujos:

RF-4.2.2.1.1.1 *Timeout* del flujo de datos:

RF-4.2.2.1.1.1.1. Se lanzará el error cuando el tiempo sin recibir ítems del flujo de entrada supere el especificado.

RF-4.2.2.1.1.1.2. El error incluirá la siguiente información:

RF-4.2.2.1.1.1.2.1. Límite de tiempo que ha sido excedido.

RF-4.2.2.1.2. Se incluyen los siguientes errores relacionados con la validación de flujos:

RF-4.2.2.1.2.1 Ítem erróneo en el flujo:

RF-4.2.2.1.2.1.1. Se lanzará el error cuando:

RF-4.2.2.1.2.1.1.1. Un ítem del flujo produzca un fallo.

RF-4.2.2.1.2.1.1.2. El sistema esté configurado para detenerse en caso de fallos.

RF-4.2.2.1.2.1.2. El error incluirá la siguiente información:

RF-4.2.2.1.2.1.2.1. Error original que ha causado la detención.

RF-4.2.2.1.2.2 Ítem inválido en el flujo:

RF-4.2.2.1.2.2.1. Se lanzará el error cuando:

RF-4.2.2.1.2.2.1.1. Un ítem del flujo no supere la validación con éxito.

RF-4.2.2.1.2.2.1.2. El sistema esté configurado para detenerse en caso de elementos no válidos.

RF-4.2.2.1.2.2.2. El error incluirá la siguiente información:

RF-4.2.2.1.2.2.2.1. Error original lanzado por la herramienta de validación que ha caudado la detención, si está disponible.

RF-4.2.2.1.2.2.2.2. El resultado de la validación del ítem inválido.

RF-4.2.2.2. Los errores definidos por la biblioteca serán promocionados para permitir a los usuarios finales manejar la respuesta.

RF-5. El sistema debe estar disponible públicamente:

RF-5.1. El código fuente del sistema debe estar publicado en un repositorio público del sistema gestor de versiones del grupo WESO.

RF-5.2. Los binarios de la biblioteca deben estar alojados en un repositorio público.

RF-5.2.1. Será posible automatizar la descarga automatizada de la biblioteca desde gestores de dependencias del ecosistema *Java*.

RF-5.2.1.1. Se soportarán, como mínimo, los siguientes gestores:

RF-5.2.1.1.1. Maven.

RF-5.2.1.1.2. Gradle.

RF-5.2.1.1.3. SBT.

RF-5.2.1.1.4. Apache Ivy.

RF-5.2.2. Los binarios serán actualizados automáticamente con cada nueva versión del sistema.

RF-6. El sistema debe tener presencia en las redes:

RF-6.1. El proyecto debe contar con un sitio web propio.

RF-6.1.1. El sitio web debe alojar:

RF-6.1.1.1. La documentación *Scaladoc* del módulo.

RF-6.1.1.2. Documentación redactada manualmente para el uso del módulo:

RF-6.1.1.2.1. Instrucciones de instalación.

RF-6.1.1.2.2. Instrucciones de ejecución.

RF-6.1.1.2.2.1 Se deben incluir ejemplos.

RF-6.1.1.2.3. Instrucciones para desarrolladores para la ampliación del sistema.

RF-6.1.2. El sitio web debe enlazar a:

RF-6.1.2.1. El repositorio del proyecto.

4.2.1.1.2 RDFShape (API)

RF-7. El sistema debe estar documentado:

RF-7.1. El sistema debe contar con documentación *Scaladoc* de todas las clases del módulo.

RF-7.2. El sistema debe contar con documentación para los clientes del módulo:

RF-7.2.1. El sistema contará con documentación del API REST expuesta:

RF-7.2.1.1. La documentación seguirá el estándar *Open API Specification*.

RF-7.2.1.2. La documentación se generará automáticamente a partir del código del módulo.

RF-8. El sistema debe contar con un mecanismo de registros o *logs*:

RF-8.1. Se implementará un sistema de *logs* mediante *Logback*.

RF-8.1.1. Se registrará la siguiente información:

RF-8.1.1.1. Peticiones recibidas por el API:

RF-8.1.1.1.1. Iniciador de la solicitud

RF-8.1.1.1.1.1 Recurso solicitado.

RF-8.1.1.1.1.2 Encabezados (*headers*) de la petición.

RF-8.1.1.1.1.3 Agente de usuario (*User Agent*).

RF-8.1.1.1.2. Respuesta producida:

RF-8.1.1.1.2.1 Código de estado HTTP de respuesta.

RF-8.1.1.1.2.2 Encabezados (*headers*) de la respuesta.

RF-8.1.1.1.3. Instante de la petición.

RF-8.1.1.2. Mensajes emitidos desde el código del proyecto.

RF-8.1.2. Los registros se almacenarán en el sistema de archivos del servidor de la aplicación:

RF-8.1.2.1. Se guardarán en el sistema de archivos los mensajes de *log* con nivel *INFO* o superior: *INFO*, *WARN*, *ERROR*.

RF-8.1.2.2. Los registros se almacenarán en una carpeta en la misma ruta que el ejecutable de la aplicación.

RF-8.1.2.2.1. El nombre de la carpeta por defecto será "*logs*".

RF-8.1.2.3. Los registros se almacenarán repartido en varios ficheros independientes:

RF-8.1.2.3.1. Cada fichero de registros tendrá un límite de tamaño configurable.

RF-8.1.2.3.1.1 El límite por defecto será de *50MB*.

RF-8.1.2.3.2. Se almacenarán una cantidad máxima y configurable de ficheros de registros:

RF-8.1.2.3.2.1 El máximo por defecto será de *10* ficheros.

RF-8.1.2.3.2.2 Cada fichero de registro tendrá un nombre único en base al nombre de la aplicación y el índice del fichero.

RF-8.1.2.3.3. Los ficheros de registros rotarán según se supere el tamaño máximo establecido.

RF-8.1.2.3.3.1 Los ficheros de registros no activos se comprimirán para ahorrar espacio.

RF-8.1.2.3.3.1.1. Se usará el formato de compresión *ZIP*.

RF-8.2. Los administradores del sistema tendrán acceso al fichero de configuración para alterar el sistema de registros según lo precisen.

RF-8.2.1. Los cambios en la configuración del sistema de registros tendrán efecto sin reiniciar la aplicación.

RF-8.2.1.1. El sistema comprobará y aplicará los cambios en la configuración periódicamente:

RF-8.2.1.1.1. El intervalo de comprobación por defecto será de *60* segundos.

RF-8.2.1.1.2. Si la nueva configuración no es válida, el sistema no aplicará la nueva configuración.

RF-9. El sistema debe ofrecer un mecanismo de enlaces permanentes acortados o *permalinks*:

RF-9.1. Los enlaces deben almacenar información sobre operaciones realizadas desde el cliente web.

RF-9.1.1. Cuerpo del enlace.

RF-9.1.1.1. URL completa.

RF-9.1.2. Fecha de uso del enlace.

RF-9.1.2.1. Inicialmente, será la fecha en que se ha creado el enlace.

RF-9.1.2.2. Se actualizará cada vez que se acceda al enlace.

RF-9.1.3. Identificador único del enlace.

RF-9.1.3.1. Será de tipo numérico entero.

RF-9.1.3.2. Se generará de forma que se mitigue el riesgo colisión entre enlaces.

RF-9.1.3.2.1. Se usarán el instante de tiempo actual y un factor aleatorio para generar el identificador.

RF-9.2. El servidor será capaz de leer y escribir de una base de datos externa para almacenar enlaces a operaciones realizadas desde el cliente web.

RF-9.2.1. El API contará con nuevas rutas para exponer la funcionalidad de crear o recuperar *permalinks* a usuarios externos.

RF-9.2.1.1. Ruta para la generación de enlaces:

RF-9.2.1.1.1. El sistema recibirá de parte del cliente la siguiente información:

RF-9.2.1.1.1.1 URL para la que se desea generar un enlace acortado.

RF-9.2.1.1.1.2 El sistema validará que la información recibida por el cliente cumple las siguientes condiciones:

RF-9.2.1.1.1.2.1. La URL recibida es válida respecto al *RFC 1738* **[91]**.

RF-9.2.1.1.1.2.2. Si no se cumplen las condiciones, el sistema devolverá un código 400 con un mensaje de error.

RF-9.2.1.1.1.3 El sistema comprobará que la URL no cuenta ya con un enlace disponible:

RF-9.2.1.1.1.3.1. Si existe un enlace para la URL, el sistema devolverá un código 200 junto con el identificador del enlace.

RF-9.2.1.1.1.4 Si no existe un enlace, el sistema procederá a crearlo:

RF-9.2.1.1.1.4.1. El sistema generará un nuevo identificador único para el enlace.

RF-9.2.1.1.1.4.2. El sistema escribirá en nuevo enlace en la base de datos:

RF-9.2.1.1.1.4.2.1. Si el enlace se inserta con éxito, el sistema devolverá un código 201 junto con el identificador del enlace.

RF-9.2.1.1.1.4.2.2. Si el enlace no puede insertarse, el sistema devolverá un código 500 con un mensaje de error.

RF-9.2.1.2. Ruta GET para la recuperación de enlaces:

RF-9.2.1.2.1. El sistema recibirá de parte del cliente la siguiente información:

RF-9.2.1.2.1.1 Identificador del enlace que desea recuperar.

RF-9.2.1.2.2. El sistema validará que la información recibida del cliente cumple las siguientes condiciones:

RF-9.2.1.2.2.1 El identificador recibido es válido.

RF-9.2.1.2.2.2 Si no se cumplen las condiciones, el sistema devolverá un código 400 con un mensaje de error.

RF-9.2.1.2.3. El sistema buscará en enlace solicitado en la base de datos:

RF-9.2.1.2.3.1 Si el enlace existe, devolverá un código 200 junto a la URL original del enlace.

RF-9.2.1.2.3.2 Si el enlace no existe, devolverá un código 404 con un mensaje de error.

RF-10. El sistema debe ser desplegable mediante contenedores:

RF-10.1. Se crearán imágenes *Docker* que permitan a los administradores del sistema desplegar el módulo con mínimo esfuerzo.

RF-10.2. Los contenedores permitirán al administrador del sistema configurar características de la aplicación.

RF-10.2.1. Se podrán cambiar las siguientes configuraciones:

RF-10.2.1.1. Puerto en el que está expuesto el módulo.

RF-10.2.1.2. Uso de HTTPS frente a HTTP.

RF-10.2.1.2.1. Si se activa, el servidor tratará de servir los contenidos bajo HTTPS.

RF-10.2.1.3. Intervalo de tiempo máximo tras el cual descartar una validación de flujos que no ha recibido nuevos elementos.

RF-10.2.2. Se podrá cambiar la configuración del sistema bajo los siguientes criterios:

RF-10.2.2.1. No será necesario regenerar las imágenes.

RF-10.2.2.2. La configuración se realizará en el momento del lanzamiento de los contenedores de la aplicación.

RF-10.2.2.3. La configuración se realizará mediante variables de entorno.

RF-11. El sistema debe ofrecer a los clientes la posibilidad de validar flujos de datos:

RF-11.1. El sistema utilizará la biblioteca de validación desarrollada para el proyecto para validar flujos de datos provenientes de *Apache Kafka*.

RF-11.1.1. El sistema recibirá los siguientes datos del cliente para iniciar la validación:

RF-11.1.1.1. Información sobre el esquema de validación:

RF-11.1.1.1.1. Esquema *ShEx* a emplear en la validación.

RF-11.1.1.1.1.1 El esquema se leerá y formateará imitando el trabajo previo del módulo.

RF-11.1.1.1.2. *ShapeMap* a emplear en la validación.

RF-11.1.1.1.2.1 El *ShapeMap* se leerá y formateará imitando el trabajo previo del módulo.

RF-11.1.1.2. Información sobre el comportamiento del validador:

RF-11.1.1.2.1. Estrategia ante errores inesperados.

RF-11.1.1.2.2. Estrategia ante ítems inválidos.

RF-11.1.1.2.3. Para cada campo, se aplican las mismas restricciones que las de la biblioteca (ver *RF-3.2*).

RF-11.1.1.3. Información para la extracción de datos del flujo:

RF-11.1.1.3.1.1 Formato de los datos RDF recibidos a través del flujo.

RF-11.1.1.3.1.1.1. Se aceptarán los formatos especificados en la biblioteca de validación (ver *RF-1.1.1.3.1*).

RF-11.1.1.3.1.2 Inferencia que aplicar sobre los datos RDF recibidos a través del flujo.

RF-11.1.1.3.1.3 Se aceptarán los motores de inferencia especificados en la biblioteca de validación (ver *RF-1.1.2*).

RF-11.1.1.4. Configuración de la extracción de datos del flujo:

RF-11.1.1.4.1. Origen del flujo:

RF-11.1.1.4.1.1 Servidor.

RF-11.1.1.4.1.2 Puerto.

RF-11.1.1.4.2. *Topic*.

RF-11.1.1.4.3. Grupo de consumidores.

RF-11.1.1.4.4. Para cada campo, se aplican las mismas restricciones que las de la biblioteca (ver *RF-2.2.1.2*).

RF-11.2. El módulo actuará como un servidor *WebSockets* para comunicarse con los clientes en la validación de flujos.

RF-11.2.1. Los clientes podrán iniciar una validación enviando un mensaje al servidor.

RF-11.2.1.1. Los clientes deberán suministrar la información necesaria para la validación (ver *RF-11.1.1*) en formato JSON.

RF-11.2.2. Los clientes recibirán los resultados de la validación desde el servidor.

RF-11.2.2.1. Cada resultado de la validación pertenecerá a un mensaje del servidor hacia el cliente.

RF-11.2.2.2. Cada resultado incluirá la información resultante de la validación (ver *RF-4.2.1.2*) en formato JSON.

RF-12. El sistema debe estar disponible públicamente:

RF-12.1. El código fuente del sistema debe estar publicado en un repositorio público del sistema gestor de versiones del grupo WESO.

RF-12.2. La documentación del API del módulo debe estar publicada en la cuenta de *SwaggerHub* del grupo WESO.

RF-12.3. Las imágenes *Docker* del módulo deben estar alojadas en un repositorio público.

RF-12.3.1. Las imágenes serán actualizadas automáticamente con cada nueva versión del sistema.

RF-13. El sistema debe tener presencia en las redes:

RF-13.1. El proyecto debe contar con un sitio web propio.

RF-13.1.1. El sitio web debe alojar:

RF-13.1.1.1. La documentación *Scaladoc* del módulo.

RF-13.1.1.2. Documentación redactada manualmente para el uso del módulo:

RF-13.1.1.2.1. Instrucciones de instalación.

RF-13.1.1.2.2. Instrucciones de ejecución.

RF-13.1.2. El sitio web debe enlazar a:

RF-13.1.2.1. El repositorio del proyecto.

RF-13.1.2.2. La documentación del API del proyecto.

4.2.1.1.3 *RDFShape* (Cliente)

RF-14. El sistema debe ser desplegable mediante contenedores:

RF-14.1. Se crearán imágenes *Docker* que permitan a los administradores del sistema desplegar el módulo con mínimo esfuerzo.

RF-14.2. Los contenedores permitirán al administrador del sistema configurar características de la aplicación.

RF-14.2.1. Se podrán cambiar las siguientes configuraciones:

RF-14.2.1.1. Ubicación a la que se dirigirán las peticiones al API.

RF-14.2.2. Se podrá cambiar la configuración del sistema bajo los siguientes criterios:

RF-14.2.2.1. No será necesario regenerar las imágenes.

RF-14.2.2.2. La configuración se realizará en el momento del lanzamiento de los contenedores de la aplicación.

RF-14.2.2.3. La configuración se realizará mediante variables de entorno.

RF-15. El sistema debe permitir a los clientes visualizar validaciones de flujos a través del API:

RF-15.1. El módulo actuará como un cliente *WebSockets* para solicitar al servidor la validación de un flujo de datos *Apache Kafka*.

RF-15.2. El sistema permitirá operar sobre validaciones de flujos a través de una interfaz web:

RF-15.2.1. El sistema permitirá especificar la configuración de la validación (ver *RF-11.1.1*).

RF-15.2.1.1. La configuración de origen del flujo y comportamiento antes errores se introducirá en un formulario web.

RF-15.2.1.1.1. El cliente validará que los datos proporcionados son válidos:

RF-15.2.1.1.1.1 El servidor origen del flujo no podrá estar vacío.

RF-15.2.1.1.1.2 El puerto origen del flujo no podrá estar vacío.

RF-15.2.2. El sistema permitirá especificar el esquema de validación:

RF-15.2.2.1. Los usuarios podrán introducir el esquema *ShEx* en un campo de texto.

RF-15.2.2.1.1. Los usuarios contarán con un editor de texto mejorado:

RF-15.2.2.1.1.1 Se utilizará el trabajo previo con *YASHE*.

RF-15.2.2.1.1.2 Detección de errores.

RF-15.2.2.1.1.3 Sintaxis coloreada.

RF-15.2.2.1.1.4 Autocompletado de prefijos.

RF-15.2.2.2. Los usuarios podrán introducir el *ShapeMap* en un campo de texto.

RF-15.2.2.2.1. Los usuarios contarán con un editor de texto mejorado:

RF-15.2.2.2.1.1 Se utilizará el trabajo previo de *YATE*.

RF-15.2.2.2.1.1.1. Sintaxis coloreada.

RF-15.2.3. El sistema permitirá operar sobre la validación del flujo que esté activa:

RF-15.2.3.1. Cancelar la validación en curso.

RF-15.2.3.1.1. Sólo se permitirá cuando haya una validación activa.

RF-15.2.3.2. Pausar la validación en curso.

RF-15.2.3.2.1. Sólo se permitirá cuando haya una validación activa reproduciéndose.

RF-15.2.3.3. Reanudad la validación en curso.

RF-15.2.3.3.1. Sólo se permitirá cuando haya una validación activa pausada.

RF-15.2.4. El sistema permitirá navegar por los resultados de la validación:

RF-15.2.4.1. Los resultados contendrán la siguiente información:

RF-15.2.4.1.1. Informe detallado de la validación.

RF-15.2.4.1.2. Datos resumen de la validación:

RF-15.2.4.1.2.1 Identificador del nodo validado.

RF-15.2.4.1.2.2 Identificador de la *Shape* contra la que se ha validado.

RF-15.2.4.1.2.3 Resultado de la validación:

RF-15.2.4.1.2.3.1. Válido.

RF-15.2.4.1.2.3.2. Inválido.

RF-15.2.4.1.2.4 Fecha de emisión del resultado.

RF-15.2.4.2. Se podrán consultar los resultados de la validación aun cuando esté en curso.

RF-15.2.4.2.1. Los resultados se actualizarán en la interfaz según lleguen resultados nuevos.

RF-15.2.4.3. Se podrán ordenar los resultados.

RF-15.2.4.3.1. Se podrán ordenar en base a los campos en *RF-15.2.4.1.2*.

RF-15.2.4.4. Se podrán realizar búsquedas textuales entre los resultados.

RF-15.2.4.4.1. La búsqueda comprenderá los campos en *RF-15.2.4.1*.

RF-15.2.4.5. Se podrán exportar los resultados en formato CSV.

RF-15.2.4.5.1. Se exportarán todos los resultados recibidos desde el inicio de la validación.

RF-15.2.4.5.2. El CSV resultante comprenderá los campos en *RF-15.2.4.1*.

RF-16. El sistema debe ser sencillo de utilizar:

RF-16.1. Los usuarios podrán navegar entre las distintas pantallas de la interfaz manteniendo sin reintroducir los datos que estaban operando.

RF-16.1.1. El sistema almacenará la siguiente información entre pantallas:

RF-16.1.1.1. Datos RDF.

RF-16.1.1.1.1. Contenido.

RF-16.1.1.1.2. Formato.

RF-16.1.1.1.3. Inferencia.

RF-16.1.1.1.4. Para flujos de datos:

RF-16.1.1.1.4.1 Origen del flujo:

RF-16.1.1.1.4.1.1. Servidor.

RF-16.1.1.1.4.1.2. Puerto.

RF-16.1.1.1.4.2 Comportamiento ante errores:

RF-16.1.1.1.4.2.1. Ítem errado.

RF-16.1.1.1.4.2.2. Ítem inválido.

RF-16.1.1.2. Esquema de validación.

RF-16.1.1.2.1. Contenido.

RF-16.1.1.2.2. Formato.

RF-16.1.1.3. *ShapeMap* de validación.

RF-16.1.1.3.1. Contenido.

RF-16.1.1.3.2. Formato.

RF-16.2. El sistema permitirá a los usuarios hacer uso del sistema de enlaces acortados implementado en el servidor (ver *RF-9*).

RF-17. El sistema debe tener presencia en las redes:

RF-17.1. El sistema incluirá un *favicon*.

RF-17.2. El sistema incluirá una imagen personalizada para las previsualizaciones de enlaces.

4.2.1.1.4 Despliegue del Sistema

RF-18. El despliegue se realizará en la infraestructura del cliente:

RF-18.1. Servidores de WESO en la red de la Universidad de Oviedo.

RF-19. El despliegue de todos los módulos del sistema estará automatizado:

RF-19.1. El sistema permitirá un despliegue coordinado mediante *Docker Compose*.

RF-19.1.1. Se suministrará un fichero de configuración listo para completar el despliegue automáticamente.

RF-19.1.2. Se utilizarán las siguientes imágenes para el despliegue:

RF-19.1.2.1. Las imágenes de cada módulo desarrollado para *RDFShape*:

RF-19.1.2.1.1. API.

RF-19.1.2.1.2. Cliente.

RF-19.1.2.2. Otras imágenes para colaborar en la monitorización del despliegue.

RF-20. El sistema permitirá gestionar los dominios de la infraestructura de forma transparente:

RF-20.1. Se utilizará un proxy inverso para redirigir las peticiones al módulo API o al cliente web en función del (sub)dominio solicitado, de forma que ambos módulos puedan convivir en los puertos 80 y 443.

RF-20.1.1. El proxy será operado por un contenedor independiente.

RF-20.1.2. El API será desplegado bajo el domino "*api.rdfshape.weso.es*".

RF-20.1.3. El cliente será desplegado bajo el domino "*rdfshape.weso.es*"

RF-21. El estado del sistema será monitorizable a los administradores

RF-21.1. El API y el cliente contarán con chequeos periódicos para comprobar que siguen operando correctamente.

RF-21.1.1. Se implementarán mediante *healthchecks* de *Docker*.

RF-21.1.2. Los chequeos consistirán en los contenedores lanzándose peticiones HTTP a sí mismos para comprobar que los módulos web responden normalmente.

RF-21.1.3. Los administradores del sistema podrán configurar el despliegue para relanzar automáticamente los contenedores que no superen los chequeos:

RF-21.1.3.1. El sistema de vigilancia y reinicio de los contenedores será operado en un contenedor independiente.

RF-22. Facilidad de uso y presencia en las redes

RF-22.1. Los módulos API y cliente serán accesibles de forma segura a través de HTTPS.

RF-22.2. Se instalarán certificados SSL en la máquina de despliegue.

4.2.1.2 *Requisitos no Funcionales*

4.2.1.2.1 Biblioteca de Validación

RNF-1. Extensibilidad:

RNF-1.1. El módulo validador debe estar desarrollado de la manera que más facilite ampliar el sistema a otros desarrolladores.

RNF-2. Publicidad:

RNF-2.1. El módulo validador debe estar desarrollado de manera que facilite su adopción por el mayor público posible.

RNF-2.1.1. Se compilará la biblioteca para ser compatible con varias versiones de *Scala*.

4.2.1.2.2 *RDFShape* (API)

RNF-3. Seguridad:

RNF-3.1. El API expuesta al exterior debe seguir la arquitectura REST.

RNF-3.2. El servidor no hará acopio de datos personales de los usuarios.

RNF-3.2.1. El servidor recogerá las mínimas estadísticas de uso de la aplicación de manera anónima.

RNF-3.2.2. No se implementarán utilidades de analítica de uso a nivel de cliente.

4.2.1.2.3 *RDFShape* (Cliente)

RNF-4. Accesibilidad:

RNF-4.1. El cliente web debe cumplir con el nivel de accesibilidad AA del estándar WCAG.

RNF-4.1.1. Se verificará el cumplimiento del estándar mediante herramientas de validación independientes:

RNF-4.1.1.1. AChecker **[92]**.

RNF-5. Interfaz web:

RNF-5.1. La interfaz debe cumplir unos mínimos de responsividad:

RNF-5.1.1. No se contempla que la interfaz esté optimizada para dispositivos móviles (*responsiveness*).

RNF-5.1.2. La interfaz debe adaptarse a cualquier resolución comercial de escritorio.

RNF-5.1.3. La interfaz debe adaptarse a cualquier relación de aspecto comercial de escritorio.

RNF-5.2. La interfaz web debe ser fácilmente usable.

RNF-5.2.1. El tiempo de formación en el uso del sistema no debe ser mayor a 30 minutos.

RNF-5.3. La interfaz debe ser completamente manejable con el uso de teclado.

RNF-6. Respuesta e interacción:

RNF-6.1. El cliente web debe responder a las acciones de usuario. Se contemplan los siguientes mecanismos:

RNF-6.1.1. Cambios visuales.

RNF-6.1.1.1. Reflejar las entradas del usuario.

RNF-6.1.1.2. Representar las salidas del sistema.

RNF-6.1.2. Scroll o desplazamiento automático hacia zonas de interés.

RNF-6.1.2.1. Se llevará al usuario a la sección de los resultados una vez iniciada la validación.

RNF-6.1.3. *Pop-ups* o elementos temporales.

RNF-6.1.4. Otros mecanismos convenientes.

RNF-6.2. Los tiempos de respuesta de la aplicación deben ser rápidos y no superar los 15 segundos en ningún caso.

RNF-7. Internacionalización:

RNF-7.1. El cliente web deberá estar internacionalizado y localizado a los siguientes idiomas:

RNF-7.1.1. Inglés.

RNF-8. Navegadores:

RNF-8.1. La aplicación web debe ser accesible desde, al menos, los navegadores de escritorio más utilizados a fecha de inicio del proyecto **[93]**:

RNF-8.1.1. Google Chrome.

RNF-8.1.2. Apple Safari.

RNF-8.1.3. Microsoft Edge.

RNF-8.1.4. Mozilla Firefox.

4.2.1.2.4 Despliegue del Sistema

RNF-9. Seguridad

RNF-9.1. Todos los módulos web del proyecto deben ofrecer sus servicios de forma encriptada a través de certificados SSL.

RNF-9.1.1. Las conexiones *HTTP* serán cifradas como *HTTPS*.

RNF-9.1.2. Las conexiones *WebSockets* (*WS*) serán cifradas como *WSS*.

RNF-9.2. El sistema intentará, por defecto, redireccionar las peticiones que usen protocolos inseguros a su equivalente encriptado.

RNF-10. Disponibilidad:

RNF-10.1. El sistema desplegado será accesible a cualquier dispositivo con conexión a internet.

RNF-10.2. El sistema desplegado tendrá una disponibilidad del 99%.

RNF-11. Eficiencia:

RNF-11.1. El sistema desplegado debe soportar al menos 20 usuarios simultáneos.

4.2.1.3 *Restricciones técnicas*

Se dan las siguientes restricciones técnicas del proyecto, completadas tras dialogar con el cliente y definir la arquitectura final del sistema. Las presentes restricciones complementan los requisitos finales:

RT-1. Se utilizará *Scala* como lenguaje de programación del módulo validador de flujos:
- RT-1.1. El sistema de construcción del proyecto será *SBT*.
- RT-1.2. El proyecto seguirá el paradigma de programación funcional.

RT-2. Se utilizará *Docker* como tecnología de contenedores para el despliegue del sistema.

RT-3. Se desplegará el sistema final en los servidores de la Universidad de Oviedo proporcionados por WESO.

RT-4. Se utilizará *GitHub* como sistema gestor de versiones del código del proyecto.
- RT-4.1. Los módulos del proyecto se publicarán en la cuenta de *GitHub* de WESO.
- RT-4.2. Las páginas web asociadas a cada proyecto se desplegarán en *GitHub Pages* **[94]**.

RT-5. El proyecto se desarrollará siguiendo la metodología Métrica v.3.

4.2.2 Identificación de Actores del Sistema

En esta sección se especifican los distintos actores que interactuarán con el sistema.

4.2.2.1 *Usuario Biblioteca*

Usuario con un perfil programador que interactúa, a través de código, con el API expuesto por la biblioteca de validación de flujos. Posee las siguientes capacidades:

- Integrar la biblioteca de validación de flujos en su proyecto.
    - Añadir la biblioteca al sistema gestor de dependencias.
- Validar flujos de datos RDF programáticamente:
    - Configurar la validación de flujos:
        - Configuración del flujo de entrada:
            - Deserialización de los datos de entrada.
            - Formato e inferencia del RDF.
            - Fuente del flujo.
            - Período de espera antes de abortar la validación.
        - Configuración del proceso de validación:
            - Esquema y *ShapeMap* de validación.
            - Interrupción ante datos erróneos o inválidos.
    - Extender la biblioteca para nuevos flujos de datos.
- Consultar la documentación de la biblioteca:
    - *Scaladoc*.
    - Manuales de uso.

4.2.2.2 *Usuario API*

Usuarios que interactúan con el sistema a través de la funcionalidad expuesta por *RDFShape*. Al estar limitados a las funcionalidades públicas de la demo del sistema, tiene la capacidad de validar flujos provenientes de *Apache Kafka*, dejando que el servidor de la aplicación haga el trabajo de computación.

Según el modo en que interactúan con el API, se subdividen en:

4.2.2.2.1 Usuario Backend

Utiliza el API de *RDFShape* manual o programáticamente a través de cualquier software con capacidades de hacer de cliente *HTTP* y *WebSockets*. Posee la capacidad de:

- Validar flujos de datos RDF provenientes de *Apache Kafka*.
    - Operar sobre la validación del flujo:
        - Iniciar la validación.

- Configurar la validación.
    - Formato e inferencia del RDF.
    - Esquema y *ShapeMap* de validación.
    - Servidor de origen del flujo.
    - Interrupción ante datos erróneos o inválidos.
- Detener la validación en curso.

- Consultar la documentación del sistema:
    - API Docs de *RDFShape*.
    - Manuales de uso.

4.2.2.2.2 Usuario Cliente

Hace uso del API de *RDFShape* a través del cliente web e interfaces gráficas facilitados para ese fin. Al hacer uso de la interfaz de usuario del sistema, posee capacidades adicionales respecto al uso programático del API:

- Validar flujos de datos RDF provenientes de *Apache Kafka*.
    - Operar sobre la validación del flujo:
        - Pausar la validación en curso.
        - Continuar la validación en curso previamente pausada.
    - Operar los resultados de la validación:
        - Ordenar los resultados.
        - Limpiar el histórico de resultados de la interfaz.
        - Exportar los resultados como CSV.
        - Buscar entre los resultados.
- Gestionar los enlaces acortados o *permalinks* para validaciones.
    - Crear un enlace a la validación existente.
    - Recuperar una validación a partir de un enlace.
- Consultar menús de ayuda para el uso de la interfaz.

4.2.2.3 *Administrador del Sistema*

Administrador con perfil técnico encargado del despliegue y mantenimiento de los módulos. Puede realizar las siguientes tareas:

- Despliegue automatizado del sistema.
- Monitorización del estado del sistema desplegado.
    - Reinicio automático configurable de los módulos fallidos.
        - Condiciones para marcar un módulo como fallido.
        - Módulos que reiniciar.
        - Tiempo transcurrido entre comprobaciones de estado.
- Configurar las propiedades de cada módulo desplegado:
    - API:

        - Tiempo de espera para abortar la validación de flujos.
    - Cliente:
        - Dirección en la que buscar el API del sistema.

4.2.2.4 *Servidor Remoto*

Un servidor ajeno al sistema actúa como actor secundario en cada validación de flujos a través de la red. El servidor remoto es el encargado de generar el flujo de datos de entrada del sistema y trasmitirlo a través de la tecnología que corresponda. En el caso de *RDFShape*, dicha tecnología es *Apache Kafka*. Posee las siguientes funcionalidades:

- Recibir conexiones del exterior.
- Autenticar al sistema para permitirle el consumo del flujo de datos.

4.2.3 Especificación de Casos de Uso

Los casos de uso definen el comportamiento del sistema cuando interactúa con los actores definidos previamente. A continuación, se listan y definen los casos de uso identificados junto con sus diagramas correspondientes.

4.2.3.1 *Usuario Biblioteca*

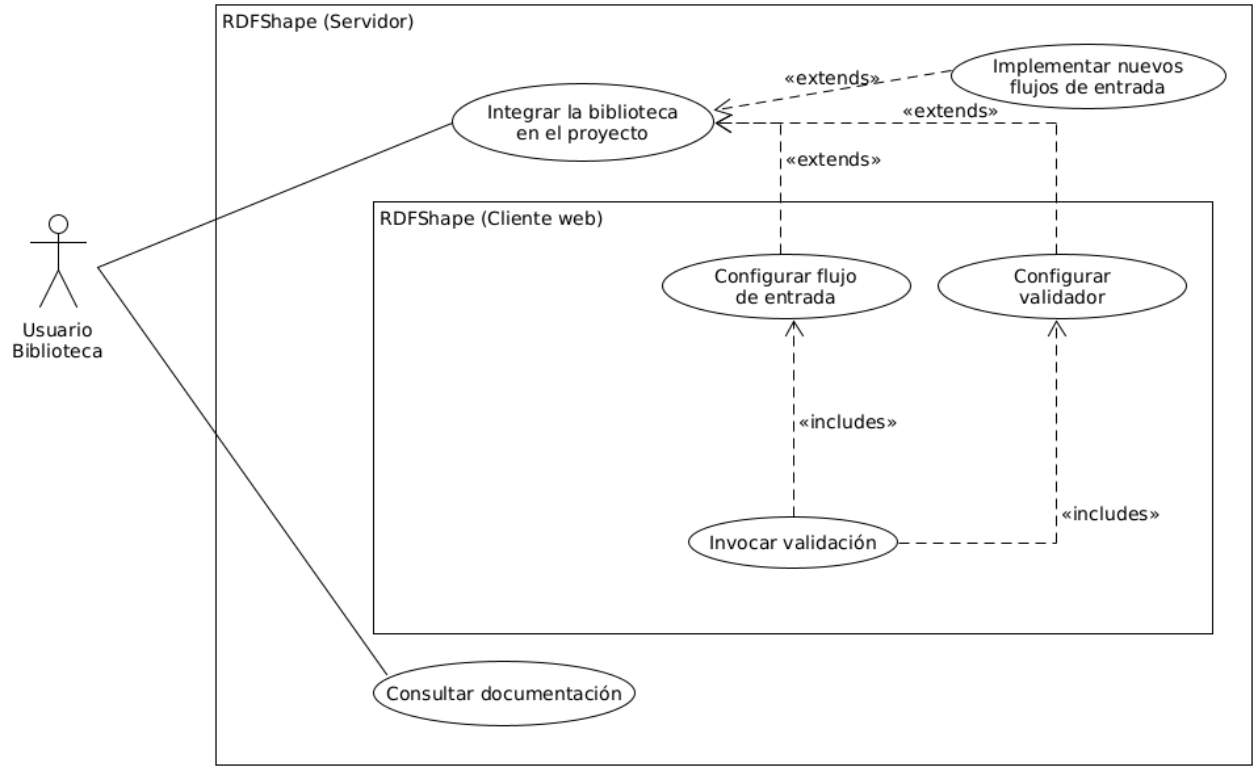


Ilustración 20. Especificación de Casos de Uso - Usuario Biblioteca

Nombre del Caso de Uso
Integrar la biblioteca de validación en el proyecto
Descripción
El programador añadirá la información de la biblioteca de validación de flujos al gestor de dependencias del proyecto, con el objetivo de emplear las funcionalidades expuestas por el módulo validador en su propio proyecto.

Tabla 3. Especificación caso de uso "Integrar la biblioteca de validación en el proyecto" (Usuario Biblioteca)

Nombre del Caso de Uso
Configurar el flujo de entrada
Descripción
El programador podrá configurar el origen y las propiedades del flujo de datos de entrada para la validación, incluyendo: - Fuente del flujo. - Mecanismo de deserialización de los datos recibidos a RDF. - Formato e inferencia del RDF recibido. - Período máximo de inactividad del flujo antes de abortar la validación.

Tabla 4. Especificación caso de uso "Configurar flujo de entrada" (Usuario Biblioteca)

Nombre del Caso de Uso
Configurar el validador de los ítems del flujo
Descripción
El programador podrá configurar el proceso de validación de los datos RDF del flujo de entrada, incluyendo: - Esquema *ShEx* contra el que validar los datos. - *ShapeMap* contra el que validar los datos. - Comportamiento del sistema ante errores en el procesamiento de los datos. - Comportamiento del sistema ante datos que no superen la validación.

Tabla 5. Especificación caso de uso "Configurar el validador de los ítems del flujo" (Usuario Biblioteca)

Nombre del Caso de Uso
Invocar una validación de flujos
Descripción
El programador podrá iniciar la validación de un flujo de datos entrante en base a la configuración del flujo y del validador. El sistema verificará la configuración y comenzará a procesar y emitir el flujo de resultados.

Tabla 6. Especificación caso de uso "Invocar una validación de flujos" (Usuario Biblioteca)

Nombre del Caso de Uso
Implementar nuevos flujos de entrada
Descripción
El programador podrá extender las clases existentes en el sistema para implementar la validación de flujos de entrada provenientes de nuevas fuentes.

Tabla 7. Especificación caso de uso "Implementar nuevos flujos de entrada" (Usuario Biblioteca)

Nombre del Caso de Uso
Consultar documentación de la biblioteca
Descripción
El programador podrá consultar la documentación de uso de la biblioteca a través del sitio web del módulo, incluyendo: - *Scaladoc* adjunto al propio código de la biblioteca para obtener información exhaustiva del sistema, incluyendo clases, métodos, parámetros y excepciones. - Manuales de instalación y uso.

Tabla 8. Especificación caso de uso "Consultar documentación de la biblioteca" (Usuario Biblioteca)

4.2.3.2 *Usuario API*

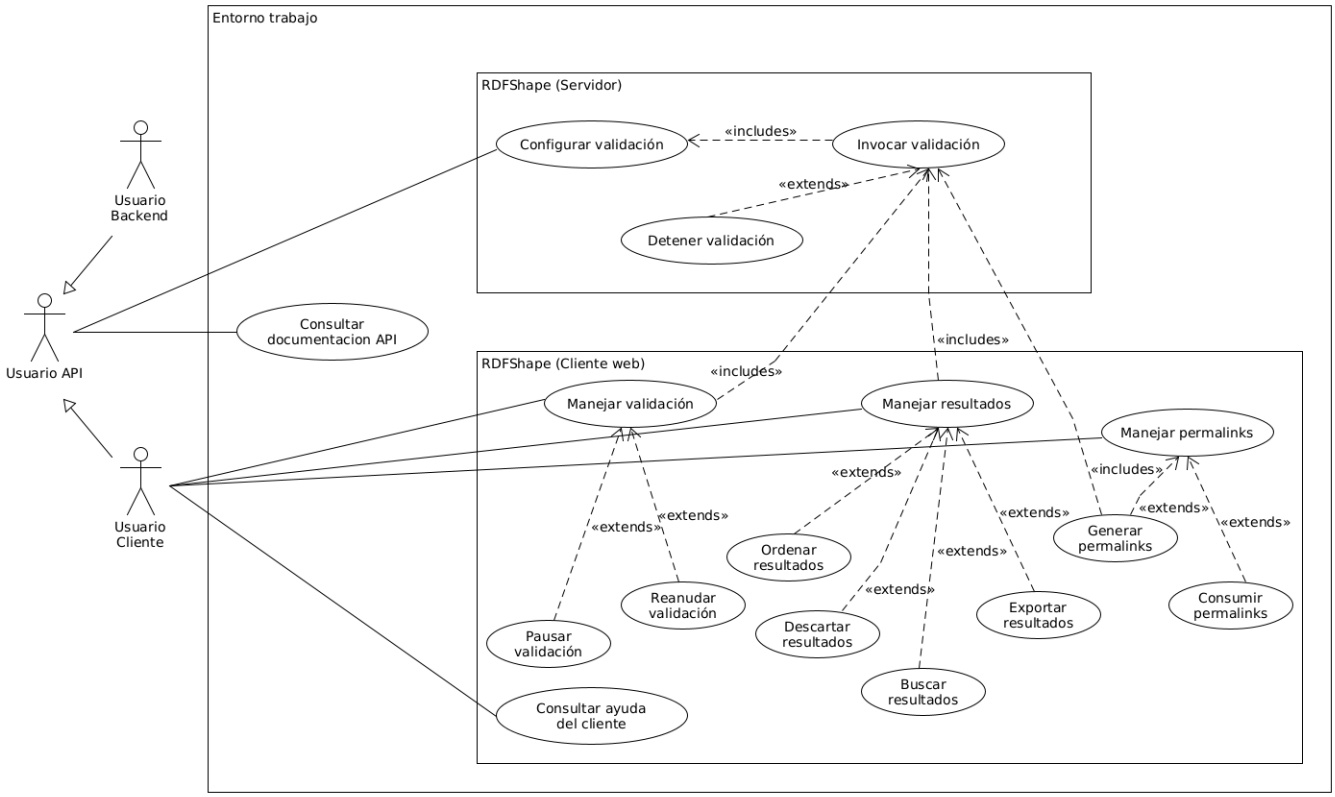


Ilustración 21. Especificación de Casos de Uso - Usuario API

4.2.3.2.1 Usuario Backend

Nombre del Caso de Uso
Configurar la validación de los ítems del flujo
Descripción
Los usuarios podrán configurar el comportamiento del servidor para leer y validar el procesamiento del flujo de entrada, incluyendo: - Formato e inferencia del RDF. - Esquema y *ShapeMap* de validación. - Servidor de origen del flujo. - Interrupción ante datos erróneos o inválidos.

Tabla 9. Especificación caso de uso "Configurar la validación de los ítems del flujo" (Usuario API - Backend)

Nombre del Caso de Uso
Invocar una validación de flujos
Descripción
Los usuarios podrán iniciar la validación de un flujo de datos proveniente de *Apache Kafka* en base a la configuración de la validación. El sistema verificará la configuración y comenzará a procesar y enviar mensajes a los clientes con el contenido del flujo de resultados.

Tabla 10. Especificación caso de uso "Invocar una validación de flujos" (Usuario API - Backend)

Nombre del Caso de Uso
Detener una validación de flujos
Descripción
Los usuarios podrán finalizar una validación de flujos que hayan invocado previamente. El sistema interrumpirá la conexión con el flujo de entrada, dejará de emplear recursos en validar elementos y detendrá el envío de resultados al cliente.

Tabla 11. Especificación caso de uso "Detener una validación de flujos" (Usuario API - Backend)

Nombre del Caso de Uso
Consultar documentación del API
Descripción
Los usuarios podrán consultar la documentación de uso del API expuesta por *RDFShape* a través de *SwaggerHub* y el sitio web del módulo, incluyendo: - *API Docs* coherentes con la *Open API Specification*. - Manuales de instalación y uso.

Tabla 12. Especificación caso de uso "Consultar documentación del API" (Usuario API - Backend)

4.2.3.2.2 Usuario Cliente

Cabe destacar que los usuarios del API de *RDFShape* a través del cliente web tienen acceso a las capacidades del resto de usuarios del *backend*, más las descritas aquí.

Nombre del Caso de Uso
Pausar una validación de flujos
Descripción
Los usuarios podrán pausar una validación de flujos que hayan invocado previamente. El sistema detendrá la validación y dejará de mostrar resultados, pero conservará los resultados y configuración previas para poder reanudar el consumo del flujo de RDF.

Tabla 13. Especificación caso de uso "Pausar una validación de flujos" (Usuario API - Cliente)

Nombre del Caso de Uso
Reanudar una validación de flujos
Descripción
Los usuarios podrán reanudar una validación de flujos que hayan pausado previamente. El sistema reconectará con la fuente del flujo usando las mismas opciones de conexión y validación que la validación pausada y volverá a mostrar los resultados procesados.

Tabla 14. Especificación caso de uso "Reanudar una validación de flujos" (Usuario API - Cliente)

Nombre del Caso de Uso
Ordenar resultados de una validación de flujos
Descripción
Los usuarios podrán ordenar los resultados de la validación en la interfaz del cliente sin necesidad de detener la validación y en torno a varios parámetros, incluyendo: - Identificador del nodo validado. - Identificador de la *Shape* contra la que se ha validado el nodo. - Resultado de la validación. - Fecha de emisión del resultado.

Tabla 15. Especificación caso de uso "Ordenar resultados de una validación de flujos" (Usuario API - Cliente)

Nombre del Caso de Uso
Descartar resultados de una validación de flujos
Descripción
Los usuarios podrán eliminar los resultados de la validación recibidos hasta el momento sin necesidad de detener la validación. El sistema descartá los resultados recibidos y desaparecerán de la interfaz del cliente.

Tabla 16. Especificación caso de uso "Descartar resultados de una validación de flujos" (Usuario API - Cliente)

Nombre del Caso de Uso
Buscar resultados de una validación de flujos
Descripción
Los usuarios podrán realizar búsquedas textuales entre los resultados de la validación desde la interfaz del cliente sin necesidad de detener la validación y en torno a varios parámetros, incluyendo: - Identificador del nodo validado. - Identificador de la *Shape* contra la que se ha validado el nodo. - Resultados detallados de la validación.

Tabla 17. Especificación caso de uso "Buscar resultados de una validación de flujos" (Usuario API - Cliente)

Nombre del Caso de Uso
Exportar resultados de una validación de flujos
Descripción
Los usuarios podrán exportar los resultados de la validación a formato CSV desde la interfaz del cliente sin necesidad de detener la validación.

Tabla 18. Especificación caso de uso "Exportar resultados de una validación de flujos" (Usuario API - Cliente)

Nombre del Caso de Uso
Generar *permalinks*
Descripción
Los usuarios podrán generar enlaces acortados permanentes asociados a una validación. El sistema generará el enlace y permitirá al usuario guardarlo.

Tabla 19. Especificación caso de uso "Generar permalinks" (Usuario API - Cliente)

Nombre del Caso de Uso
Consumir *permalinks*
Descripción
Los usuarios podrán acceder a una validación accediendo al enlace permanente asociado a la misma. El sistema redirigirá a los usuarios a la validación y la iniciará.

Tabla 20. Especificación caso de uso "Consumir permalinks" (Usuario API - Cliente)

Nombre del Caso de Uso
Consultar ayuda del cliente
Descripción
Los usuarios podrán acceder a menús de ayuda sobre las capacidades y uso de la herramienta de validación de flujos del cliente.

Tabla 21. Especificación caso de uso "Consultar ayuda del cliente" (Usuario API - Cliente)

4.2.3.3 *Administrador del Sistema*

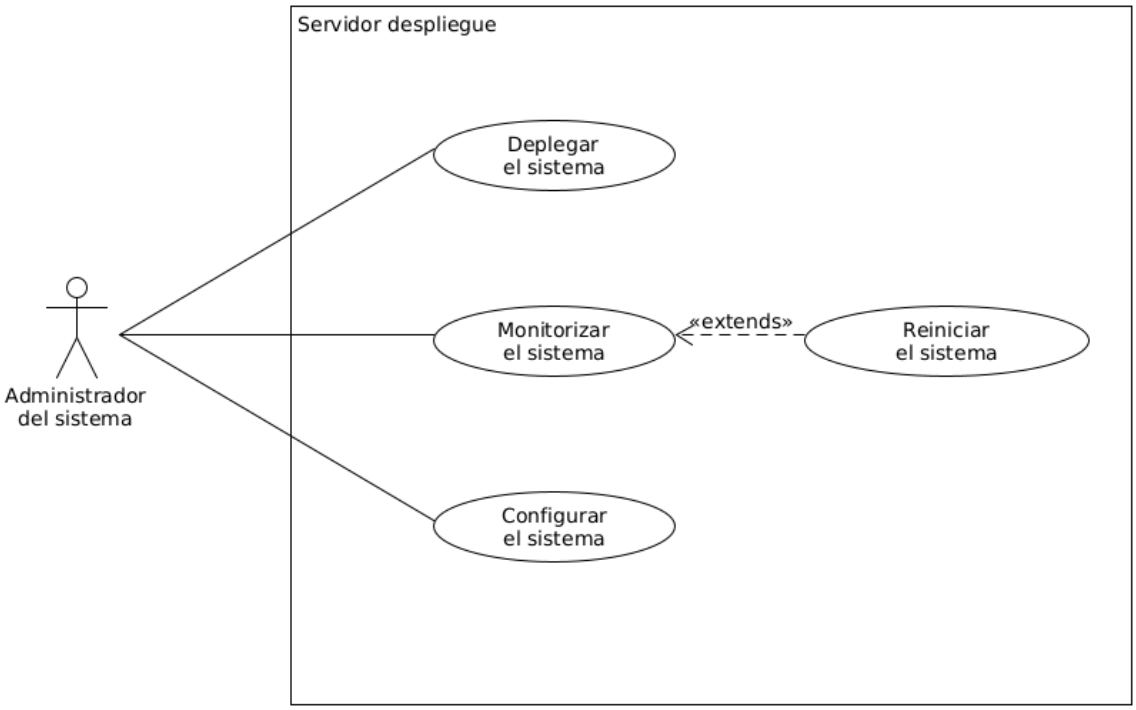


Ilustración 22. Especificación de Casos de Uso - Administrador del Sistema

Nombre del Caso de Uso
Desplegar el sistema
Descripción
El administrador del sistema podrá desplegar todos los módulos automáticamente y en el orden deseado. El sistema se iniciará y será accesible en el servidor de despliegue.

Tabla 22. Especificación caso de uso "Desplegar el sistema" (Administrador del Sistema)

Nombre del Caso de Uso
Monitorizar el sistema
Descripción
El administrador del sistema podrá visualizar el estado de cada uno de los módulos del sistema y configurar las condiciones para marcar un módulo como caído.

Tabla 23. Especificación caso de uso "Monitorizar el sistema" (Administrador del Sistema)

Nombre del Caso de Uso
Reiniciar el sistema
Descripción
El administrador del sistema podrá configurar el reinicio automático de los módulos caídos, incluyendo: - Qué módulos reiniciar automáticamente. - El intervalo de tiempo entre comprobaciones de estado de los módulos.

Tabla 24. Especificación caso de uso "Reiniciar el sistema" (Administrador del Sistema)

Nombre del Caso de Uso
Configurar el sistema
Descripción
El administrador del sistema podrá configurar ciertas propiedades del sistema durante en el momento del despliegue sin manipular el código de la aplicación, incluyendo: - El tiempo de espera del servidor antes de abortar una validación de flujos por inactividad. - La URL base en la que el cliente de *RDFShape* intentará contactar con el API.

Tabla 25. Especificación caso de uso "Configurar el sistema" (Administrador del Sistema)

4.2.3.4 *Servidor Remoto*

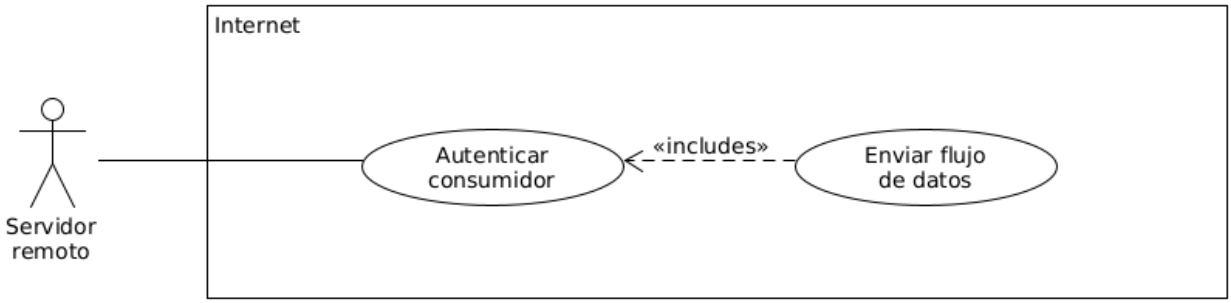


Ilustración 23. Especificación de Casos de Uso - Servidor Remoto

Nombre del Caso de Uso
Autenticar consumidor
Descripción
El sistema usará la configuración suministrada para contactar con el servidor remoto que está generando el flujo de entrada de la validación. El servidor utilizará la configuración para autenticar al consumidor del flujo.

Tabla 26. Especificación caso de uso "Autenticar consumidor" (Servidor Remoto)

Nombre del Caso de Uso
Enviar flujo de datos
Descripción
El servidor y permitirá al sistema consumir el flujo de datos y lo transmitirá mediante la tecnología y/o protocolo correspondiente.

Tabla 27. Especificación caso de uso "Enviar flujo de datos" (Servidor Remoto)

4.2.4 Diagramas de Estado y Flujo

En este apartado se incluyen diagramas para ayudar a comprender los flujos y cambios de estado que se dan en el funcionamiento de algunos componentes sin entrar en detalles de implementación.

4.2.4.1 *Validación de un elemento del flujo entrante de RDF en la biblioteca de validación*

Se incluye el diagrama de los estados por los que pasa la biblioteca de validación al procesar un flujo de datos externo desde que el sistema inicia la validación hasta que se generan los resultados de las validaciones de cada ítem.

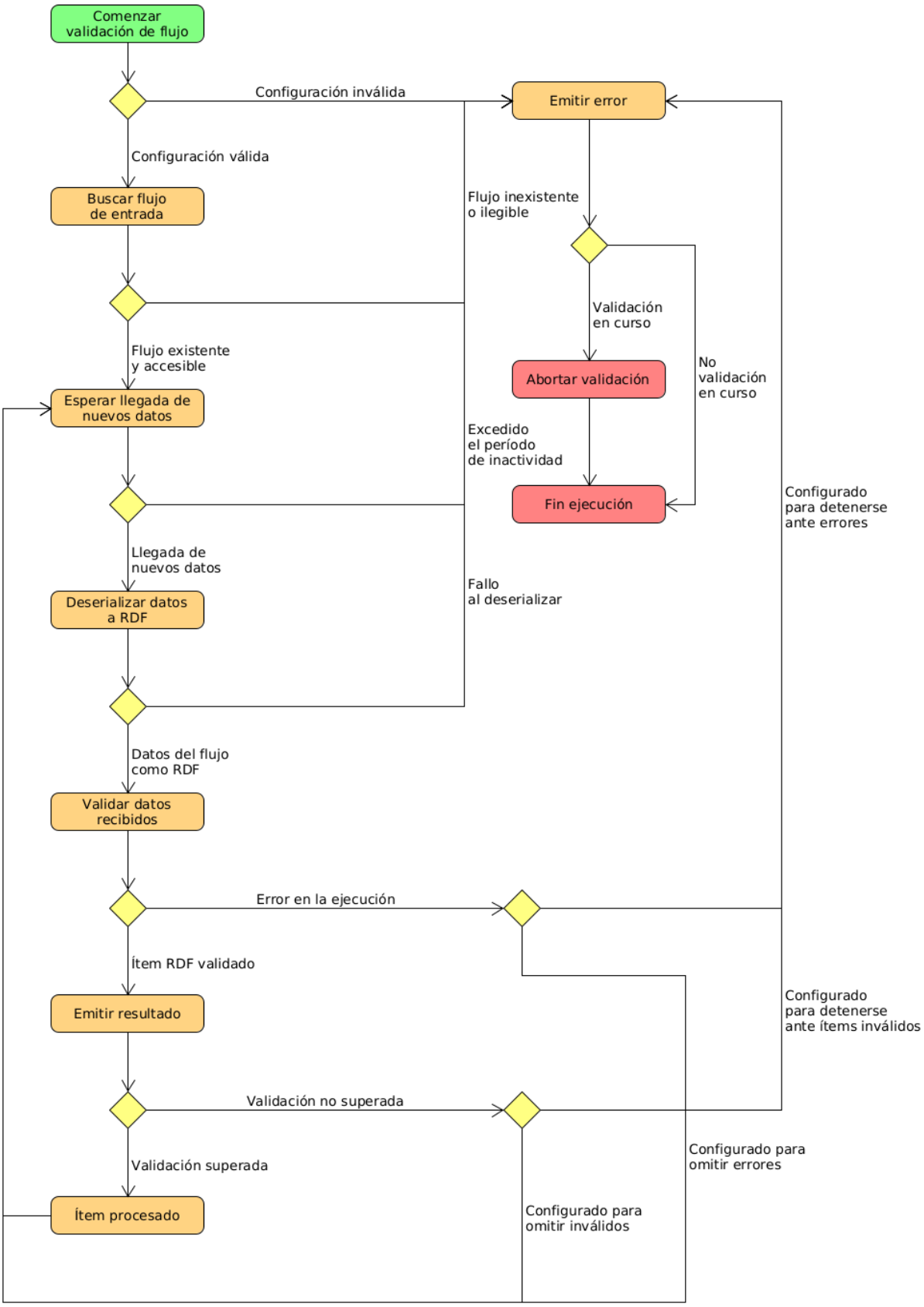


Ilustración 24. Diagrama de flujo de una validación con Comet

4.2.4.2 *Procesamiento de una nueva validación en el servidor*

El siguiente diagrama presenta los estados por los que pasa el servidor de *RDFShape* cuando es contactado para iniciar una validación de flujos.

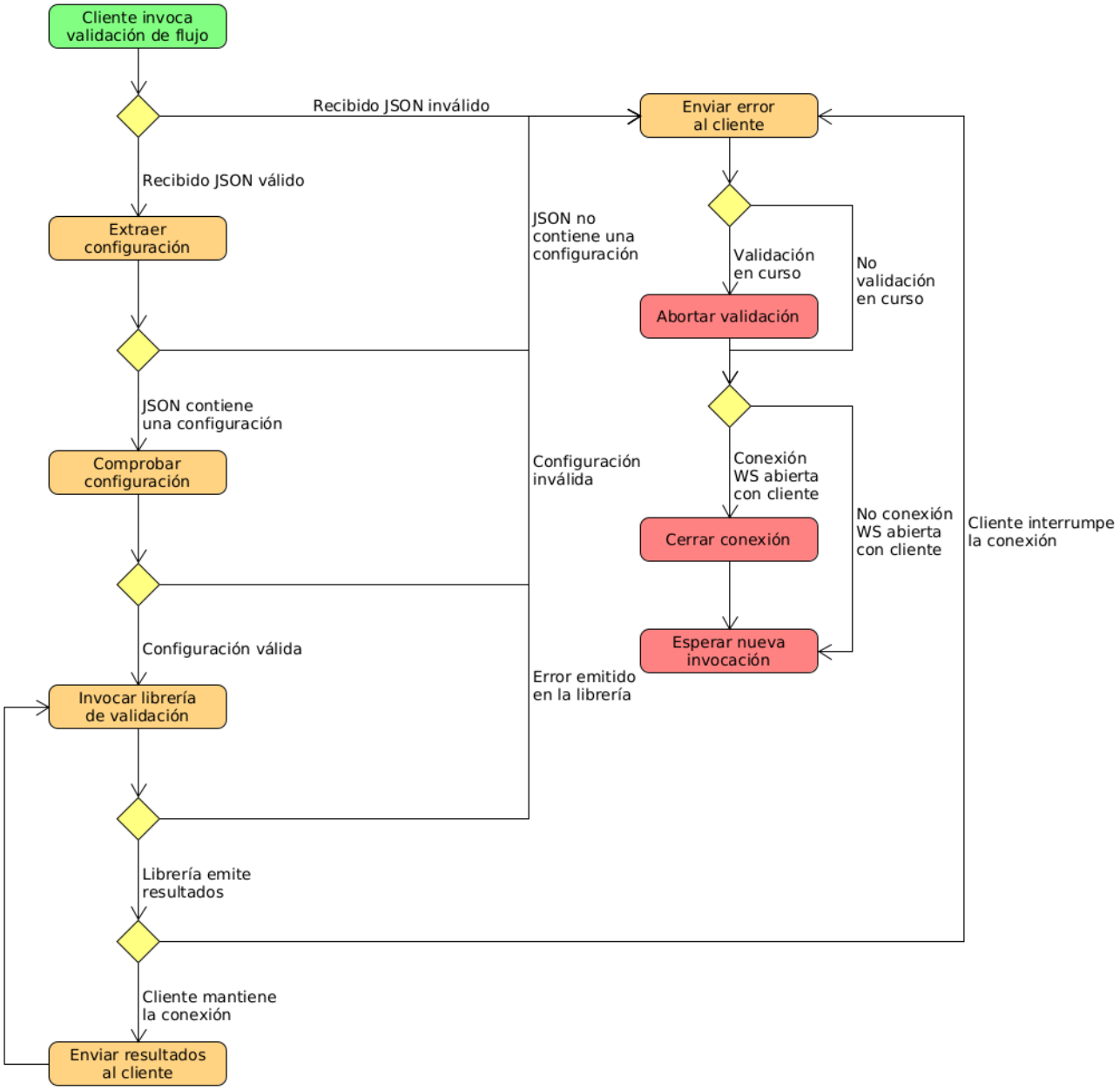


Ilustración 25. Diagrama de flujo - Procesamiento de una validación en el servidor

4.2.4.3 *Procesamiento de una nueva validación en el cliente*

Este diagrama representa los estados por los que pasa el cliente de *RDFShape* cuando es empleado para invocar una validación de flujos en el API, atendiendo a las posibilidades adicionales que brinda el cliente y su interfaz gráfica.

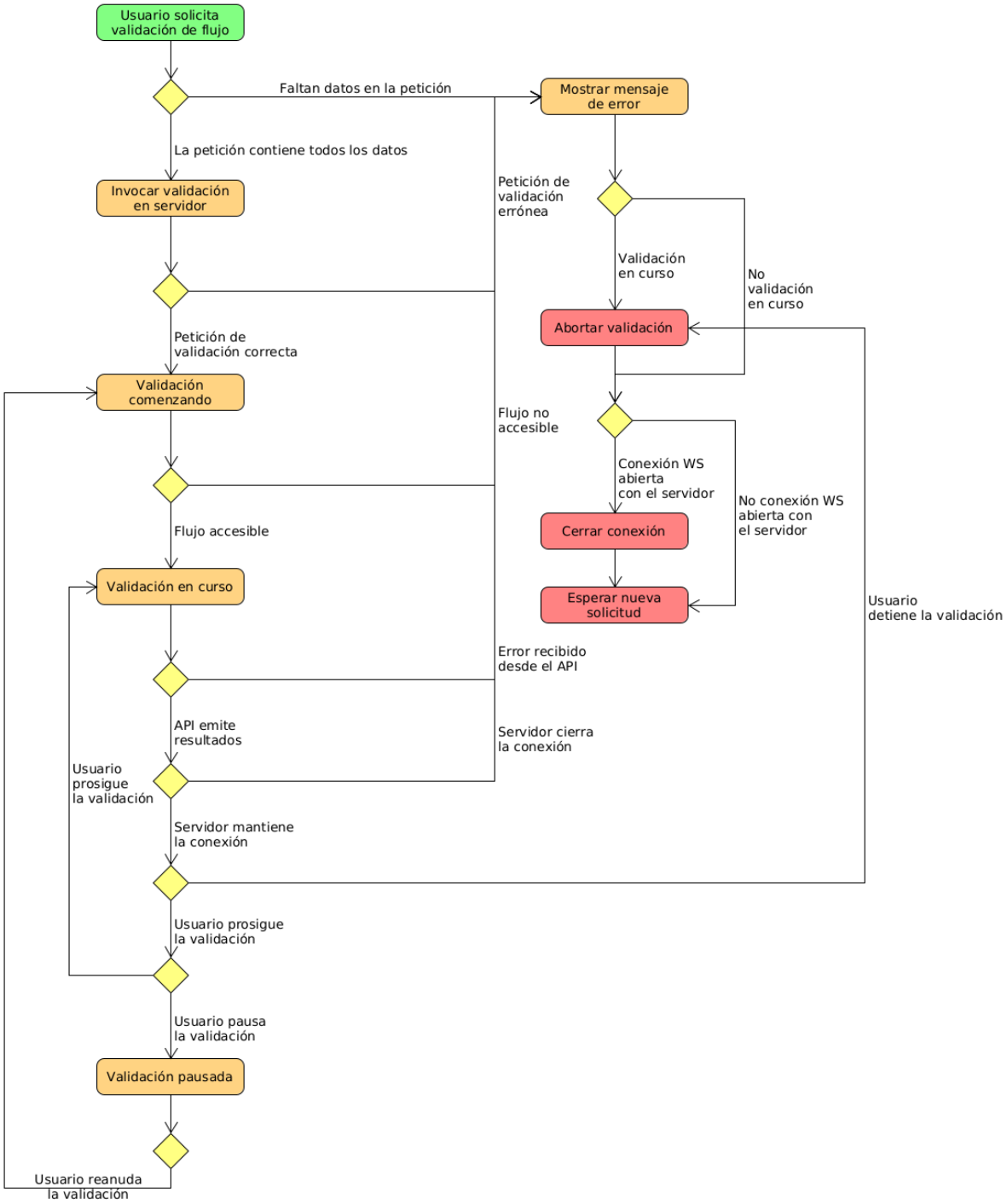


Ilustración 26. Diagrama de flujo - Procesamiento de una validación en el cliente

4.3 PLANIFICACIÓN Y GESTIÓN DEL TFM

4.3.1 Planificación del Proyecto

4.3.1.1 *Identificación de Interesados*

En esta sección, se listan los *stakeholders* o interesados identificados para el proyecto:

- **Desarrolladores del proyecto**
  - Eduardo Ulibarri Toledo: persona encargada de llevar a cabo la planificación, construcción y mantenimiento del sistema.
- **Clientes del proyecto**
  - José Emilio Labra Gayo: tutor encargado de supervisar, apoyar y aceptar los resultados del proyecto.
  - Grupo de investigación Web Semántica Oviedo (WESO): principales interesados en la viabilidad, futuros usos y publicidad generada por el desarrollo del proyecto.
  - Universidad de Oviedo / Escuela de Ingeniería Informática: interesados por parte del tutor y de WESO, así como en el uso de la herramienta en la formación de alumnos en algunas asignaturas del grado en ingeniería informática y el Máster en Ingeniería Web (MIW).
- **Usuarios finales de la aplicación**
  - Otros grupos de investigación especializados en tecnologías de web semántica.
  - Particulares y empresas interesados en tecnologías de web semántica.
  - Estudiantes de la escuela de Ingeniería Informática de Oviedo.

4.3.1.2 *OBS y PBS*

4.3.1.2.1 OBS

4.3.1.2.1.1 Roles del proyecto

Para comprender la estructura de organización del proyecto cabe destacar que, a efectos prácticos, sólo una persona va a trabajar en la realización del proyecto de fin de máster: el alumno.

Sin embargo, a la hora de elaborar la planificación y el presupuesto, el papel del alumno se ha desglosado en varios roles. La finalidad de este cambio es construir una planificación y un presupuesto más verosímiles respecto a los de un proyecto real.

Los recursos humanos contemplados en la planificación y presupuesto interno del proyecto son los siguientes:

Nombre del recurso	Tasa estándar (€ / hora)
Jefe de proyecto	65,00 €/h
Arquitecto de software	60,00 €/h
Ingeniero de sistemas	56,00 €/h
Consultor de tecnología	52,00 €/h
Analista	45,00 €/h
Programador	38,00 €/h
Tester	38,00 €/h
Diseñador gráfico	34,00 €/h
Auxiliar de administración	20,00 €/h

Tabla 28. Roles del proyecto

Los recursos humanos expuestos anteriormente pueden consultarse en detalle, junto con los recursos materiales y de software, en el fichero de *Microsoft Project* que se adjunta a la presente documentación.

4.3.1.2.1.2 Responsabilidades del proyecto

Cada miembro del personal del proyecto se responsabiliza de distintas tareas en función de su especialización. A continuación, se muestran, de forma resumida, las tareas del proyecto y su(s) interviniente(s) principal(es).

Tarea	Responsables principales
Reunión inicial con el cliente	Jefe de proyecto
Preparación del entorno físico de trabajo	Auxiliar de administración
Preparación del entorno informático de trabajo	Programador
Estudios en web semántica y datos RDF	Consultor de tecnología, Analista
Estudios en tecnologías de validación de datos RDF	Consultor de tecnología, Analista
Estudios en software disponible para el envío y recepción de flujos RDF	Consultor de tecnología, Analista
Estudio inicial de tecnologías para la comunicación entre clientes y servidor con posibilidad de tiempo real	Consultor de tecnología, Analista
Formación en las tecnologías presentes en el validador actual del cliente	Consultor de tecnología, Programador
Seguimiento del proyecto con el cliente (+ reuniones periódicas)	Jefe de proyecto
Definir una plantilla para la memoria del proyecto	Jefe de proyecto, Auxiliar de administración

Planificación del sistema	Jefe de proyecto, Analista
Definición de la arquitectura tecnológica del sistema	Arquitecto de software
Estudio de viabilidad del sistema	Consultor de tecnología, Arquitecto de software
Identificación de actores y casos de uso	Analista
Establecimiento de los requisitos del sistema	Arquitecto de software, Analista
Planificación y gestión del proyecto	Jefe de proyecto
Identificación de los subsistemas	Arquitecto de software, Analista
Análisis preliminar de clases	Arquitecto de software, Analista
Definición inicial de interfaces de usuario	Programador, Diseñador gráfico
Especificación inicial del plan de pruebas	Tester
Diseño de casos de uso finales	Analista
Diseño de clases	Arquitecto de software, Analista
Diseño de la arquitectura de módulos del sistema	Arquitecto de software, Programador
Diseño físico de los datos	Analista
Diseño del plan de pruebas	Tester, Programador
Definición del entorno de generación y construcción del sistema	Analista
Módulo 1: Biblioteca de validación de flujos RDF (Comet)	Programador
Módulo 2: Servidor web API REST (RDFShape-API)	Programador, Ingeniero de sistemas
Módulo 3: Cliente web (RDFShape-Client)	Programador, Ingeniero de sistemas
Stream de datos RDF pseudoaleatorios para alimentar el sistema de validación	Programador
Ejecución del plan de pruebas y corrección de errores	Tester, Programador
Elaboración de los manuales de usuario	Programador, Ingeniero de sistemas
Implantación del sistema desarrollado en los servidores del cliente	Ingeniero de sistemas
Preparación del mantenimiento del sistema	Ingeniero de sistemas
Revisar y completar la memoria del proyecto	Jefe de proyecto, Auxiliar de administración
Visto bueno del cliente	Jefe de proyecto
Defensa del proyecto	Jefe de proyecto

Tabla 29. OBS del proyecto (resumen)

Todas las tareas del proyecto desglosadas al completo junto con sus responsables se encuentran en el fichero de *Microsoft Project* adjunto.

4.3.1.2.2 PBS

Los siguientes esquemas muestran resumidamente los productos obtenidos en la realización de las tareas del proyecto. EL PBS se encuentra subdividido para facilitar su lectura.

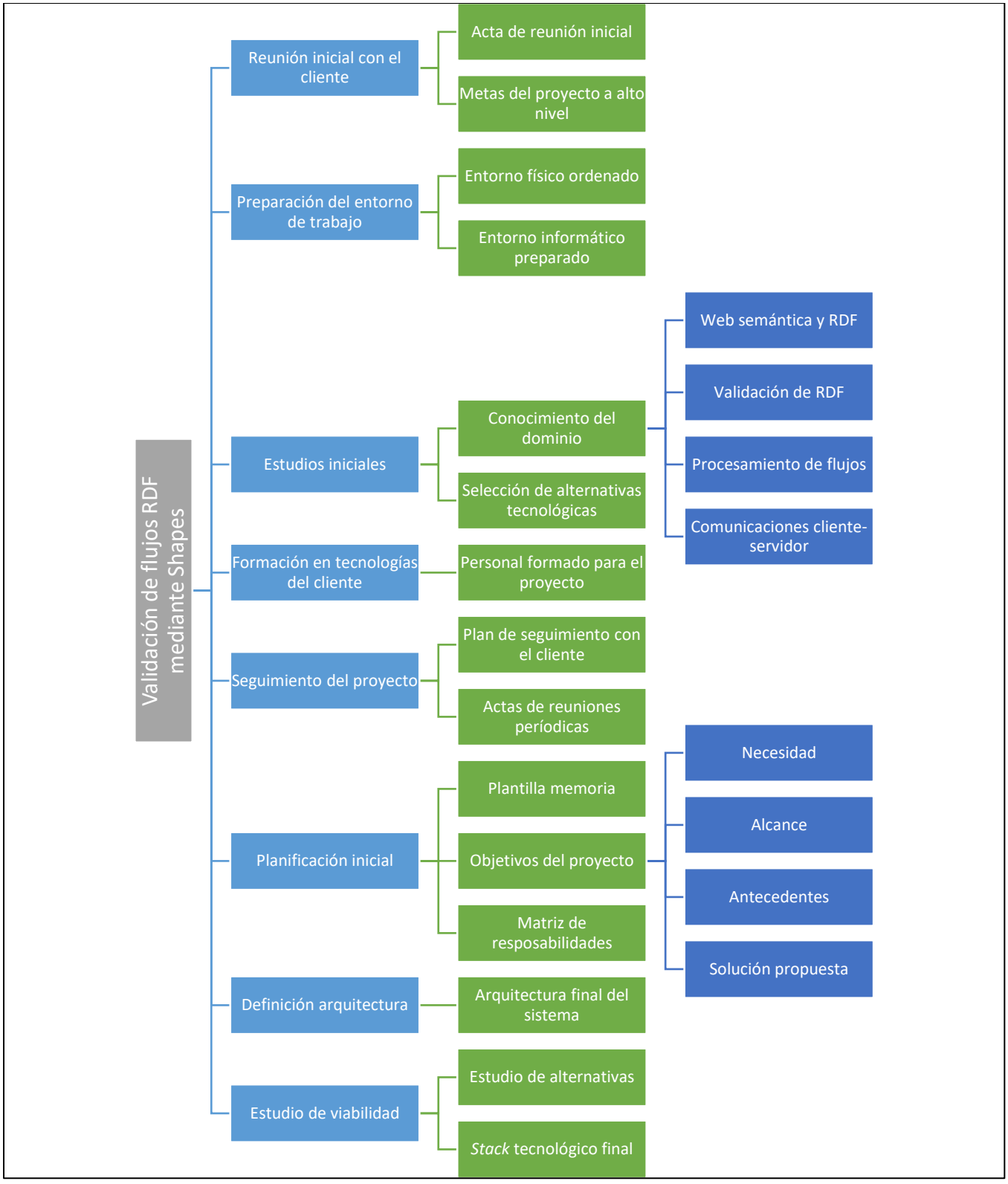


Ilustración 27. PBS del proyecto - Inicio

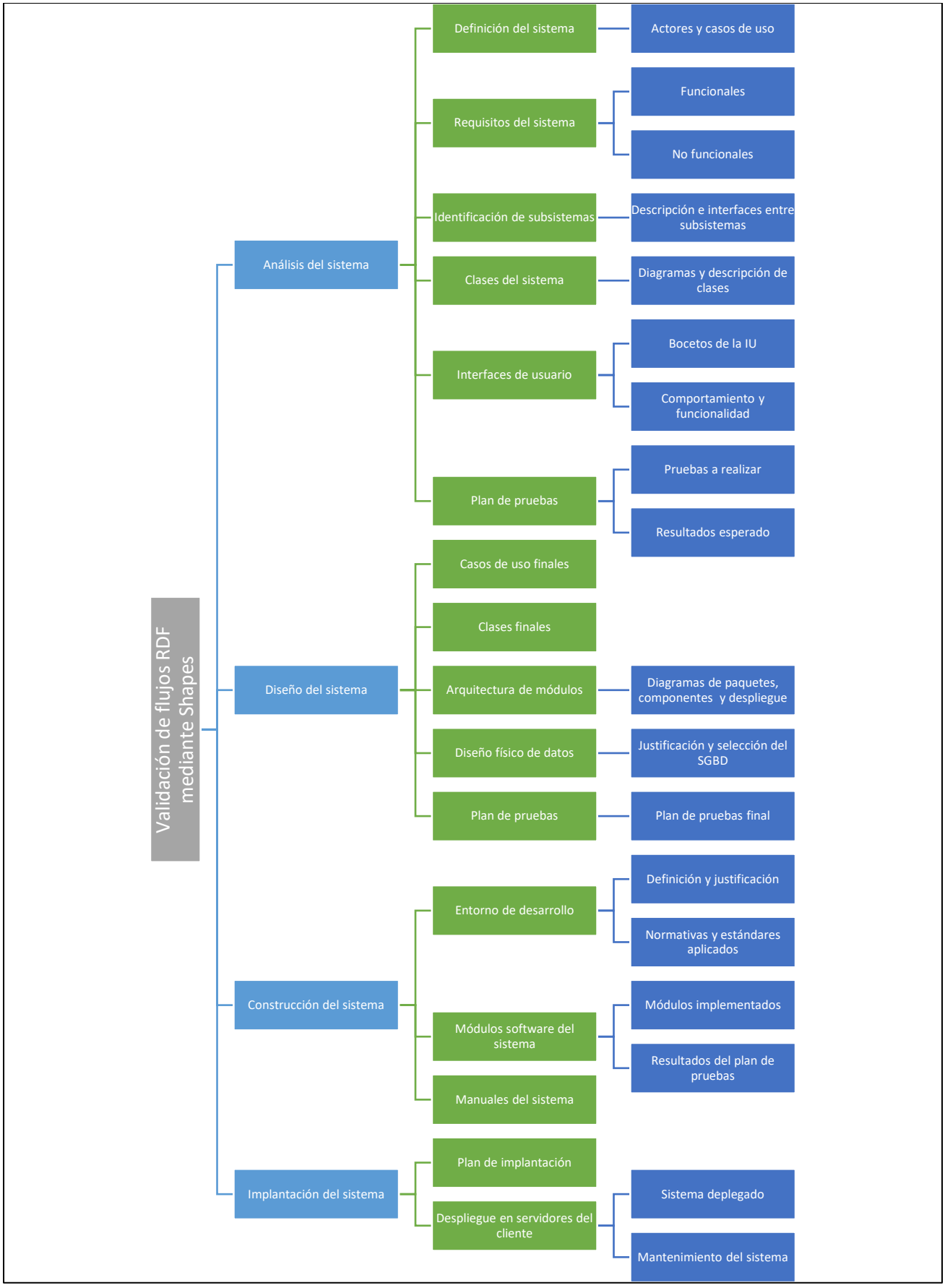


Ilustración 28. PBS del proyecto - Ejecución

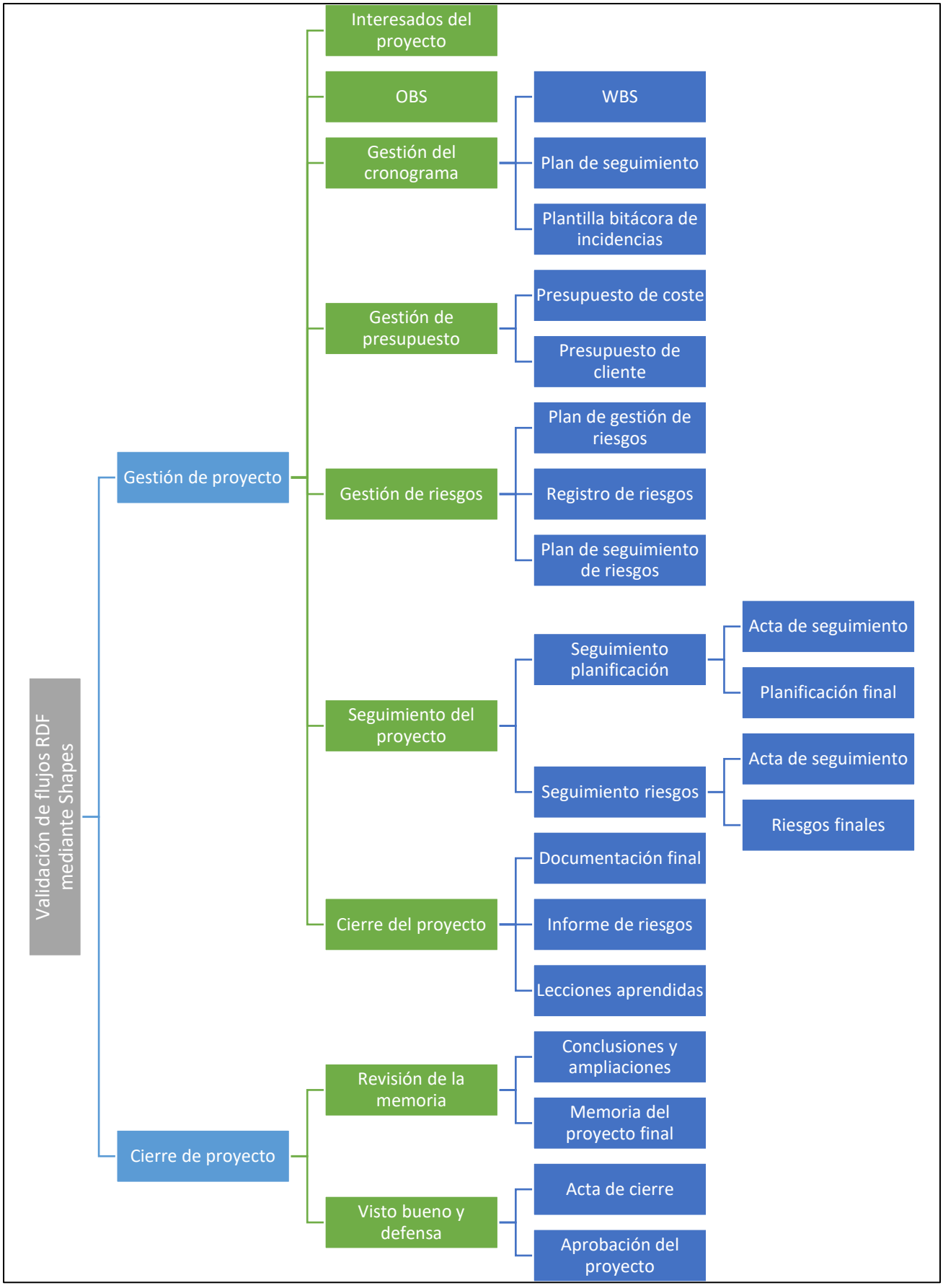


Ilustración 29. PBS del proyecto - Gestión y cierre

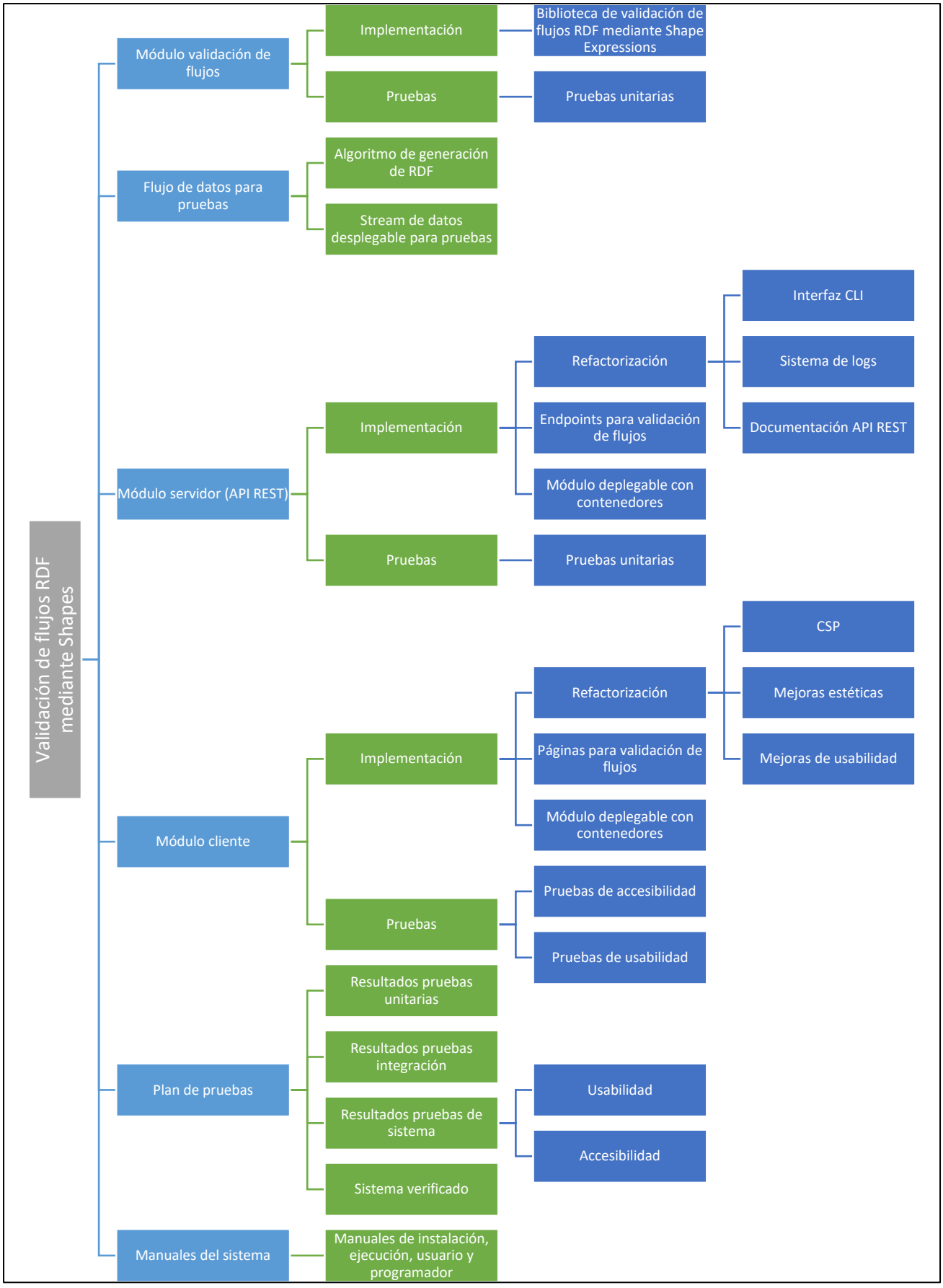


Ilustración 30. PBS del proyecto – Desarrollo software

4.3.1.3 *Planificación Inicial. WBS*

En esta sección de describe y visualiza la planificación inicial elaborada para el proyecto.

4.3.1.3.1 Aspectos generales

La intención de la planificación del TFM es cubrir todos los procesos involucrados en la realización del proyecto, desde su adjudicación al alumno hasta su defensa en la convocatoria.

Al tratarse de una planificación inicial, es de esperar que a lo largo del desarrollo del proyecto se maticen la planificación y presupuestos aquí presentados, empleando las reservas de tiempo de las tareas de revisión y seguimiento.

4.3.1.3.2 Calendario del alumno

El proyecto se inicia el día **6 de septiembre de 2021** con la reunión inicial con el grupo de investigación WESO, representado por su investigador principal y tutor del proyecto. En la concepción de la planificación, se plantea la ejecución del proyecto a lo largo del curso académico 2021-2022. La fecha estimada para el cierre del proyecto con la aceptación del cliente es el **11 de mayo de 2022**, si bien se reserva una última tarea para la defensa.

El calendario de proyecto consta de tres fases o ventanas temporales diferenciadas que se adaptan al contexto del alumno a lo largo del curso, que son:

1. **01/09/2021 – 24/12/2021**. Cubre las etapas iniciales y la mayoría del análisis del proyecto. La jornada laboral planificada es la siguiente:
    a. Jornada de **9:00 a 14:00**.
    b. Se contempla una pausa entre los días 24 de diciembre de 2021 y 9 de enero de 2022 (ambos inclusive), dedicada a asuntos personales y las festividades navideñas.
2. **10/01/2022 – 08/04/2022**. Cubre las etapas de diseño y la construcción de los módulos software. La jornada laboral planificada es la siguiente:
    a. Jornada de **9:00 a 15:00**.
    b. Se contempla una pausa entre los días 9 de abril de 2022 y 21 de abril de 2022 (ambos inclusive), dedicada a asuntos personales y las festividades de semana santa.
3. **22/04/2022 – 31/07/2022**. Cubre las etapas de implantación del sistema y cierre de la memoria hasta la defensa del proyecto. La jornada laboral planificada es la siguiente:
    a. Jornada de **9:00 a 14:00**.

Así mismo, se tuvieron en cuenta las siguientes observaciones a la hora de elaborar las planificaciones del proyecto:

- Hay varios roles que participan en el proyecto, pero todos son encarnados por el alumno (el tutor es un agente externo). Por tanto, es consecuencia lógica que todas las tareas sean secuenciales.
    - Se salen de la ejecución lineal del proyecto las tareas de seguimiento, restringidas a fechas periódicas a lo largo del transcurso del proyecto.

- La jornada laboral establecida va de lunes a viernes, dejando los fines de semana para el descanso. Quedan excluidos del descanso los festivos nacionales u otras festividades fuera de las mencionadas en las ventanas temporales.
- No se contemplan viajes dada la cercanía física del alumno y el tutor, así como la posibilidad de teleconferencias por comodidad y por la situación sanitaria.

Las ventanas temporales descritas anteriormente pueden revisarse en el calendario de trabajo empleado en el documento de *Microsoft Project* adjunto al proyecto.

4.3.1.3.3 Diagrama de Gantt

A continuación, se muestra, mediante capturas de pantalla, el diagrama de Gantt para la planificación inicial del proyecto.

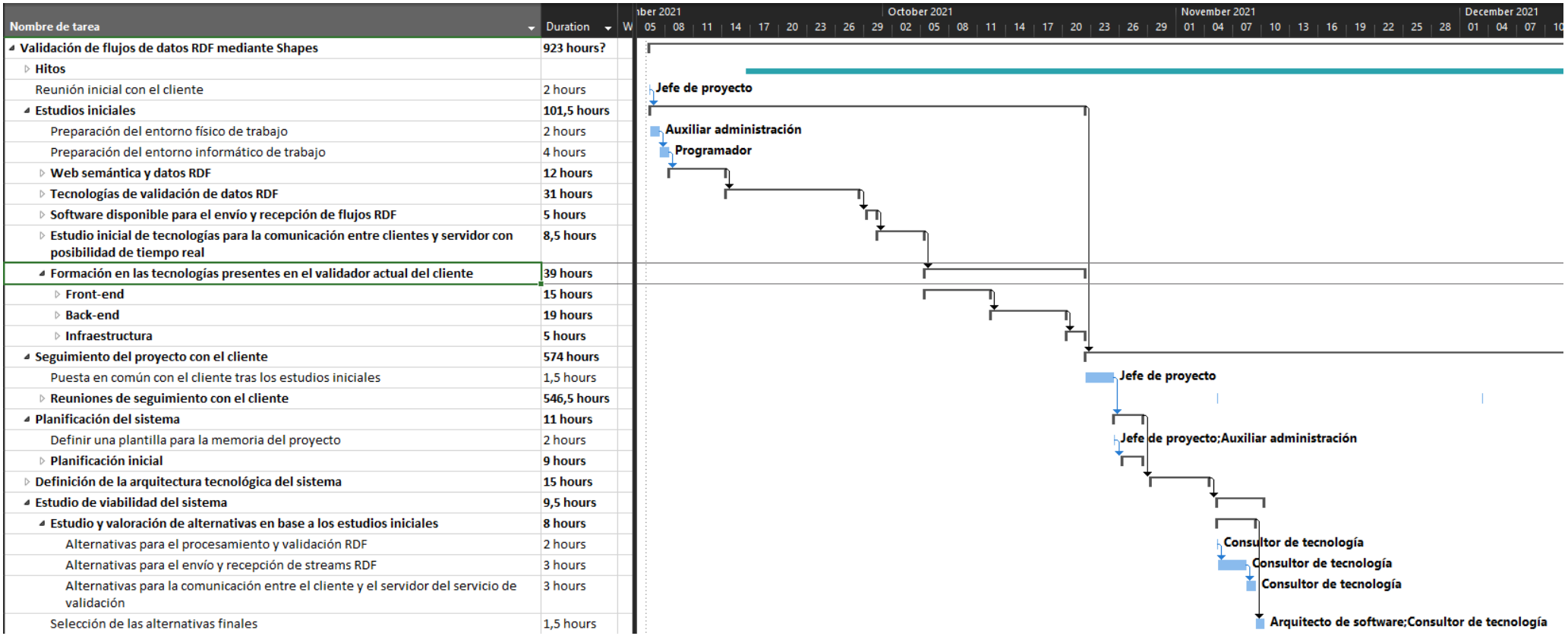


Ilustración 31. Diagrama de Gantt - Planificación Inicial I

Nombre de tarea	Duration	W
Análisis del sistema	**669,74 hours**	
Definición final del sistema	2 hours	
Establecimiento de los requisitos del sistema	**16 hours**	
Requisitos funcionales	8 hours	
Requisitos no funcionales	8 hours	
Identificación de actores del sistema	5 hours	
Definición de casos de uso iniciales	6 hours	
Planificación y gestión del proyecto	**669,74 hours**	
Inicio del proyecto	**218 hours**	
Identificación de interesados	2 hours	
Definición del PBS y OBS	4 hours	
Gestión del cronograma inicial	**19 hours**	
Planificación temporal inicial, WBS	16 hours	
Definición del plan de seguimiento de la planificación	2 hours	
Definición de la estructura de la bitácora de incidencias del proyecto	1 hour	
Gestión de riesgos inicial	**193 hours**	
Definición del plan de gestión de riesgos	6 hours	
Identificación y registro de riesgos	17 hours	
Definición del plan de seguimiento de riesgos	2 hours	
Presupuesto inicial	**15 hours**	
Presupuesto de coste (interno)	10 hours	
Presupuesto de cliente	4 hours	
Ejecución del proyecto	**303 hours**	
Cierre del proyecto	**13 hours**	
Identificación de los subsistemas a analizar	**17 hours**	
Enumeración y descripción de los subsistemas	3 hours	
Descripción de las interfaces entre subsistemas	3 hours	
Análisis preliminar de clases del sistema	**20,5 hours**	
Diagramas de clases preliminar	10 hours	
Descripción de las clases identificadas	5 hours	
Definición inicial de interfaces de usuario	**20 hours**	
Especificación inicial del plan de pruebas	**11 hours**	

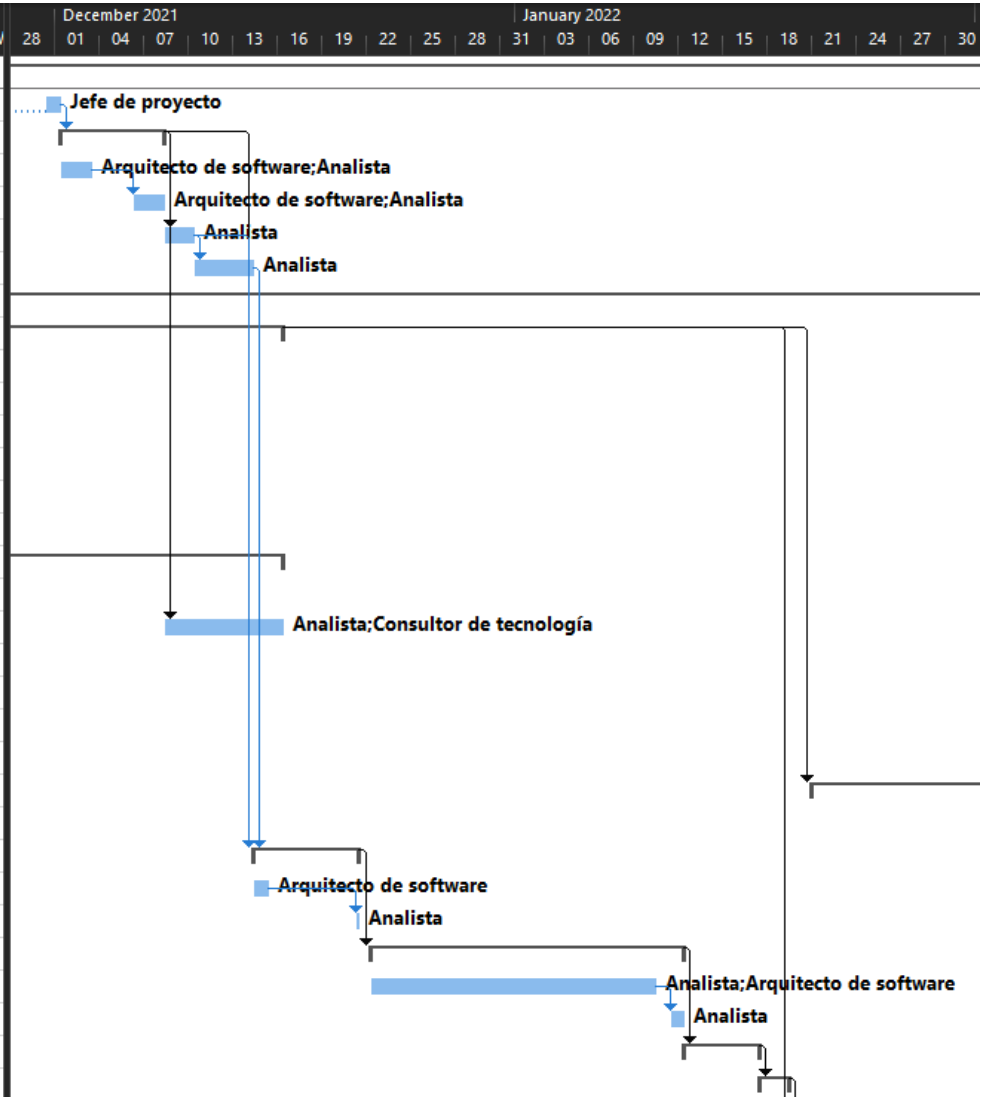


Ilustración 32. Diagrama de Gantt - Planificación Inicial II

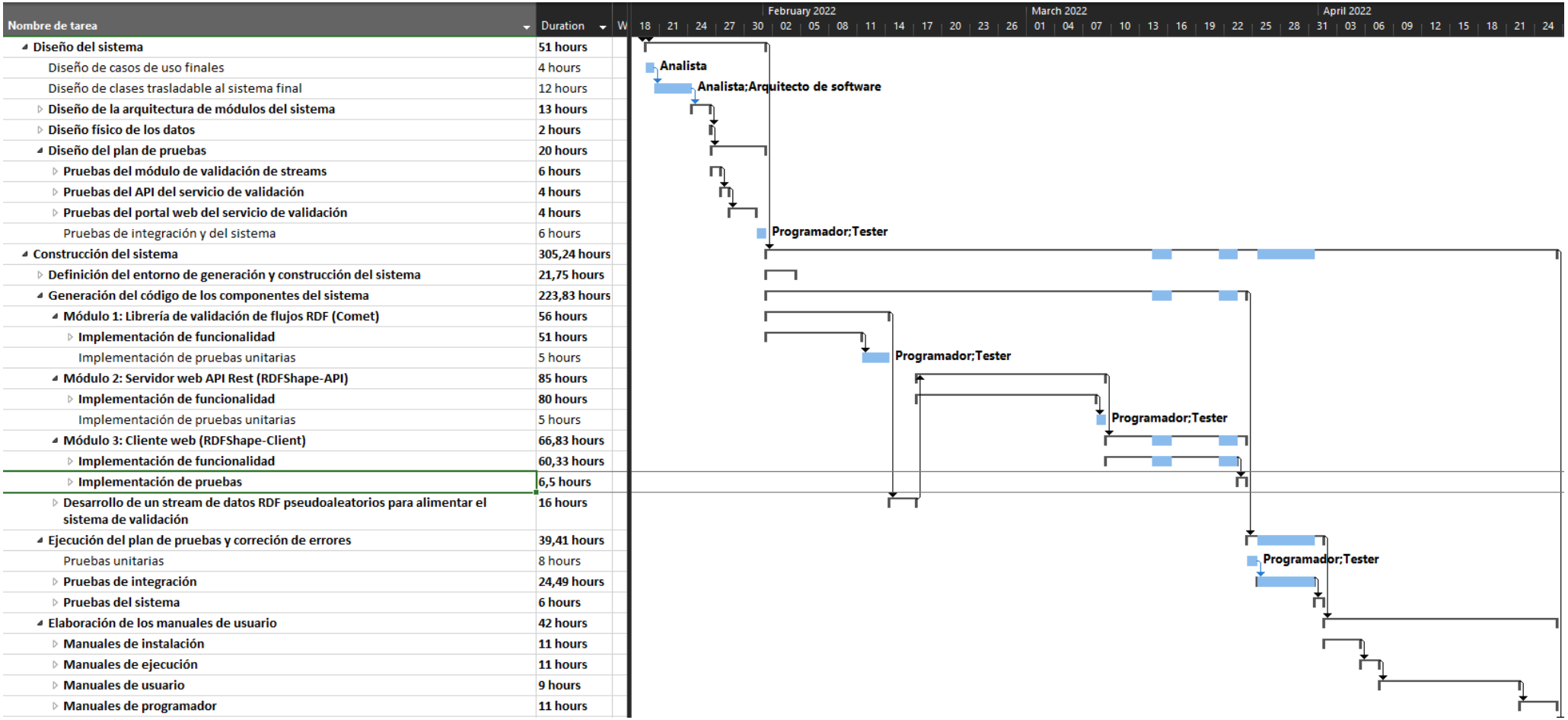


Ilustración 33. Diagrama de Gantt - Planificación Inicial III

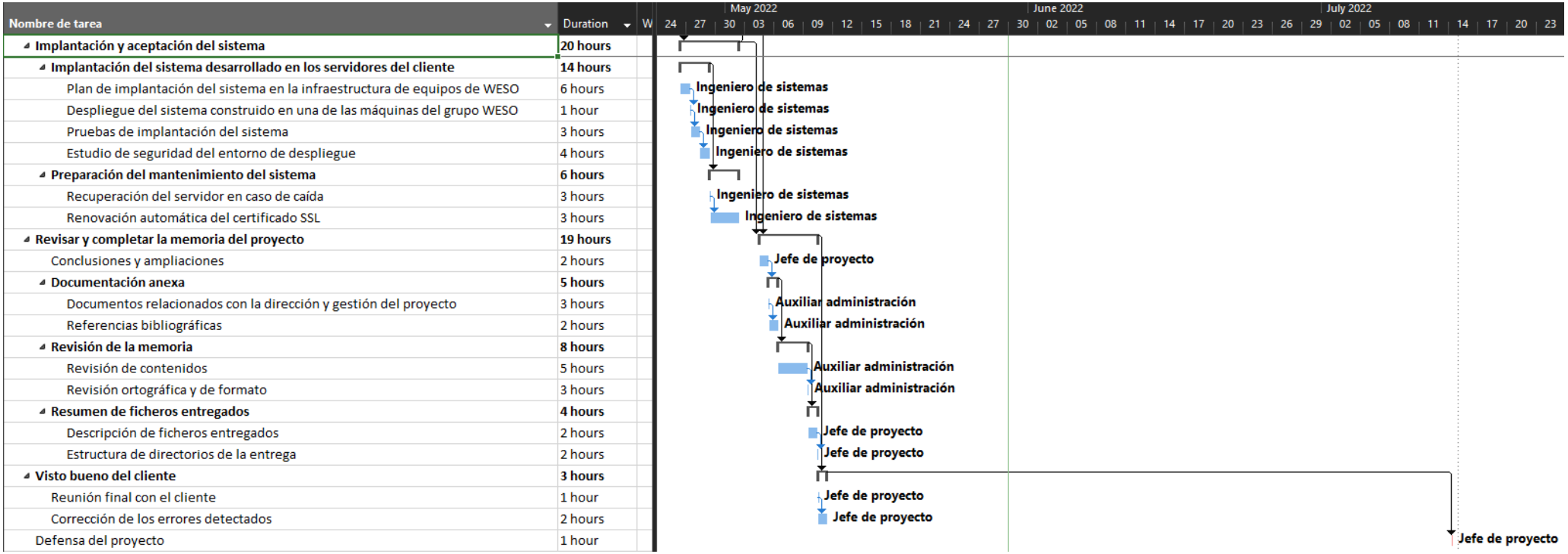


Ilustración 34. Diagrama de Gantt - Planificación Inicial IV

4.3.1.3.4 Cuadro resumen

El siguiente cuadro muestra, en forma de tabla, las tareas principales del proyecto y las horas de trabajo dedicadas a las mismas. Las tareas aquí mostradas son un resumen de las totales y se corresponden con las vistas en *4.3.1.2.1.2-Responsabilidades del proyecto*.

ID Tarea	Nombre	Fecha inicio	Fecha fin	Horas de trabajo
15	Reunión inicial con el cliente	06/09/21	06/09/21	2 horas
17	Preparación del entorno físico de trabajo	06/09/21	07/09/21	2 horas
18	Preparación del entorno informático de trabajo	07/09/21	08/09/21	4 horas
19	Estudios en web semántica y datos RDF	08/09/21	14/09/21	17 horas
24	Estudios en tecnologías de validación de datos RDF	14/09/21	28/09/21	43 horas
35	Estudios en software disponible para el envío y recepción de flujos RDF	29/09/21	30/09/21	7 horas
40	Estudio inicial de tecnologías para la comunicación entre clientes y servidor con posibilidad de tiempo real	30/09/21	05/10/21	15 horas
47	Formación en las tecnologías presentes en el validador actual del cliente	05/10/21	22/10/21	44,5 horas
60	Seguimiento del proyecto con el cliente (+ reuniones periódicas)	22/10/21	06/05/22	21 horas
77	Definir una plantilla para la memoria del proyecto	25/10/21	25/10/21	3 horas
78	Planificación del sistema	26/10/21	28/10/21	15 horas
84	Definición de la arquitectura tecnológica del sistema	29/10/21	04/11/21	15 horas
87	Estudio de viabilidad del sistema	05/11/21	10/11/21	11 horas

98	Identificación de actores y casos de uso	08/12/21	14/12/21	11 horas
95	Establecimiento de los requisitos del sistema	01/12/21	08/12/21	24 horas
100	Planificación y gestión del proyecto	06/09/21	04/05/22	100 horas
137	Identificación de los subsistemas	14/12/21	21/12/21	6 horas
140	Análisis preliminar de clases	22/12/21	12/01/22	15 horas
143	Definición inicial de interfaces de usuario	12/01/22	17/01/22	22 horas
149	Especificación inicial del plan de pruebas	17/01/22	19/01/22	11 horas
153	Diseño de casos de uso finales	19/01/22	20/01/22	4 horas
154	Diseño de clases	20/01/22	24/01/22	16 horas
155	Diseño de la arquitectura de módulos del sistema	24/01/22	26/01/22	17 horas
161	Diseño físico de los datos	26/01/22	26/01/22	2 horas
163	Diseño del plan de pruebas	26/01/22	01/02/22	40 horas
174	Definición del entorno de generación y construcción del sistema	01/02/22	04/02/22	7 horas
179	Módulo 1: Biblioteca de validación de flujos RDF (Comet)	01/02/22	14/02/22	61 horas
187	Módulo 2: Servidor web API REST (RDFShape-API)	17/02/22	09/03/22	92 horas
204	Módulo 3: Cliente web (RDFShape-Client)	09/03/22	24/03/22	67,5 horas
223	Stream de datos RDF pseudoaleatorios para alimentar el sistema de validación	14/02/22	17/02/22	16 horas

226	Ejecución del plan de pruebas y corrección de errores	24/03/22	01/04/22	46 horas
235	Elaboración de los manuales de usuario	01/04/22	26/04/22	66 horas
253	Implantación del sistema desarrollado en los servidores del cliente	26/04/22	29/04/22	14 horas
258	Preparación del mantenimiento del sistema	29/04/22	02/05/22	6 horas
261	Revisar y completar la memoria del proyecto	04/05/22	10/05/22	19 horas
272	Visto bueno del cliente	10/05/22	11/05/22	3 horas
275	Defensa del proyecto	14/07/22	14/07/22	1 hora

Tabla 30. Cuadro resumen de la planificación inicial

4.3.1.4 *Gestión de Riesgos*

4.3.1.4.1 Plan de Gestión de Riesgos

El plan de gestión de riesgos se encuentra detallado en el anexo *A1-Plan de gestión de riesgos*.

4.3.1.4.2 Identificación de Riesgos

Identificador	Nombre
1	Inexperiencia del programador en las tecnologías empleadas por el cliente en el módulo API
2	Errores o "bugs" en las bibliotecas de código de terceros empleadas en el desarrollo del módulo de validación
3	Documentación insuficiente de las bibliotecas de código empleadas para manipular y validar RDF mediante Shape Expressions
4	Cambios en los requisitos iniciales (se incluyen nuevos mecanismos de validación y formatos de entrada o salida adicionales)
5	Mejoras en las herramientas de validación de RDF mediante Shape Expressions
6	El módulo validador de streams se ha desarrollado introduciendo una alta complejidad, lo que repercute en un uso elevado de recursos.
7	Verificar el funcionamiento del módulo validador de streams supone una complejidad añadida
8	Se producen demoras en el desarrollo al tener que mantener el código e interfaz del módulo Cliente coherentes con el módulo API
9	Problemas de seguridad relevantes en las bibliotecas de código de terceros empleadas en el sistema
10	Cambios en la infraestructura de servidores del cliente
11	Demoras en el despliegue del sistema por la complejidad de integrar 3 módulos
12	Las interfaces gráficas y de usuario del módulo Cliente no cumple con el nivel de accesibilidad AA del WCAG
13	El cliente web presenta problemas de rendimiento o renderizado en alguno de los navegadores más populares
14	Retrasos en la planificación del proyecto por causas personales y/o sanitarias
15	El SGBD adoptado para cumplir los requisitos del cliente no es lo suficientemente flexible para adaptarse a cambios futuros

Tabla 31. Listado de riesgos

4.3.1.4.3 Registro de Riesgos

En la tabla a continuación, se incluye el registro detallado de los riesgos identificados previamente, con el análisis cuantitativo de los mimos.

ID	Nombre	Responsable	Probabilidad	Impacto				Impacto	0,40	Respuesta	Notas
				Presupuesto	Planificación	Alcance	Calidad		Priorización		
1	Inexperiencia del programador en las tecnologías empleadas por el cliente en el módulo API	Consultor de tecnología	Muy Alta	Medio	Alto	Alto	Alto	0,50		Mitigar. Dentro del equipo de proyecto se contará con un consultor de tecnologías con experiencia previa en el campo que definirá, antes de iniciarse el desarrollo y con el beneplácito del jefe de proyecto, un plan de formación para el programador del sistema.	En el servidor que monta el API para la validación de RDF, el cliente hace uso de tecnologías muy potentes, pero poco populares en el ámbito general, tales como el lenguaje de programación Scala y la metodología de programación funcional con las herramientas del ecosistema Typelevel. La complejidad y el desconocimiento de las bases de estas tecnologías puede frustrar y demorar las tareas del programador.
2	Errores o "bugs" en las bibliotecas de código de terceros empleadas en el desarrollo del módulo de validación	Consultor de tecnología	Media	Medio	Alto	Muy alto	Muy alto	0,45		Mitigar. Se contará con un consultor de tecnologías con experiencia previa en el campo que decidirá, antes de iniciarse el desarrollo, qué bibliotecas de código se utilizarán atendiendo a los antecedentes disponibles. En dichas decisiones se priorizará el software open-source y se exigirá el beneplácito del analista y el arquitecto del proyecto.	El módulo de validación de streams se desarrolla desde cero e implementa dependencias de código de terceros. Es posible que este código, desarrollado de forma externa al proyecto, tenga limitaciones o fallos que lastren al mismo.

3	Documentación insuficiente de las bibliotecas de código empleadas para manipular y validar RDF mediante Shape Expressions	Consultor de tecnología	Alta	Medio	Alto	Bajo	Medio	0,39		Mitigar. Se contará con un consultor de tecnologías con experiencia previa en el campo que decidirá, antes de iniciarse el desarrollo, qué bibliotecas de código se utilizarán atendiendo a los antecedentes disponibles. En dichas decisiones se priorizará el software open-source y se exigirá el beneplácito del analista y el arquitecto del proyecto.	Debido a que nos encontramos en terreno de investigación, el estándar RDF goza de una acogida limitada y las bibliotecas disponibles pueden estar poco documentadas, incrementando la curva de aprendizaje del equipo programador.
4	Cambios en los requisitos iniciales (se incluyen nuevos mecanismos de validación y formatos de entrada o salida adicionales)	Jefe de proyecto	Muy Baja	Alto	Alto	Muy alto	Alto	0,09		Mitigar. Dado el corto período de realización del proyecto y la baja probabilidad del riesgo, cabría la posibilidad de aceptar el riesgo. Sin embargo, de cara a la calidad del presente y futuros proyectos, se construirá el software supervisado por un arquitecto y de forma modular tal que sus partes sean fáciles de modificar según proceda. Así mismo, se destinarán reservas de tiempo del jefe de proyecto y del arquitecto para reuniones con el cliente y captura de requisitos.	Puede ocurrir que, durante la realización del proyecto, los avances en investigación o en la competencia motiven al cliente a actualizar los requisitos del producto, aumentando el alcance.
5	Mejoras en las herramientas de validación de RDF mediante Shape Expressions	Arquitecto de software	Muy Baja	Bajo	Medio	Alto	Muy alto	0,09		Aceptar. Dado el relativamente corto período de realización del proyecto y la baja probabilidad de la oportunidad, se asume que no va a haber cambios en las tecnologías implicadas que justifiquen modificaciones en la arquitectura del proyecto. La incorporación de nuevos avances al proyecto se contemplará en	Oportunidad. Puede ocurrir que, durante la realización del proyecto y al movernos en un entorno dinámico, los avances en investigación abran la posibilidad de mejorar el rendimiento del validador RDF y/o acelerar la consecución de los objetivos.

										los planes de mantenimiento, en función de la escala de los cambios.	
6	El módulo validador de streams se ha desarrollado introduciendo una alta complejidad, lo que repercute en un uso elevado de recursos.	Arquitecto de software	Baja	Bajo	Bajo	Alto	Muy alto	0,27		Evitar. Se contará dentro del equipo de proyecto con perfiles suficientemente preparados y se seguirá una metodología de desarrollo que garantice la calidad del producto del producto desde su concepción. Además, junto al equipo de desarrollo trabajará un tester encargado de diseñar y dirigir las pruebas contra el sistema.	El módulo de validación de streams RDF funciona correctamente, pero al ser una implementación pionera basada en tecnologías poco exploradas, tiene una complejidad que impide su uso práctico.
7	Verificar el funcionamiento del módulo validador de streams supone una complejidad añadida	Arquitecto de software	Alta	Medio	Muy alto	Muy Bajo	Bajo	0,63		Aceptar. Para comprobar el funcionamiento de este módulo es necesario hacer pruebas de integración contra un stream de datos externo. Se destinarán reservas de tiempo al desarrollo de un stream de datos manipulable de forma programática y que el tester pueda tener en cuenta en el diseño del plan de pruebas.	Para verificar el funcionamiento del módulo de validación de streams se requiere de un stream de datos de prueba, implementado en la tecnología que se escoja finalmente y moldeable para realizar pruebas. Esto implica un tiempo de desarrollo y de testing añadido.

8	Se producen demoras en el desarrollo al tener que mantener el código e interfaz del módulo Cliente coherentes con el módulo API	Jefe de proyecto	Alta	Bajo	Medio	Muy Bajo	Bajo	0,21		Evitar. El desarrollo de los módulos se realizará de forma secuencial, de manera que no se trabajará exhaustivamente en el módulo Cliente hasta que no se tenga la versión final del módulo API. Además, se plantean fase de análisis y diseño detalladas que minimicen los problemas, así como el uso de tecnología de contenedores para flexibilizar el despliegue de ambos módulos.	Dentro de que el proyecto es multimodular, los módulos Cliente y API se encuentran fuertemente cohesionados ya que los cambios en el API afectan a cómo debe ser invocada desde el Cliente.
9	Problemas de seguridad relevantes en las bibliotecas de código de terceros empleadas en el sistema	Jefe de proyecto	Baja	Medio	Medio	Alto	Muy alto	0,27		Mitigar. Además de las recomendaciones combinadas del analista, el consultor tecnológico y el arquitecto, el jefe de proyecto revisará las vulnerabilidades de las dependencias utilizadas durante las sesiones de seguimiento de riesgos. Se reevaluarán y se tomarán acciones correctivas para las dependencias que lo requieran.	El proyecto por desarrollar es multimodular e implica operar con módulos ya existentes relacionados con el desarrollo web (front-end y back-end) y construidos en JavaScript o Scala/Java, dos blancos de ataque muy comunes.
10	Cambios en la infraestructura de servidores del cliente	Jefe de proyecto	Media	Bajo	Alto	Muy Bajo	Muy Bajo	0,28		Evitar. Se destinarán reservas de tiempo del jefe de proyecto para reuniones con el cliente, donde puedan ser informados de cambios que afecten al despliegue.	Dada la situación sanitaria causada por la COVID-19, el cliente se encuentra implementando la modalidad de teletrabajo, así como varios cambios de su infraestructura de redes y servidores que pueden afectar al entorno de pruebas del proyecto.

11	Demoras en el despliegue del sistema por la complejidad de integrar 3 módulos	Ingeniero de sistemas	Alta	Alto	Muy alto	Muy Bajo	Bajo	0,63		Evitar. Se contará con un arquitecto de software y con un ingeniero de sistemas dentro del equipo de desarrollo para elaborar la arquitectura del sistema y apoyar al programador en el desarrollo, despliegue y creación de manuales. Aparte, se formará al programador en el entorno de despliegue del cliente y se dedicarán reservas de tiempo a generar imágenes de cada módulo compatibles con tecnología de contenedores como Docker, para conseguir un despliegue sencillo y repetible.	El proyecto por desarrollar es multimodular e implica la comunicación entre tres módulos que se ejecutan independientemente: Cliente, API y biblioteca de validación. Además, los streams validados por el servidor provendrán, mayoritariamente, del exterior. Facilitar la correcta comunicación entre módulos y de estos con el exterior una vez desplegados puede causar complicaciones.
12	Las interfaces gráficas y de usuario del módulo Cliente no cumple con el nivel de accesibilidad AA del WCAG	Tester	Media	Medio	Medio	Bajo	Alto	0,28		Evitar. A la hora de refactorizar y desarrollar nueva funcionalidad para el cliente se tendrán en cuenta las directrices del WCAG. Así mismo, se contará con la ayuda de un diseñador gráfico para el desarrollo de interfaces y el tester del equipo deberá reservar parte del plan de pruebas a la accesibilidad del cliente.	El módulo Cliente se opera a través de una interfaz web que debe cumplir los estándares de accesibilidad AA del WCAG.
13	El cliente web presenta problemas de rendimiento o renderizado en alguno de los navegadores más populares	Programador	Media	Bajo	Bajo	Medio	Medio	0,15		Evitar. El plan de pruebas diseñado para el módulo Cliente llevará a cabo las pruebas se llevará a cabo utilizando cada uno de los cinco navegadores de escritorio más populares del mercado, a fecha de inicio del proyecto. Preventivamente, se formará a los programadores en	El módulo Cliente debe ser totalmente funcional independientemente del navegador con el que se acceda al mismo.

										las tecnologías web usadas en este módulo y se promoverá el uso de estándares.	
15	Retrasos en la planificación del proyecto por causas personales y/o sanitarias	Jefe de proyecto	Baja	Medio	Alto	Muy Bajo	Muy Bajo	0,17		Aceptar. A la hora de planificar el proyecto, se partirá de horarios de trabajo flexibles para los trabajadores, que incluyan suficientes horas de descanso al día, posibilidad de teletrabajo en caso de necesidad y libertad en los fines de semana. Además, las estimaciones de tiempo iniciales serán holgadas en la medida de lo posible para poder compensar dichos contratiempos. Más allá de estas medidas estándar, no se pueden predecir ni prevenir estos incidentes.	Si cualquiera de los perfiles del proyecto queda temporalmente inhabilitado por motivos personales o de salud (estado de pandemia), puede retrasar la finalización de este.
16	El SGBD adoptado para cumplir los requisitos del cliente no es lo suficientemente flexible para adaptarse a cambios futuros	Consultor de tecnología	Baja	Medio	Medio	Medio	Medio	0,09		Evitar. Reservar tiempo del jefe de proyecto y del arquitecto para reuniones con el cliente y una correcta captura de requisitos. Además, permitir que los perfiles expertos como el consultor tecnológico y el arquitecto decidan el SGBD a emplear en las fases de análisis del proyecto.	El cliente ha expresado su voluntad de que el SGBD empleado para guardar operaciones de los usuarios sea flexible de cara a futuras ampliaciones al proyecto. Incumplir este requisito y lastraría la calidad del producto final de cara al cliente.

4.3.1.5 *Presupuesto Inicial*

Para realizar los presupuestos iniciales del proyecto se han tenido en cuenta las siguientes consideraciones:

- Las tarifas aplicadas al trabajo del alumno dependerán del rol que esté realizando dentro de los especificados para el proyecto. Los roles y tarifas pueden encontrarse en el OBS, en *4.3.1.2.1.1-Roles del proyecto*.
- Las tareas presupuestadas, así como su duración y personal asignado, son las especificadas en la planificación y WBS iniciales del proyecto, documentados en *4.3.1.3-Planificación Inicial. WBS*.
- A fin de considerar la amortización de productos hardware y licencias software vitalicias, se ha estimado un período de vida útil de:
    - 4 años para equipos informáticos.
    - 2 años para periféricos.
    - 2 años para software informático.
- No se contempla la adquisición de ningún recurso hardware dado que la infraestructura de despliegue es proporcionada por el cliente.

La hoja de cálculos con el presupuesto completo y otras anotaciones se puede encontrar en el documento de *Microsoft Excel* adjunto a esta memoria.

4.3.1.5.1 Presupuesto de Costes

El presupuesto de costes se computa en base a:

1. Las tarifas de cada perfil del proyecto, multiplicadas por las horas de trabajo que dedican a las tareas del cronograma.
2. Los costes de los recursos materiales (hardware) y licencias (software) empleados en el desarrollo del proyecto, computados de una de las siguientes maneras:
    a. Las tarifas periódicas derivadas del uso del recurso, multiplicadas por el número de periodicidades que se estima que durará el proyecto.
    b. Según un factor de amortización, calculado en base a la vida útil considerada para el hardware y las licencias software de compra única.
3. Costes indirectos, derivados principalmente del mantenimiento de la compañía durante el desarrollo del proyecto.

4.3.1.5.1.1 Presupuesto de costes resumido

En la siguiente tabla se muestra el presupuesto de costes final, resumido en las partidas definitivas.

Partida	*Ítem (1)*	*Ítem (2)*	*Descripción*	*Importe (2)*	*Importe (1)*	*Total partida*
1			**Planificación y seguimiento del proyecto**			**10.437,00 €**
	1		Definición del sistema		1.860,00 €	
		1	Planificación inicial	960,00 €		
		2	Definición de la arquitectura tecnológica	900,00 €		
	2		Gestión del proyecto		6.822,00 €	
		1	Inicio del proyecto	4.157,00 €		
		2	Ejecución del proyecto	1.170,00 €		
		3	Cierre del proyecto	1.495,00 €		
	3		Seguimiento con el cliente		1.755,00 €	
		1	Reuniones iniciales y finales con el cliente	390,00 €		
		2	Reuniones de seguimiento	1.365,00 €		
2			**Análisis del sistema**			**4.736,00 €**
	1		Estudio de viabilidad del sistema		584,00 €	
	2		Análisis del sistema		4.152,00 €	
		1	Definición final del sistema	130,00 €		
		2	Establecimiento de requisitos del sistema	1.200,00 €		
		3	Identificación de actores y casos de uso	495,00 €		
		4	Identificación y descripción de subsistemas	315,00 €		
		5	Definición preliminar de clases	750,00 €		
		6	Definición preliminar de interfaces de usuario	844,00 €		
		7	Especificación inicial del plan de pruebas	418,00 €		
3			**Desarrollo del sistema**			**23.476,50 €**
	1		Estudios iniciales de tecnologías del sistema		6.045,50 €	
		1	Preparación del entorno de trabajo	192,00 €		
		2	Web semántica y procesamiento y validación de RDF	3.001,00 €		
		3	Flujos de datos aplicables a RDF	350,00 €		
		4	Tecnologías web de comunicación en tiempo real	734,50 €		
		5	Formación en las tecnologías del cliente	1.768,00 €		

	2		Diseño del sistema		3.282,00 €	
		1	Diseño de casos de uso	180,00 €		
		2	Diseño de clases	780,00 €		
		3	Diseño de arquitectura y datos	818,00 €		
		4	Diseño del plan de pruebas	1.504,00 €		
	3		Construcción del sistema		9.473,00 €	
		1	Definición del entorno	315,00 €		
		2	Módulo validador	2.318,00 €		
		3	Módulo API	3.586,00 €		
		4	Módulo Cliente	2.646,00 €		
		5	Stream de pruebas	608,00 €		
	4		Plan de pruebas y corrección de errores		1.736,00 €	
		1	Pruebas unitarias	380,00 €		
		2	Pruebas de integración	912,00 €		
		3	Pruebas de sistema	444,00 €		
	5		Manuales del sistema		2.940,00 €	
4			**Despliegue del sistema**			**1.120,00 €**
	1		Implantación del sistema en los servidores del cliente		784,00 €	
		1	Plan de implantación	336,00 €		
		2	Despliegue y verificación	448,00 €		
	2		Planificación del mantenimiento del sistema		336,00 €	
5			**Licencias y recursos materiales**			**1.707,20 €**
	1		Licencias software y servicios		1.411,30 €	
	2		Recursos materiales		295,90 €	
TOTAL INTERNO (COSTES DIRECTOS)						**41.476,70 €**
CI	**-**	**-**	**Costes indirectos**			**5.127,65 €**
TOTAL INTERNO (COSTES DIRECTOS E INDIRECTOS)						**46.604,35 €**

Tabla 32. Presupuesto de costes (inicial)

4.3.1.5.1.2 Costes indirectos

En la siguiente tabla se desglosan los costes indirectos para el proyecto, que corresponden un **11,00%** del total.

I1	*I2*	*I3*	*Descripción*	*Cantidad*	*Unidades*	*Precio*	*Subtotal (3)*	*Subtotal (2)*	*Total*
1			**Costes indirectos**						**7.083,16 €**
	1		Gastos de oficina					6.957,16 €	
		1	*Local oficina*	9	meses	550,00 €	4.950,00 €		
		2	*Agua*	9	meses	36,60 €	329,40 €		
		3	*Electricidad*	9	meses	147,52 €	1.327,66 €		
		4	*Internet / Telefonía*	9	meses	38,90 €	350,10 €		
	2		Gastos de papelería	9				126,00 €	
		1	*Lapiceros, bolígrafos y rotuladores*	9	meses	6,00 €	54,00 €		
		2	*Papel, folios y gastos de impresión y fotocopias*	9	meses	8,00 €	72,00 €		
								Prop.	**72,39%**
								TOTAL	**5.127,65 €**

Tabla 33. Costes indirectos (inicial)

Al final de los cálculos de puede observar un "porcentaje de proporcionalidad". Sería injusto facturar todos los gastos de alquiler, luz, copistería, etc. de la empresa a este proyecto, dado que nuestro modelo de empresa es capaz de manejar múltiples proyectos al mismo tiempo. Por tanto, se computa un factor de proporcionalidad en base a las horas de cada mes que la empresa destina de media al proyecto respecto a las horas laborables totales del mes.

4.3.1.5.2 Presupuesto de Cliente

Para la elaboración del presupuesto del cliente, se han tenido en cuenta las siguientes consideraciones:

- El beneficio económico esperado de la realización del proyecto es del **20%**.
- Varias de las (sub)partidas especificadas en el presupuesto interno son ocultadas al cliente deliberadamente y su precio redistribuido o agrupado dentro de otras partidas generales.
- El coste de los recursos y licencias no es incrementado en relación con el beneficio esperado y los costes indirectos, pues el incremento sería fácilmente contrastable por el cliente. El incremento que debería aplicarse a estas partidas se redistribuye proporcionalmente entre las demás para alcanzar el beneficio deseado.
- En última instancia, se incluye el IVA (21%).

4.3.1.5.2.1 Presupuesto de cliente detallado

Partida	*Ítem*	*Descripción*	*Importe*	*Total*
1		**Planificación y seguimiento del proyecto**		**13.959,69 €**
	1	Definición del sistema	2.487,79 €	
	2	Gestión y seguimiento del proyecto	11.471,91 €	
2		**Análisis del sistema**		**6.334,49 €**
	1	Estudio de viabilidad	781,11 €	
	2	Análisis del sistema	5.553,38 €	
3		**Desarrollo del sistema**		**31.400,28 €**
	1	Estudios tecnológicos y formación	8.085,98 €	
	2	Módulo Validador	7.602,02 €	
	3	Módulo API	8.484,78 €	
	4	Módulo Cliente	7.227,51 €	
4		**Despliegue del sistema**		**1.498,02 €**
	1	Implantación del sistema	1.048,62 €	
	2	Mantenimiento del sistema	449,41 €	
5		**Licencias y recursos materiales**		**1.707,20 €**
	1	Licencias software y servicios	1.411,30 €	
	2	Recursos materiales	295,90 €	
			TOTAL CLIENTE	**54.899,69 €**
			IVA	**11.528,94 €**
			TOTAL CLIENTE + IVA	**66.428,63 €**

Tabla 34. Presupuesto de cliente detallado (inicial)

4.3.1.5.2.2 Presupuesto de cliente resumido

Partida	*Descripción*	*Total*
1	Planificación y seguimiento del proyecto	13.959,69 €
2	Análisis del sistema	6.334,49 €
3	Desarrollo del sistema	31.400,28 €
4	Despliegue del sistema	1.498,02 €
5	Licencias y recursos materiales	1.707,20 €
	TOTAL CLIENTE	**54.899,69 €**
	IVA	**11.528,94 €**
	TOTAL CLIENTE (+ IVA)	**66.428,63 €**

Tabla 35. Presupuesto de cliente resumido (inicial)

4.3.2 Ejecución del Proyecto

4.3.2.1 *Plan Seguimiento de Planificación*

Atendiendo a la planificación inicial, se puede ver que se han asignado reservas temporales para hacer un seguimiento de la planificación del proyecto una vez definida. En concreto, el jefe de proyecto será en encargado de, en estos períodos, revisar que se estén cumpliendo los objetivos y corregir la planificación según corresponda.

ID Tarea	Descripción	Duración
118	Sesión de seguimiento de la planificación (1)	21/01/2022 – 21/01/2022
120	Sesión de seguimiento de la planificación (2)	25/02/2022 – 25/02/2022
122	Sesión de seguimiento de la planificación (3)	01/04/2022 – 01/04/2022
133	Revisión y documentación de la planificación final	02/05/2022 – 03/05/2022

Tabla 36. Tareas de seguimiento de la planificación

Para el plan de seguimiento del proyecto se definen un total de 3 líneas base:

1. **Línea base inicial**: antes del inicio de las tareas del proyecto.
2. **Línea base a medio proyecto**: se corresponde con la segunda auditoría de seguimiento de la planificación.
3. **Línea base final**: tras la conclusión de todas las tareas del proyecto y antes de la defensa.

Las diferencias entre líneas base en la duración y fecha de finalización del proyecto se deben a las correcciones aplicadas para paliar las incidencias surgidas durante el desarrollo, las cuales se pueden seguir cronológicamente en *4.3.2.2-Bitácora de Incidencias del Proyecto*.

4.3.2.1.1 Línea Base Inicial

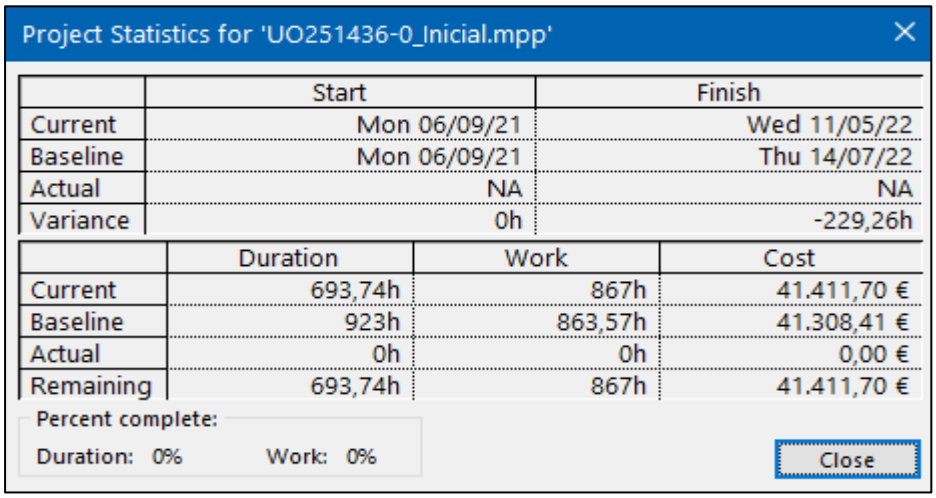


Ilustración 35. Estadísticas de la línea base inicial

4.3.2.1.2 Línea Base a Medio Proyecto

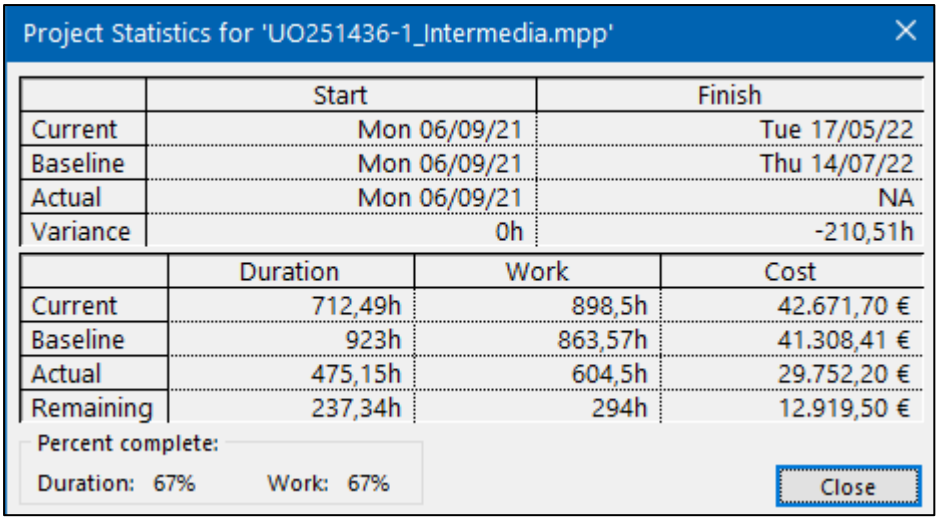


Ilustración 36. Estadísticas de la línea base intermedia

4.3.2.1.3 Línea Base a Final de Proyecto

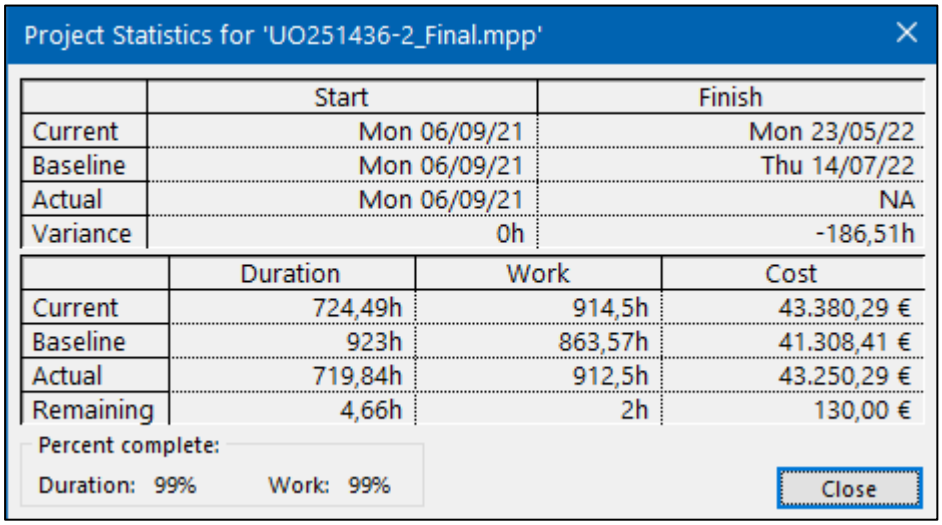


Ilustración 37. Estadísticas de la línea base final

4.3.2.2 *Bitácora de Incidencias del Proyecto*

Fecha	Descripción
06/09/2021	- Reunión inicial con el cliente donde se comprenden las necesidades del proyecto a alto nivel. - Se adquieren las bases para empezar a trabajar en el cronograma y el presupuesto iniciales.
08/09/2021	- Se completa la preparación del entorno de trabajo para el proyecto, tanto físico como tecnológico.
05/10/2021	- Finalizan los estudios tecnológicos iniciales sobre RDF, Shape Expressions y manipulación de *streams*.
16/09/2021	- Se elabora la planificación inicial del proyecto.
21/09/2021	- Se finalizan el plan de gestión y de seguimiento de riesgos del proyecto.
28/09/2021	- Se elabora el presupuesto inicial del proyecto.
01/11/2021	- Termina el período de formación del programador en las tecnologías empleadas por el cliente, tomando una semana más de la esperado. - Se anota el suceso para reestimar las reservas de tiempo destinadas al desarrollo del módulo API. - Se atrasa una semana el plan de reuniones y auditorías de seguimiento para ajustarse a este retraso inicial.
02/11/2021	- Primera reunión de puesta al día con el cliente tras el período de estudios y formación. - Se aclaran cuestiones y se inicia el plan de reuniones periódicas quincenales con el cliente.
05/11/2021	- Algunos perfiles del proyecto informan de que tendrán menos productividad por encontrarse en proceso de mudanza. - Se recurre a la flexibilidad horaria y posibilidad de teletrabajo. - Se anota el suceso para revisar la planificación temporal en las sesiones de seguimiento.
12/11/2021	- Se decide la arquitectura tecnológica final del sistema.
17/11/2021	- Concluye el estudio de viabilidad del sistema y la elección de tecnologías del proyecto (aquellas tecnologías no limitadas por el cliente).
24/11/2021	- Se publica la vulnerabilidad *zero-day* Log4Shell, capaz de facilitar la ejecución de código remoto a un agente externo al sistema. - Se anota el suceso para revisar la arquitectura planeada para el sistema de logs en la primera auditoría de seguimiento.

16/12/2021	- Se catalogan y documentan formalmente los requisitos finales del sistema.
23/12/2021	- Conocidos los requisitos y arquitectura finales del sistema, se completan el catálogo de riesgos y las hojas (iniciales) de riesgos.
24/12/2021	- Se detiene el desarrollo del proyecto por festividades navideñas.
09/01/2022	- Se reanuda el desarrollo del proyecto tras las festividades navideñas. - Se amplía el horario de trabajo, según lo planeado.
21/01/2021	- Concluye la etapa de análisis del sistema según lo planificado.
28/01/2022	- Primera auditoría de seguimiento del proyecto. - Se deciden mantener las estimaciones iniciales de desarrollo para el módulo API. - No se considera atrasar el proyecto, los efectos de la mudanza de algunos trabajadores sobre la planificación han sido mitigados. - Se añaden tareas al desarrollo del sistema de logs para verificar el impacto de Log4Shell sobre el proyecto y aplicar medidas correctivas. - Ahora que se conocen las tecnologías a utilizar, se decide que el riesgo de retrasos en las pruebas del módulo validador es elevado. o Dado que se desea un *stream* de pruebas fácilmente desplegable y personalizable, se decide crear una solución propia con *Docker*, basándose en implementaciones previas de *Apache Kafka* y *Zookeeper*. o Se añaden reservas temporales para crear el stream de pruebas.
02/02/2022	- Concluye la etapa de diseño del sistema, según lo planeado.
16/02/2022	- Concluye el desarrollo del módulo validador, con un día de retraso respecto a lo planeado. - Se anota el suceso para tomar las medidas correctivas a la planificación en la siguiente auditoría.
22/02/2022	- Concluye el desarrollo del *stream* de pruebas para el módulo validador.
04/03/2022	- Segunda auditoría de seguimiento del proyecto. - Se reajusta la planificación del proyecto, contando con el tiempo extra dedicado al módulo validador.
07/03/2022	- Varios perfiles del proyecto presentan síntomas moderados de COVID, causando la pérdida de dos jornadas laborales: del 07/03 al 08/03. - Se anota el suceso para tomar las medidas correctivas a la planificación en la siguiente auditoría.

16/03/2022	- Concluye el desarrollo del módulo API, según lo planeado.
31/03/2022	- Concluye el desarrollo del módulo Cliente, según lo planeado.
04/04/2022	- Caída del despliegue en los servidores de pruebas del cliente. - Se anota el suceso para plantear medidas correctivas en la futura sesión de auditoría.
06/04/2022	- Los logs del módulo API dejan de funcionar silenciosamente tras una actualización de la dependencia. - Se anota el suceso para su evaluación en la siguiente auditoría.

08/04/2022	- Tercera auditoría de seguimiento del proyecto.
	- Se decide hacer cambios en base al incidente del día 4 para garantizar un futuro despliegue exitoso en el entorno de producción:
	  o Más tiempo destinado a automatizar la renovación de certificados SSL.
	  o Más tiempo destinado a elaborar manuales de instalación y ejecución.
	  o Se altera la planificación para reflejar estos cambios.
	- La planificación general del proyecto se retrasa dos días para compensar los efectos del COVID. No se modifican las estimaciones temporales.
	- Se definen tareas adicionales para diagnosticar y solventar los errores en los logs del módulo API.

09/04/2022	- Se detiene el desarrollo del proyecto por semana santa.
21/04/2022	- Se reanuda el desarrollo del proyecto tras semana santa.
22/04/2022	- Concluye la ejecución del plan de pruebas del sistema. - Los resultados son satisfactorios y las reservas de tiempo destinadas a la corrección de fallos son suficientes.
05/05/2022	- Concluye la elaboración de los manuales del sistema.
12/05/2022	- Concluye el despliegue del sistema en producción. - El sistema comienza a ser empleado por el cliente para preparar las clases de la asignatura de Web Semántica del Máster.
20/05/2022	- Termina la documentación y revisión de la memoria del proyecto.
23/05/2022	- Reunión de cierre con el cliente. - Los resultados del proyecto son suficientes como para su defensa. - Se decide mantener la defensa del proyecto en Julio, por conveniencia.

Tabla 37. Resumen temporal de las incidencias del proyecto

4.3.2.3 *Riesgos*

El seguimiento de riesgos tiene lugar periódicamente de acuerdo con lo estipulado en el plan de gestión de riesgos del proyecto.

De entre los riesgos listados en *4.3.1.4.2-Identificación de Riesgos*, se ha realizado un seguimiento pormenorizado de los identificadores **1, 2, 7, 11** y **15**, cuyas hojas de riesgos se hallan en el anexo *A2-Hojas de riesgos*.

4.3.3 Cierre del proyecto

4.3.3.1 *Planificación Final*

A continuación, se especifica la planificación final del proyecto, elaborada cerca del término de este, y por tanto definitiva.

4.3.3.1.1 Diagrama de Gantt

A continuación, se muestra, mediante capturas de pantalla, el diagrama de Gantt para la planificación final del proyecto.

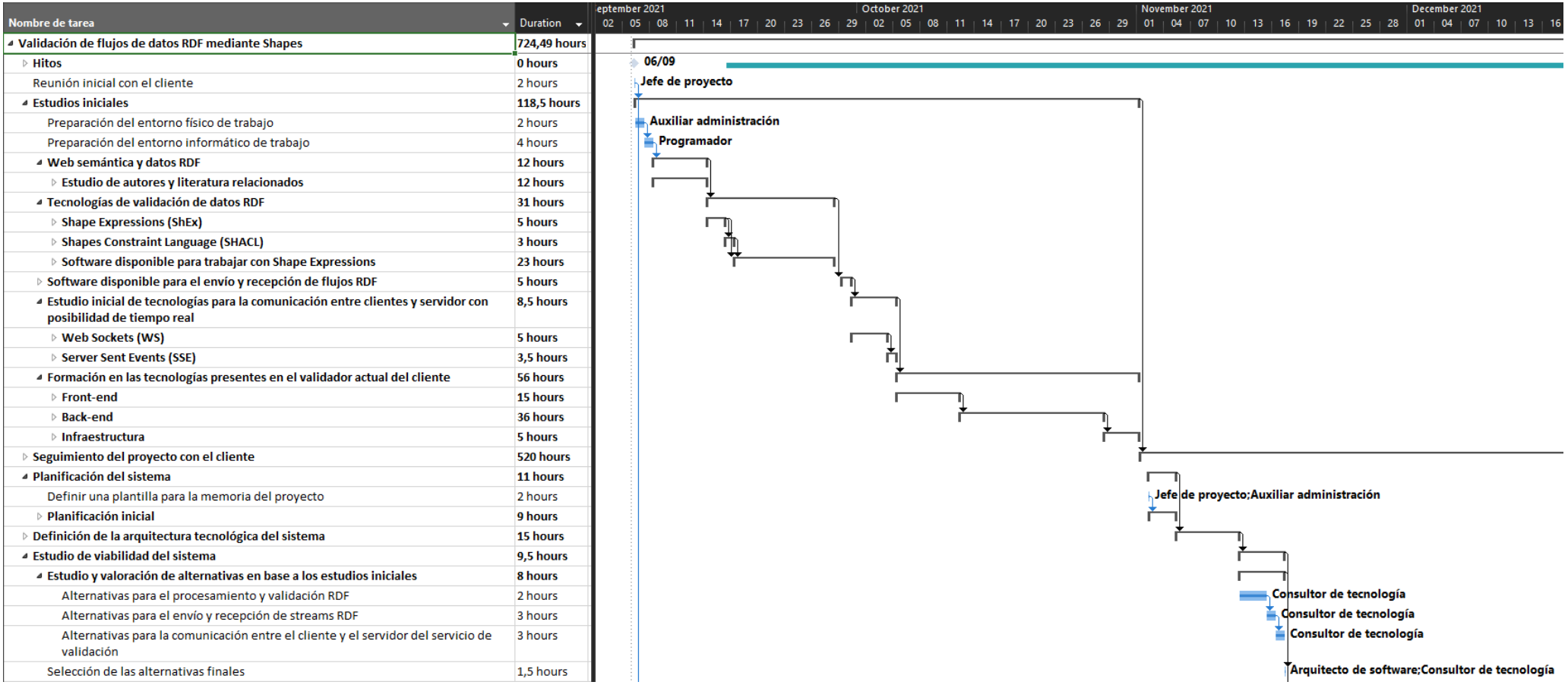


Ilustración 38. Diagrama de Gantt - Planificación final I

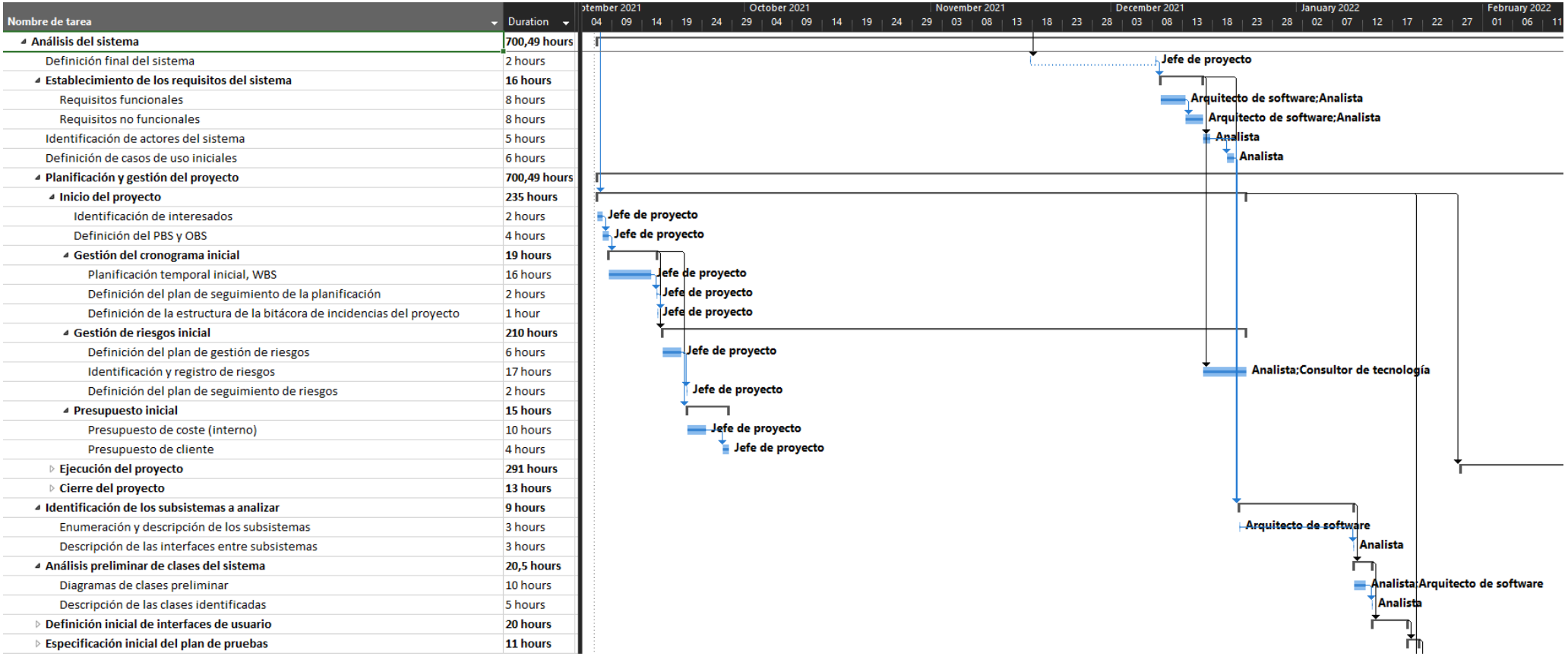


Ilustración 39. Diagrama de Gantt - Planificación final II

Nombre de tarea	Duration
Diseño del sistema	**51 hours**
Diseño de casos de uso finales	4 hours
Diseño de clases trasladable al sistema final	12 hours
Diseño de la arquitectura de módulos del sistema	**13 hours**
Diseño físico de los datos	**2 hours**
Diseño del plan de pruebas	**20 hours**
Pruebas del módulo de validación de streams	**6 hours**
Pruebas del API del servicio de validación	**4 hours**
Pruebas del portal web del servicio de validación	**4 hours**
Pruebas de integración y del sistema	6 hours
Construcción del sistema	**324,99 hours**
Definición del entorno de generación y construcción del sistema	**16,5 hours**
Generación del código de los componentes del sistema	**239,83 hours**
Módulo 1: Librería de validación de flujos RDF (Comet)	**58 hours**
Implementación de funcionalidad	**53 hours**
Implementación de pruebas unitarias	5 hours
Módulo 2: Servidor web API Rest (RDFShape-API)	**91 hours**
Implementación de funcionalidad	**86 hours**
Implementación de pruebas unitarias	5 hours
Módulo 3: Cliente web (RDFShape-Client)	**66,83 hours**
Implementación de funcionalidad	**60,33 hours**
Implementación de pruebas	**6,5 hours**
Desarrollo de un stream de datos RDF pseudoaleatorios para alimentar el sistema de validación	**24 hours**
Ejecución del plan de pruebas y corrección de errores	**39,16 hours**
Pruebas unitarias	8 hours
Pruebas de integración	**24,49 hours**
Pruebas del sistema	**6 hours**
Elaboración de los manuales de usuario	**46 hours**
Manuales de instalación	**13 hours**
Manuales de ejecución	**13 hours**
Manuales de usuario	**9 hours**
Manuales de programador	**11 hours**

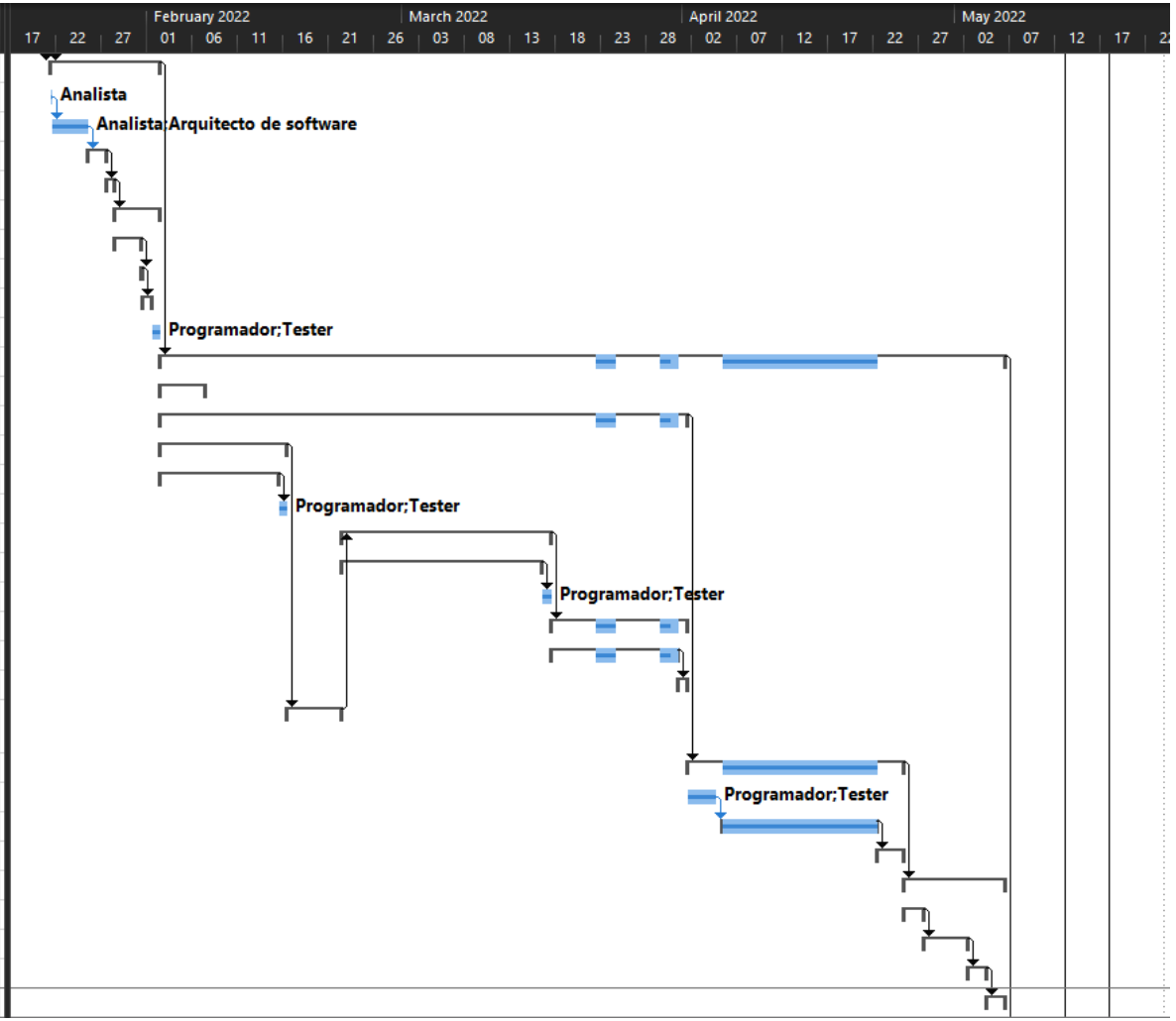


Ilustración 40. Diagrama de Gantt -Planificación final III

Nombre de tarea	Duration
Implantación y aceptación del sistema	**22 hours**
Implantación del sistema desarrollado en los servidores del cliente	**14 hours**
Plan de implantación del sistema en la infraestructura de equipos de WESO	6 hours
Despliegue del sistema construido en una de las máquinas del grupo WESO	1 hour
Pruebas de implantación del sistema	3 hours
Estudio de seguridad del entorno de despliegue	4 hours
Preparación del mantenimiento del sistema	**8 hours**
Recuperación del servidor en caso de caída	3 hours
Renovación automática del certificado SSL	5 hours
Revisar y completar la memoria del proyecto	**19 hours**
Conclusiones y ampliaciones	2 hours
Documentación anexa	**5 hours**
Documentos relacionados con la dirección y gestión del proyecto	3 hours
Referencias bibliográficas	2 hours
Revisión de la memoria	**8 hours**
Revisión de contenidos	5 hours
Revisión ortográfica y de formato	3 hours
Resumen de ficheros entregados	**4 hours**
Descripción de ficheros entregados	2 hours
Estructura de directorios de la entrega	2 hours
Visto bueno del cliente	**3 hours**
Reunión final con el cliente	1 hour
Corrección de los errores detectados	2 hours
~~Defensa del proyecto~~	~~1 hour~~

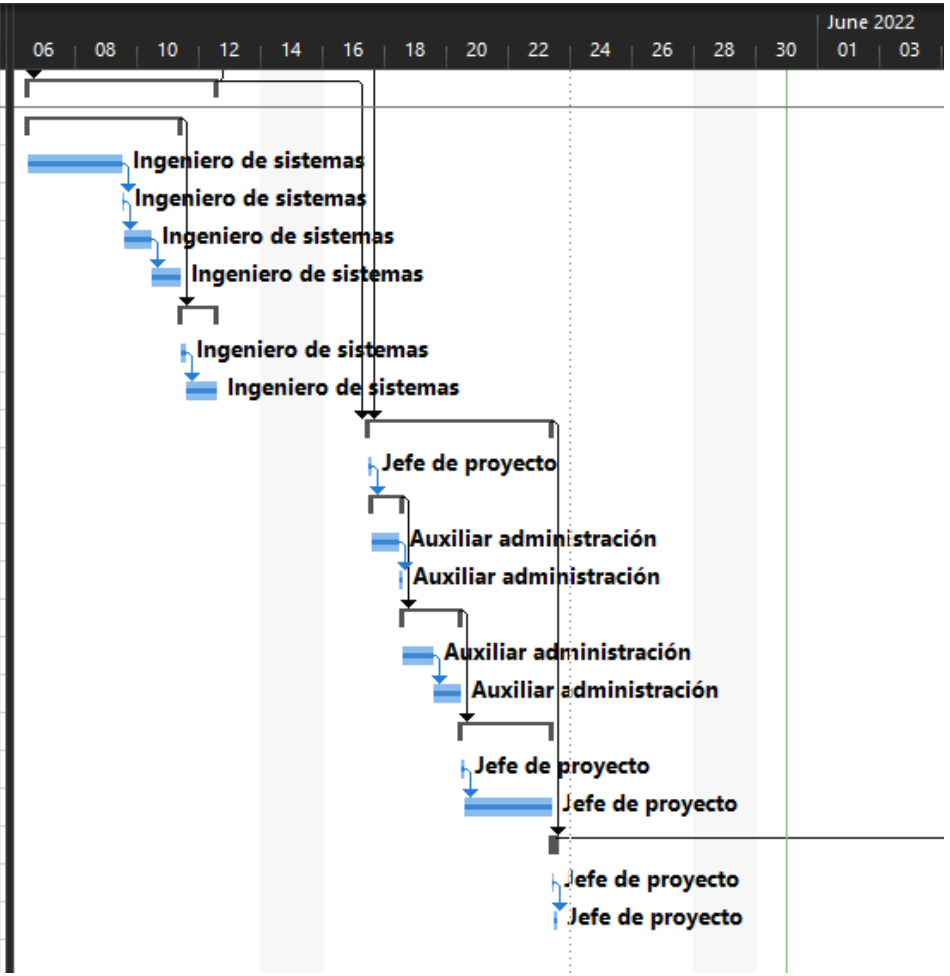


Ilustración 41. Diagrama de Gantt - Planificación final IV

4.3.3.1.2 Cuadro resumen

El siguiente cuadro muestra, en forma de tabla, las tareas principales finales del proyecto y las horas de trabajo dedicadas a las mismas. Las tareas aquí mostradas son un resumen de las totales.

ID Tarea	Nombre	Fecha inicio	Fecha fin	Horas de trabajo
15	Reunión inicial con el cliente	06/09/21	06/09/21	2 horas
17	Preparación del entorno físico de trabajo	06/09/21	07/09/21	2 horas
18	Preparación del entorno informático de trabajo	07/09/21	08/09/21	4 horas
19	Estudios en web semántica y datos RDF	08/09/21	14/09/21	17 horas
24	Estudios en tecnologías de validación de datos RDF	14/09/21	28/09/21	43 horas
35	Estudios en software disponible para el envío y recepción de flujos RDF	29/09/21	30/09/21	7 horas
40	Estudio inicial de tecnologías para la comunicación entre clientes y servidor con posibilidad de tiempo real	30/09/21	05/10/21	15 horas
47	Formación en las tecnologías presentes en el validador actual del cliente	05/10/21	01/11/21	66 horas
60	Seguimiento del proyecto con el cliente (+ reuniones periódicas)	01/11/21	29/04/22	21 horas
77	Definir una plantilla para la memoria del proyecto	02/11/21	02/11/21	3 horas
78	Planificación del sistema	02/11/21	05/11/21	15 horas
84	Definición de la arquitectura tecnológica del sistema	05/11/21	12/11/21	15 horas
87	Estudio de viabilidad del sistema	12/11/21	17/11/21	11 horas

98	Identificación de actores y casos de uso	16/12/21	21/12/21	11 horas
95	Establecimiento de los requisitos del sistema	09/12/21	16/12/21	24 horas
100	Planificación y gestión del proyecto	06/09/21	17/05/22	100 horas
137	Identificación de los subsistemas	22/12/21	10/01/21	6 horas
140	Análisis preliminar de clases	10/12/21	13/01/22	15 horas
143	Definición inicial de interfaces de usuario	13/01/22	19/01/22	22 horas
149	Especificación inicial del plan de pruebas	19/01/22	21/01/22	11 horas
153	Diseño de casos de uso finales	21/01/22	21/01/22	4 horas
154	Diseño de clases	21/01/22	25/01/22	16 horas
155	Diseño de la arquitectura de módulos del sistema	25/01/22	27/01/22	17 horas
161	Diseño físico de los datos	27/01/22	28/01/22	2 horas
163	Diseño del plan de pruebas	28/01/22	02/02/22	40 horas
174	Definición del entorno de generación y construcción del sistema	02/02/22	07/02/22	7 horas
179	Módulo 1: Biblioteca de validación de flujos RDF (Comet)	02/02/22	16/02/22	63 horas
187	Módulo 2: Servidor web API REST (RDFShape-API)	22/02/22	17/03/22	98 horas
204	Módulo 3: Cliente web (RDFShape-Client)	17/03/22	01/04/22	67,5 horas
223	Stream de datos RDF pseudoaleatorios para alimentar el sistema de validación	16/02/22	22/02/22	24 horas

226	Ejecución del plan de pruebas y corrección de errores	01/04/22	25/04/22	46 horas
235	Elaboración de los manuales de usuario	25/04/22	06/05/22	74 horas
253	Implantación del sistema desarrollado en los servidores del cliente	06/05/22	11/05/22	14 horas
258	Preparación del mantenimiento del sistema	11/05/22	12/05/22	8 horas
261	Revisar y completar la memoria del proyecto	17/05/22	23/05/22	19 horas
272	Visto bueno del cliente	23/05/22	23/05/22	3 horas
275	Defensa del proyecto	14/07/22	14/07/22	1 hora

Tabla 38. Cuadro resumen de la planificación final

4.3.3.2 *Informe Final de Riesgos*

A lo largo del desarrollo del proyecto, se han identificado un total de quince riesgos potenciales, de los cuales se han seleccionado cinco riesgos significativos para su análisis cuantitativo y seguimiento mediante la elaboración de hojas de riesgos.

4.3.3.2.1 Presupuesto de riesgos

La siguiente tabla resume el presupuesto final destinado a la gestión de riesgos del proyecto. Se sigue lo presupuestado en el plan de gestión de riesgos.

Tarea (ID MPP)	Descripción	Asignación (€)
109	Definición del plan de gestión de riesgos	390,00 €
110	Identificación y registro de riesgos	1.102,00 €
111	Definición del plan de seguimiento de riesgos	130,00 €
125	Sesiones de seguimiento de riesgos	585,00 €
135	Informe final de riesgos	260,00 €
	TOTAL	**2467,00 €**

Tabla 39. Presupuesto de riesgos (final)

4.3.3.2.2 Calendario de riesgos

La siguiente tabla muestra el calendario final dedicado a la gestión de riesgos del proyecto. Se sigue lo planificado en el plan de gestión de riesgos, con retrasos para compensar las desviaciones ocurridas en la planificación.

ID Tarea	Descripción	Duración
109	Definición del plan de gestión de riesgos	17/09/2021 – 20/09/2021
110	Identificación y registro de riesgos	16/12/2021 – 23/12/2021
111	Definición del plan de seguimiento de riesgos	21/09/2021 – 21/09/2021
125	Sesiones de seguimiento de riesgos	28/01/2022 – 08/04/2022
135	Informe final de riesgos	16/05/2022 – 17/05/2022

Tabla 40. Calendario de riesgos (final)

4.3.3.3 *Presupuesto Final*

Se mantienen las partidas, así como las premisas y consideraciones explicadas en la elaboración del presupuesto inicial (ver *4.3.1.5-Presupuesto Inicial*) y se ajustan los presupuestos a las correcciones de planificación ejecutadas a raíz de la gestión de riesgos.

4.3.3.3.1 Presupuesto de Costes

A continuación, se muestra desglosado el presupuesto de costes final del proyecto.

Partida	***Ítem (1)***	***Ítem (2)***	***Descripción***	***Importe (2)***	***Importe (1)***	***Total partida***
1			**Planificación y seguimiento del proyecto**			**10.437,00 €**
	1		Definición del sistema		1.860,00 €	
		1	Planificación inicial	960,00 €		
		2	Definición de la arquitectura tecnológica	900,00 €		
	2		Gestión del proyecto		6.822,00 €	
		1	Inicio del proyecto	4.157,00 €		
		2	Ejecución del proyecto	1.170,00 €		
		3	Cierre del proyecto	1.495,00 €		
	3		Seguimiento con el cliente		1.755,00 €	
		1	Reuniones iniciales y finales con el cliente	390,00 €		
		2	Reuniones de seguimiento	1.365,00 €		
2			**Análisis del sistema**			**4.736,00 €**
	1		Estudio de viabilidad del sistema		584,00 €	
	2		Análisis del sistema		4.152,00 €	
		1	Definición final del sistema	130,00 €		

		2	Establecimiento de requisitos del sistema	1.200,00 €		
		3	Identificación de actores y casos de uso	495,00 €		
		4	Identificación y descripción de subsistemas	315,00 €		
		5	Definición preliminar de clases	750,00 €		
		6	Definición preliminar de interfaces de usuario	844,00 €		
		7	Especificación inicial del plan de pruebas	418,00 €		
3			**Desarrollo del sistema**			**25.340,50 €**
	1		Estudios iniciales de tecnologías del sistema		6.925,50 €	
		1	Preparación del entorno de trabajo	192,00 €		
		2	Web semántica y procesamiento y validación de RDF	3.001,00 €		
		3	Flujos de datos aplicables a RDF	350,00 €		
		4	Tecnologías web de comunicación en tiempo real	734,50 €		
		5	Formación en las tecnologías del cliente	2.648,00 €		
	2		Diseño del sistema		3.282,00 €	
		1	Diseño de casos de uso	180,00 €		
		2	Diseño de clases	780,00 €		
		3	Diseño de arquitectura y datos	818,00 €		
		4	Diseño del plan de pruebas	1.504,00 €		
	3		Construcción del sistema		10.081,00 €	
		1	Definición del entorno	315,00 €		
		2	Módulo validador	2.622,00 €		
		3	Módulo API	3.586,00 €		
		4	Módulo Cliente	2.646,00 €		
		5	Stream de pruebas	912,00 €		
	4		Plan de pruebas y corrección de errores		1.736,00 €	
		1	Pruebas unitarias	380,00 €		
		2	Pruebas de integración	912,00 €		
		3	Pruebas de sistema	444,00 €		
	5		Manuales del sistema		3.316,00 €	
4			**Despliegue del sistema**			**1.232,00 €**
	1		Implantación del sistema en los servidores del cliente		784,00 €	
		1	Plan de implantación	336,00 €		
		2	Despliegue y verificación	448,00 €		

	2		Planificación del mantenimiento del sistema		448,00 €	
5			**Licencias y recursos materiales**			**1.707,20 €**
	1		Licencias software y servicios		1.411,30 €	
	2		Recursos materiales		295,90 €	
					TOTAL INTERNO (COSTES DIRECTOS)	**45.452,70 €**
CI	**-**	**-**	**Costes indirectos**			**5.127,65 €**
					TOTAL INTERNO (COSTES DIRECTOS E INDIRECTOS)	**48.580,35 €**

Tabla 41. Presupuesto de costes (final)

Al comparar el presupuesto de costes inicial con el final se puede apreciar un incremento de 46.604,35€ a 48.580,35€, es decir, de **1.976€**.

4.3.3.3.2 Presupuesto de Cliente

La siguiente tabla muestra el presupuesto de cliente (resumido) final.

Partida	***Descripción***	***Total***
1	Planificación y seguimiento del proyecto	13.891,75 €
2	Análisis del sistema	6.303,66 €
3	Desarrollo del sistema	33.728,47 €
4	Despliegue del sistema	1.639,80 €
5	Licencias y recursos materiales	1.707,20 €
	TOTAL CLIENTE	**57.270,89 €**
	IVA	**12.026,89 €**
	TOTAL CLIENTE (+ IVA)	**69.297,78 €**

Tabla 42. Presupuesto de cliente resumido (final)

Al comparar el presupuesto de cliente inicial con el final se puede apreciar un incremento de 66.428,63€ a 69.297,78€, es decir, de **2.869,15€**.

4.3.3.4 *Informe de Lecciones Aprendidas*

Una vez llegados al cierre del proyecto, se pueden extraer las siguientes lecciones.

4.3.3.4.1 Aspectos positivos

- Se ha conseguido completar el proyecto con una desviación de la planificación mucho menor que en otros trabajos previos del alumno (proyectos propios, trabajo de fin de grado, etc.). Esto puede deberse a una mayor experiencia y conocimiento de gestión de proyectos, aunque aún queda mucho por mejorar.
- El modelo de reuniones periódicas breves con el cliente/tutor ha demostrado ser un acierto para refinar requisitos y recibir *feedback* regularmente.

4.3.3.4.2 Aspectos negativos

- Es necesario subdividir más las tareas, especialmente las de desarrollo software, para poder gestionar mejor los recursos y medir el avance. Aunque en la planificación inicial sea complicado crear las tareas de desarrollo, estas deberían haberse refinado más en las auditorías de seguimiento. Por suerte, los objetivos del proyecto estaban claros y varios de los módulos eran refactorizaciones más fáciles de estimar.
- Se infravaloró el tiempo dedicado a tareas de formación en tecnologías en comparación con el tiempo estimado para las tareas de desarrollo. Esto se debe, presumiblemente, al perfil todavía *junior* del alumno, demasiado obcecado con las partes de programación en cuestión.
- Hay que destinar más reservas de tiempo y/o presupuesto para corregir la materialización de los riesgos del proyecto. Durante el desarrollo ha habido problemas de salud, caídas de entornos de despliegue, así como vulnerabilidades o deprecación de bibliotecas de terceros. Afortunadamente, se han podido corregir fácilmente retrasando la planificación, dado que el proyecto contaba con un amplio margen de tiempo de finalización, si bien esto no es lo ideal.

4.4 ASI 3: IDENTIFICACIÓN DE SUBSISTEMAS DE ANÁLISIS

El producto resultante del proyecto no es un único sistema monolítico, sino un conjunto de varios subsistemas con responsabilidades únicas y una serie de interfaces que los comunican entre sí. Estos subsistemas se corresponden con los tres módulos identificados para el proyecto: biblioteca de validación de flujos, API y cliente.

Adicionalmente, cada uno de estos subsistemas puede dividirse internamente en más subsistemas, de nuevo con características y funcionalidad propia dentro de su contexto.

4.4.1 Descripción de los Subsistemas

4.4.1.1 *Subsistema validador de flujos*

Módulo encargado de consumir flujos de datos RDF y emitir los resultados de la validación de estos. Está construido como una biblioteca de código extensible y reutilizable, pensada para ser integrada en programas de terceros.

La biblioteca se divide en dos principales subsistemas:

4.4.1.1.1 Extractor

Comprende el código y utilidades para, dada la información necesaria, conectar con un flujo de entrada, consumir los datos que van llegando y convertirlos en objetos RDF del dominio de la aplicación.

El extractor de flujos depende de varias configuraciones base, tales como el formato y tipo de datos RDF esperados, el nivel de paralelización en el consumo de ítems del flujo o el tiempo de esperar antes de abortar la conexión con el flujo. Así mismo, proporciona una interfaz común basada en *Fs2* para preprocesar los flujos antes de transferirlos al subsistema Validador.

Cabe destacar que el módulo extractor permite ser extendido para ser capaz de interactuar con nuevos flujos de entrada de datos.

4.4.1.1.2 Validador

Subsistema encargado de convertir un flujo de datos RDF en un flujo de resultados de validaciones. El Validador opera después del Extractor, lo que le permite:

- Partir de la premisa de que sus datos de entrada son fragmentos RDF válidos.
- Operar independientemente de la procedencia del flujo de datos RDF, pues la conexión, extracción y conversión de los contenidos a RDF son tarea del Extractor.

El validador de flujos necesita una serie de configuraciones para funcionar, principalmente el esquema contra el que va a validar los datos que procese y cómo debe reaccionar ante errores de validación.

4.4.1.2 *Subsistema API*

Subsistema que reside en el servidor de *RDFShape*. Es un programa complejo pero en lo que atañe al proyecto destaca por integrar la biblioteca de validación de flujos y exponerla al exterior a través de una API REST y un servidor *WebSockets*.

Este módulo se divide a su vez en varios subsistemas:

4.4.1.2.1 Servidor HTTP

Recibe las peticiones HTTP enviadas por los clientes del API expuesta y las procesa en función de la operación solicitada por el cliente y sus datos asociados. Implementa la lógica de negocio de la aplicación para:

- **Procesamiento de RDF, *Shape Expressions* y *ShapeMaps***
  Al margen de las validaciones de flujos, los clientes pueden invocar el API de *RDFShape* para realizar diversas operaciones con tecnologías semánticas.
- **Procesamiento de *Permalinks***
  El Servidor implementa la conexión con la base de datos para el almacenamiento y recuperación de información relacionada con enlaces guardados.

4.4.1.2.2 Servidor *WebSockets*

De forma análoga al servidor HTTP, escucha las solicitudes de conexión de los clientes a través de *WebSockets*. Implementa la lógica de negocio para:

- **Realizar validaciones de flujos**
  El servidor recibe peticiones de validación de los clientes y, en caso de ser válidas, corresponde el intento de conexión enviando los resultados de la validación al cliente en tiempo real hasta que una de las partes aborta el proceso.

4.4.1.2.3 Recogida de datos

El servidor cuenta con herramientas para almacenar, a alto nivel, información de uso y de las peticiones realizadas contra el API. Los datos recogidos no son personales bajo ningún concepto y se almacenan en ficheros de *log* dentro del sistema de ficheros del servidor para su revisión o post-procesado.

4.4.1.3 *Subsistema Cliente*

Aplicación web servida desde el servidor de *RDFShape*. Al igual que el API, es un programa multidisciplinar, si bien las partes refactorizadas y ampliadas para este trabajo incluyen la invocación

y manipulación de validaciones de flujos a través de una interfaz web que actúa de cliente HTTP/*WebSockets* del subsistema API. Así mismo, permite la visualización y trata de los resultados de la validación en tiempo real.

El portal web contiene dos subsistemas principales:

4.4.1.3.1 Cliente HTTP

Envía peticiones HTTP contra el API REST del servidor en base a las operaciones y parámetros solicitados por los usuarios desde la interfaz gráfica. Así mismo, interpreta las respuestas del API y las convierte en elementos de la interfaz y eventos que facilitan la interacción de los usuarios con el sistema.

4.4.1.3.2 Cliente *WebSockets*

Envía mensajes *WebSockets* contra el servidor de la aplicación. Sigue los mismos principios que el cliente HTTP, sólo que está especializado en operaciones de validación de flujos que requieren de una conexión bidireccional en tiempo real.

4.4.2 Descripción de los Interfaces entre Subsistemas

Se procede a mostrar gráficamente cómo se va a efectuar la comunicación entre los subsistemas expuestos previamente.

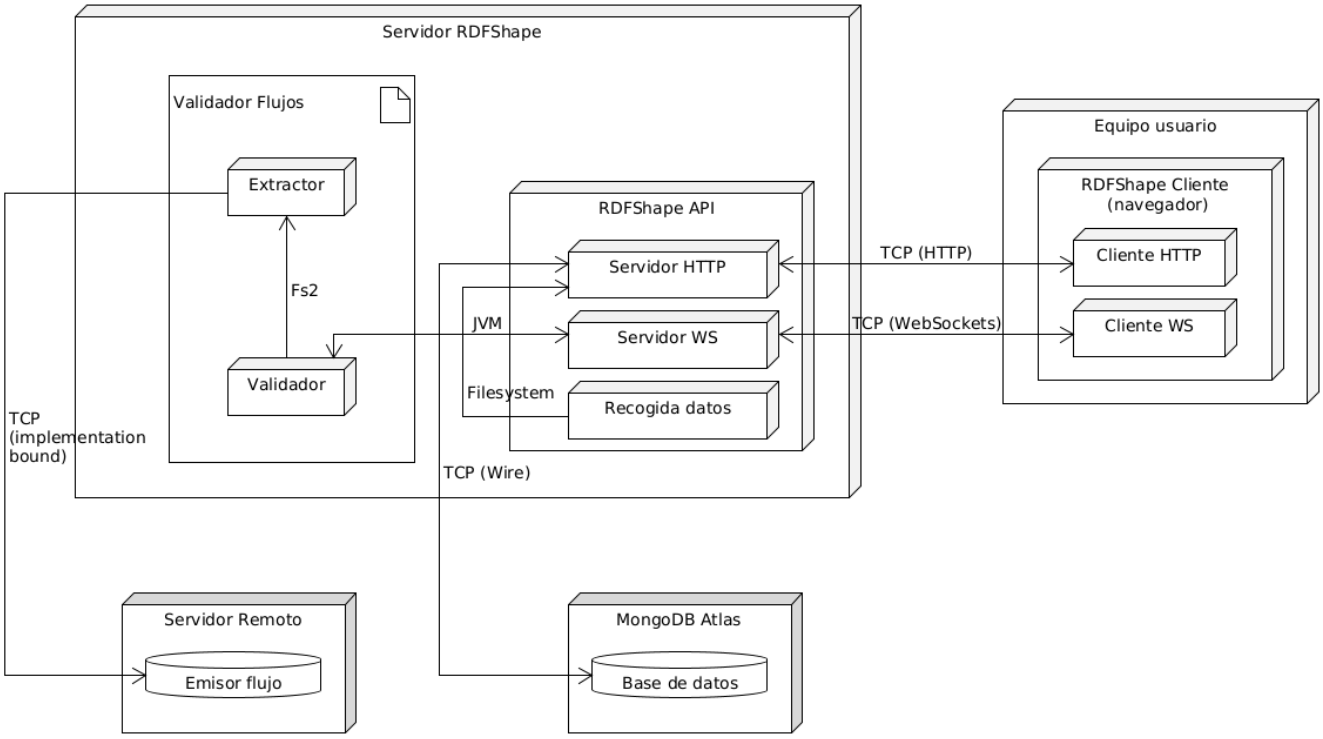


Ilustración 42. Interfaces entre subsistemas

La ilustración anterior emula un diagrama de despliegue simplificado para el sistema. Es necesario hacer las siguientes observaciones generales:

- Se pueden observar dos entornos de ejecución diferenciados: el servidor de *RDFShape*, donde se va a ejecutar el API, y el equipo de trabajo del usuario final, que ejecutará el cliente web.
- La biblioteca de validación de flujos contacta con un servidor externo que está a cargo de generar el flujo de datos de entrada. Dicha conexión es unidireccional, pues la biblioteca consume del emisor. La comunicación se realiza a través del protocolo TCP, si bien el protocolo específico dependerá de la implementación del flujo a consumir.
- Dentro del módulo de validación, los subsistemas Extractor y Validador se comunican en un único sentido: primero el Extractor procesa y luego el Validador valida. Esta comunicación se vale de *Fs2* como interfaz de procesamiento de flujos.
- Dentro del API, es el servidor *WebSockets* el que se hace cargo de las validaciones de flujos, invocando e intercambiando mensajes con el subsistema Validador, que es una dependencia del programa. Así mismo, el servidor HTTP genera información que luego es almacenada por el subsistema de recogida de datos.
- Los clientes y servidores de HTTP y *WebSockets* se comunican mutuamente con su contraparte mediante el protocolo correspondiente.
- La base de datos del sistema está alojada en MongoDB Atlas y es contactada por la lógica del servidor HTTP, que atiende las peticiones relacionadas con *permalinks*, a través del protocolo *Wire* **[95]** de MongoDB.

4.5 ASI 4: ANÁLISIS DE LOS CASOS DE USO

4.5.1 Usuario Biblioteca

Se desglosan los escenarios de los casos de uso especificados en *4.2.3.1-Usuario* .

4.5.1.1 *Integrar la biblioteca de validación en el proyecto*

INTEGRAR LA BIBLIOTECA DE VALIDACIÓN EN EL PROYECTO	
PRECONDICIONES	-
POSCONDICIONES	- El sistema tiene acceso al código y documentación de la biblioteca de validación de flujos para su uso en el proyecto del usuario.
ACTOR PRINCIPAL	- Usuario biblioteca.
DESCRIPCIÓN	1. El usuario crea un nuevo proyecto de desarrollo software basado en uno de los lenguajes de la *JVM*. 2. El usuario configura el mecanismo de gestión de dependencias del proyecto. 3. El usuario introduce los datos de la biblioteca de validación dentro del listado de dependencias. 4. El sistema descarga los binarios y documentación de la biblioteca de validación. 5. El sistema permite: a. Utilizar la biblioteca de validación en el proyecto a través de su API. b. Consultar la documentación del código de la biblioteca.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Proyecto existente:** - El usuario abre un proyecto de desarrollo software ya existente basado en uno de los lenguajes de la *JVM*. - Proseguir desde el paso 2 del escenario principal.
EXCEPCIONES	**Biblioteca de validación no disponible:** - No se puede contactar con el servicio en que están alojados los binarios de la biblioteca. - El sistema mostrará un mensaje de error y la dependencia no será descargada ni utilizable.
NOTAS	- La biblioteca contará con un identificador único para su adición como dependencia. - El servicio de alojamiento de los binarios proporcionará al programador los datos que debe añadir al listado de dependencias.

Tabla 43. Análisis caso de uso "Integrar la biblioteca de validación en el proyecto" (Usuario Biblioteca)

4.5.1.2 *Configurar el flujo de entrada*

CONFIGURAR EL FLUJO DE ENTRADA	
PRECONDICIONES	- El usuario cuenta con un proyecto de desarrollo software - La biblioteca de validación de flujos está instalada como dependencia del proyecto.
POSCONDICIONES	- El sistema está configurado para buscar el flujo de origen de los datos de la validación. - Los parámetros de la configuración son válidos.
ACTOR PRINCIPAL	- Usuario biblioteca.
DESCRIPCIÓN	1. El usuario crea un nuevo extractor mediante el API de la biblioteca de validación. 2. El sistema solicita la siguiente información: a. Fuente del flujo de datos de entrada. b. Función de deserialización de los datos recibidos a RDF. c. Formato esperado en el RDF recibido. d. Inferencia aplicable al RDF recibido. e. Nivel máximo de paralelismo en la extracción de elementos del flujo. f. Período máximo de inactividad del flujo antes de abortar la validación. 3. El usuario introduce los datos solicitados. 4. El compilador del sistema comprueba que los datos introducidos son del tipo correcto. 5. El sistema verifica que los datos introducidos son válidos. 6. El sistema genera un nuevo extractor de flujos.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Datos suministrados insuficientes:** - En el paso 3, el usuario no introduce todos los datos solicitados por el sistema. - El sistema comprueba la existencia de valores por defecto para los datos no solicitados. ○ Si existen alternativas predeterminadas para los valores faltantes, proseguir desde el paso 5. ○ Si los valores faltantes son imprescindibles, el compilador del sistema señala al usuario los valores faltantes. **Nivel de paralelismo erróneo:** - En el paso 5, el sistema verifica que el número especificado de ítems procesables en paralelo es igual o menor que 0.

	- El sistema notificará al usuario del problema al inicio de la validación. - Proseguir desde el paso 2. **Período de inactividad erróneo:** - En el paso 5, el sistema verifica que el período especificado antes de detener la validación por inactividad es igual o menor que 0 milisegundos. - El sistema notificará al usuario del problema al inicio de la validación. - Proseguir desde el paso 2.
EXCEPCIONES	-
NOTAS	-

Tabla 44. Análisis caso de uso "Configurar el flujo de entrada" (Usuario Biblioteca)

4.5.1.3 *Configurar el validador de los ítems del flujo*

CONFIGURAR EL VALIDADOR DE LOS ÍTEMS DEL FLUJO	
PRECONDICIONES	- El usuario cuenta con un proyecto de desarrollo software - La biblioteca de validación de flujos está instalada como dependencia del proyecto.
POSCONDICIONES	- El sistema está configurado para validar los datos que llegan del flujo de origen. - Los parámetros de la configuración son válidos.
ACTOR PRINCIPAL	- Usuario biblioteca.
DESCRIPCIÓN	1. El usuario crea un nuevo validador mediante el API de la biblioteca de validación. 2. El sistema solicita la siguiente información: a. Extractor del que proceden los ítems de la validación. b. Esquema *Shape Expressions* a emplear en la validación. c. *ShapeMap* a emplear en la validación. d. Estrategia que aplicar antes errores en la validación. e. Estrategia que aplicar ante ítems que no superen la validación. f. Nivel máximo de paralelismo en la validación de elementos del flujo. 3. El usuario introduce los datos solicitados. 4. El compilador del sistema comprueba que los datos introducidos son del tipo correcto. 5. El sistema verifica que los datos introducidos son válidos. 6. El sistema genera un nuevo validador de flujos.

VARIACIONES (ESCENARIOS SECUNDARIOS)	**Datos suministrados insuficientes:**
	- En el paso 3, el usuario no introduce todos los datos solicitados por el sistema.
	- El sistema comprueba la existencia de valores por defecto para los datos no solicitados.
	  o Si existen alternativas predeterminadas para los valores faltantes, proseguir desde el paso 5.
	  o Si los valores faltantes son imprescindibles, el compilador del sistema señala al usuario los valores faltantes.
	Nivel de paralelismo erróneo:
	- En el paso 5, el sistema verifica que el número especificado de ítems validables en paralelo es igual o menor que 0.
	- El sistema notificará al usuario del problema al inicio de la validación.
	- Proseguir desde el paso 2.

EXCEPCIONES	-
NOTAS	-

Tabla 45. Análisis caso de uso "Configurar el validador de los ítems del flujo" (Usuario Biblioteca)

4.5.1.4 *Invocar una validación de flujos*

INVOCAR UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario cuenta con un extractor y validador configurados.
POSCONDICIONES	- El sistema ha conectado con el origen del flujo y comienza a emitir un flujo de resultados. - El sistema valida ítems hasta que dejan de llegar o se cumplen las condiciones de parada del validador.
ACTOR PRINCIPAL	- Usuario biblioteca.

DESCRIPCIÓN	1. El usuario inicia el validador.
	2. El sistema conecta con el origen del flujo de datos.
	3. El sistema comienza a consumir ítems del flujo de datos. Para cada ítem:
	  a. El sistema reinicia la cuenta atrás para cerrar la validación por inactividad.
	  b. El sistema convierte los ítems entrantes en objetos del dominio con contenidos RDF.
	  c. El sistema valida cada ítem contra la configuración del validador.
	  d. El sistema emite el resultado de la validación de cada ítem en un nuevo flujo de resultados.
	4. El usuario aborta la validación.

	5. El sistema finaliza la validación.

VARIACIONES (ESCENARIOS SECUNDARIOS)	**Error durante la validación:**
	- En el paso 3, se produce un error inesperado en el sistema mientras se maneja y valida los ítems.
	- El sistema comprueba la configuración del validador.
	  ○ Si el validador está configurado para detenerse en caso de errores, el sistema emite una excepción conteniendo el error original, llegando al paso 5.
	  ○ Si el validador está configurado para omitir los errores, el sistema prosigue con la validación del siguiente elemento, volviendo al paso 3.
	Validación fallida:
	- En el paso 3, el ítem no supera el proceso de validación con éxito.
	- El sistema comprueba la configuración del validador.
	  ○ Si el validador está configurado para detenerse en caso de ítems inválidos, el sistema emite una excepción conteniendo el resultado fallido, llegando al paso 5.
	  ○ Si el validador está configurado para omitir los ítems inválidos, el sistema prosigue con la validación del siguiente elemento, volviendo al paso 3.
	Inactividad del flujo:
	- En el paso 3, el sistema no recibe nuevos elementos durante un período superior al máximo configurado.
	- El sistema emite una excepción conteniendo el tiempo de espera máximo permitido en la validación, llegando al paso 5.

EXCEPCIONES	**Flujo de entrada no disponible:** - En el paso 2, el sistema es incapaz de establecer conexión con el flujo de datos especificado. - El sistema emite una excepción notificando el error, llegando al paso 5.
NOTAS	-

Tabla 46. Análisis caso de uso "Invocar una validación de flujos" (Usuario Biblioteca)

4.5.1.5 *Implementar nuevos flujos de entrada*

IMPLEMENTAR NUEVOS FLUJOS DE ENTRADA	
PRECONDICIONES	- El usuario cuenta con un proyecto de desarrollo software - La biblioteca de validación de flujos está instalada como dependencia del proyecto.

POSCONDICIONES	- El sistema está configurado para validar datos procedentes de un nuevo tipo de flujo de origen.
ACTOR PRINCIPAL	- Usuario biblioteca.
DESCRIPCIÓN	1. El usuario crea una nueva clase que extiende un extractor de los proporcionados por el API de la biblioteca de validación. 2. El usuario sobrescribe el valor del flujo de entrada del nuevo extractor. 3. El compilador del sistema comprueba que los datos introducidos son del tipo correcto. 4. El sistema permite utilizar el nuevo extractor en conjunto con el resto de código del programa.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Datos suministrados incorrectos:** - En el paso 3, el usuario no introduce todos los datos solicitados por el sistema o estos son del tipo incorrecto. - El compilador del sistema señala al usuario los valores faltantes o erróneos, volviendo al paso 2.
EXCEPCIONES	-
NOTAS	-

Tabla 47. Análisis caso de uso "Implementar nuevos flujos de entrada" (Usuario Biblioteca)

4.5.1.6 *Consultar documentación de la biblioteca*

CONSULTAR DOCUMENTACIÓN DE LA BIBLIOTECA	
PRECONDICIONES	- El usuario cuenta con un proyecto de desarrollo software - La biblioteca de validación de flujos está instalada como dependencia del proyecto.
POSCONDICIONES	- El sistema muestra la documentación de la biblioteca de validación de flujos al usuario.
ACTOR PRINCIPAL	- Usuario biblioteca.
DESCRIPCIÓN	1. El usuario navega hasta un fragmento de código del que quiere obtener información de la documentación. 2. El usuario elige ver más información del fragmento de código. 3. El sistema muestra al usuario la documentación de la biblioteca para la sección seleccionada.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Documentación en línea - Scaladoc:** - Previamente al paso 1, el usuario navega al sitio web del proyecto. - El usuario accede a la sección con la documentación del código del proyecto. - Proseguir desde el paso 1. **Documentación en línea - Manuales:**

	- Previamente al paso 1, el usuario navega al sitio web del proyecto. - El usuario accede a la sección con los manuales del módulo. - Proseguir desde el paso 1.
EXCEPCIONES	-
NOTAS	- Se contempla la posibilidad de visitar la documentación tanto del código como los manuales, así como tanto en local como en línea donde corresponda.

Tabla 48. Análisis caso de uso "Consultar documentación de la biblioteca" (Usuario Biblioteca)

4.5.2 Usuario API

Se desglosan los escenarios de los casos de uso especificados en *4.2.3.2-Usuario API*.

4.5.2.1 *Usuario Backend*

4.5.2.1.1 Configurar la validación de los ítems del flujo

CONFIGURAR LA VALIDACIÓN DE LOS ÍTEMS DEL FLUJO	
PRECONDICIONES	- El usuario cuenta con acceso al API de *RDFShape*.
POSCONDICIONES	- El sistema está preparado para invocar el API con una configuración del flujo de cadenas de texto provenientes de *Apache Kafka*. - El sistema está preparado para invocar el API con una configuración del validador.
ACTOR PRINCIPAL	- Usuario backend.
DESCRIPCIÓN	1. El usuario establece una conexión con el servidor *WebSockets* de *RDFShape*. 2. El sistema solicita la siguiente información: a. Fuente del flujo de datos *Apache Kafka*: i. Servidor (URL o dominio). ii. Puerto. iii. *Topic*. iv. Identificador del grupo de consumidores. b. Configuración base del extractor del flujo: i. Formato esperado en el RDF recibido. ii. Inferencia aplicable al RDF recibido. c. Configuración base del validador del flujo: i. Esquema *Shape Expressions* a emplear en la validación. ii. *ShapeMap* a emplear en la validación. iii. Estrategia que aplicar antes errores en la validación.

	iv. Estrategia que aplicar ante ítems que no superen la validación. 3. El usuario introduce los datos solicitados. 4. El sistema comprueba que la configuración está en formato JSON. 5. El sistema verifica que los datos de la configuración son válidos. 6. El sistema genera un nuevo validador de flujos en base a la configuración recibida.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Datos suministrados insuficientes:** - En el paso 3, el usuario no introduce todos los datos solicitados por el sistema. - En el paso 5, el sistema comprueba la existencia de valores por defecto para los datos no solicitados. o Si existen alternativas predeterminadas para los valores faltantes, proseguir desde el paso 5. o Si los valores faltantes son imprescindibles, el sistema responde con un mensaje *WebSockets* conteniendo información del error. ▪ El sistema cierra la conexión *WebSockets* incluyendo un código de error específico. ▪ Retroceder al paso 1. **Datos suministrados inválidos:** - En el paso 3, el usuario introduce datos inválidos o de tipo incorrecto. - En el paso 5, el sistema descubre los datos incorrectos. - El sistema cierra la conexión *WebSockets* incluyendo un código de error específico. - Retroceder al paso 1. **Datos suministrados con sintaxis inválida:** - En el paso 3, el usuario introduce datos que no corresponden a la sintaxis JSON. - En el paso 4, el sistema es incapaz de procesar el mensaje del usuario como JSON. - El sistema cierra la conexión *WebSockets* incluyendo un código de error específico. - Retroceder al paso 1.
EXCEPCIONES	**Servidor API no disponible:** - No se puede contactar con el API de *RDFShape*. - El cliente que esté usando el usuario mostrará un mensaje de error según corresponda.
NOTAS	-

Tabla 49. Análisis caso de uso "Configurar la validación de los ítems del flujo" (Usuario API - Backend)

4.5.2.1.2 Invocar una validación de flujos

INVOCAR UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario cuenta con una conexión *WebSockets* activa con el servidor de *RDFShape*. - El usuario ha enviado un mensaje al servidor con la configuración de la validación.
POSCONDICIONES	- El sistema ha conectado con el origen del flujo y comienza a emitir un flujo de resultados. - El sistema valida ítems hasta que dejan de llegar o se cumplen las condiciones de parada del validador.
ACTOR PRINCIPAL	- Usuario backend.
DESCRIPCIÓN	1. El sistema conecta con el origen del flujo de datos. 2. El sistema comienza a consumir ítems del flujo de datos. Para cada ítem: a. El sistema emite el resultado de la validación a través de un mensaje *WebSockets* destinado al usuario.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Error durante la validación:** - En el paso 2, la biblioteca de validación emite un error inesperado, estando el sistema configurado para detenerse en caso de errores. - El sistema envía al usuario un mensaje *WebSockets* con información del error original. - El sistema cierra la conexión *WebSockets* incluyendo un código de error específico, concluyendo la validación. **Ítem inválido durante la validación:** - En el paso 2, la biblioteca de validación emite un error a causa de un ítem del flujo que no supera el proceso de validación, estando el sistema configurado para detenerse en caso de ítems inválidos. - El sistema envía al usuario un mensaje *WebSockets* con el resultado de la validación problemático. - El sistema cierra la conexión *WebSockets* incluyendo un código de error específico, concluyendo la validación. **Inactividad del flujo:** - En el paso 2, la biblioteca de validación emite un error por haberse superado el período de inactividad máximo del flujo de entrada.

	- El sistema cierra la conexión *WebSockets* incluyendo un código de error específico, concluyendo la validación.
EXCEPCIONES	**Flujo de entrada no disponible:** - En el paso 2, la biblioteca de validación emite un error por no ser capaz de establecer conexión con el flujo de datos especificado. - El sistema cierra la conexión *WebSockets* incluyendo un código de error específico.
NOTAS	- Antes de cerrar una conexión a causa de un fallo, el sistema enviará un último mensaje con información del error si es posible.

Tabla 50. Análisis caso de uso "Invocar una validación de flujos" (Usuario API - Backend)

4.5.2.1.3 Detener una validación de flujos

DETENER UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario cuenta con una validación de flujos activa en el servidor de *RDFShape*.
POSCONDICIONES	- El sistema se ha desconectado de flujo de datos de entrada y dejado de consumir ítems, deteniendo de forma efectiva la validación y los recursos dedicados a esta.
ACTOR PRINCIPAL	- Usuario backend.
DESCRIPCIÓN	2. El usuario cierra la conexión con el servidor. 3. El sistema finaliza la validación. 4. El sistema emite un último mensaje *WebSockets* de cierre con un código de éxito (1000).
VARIACIONES (ESCENARIOS SECUNDARIOS)	-
EXCEPCIONES	-
NOTAS	-

Tabla 51. Análisis caso de uso "Detener una validación de flujos" (Usuario API - Backend)

4.5.2.1.4 Consultar documentación del API

CONSULTAR DOCUMENTACIÓN DEL API	
PRECONDICIONES	-
POSCONDICIONES	- El sistema muestra la documentación del API al usuario.
ACTOR PRINCIPAL	- Usuario backend.
DESCRIPCIÓN	1. El usuario navega a la documentación en *SwaggerHub* del proyecto.

	2. El usuario accede a la sección con la documentación del API del proyecto. 3. El usuario navega hasta un fragmento del API del que quiere obtener información adicional. 4. El sistema muestra al usuario la documentación del fragmento seleccionado.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Documentación en línea - Scaladoc:** - En el paso 1, el usuario navega al sitio web del proyecto. - El usuario accede a la sección con la documentación del código del proyecto. - Proseguir desde el paso 3. **Documentación en línea - Manuales:** - En el paso 1, el usuario navega al sitio web del proyecto. - El usuario accede a la sección con los manuales del módulo. - Proseguir desde el paso 3.
EXCEPCIONES	-
NOTAS	Se contempla principalmente la posibilidad de visitar en línea la documentación del API en *SwaggerHub*, seguida del *Scaladoc* y manuales del módulo.

Tabla 52. Análisis caso de uso "Consultar documentación del API" (Usuario API - Backend)

4.5.2.2 *Usuario Cliente*

4.5.2.2.1 Pausar una validación de flujos

PAUSAR UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario cuenta con una validación de flujos activa en el servidor de *RDFShape*. - El usuario ha iniciado la validación desde el cliente de *RDFShape*.
POSCONDICIONES	- El sistema se ha desconectado de flujo de datos de entrada y dejado de consumir ítems, deteniendo de forma efectiva la validación y los recursos dedicados a esta. - El sistema ha conservado la configuración de la validación de cara a una posible reanudación.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario selecciona la opción de pausar la validación. 2. El sistema almacena la última configuración de validación utilizada para invocar el API. 3. El sistema preserva los últimos resultados recibidos de la validación.

	4. El sistema cierra la conexión *WebSockets* con el API.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Validación no en curso:** - El usuario del cliente no tiene una validación en curso. - El sistema oculta y desactiva la opción de pausar la validación.
EXCEPCIONES	-
NOTAS	- De cara al API, la pausa de una validación desde el cliente es similar a la detención de una validación (ver *4.5.2.1.3-Detener una validación de flujos*), con el añadido de que el cliente web puede recordar el estado de la última validación para invocarla de nuevo, simulando un sistema de pausa-continuación.

Tabla 53. Análisis caso de uso "Pausar una validación de flujos" (Usuario API - Cliente)

4.5.2.2.2 Reanudar una validación de flujos

REANUDAR UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario no cuenta con una validación de flujos activa en el servidor de *RDFShape*. - El usuario ha pausado una validación desde el cliente de *RDFShape*.
POSCONDICIONES	- El sistema se ha reconectado al flujo de datos de entrada y empezado a consumir ítems.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario selecciona la opción de reanudar la validación. 2. El sistema abre una conexión *WebSockets* con el API. 3. El sistema envía un mensaje *WebSockets* al API con la configuración de la validación previamente pausada. 4. El sistema comienza a recibir mensajes con los resultados de la validación, para cada mensaje: a. Formatea el contenido. b. Lo añade al listado de resultados recibidos.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Validación en curso:** - El usuario del cliente tiene una validación en curso. - El sistema oculta y desactiva la opción de reanudar la validación. **Error de servidor:** - A lo largo del paso 4, el servidor genera un error en el proceso de validación. - El sistema recibe el mensaje *WebSockets* con el error y muestra la información formateada al usuario.
EXCEPCIONES	-

NOTAS	- De cara al API, reanudar una validación desde el cliente es similar a iniciar una validación nueva (ver *4.5.2.1.2-Invocar una validación de flujos*), con el añadido de que el cliente web puede recordar el estado de la última validación para invocarla de nuevo, simulando un sistema de pausa-continuación.

Tabla 54. Análisis caso de uso "Reanudar una validación de flujos" (Usuario API - Cliente)

4.5.2.2.3 Ordenar resultados de una validación de flujos

ORDENAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario cuenta con los resultados de una validación previa o actualmente activa.
POSCONDICIONES	- El sistema muestra los resultados de la validación que tiene almacenados en función del criterio seleccionado por el usuario.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario selecciona uno de los campos representados en el listado de resultados de la validación. 2. El sistema ordena los resultados que posee en memoria y los que llegan siguiendo un criterio alfabético ascendente en torno al campo seleccionado por el usuario.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Resultados ya ordenados – Otro criterio:** - Inicialmente, se da la situación de que los resultados ya están ordenados en base a otro campo distinto al seleccionado por el usuario. - El sistema ejecutará el paso 2 con normalidad: reordenará los resultados en base al nuevo criterio, ignorando el orden anterior. **Resultados ya ordenados – Mismo criterio:** - Inicialmente, se da la situación de que los resultados ya están ordenados en base al campo seleccionado por el usuario. - En el paso 2, el sistema reordenará los resultados en orden descendente en caso de estar ordenados de manera ascendente, y viceversa.
EXCEPCIONES	-
NOTAS	-

Tabla 55. Análisis caso de uso "Ordenar resultados de una validación de flujos" (Usuario API - Cliente)

4.5.2.2.4 Descartar resultados de una validación de flujos

DESCARTAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS

PRECONDICIONES	- El usuario cuenta con los resultados de una validación previa o actualmente activa.
POSCONDICIONES	- El sistema muestra el listado de resultados de la validación vacío y elimina los viejos resultados de memoria.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario selecciona la opción de descartar resultados. 2. El sistema retira de la interfaz todos los resultados presentes en el listado de resultados previos. 3. El sistema elimina de memoria todos los resultados previos.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**No hay resultados previos:** - Inicialmente, se da la situación de que no hay ningún resultado almacenado en el sistema ni mostrándose en la interfaz. - El sistema oculta e inhabilita la opción de descartar resultados.
EXCEPCIONES	-
NOTAS	-

Tabla 56. Análisis caso de uso "Descartar resultados de una validación de flujos" (Usuario API - Cliente)

4.5.2.2.5 Buscar resultados de una validación de flujos

BUSCAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario cuenta con los resultados de una validación previa o actualmente activa.
POSCONDICIONES	- El sistema muestra sólo los resultados de la búsqueda en el listado de resultados de la validación.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario selecciona la opción de buscar resultados. 2. El usuario introduce el texto que desea buscar entre los resultados. 3. El sistema filtra los resultados que contengan el texto introducido en uno de los campos: a. Identificador del nodo validado. b. Identificar de la *Shape* contra la que se ha validado. c. Informe detallado de la validación. 4. El sistema retira de la interfaz todos los resultados que no sean resultados de la búsqueda. 5. El sistema mantiene en memoria todos los resultados previos.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Búsqueda vacía:** - En el paso 2, el usuario no introduce ningún texto como filtro para la búsqueda.

	- El sistema se comporta como si no hubiera ningún filtro de búsqueda y muestra todos los resultados en la interfaz.
EXCEPCIONES	-
NOTAS	-

Tabla 57. Análisis caso de uso "Buscar resultados de una validación de flujos" (Usuario API - Cliente)

4.5.2.2.6 Exportar resultados de una validación de flujos

EXPORTAR RESULTADOS DE UNA VALIDACIÓN DE FLUJOS	
PRECONDICIONES	- El usuario cuenta con los resultados de una validación previa o actualmente activa.
POSCONDICIONES	- El usuario puede descargar un fichero CSV con los resultados de la validación.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario selecciona la opción de exportar resultados. 2. El sistema genera un fichero CSV usando el carácter ";" como separador de columnas. El fichero contiene todos los resultados de la validación presentes en el sistema en el momento. Cada columna del fichero estará asociada a cada uno de los campos de los resultados de la validación. 3. El sistema permite al usuario descargar el fichero generado.
VARIACIONES (ESCENARIOS SECUNDARIOS)	-
EXCEPCIONES	-
NOTAS	-

Tabla 58. Análisis caso de uso "Exportar resultados de una validación de flujos" (Usuario API - Cliente)

4.5.2.2.7 Generar permalinks

GENERAR PERMALINKS	
PRECONDICIONES	- El usuario cuenta con una validación de flujos activa o ya detenida en el servidor de *RDFShape*.
POSCONDICIONES	- El usuario posee un enlace único para volver a acceder al cliente y ejecutar la misma validación.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario selecciona la opción de generar un enlace a la validación. 2. El sistema abre una petición HTTP al API solicitando un enlace con los parámetros de la validación actual.

	3. El API escribe una nueva entrada en la base de datos conteniendo el identificador del enlace, los parámetros de la validación y la fecha de creación/uso. 4. El sistema recibe del API una respuesta con el identificador único del nuevo enlace generado. 5. El sistema genera el enlace final uniendo el dominio de despliegue del cliente con la ruta de recuperación de enlaces y el identificador único del enlace. 6. El sistema deposita el enlace final en el portapapeles del usuario.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Enlace para la validación ya existente:** - En el paso 3, el API detecta que ya existe un enlace para los parámetros de validación solicitados. - En el paso 4, el cliente recibe del API el enlace ya existente, de manera que no se crean nuevas entradas en base de datos. - Proseguir desde el paso 5. **Error en el API:** - A lo largo del paso 3, el API no está disponible o es incapaz de contactar con la base de datos. - En el paso 4, el cliente recibe una respuesta errónea. - En el paso 5, el cliente genera el enlace a partir de la ruta existente en la barra de direcciones del navegador del usuario. - Proseguir desde el paso 6.
EXCEPCIONES	**Error en el API:** - A lo largo del paso 3, el API no está disponible o es incapaz de contactar con la base de datos. - En el paso 4, el cliente recibe una respuesta errónea. - En el paso 5, el cliente genera el enlace a partir de la ruta existente en la barra de direcciones del navegador del usuario. - Proseguir desde el paso 6.
NOTAS	-

Tabla 59. Análisis caso de uso "Generar permalinks" (Usuario API - Cliente)

4.5.2.2.8 Consumir permalinks

CONSUMIR PERMALINKS	
PRECONDICIONES	- El usuario cuenta con un enlace permanente generado previamente desde el cliente.
POSCONDICIONES	- El usuario ejecuta la validación que almacenó previamente sin introducir sus parámetros manualmente.
ACTOR PRINCIPAL	- Usuario cliente.

DESCRIPCIÓN	1. El usuario introduce el enlace en la barra de direcciones del navegador. 2. El sistema abre una petición HTTP al API solicitando los datos de la validación asociada con el identificador del enlace recibido. 3. El API lee la entrada correspondiente en la base de datos y actualiza la fecha de creación/uso del enlace. 4. El sistema recibe del API una respuesta con el enlace original de la validación. 5. El sistema redirige al cliente a la operación de validación original. 6. El sistema recupera del enlace los parámetros de la validación y rellena la interfaz de usuario con ellos. 7. El sistema arranca la validación.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Enlace inexistente:** - En el paso 3, el API no encuentra ninguna entrada en la base de datos asociada al identificador de enlace proporcionado. - En el paso 4, el cliente recibe del API un error. - El sistema muestra una pantalla de error al usuario informando de que el enlace no existe.
EXCEPCIONES	**Error en el API:** - A lo largo del paso 3, el API no está disponible o es incapaz de contactar con la base de datos. - El sistema muestra una pantalla de error al usuario informando de que no se pudo obtener el enlace.
NOTAS	-

Tabla 60. Análisis caso de uso "Consumir permalinks" (Usuario API - Cliente)

4.5.2.2.9 Consultar la ayuda del cliente

CONSULTAR LA AYUDA DEL CLIENTE	
PRECONDICIONES	- El usuario tiene acceso al cliente de *RDFShape*.
POSCONDICIONES	- El sistema muestra información de uso de la herramienta al usuario.
ACTOR PRINCIPAL	- Usuario cliente.
DESCRIPCIÓN	1. El usuario navega a la página de validación de flujos del cliente. 2. El usuario interactúa con el menú de ayuda. 3. El sistema muestra información textual sobre las capacidades de la herramienta de validación.

VARIACIONES (ESCENARIOS SECUNDARIOS)	-
EXCEPCIONES	-
NOTAS	-

Tabla 61 . Análisis caso de uso "Consultar la ayuda del cliente" (Usuario API - Cliente)

4.5.3 Administrador del Sistema

Se desglosan los escenarios de los casos de uso especificados en *4.2.3.3-Administrador del Sistema*.

4.5.3.1 *Desplegar el sistema*

DESPLEGAR EL SISTEMA	
PRECONDICIONES	- El sistema no se encuentra desplegado. - El sistema tiene instalado *Docker* (*Compose*) y posee la configuración de despliegue para el proyecto.
POSCONDICIONES	- El sistema se encuentra desplegado en contenedores. - El sistema es accesible a través de internet de forma segura.
ACTOR PRINCIPAL	- Administrador del sistema
DESCRIPCIÓN	1. El administrador accede al sistema donde se quieren desplegar los módulos. 2. El administrador ejecuta *Docker Compose* usando el fichero de configuración del despliegue del proyecto. 3. El sistema inicializa los contenedores que operan los módulos del proyecto.
VARIACIONES (ESCENARIOS SECUNDARIOS)	-
EXCEPCIONES	**Error de despliegue:** - En el paso 3, alguno de los contenedores falla al inicializarse. - El sistema emite un mensaje con el error producido para informar al usuario.
NOTAS	- El despliegue automatizado pone en marcha varios contenedores aparte de los ligados a los módulos del proyecto. No se repiten aquí por evitar la redundancia, pues están especificados en *2.2.3.2.1.1-Arquitectura final del despliegue*.

Tabla 62. Análisis caso de uso "Desplegar el sistema" (Administrador del Sistema)

4.5.3.2 *Monitorizar el sistema*

MONITORIZAR EL SISTEMA	
PRECONDICIONES	- El sistema se encuentra desplegado.
POSCONDICIONES	- El administrador puede ver el estado de cada contenedor del despliegue.
ACTOR PRINCIPAL	- Administrador del sistema
DESCRIPCIÓN	1. El administrador accede al sistema donde se encuentran desplegados los módulos. 2. El administrador accede al listado de procesos de *Docker*. 3. El sistema muestra todos los contenedores en ejecución, distinguiendo entre saludables y no saludables en función de si superan los chequeos de estado. El sistema inicializa los contenedores que operan los módulos del proyecto.
VARIACIONES (ESCENARIOS SECUNDARIOS)	-
EXCEPCIONES	-
NOTAS	-

Tabla 63. Análisis caso de uso "Monitorizar el sistema" (Administrador del Sistema)

4.5.3.3 *Reiniciar el sistema*

REINICIAR EL SISTEMA	
PRECONDICIONES	- El sistema se encuentra desplegado.
POSCONDICIONES	- Los módulos del sistema que han fallado han sido reiniciados y funcionan correctamente.
ACTOR PRINCIPAL	- Administrador del sistema
DESCRIPCIÓN	1. El administrador del sistema ejecuta *Docker Compose* para detener todos los contenedores que conforman el sistema final. 2. El sistema detiene todos los contenedores que operan los módulos del proyecto. 3. El administrador del sistema ejecuta *Docker Compose* para lanzar todos los contenedores que conforman el sistema final. 4. El sistema inicia todos los contenedores que operan los módulos del proyecto.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Reinicio automatizado:** 1. El administrador del sistema configura el reinicio automático de contenedores, incluyendo:

	a. Contenedores afectados. b. Intervalo de tiempo entre comprobaciones de estado del sistema. 2. El sistema comprueba el estado de salud de los contenedores en ejecución. 3. El sistema reinicia los contenedores cuyo estado sea no saludable. 4. El sistema programa la siguiente comprobación de estado de los contenedores.
EXCEPCIONES	**Reinicio automatizado - Parámetros erróneos:** - En el escenario alternativo donde se automatiza el reinicio, el administrador introduce un valor inválido para el período de tiempo entre comprobaciones. - El sistema es incapaz de monitorizar el estado del despliegue.
NOTAS	-

Tabla 64. Análisis caso de uso "Reiniciar el sistema" (Administrador del Sistema)

4.5.3.4 *Configurar el sistema*

CONFIGURAR EL SISTEMA	
PRECONDICIONES	- El sistema posee la configuración de despliegue para el proyecto.
POSCONDICIONES	- El sistema se encuentra configurado para futuros despliegues.
ACTOR PRINCIPAL	- Administrador del sistema
DESCRIPCIÓN	1. El administrador accede al sistema donde se quieren desplegar los módulos. 2. El administrador del sistema modifica el fichero de configuración de *Docker Compose*. 3. El sistema almacena los cambios en la configuración para ejecutarlos en los siguientes despliegues.
VARIACIONES (ESCENARIOS SECUNDARIOS)	-
EXCEPCIONES	-
NOTAS	- El administrador es capaz de modificar sólo algunos parámetros de los sistemas desplegados, que no se citan por redundancia pues están descritos en *Tabla 25. Especificación caso de uso "Configurar el sistema" (Administrador del Sistema)*.

Tabla 65. Análisis caso de uso "Configurar el sistema" (Administrador del Sistema)

4.5.4 Servidor Remoto

Se desglosan los escenarios de los casos de uso especificados en *4.2.3.4-Servidor Remoto*.

4.5.4.1 *Autenticar consumidor*

AUTENTICAR CONSUMIDOR	
PRECONDICIONES	- El sistema no está consumiendo el flujo emitido por el servidor.
POSCONDICIONES	- El sistema conecta con éxito con el servidor emisor del flujo.
ACTOR PRINCIPAL	- Servidor remoto
DESCRIPCIÓN	1. El sistema inicia un intento de conexión contra el servidor remoto. 2. El servidor remoto recibe la conexión del sistema. 3. El servidor remoto acepta la conexión del sistema y le permite consumir el flujo de datos que emite.
VARIACIONES (ESCENARIOS SECUNDARIOS)	**Sistema no autorizado:** - El servidor remoto no permite al sistema conectar con el flujo de datos. - El sistema notificará el error.
EXCEPCIONES	**Servidor remoto no disponible:** - El sistema es incapaz de contactar con el servidor remoto. - El sistema notificará el error.
NOTAS	

Tabla 66. Análisis caso de uso "Autenticar consumidor" (Servidor Remoto)

4.5.4.2 *Enviar flujo de datos*

ENVIAR FLUJO DE DATOS	
PRECONDICIONES	- El servidor remoto es visible al sistema. - El flujo emitido por servidor remoto es accesible al sistema
POSCONDICIONES	- El sistema puede leer los datos emitidos desde el servidor remoto.
ACTOR PRINCIPAL	- Servidor remoto.
DESCRIPCIÓN	1. El servidor remoto emite un flujo de datos a través de la red. 2. El sistema recibe los datos emitidos por el servidor remoto para su procesamiento posterior.
VARIACIONES (ESCENARIOS SECUNDARIOS)	-
EXCEPCIONES	-
NOTAS	-

Tabla 67. Análisis caso de uso "Enviar flujo de datos" (Servidor Remoto)

4.6 ASI 5: ANÁLISIS DE CLASES

En esta sección se adjuntan los diagramas de clases propuestos en la fase de análisis del sistema. Cabe destacar el que sistema en su conjunto consta de tres módulos de naturalezas muy diferentes, por lo que el análisis de clases se dividirá por módulos.

Finalmente, resaltar el hecho de que al ser un diagrama de análisis se realizará a alto nivel y con nombres no definitivos para atributos y métodos.

4.6.1 Módulo Validador (Comet)

4.6.1.1 *Diagrama de Clases*

El diagrama de clases de la biblioteca de validación se elabora con total libertad dado que es el único módulo del proyecto cuyo desarrollo parte desde cero.

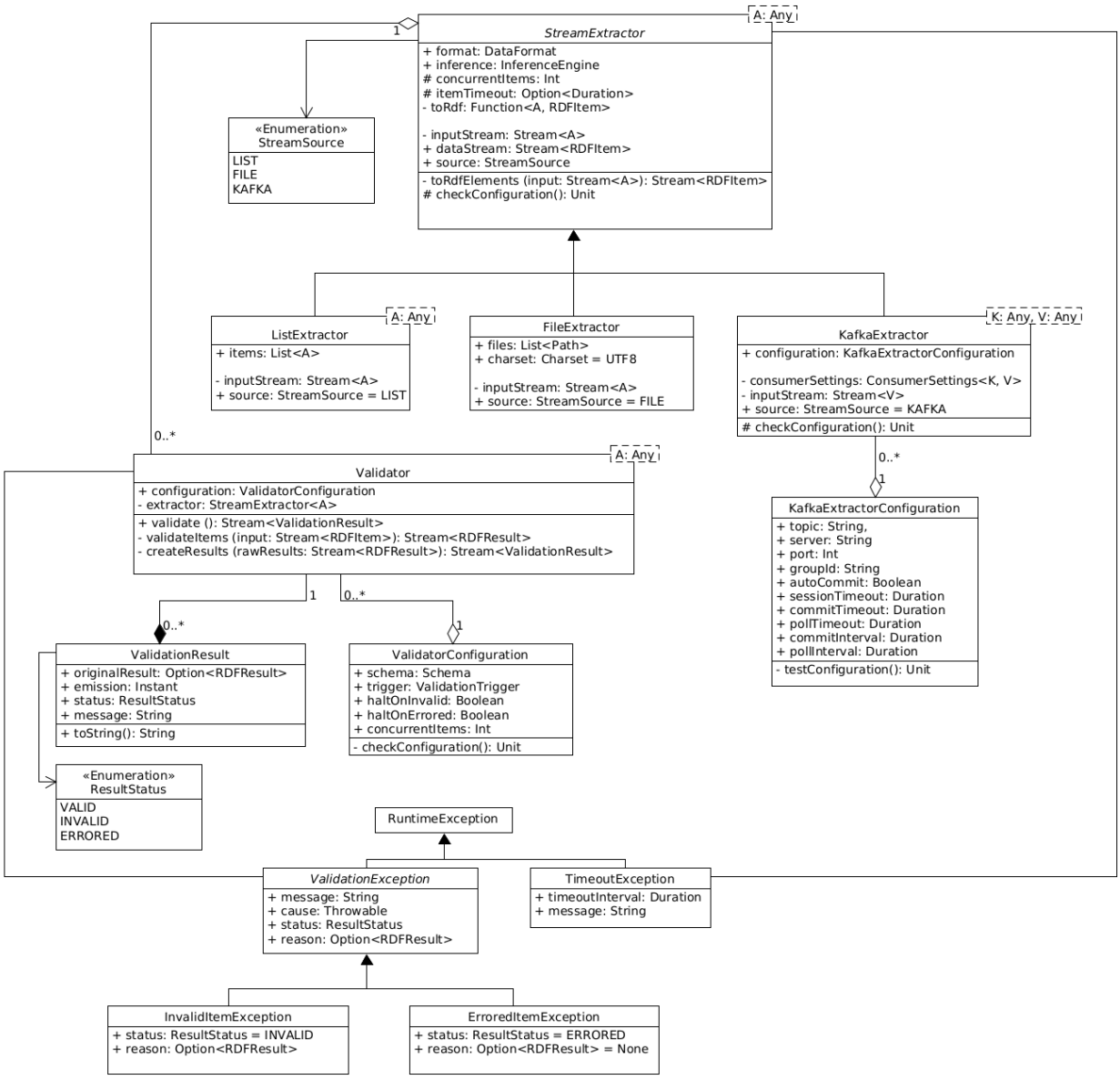


Ilustración 43. Diagrama de clases de análisis - Comet

4.6.1.2 *Descripción de Clases*

Nombre de la clase
Validator
Descripción
El validador recibe un flujo de elementos RDF y los procesa, emitiendo un flujo de resultados. Es la clase principal del módulo, cuya funcionalidad es expuesta públicamente por la biblioteca.
Responsabilidades
- Guarda la referencia a un extractor, el cual le irá suministrando elementos RDF para su validación final. - Contiene en su configuración el esquema contra el que se van a validar todos los ítems que entren al validador, además del comportamiento del sistema ante ítems errados o inválidos. - Transforma los elementos que le llegan en el resultado de validarlos contra el esquema y condiciones configuradas, de forma que el flujo de salida pueda seguir siendo transformado o consumido por terceras partes.
Atributos Propuestos
configuration: Representa la configuración del validador. **extractor**: Representa el extractor que suministrará nuevos elementos al validador según estén disponibles.
Métodos Propuestos
validate: Comienza a consumir ítems del extractor, devolviendo el flujo de salida de resultados. **validateItems**: Dado un flujo de entrada de datos RDF, transforma cada elemento para dar lugar a un flujo de resultados de *SHaclEX*. **createResults**: Dado un flujo de entrada de resultados de *SHaclEX*, mapea cada elemento a su resultado equivalente dentro de la biblioteca.

Nombre de la clase
ValidatorConfiguration
Descripción
Configuración asociada al funcionamiento de un validador.
Responsabilidades
- Especifica contra qué *Shape Expression* y *ShapeMap* debe operar un validador. - Especifica el comportamiento del validador ante excepciones. - Especifica el nivel de paralelismo del validador.
Atributos Propuestos
schema: Esquema *ShEx* contra el que se comprobarán los ítems que procese el validador asociado. **trigger**: *ShapeMap* contra el que se comprobarán los ítems que procese el validador asociado. **haltOnInvalid**: Indica si el validador asociado debe detenerse en caso de que un ítem no supere la validación.

haltOnErrored: Indica si el validador asociado debe detenerse en caso de que ocurra un error inesperado en el procesamiento de un ítem. **concurrentItems**: Número máximo de elementos que el validador intentará validar en paralelo.
Métodos Propuestos
checkConfiguration: Comprueba, en el momento en que se instancia una nueva configuración, si esta contiene valores incorrectos y detiene la ejecución de ser el caso.

Nombre de la clase
ValidationResult
Descripción
Resultado emitido por una instancia de validador
Responsabilidades
- Contiene el resultado original emitido por SHaclEX para la validación de un ítem. - Contiene información adicional sobre el resultado emitido.
Atributos Propuestos
originalResult: Opcionalmente, contiene el resultado de la validación del elemento. Puede no contener ningún valor en caso de que se hayan dado errores en el proceso de validación. **emission**: Instante temporal de emisión del resultado. **status**: Indica el estado del resultado, de entre los posibles. **message**: Ofrece información resumida sobre el estado del resultado en formato textual.
Métodos Propuestos
toString: Presenta el resultado en un formato textual inteligible para su lectura por humanos.

Nombre de la clase
ResultStatus
Descripción
Enumeración con los posibles valores que puede tener el estado de un resultado
Responsabilidades
-
Atributos Propuestos
VALID: El ítem en el resultado asociado se ha podido validar sin errores de ejecución y ha superado la validación. **INVALID**: El ítem en el resultado asociado se ha podido validar sin errores de ejecución y no ha superado la validación. **ERRORED**: El ítem en el resultado asociado no se ha podido validar completamente debido a errores de ejecución en el proceso.
Métodos Propuestos
-

Nombre de la clase
StreamExtractor
Descripción
Clase abstracta base para el resto de los extractores encargados de consumir elementos de un flujo de datos e interpretarlos como RDF. Es la segunda clase principal del módulo, cuya funcionalidad es base para el procesamiento de flujos y está pensada para ser extendida.
Responsabilidades
- Establecer una plantilla para el resto de las implementaciones de extractores, de forma que sólo tengan que implementar el proceso de lectura del flujo y el resto les venga dado de serie. - Contactar con un flujo de datos y leer sus contenidos. - Interpretar los contenidos del flujo de datos objetivo y convertirlos a RDF, según proceda. - Establecer el límite de tiempo antes de abortar el procesamiento del flujo por inactividad. - Contiene en su configuración el esquema contra el que se van a validar todos los ítems que entren al validador, además del comportamiento del sistema ante ítems errados o inválidos.
Atributos Propuestos
A: Argumento genérico aplicado a toda la clase. Especifica el tipo de ítems que el extractor espera que lleguen a través del flujo de entrada. **format**: Formato RDF en que el extractor asume que llegarán los elementos del flujo. **inference**: Inferencia que el extractor aplicará a los elementos que llegan del flujo. **concurrentItems**: Número máximo de elementos que el extractor tratará de leer e interpretar como RDF en paralelo. **itemTimeout**: Período de tiempo sin recibir elementos tras el cual el extractor abortará el consumo del flujo y emitirá un error. **toRdf**: Función empleada por el extractor para convertir los ítems entrantes del flujo, de tipo **A**, en datos RDF. **source**: Identificador del tipo de flujo procesado por el extractor. **inputStream**: Flujo de datos con los elementos tal cual llegan del flujo antes de ser procesados. **dataStream**: Flujo de datos con los elementos del flujo interpretados como RDF en base a **toRdf**.
Métodos Propuestos
toRdfElements: Dado un flujo de datos de tipo **A**, lo transforma en un flujo de datos RDF en base a **toRdf**. **checkConfiguration**: Comprueba, en el momento en que se instancia una nueva configuración, si esta contiene valores incorrectos y detiene la ejecución de ser el caso.

Nombre de la clase
StreamSource
Descripción
Enumeración con los posibles tipos de flujo manejables por la aplicación
Responsabilidades

-
Atributos Propuestos
LIST: El flujo de datos de entrada proviene de un listado de objetos en memoria. **FILES**: El flujo de datos de entrada proviene de un listado de ficheros. **KAFKA**: El flujo de datos de entrada proviene de *Apache Kafka*.
Métodos Propuestos
-

Nombre de la clase
ListExtractor
Descripción
Implementación del extractor con la lógica necesaria para extraer elementos de un listado de objetos en memoria.
Responsabilidades
- Extraer los datos de una lista y adaptarlos al marco de manejo de flujos empleado en la biblioteca. - Mostrar un ejemplo claro y directo del funcionamiento de la biblioteca.
Atributos Propuestos
items: Listado de elementos de tipo **A** que el extractor interpretará como un flujo. **source**: Identificador del tipo de flujo procesado por el extractor. Toma el valor **LIST**. **inputStream**: Flujo de datos de entrada, extraído de **items**.
Métodos Propuestos
-

Nombre de la clase
FileExtractor
Descripción
Implementación del extractor con la lógica necesaria para extraer elementos de un listado de ficheros en disco o accesibles a través de red.
Responsabilidades
- Extraer los datos de un conjunto de ficheros de texto y adaptarlos al marco de manejo de flujos empleado en la biblioteca.
Atributos Propuestos
files: Listado de rutas a los ficheros que el extractor leerá como un flujo. **charset**: Juego de caracteres empleado en los ficheros, necesario para leerlos como texto. **source**: Identificador del tipo de flujo procesado por el extractor. Toma el valor **FILES**. **inputStream**: Flujo de datos de entrada, extraído de leer **files** como ficheros de texto codificado en **charset**.
Métodos Propuestos

-

Nombre de la clase
KafkaExtractor
Descripción
Implementación del extractor con la lógica necesaria para extraer elementos de un flujo de datos publicado desde *Apache Kafka*.
Responsabilidades
- Extraer los datos de un flujo de *Apache Kafka* y adaptarlos al marco de manejo de flujos empleado en la biblioteca. - Actuar como un consumidor de forma transparente al productor de datos de *Kafka*. - Mostrar un ejemplo de consumo de datos en remoto y tiempo real con la biblioteca.
Atributos Propuestos
K: Tipo que el extractor espera que tenga la clave de cada mensaje recibido de *Kafka*. **V**: Tipo que el extractor espera que tenga el contenido de cada mensaje recibido de *Kafka*. **configuration**: Objeto con los datos necesarios para construir **consumerSettings**. **consumerSettings**: Configuración de cómo debe comportarse el extractor al consumir datos de *Kafka*. **source**: Identificador del tipo de flujo procesado por el extractor. Toma el valor **KAFKA**. **inputStream**: Flujo de datos de entrada, extraído de consumir datos de un productor de *Kafka* en base a **consumerSettings** e insertar cada mensaje como un ítem del flujo. **checkConfiguration**: Comprueba la configuración de este extractor, complementando los tests definidos en la clase base con la comprobación de **configuration**.
Métodos Propuestos
-

Nombre de la clase
KafkaExtractorConfiguration
Descripción
Objeto contenedor de la configuración de un **KafkaExtractor** para operar contra *Apache Kafka*.
Responsabilidades
- Definir el comportamiento del extractor a la hora de conectar, desconectar y sincronizar la información con *Kafka*.
Atributos Propuestos
topic: *topic* de *Kafka* del que el extractor intentará consumir elementos. **server**: IP o dominio donde se aloja el productor de *Kafka* que el extractor debe consumir. **port**: Puerto, dentro de **server**, desde el que emite el productor de *Kafka*. **groupId**: Identificador bajo el cual el extractor debe consumir datos de *Kafka*.

autoCommit: Define si el extractor debe sincronizar automáticamente con *Kafka* qué mensajes ha consumido o no. **sessionTimeout**: Intervalo de tiempo para conectar o consumir un elemento tras el cual se considera que el productor no está disponible. **commitTimeout**: Intervalo de tiempo tras el que se considera que el proceso de sincronización con *Kafka* ha fallado. **pollTimeout**: Intervalo de tiempo máximo que se permite para traer los ítems del productor al extractor. **commitInterval**: Intervalo de tiempo máximo entre operaciones de sincronización con *Kafka*. **pollInterval**: Intervalo de tiempo máximo entre comprobaciones de si hay nuevos ítems disponibles en el servidor.
Métodos Propuestos
testConfiguration: Comprueba, en el momento en que se instancia una nueva configuración, si esta contiene valores incorrectos y detiene la ejecución de ser el caso.

Nombre de la clase
TimeoutException
Descripción
Excepción personalizada para el manejo de errores relacionados con la inactividad del flujo de entrada
Responsabilidades
- Proveer al sistema y sus usuarios de un error distinguible y con información de depuración para los casos en que se aborte una validación por inactividad del origen de los datos.
Atributos Propuestos
timeoutInterval: Intervalo de tiempo en el que el flujo ha fallado en emitir elementos, causando el error. **message**: Mensaje de error amistoso para la lectura por humanos.
Métodos Propuestos
-

Nombre de la clase
ValidationException
Descripción
Clase abstracta base para excepciones personalizadas para el manejo de errores en el transcurso de la validación de los datos del flujo de entrada
Responsabilidades
- Proveer al sistema y sus usuarios de un error distinguible y con información de depuración para los casos en que se aborte una validación por errores en el validador.
Atributos Propuestos

message: Mensaje de error amistoso para la lectura por humanos. **cause**: Excepción original que ha causado el error en la validación. **status**: Estado del resultado del elemento procesado por el validador y causante del error. **reason**: Resultados de la validación del elemento causante del fallo en el validador, si están disponibles.
Métodos Propuestos
-

Nombre de la clase
InvalidItemException
Descripción
Excepción personalizada para el manejo de errores en el transcurso de la validación en casos en que un ítem no supera la validación y el validador está configurado para detenerse.
Responsabilidades
- Proveer al sistema y sus usuarios de un error distinguible y con información de depuración para los casos en que se aborte una validación por ítem inválido.
Atributos Propuestos
status: Estado del resultado del elemento procesado por el validador y causante del error. Toma el valor **INVALID**. **reason**: Resultado de la validación del ítem problemático.
Métodos Propuestos
-

Nombre de la clase
ErroredItemException
Descripción
Excepción personalizada para el manejo de errores en el transcurso de la validación en casos en que un ítem provoca un error inesperado durante la validación y el validador está configurado para detenerse.
Responsabilidades
- Proveer al sistema y sus usuarios de un error distinguible y con información de depuración para los casos en que se aborte una validación por error de ejecución.
Atributos Propuestos
status: Estado del resultado del elemento procesado por el validador y causante del error. Toma el valor **ERRORED**. **reason**: Resultado de la validación del ítem problemático. Al no haberse completado la validación no contendrá contiene ningún valor.
Métodos Propuestos
-

4.6.2 Módulo API

4.6.2.1 *Diagrama de Clases*

Para el diagrama de clases del API, se omite gran parte de la infraestructura del sistema original para centrarse en las partes afectadas por el proyecto. Por tanto, el diagrama se focaliza en el nuevo servicio de *permalinks* y las funcionalidades adicionales del servicio de esquemas para validar flujos.

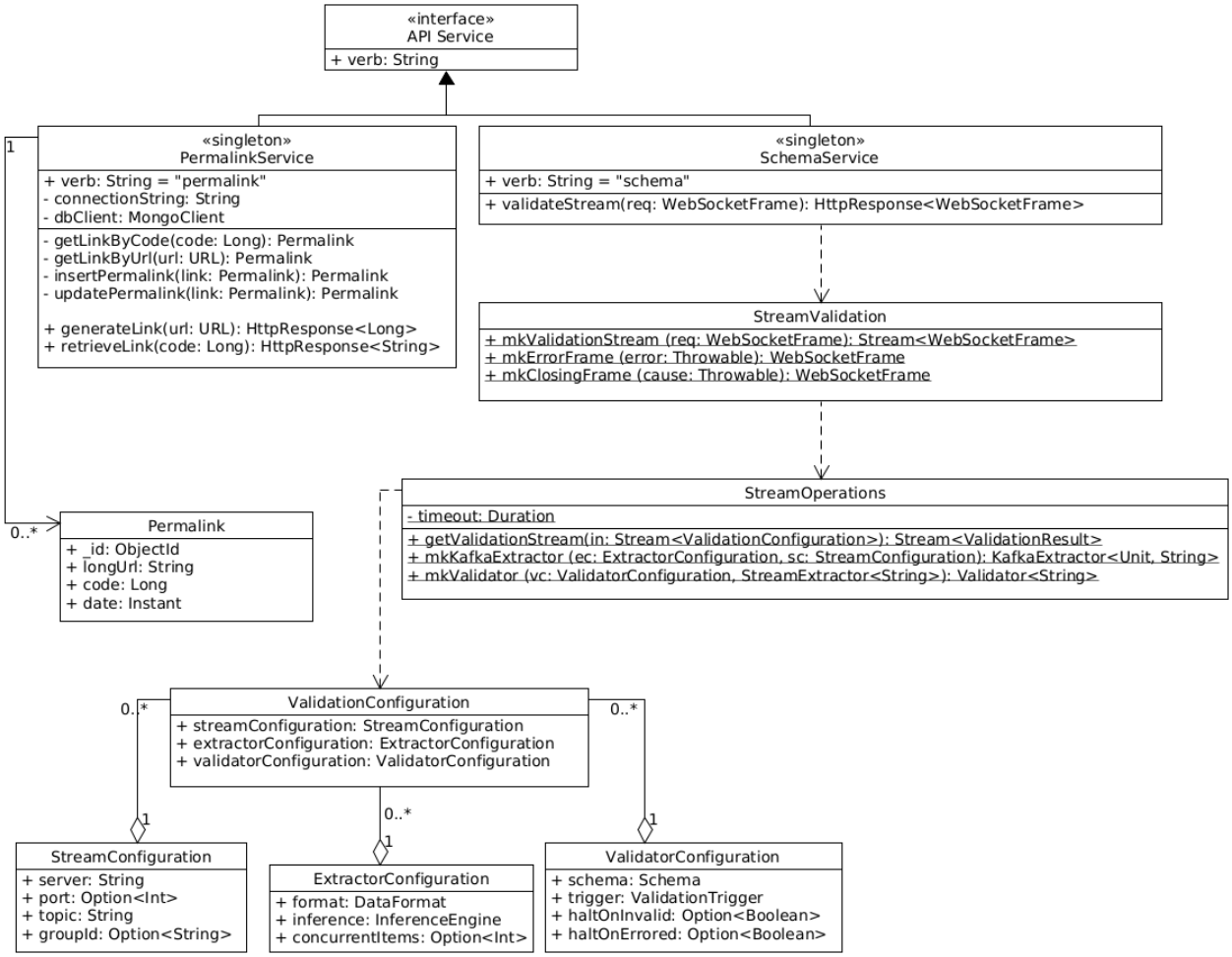


Ilustración 44. Diagrama de clases de análisis - API

4.6.2.2 *Descripción de Clases*

Nombre de la clase
API Service
Descripción
Interfaz común a todos los servicios que operan bajo el API
Responsabilidades
- Ofrece un marco de trabajo y propiedades comunes a todos los servicios definidos bajo el API.
Atributos Propuestos
verb: Verbo de acción con el que se invoca al servicio desplegado tras el API. En otras palabras, el *path* bajo el cual se ubican todas las rutas definidas por el servicio.
Métodos Propuestos
-

Nombre de la clase
PermalinkService
Descripción
Servicio del API que expone operaciones relacionadas con el manejo de *permalinks*
Responsabilidades
- Permitir a los clientes solicitar la creación de *permalinks*. - Permitir a los clientes recuperar la información de una validación dado un *permalink*.
Atributos Propuestos
verb: Verbo de acción con el que se invoca al servicio desplegado tras el API. Toma el valor **permalink**. **connectionString**: URL de conexión con la base de datos, formada en base a las credenciales y servidor de acceso. **dbClient**: Cliente para operar programáticamente con la base de datos de MongoDB, suministrado por los drivers oficiales.
Métodos Propuestos
getLinkByCode: Dado el identificador único de un *permalink*, recuperar de la base de datos el enlace correspondiente y transformarlo en un objeto del dominio. **getLinkByUrl**: Dada la URL a la que dirige un *permalink*, recuperar de la base de datos el enlace correspondiente y transformarlo en un objeto del dominio. **insertPermalink**: Dado un objeto del dominio de tipo **Permalink**, insertarlo en la base de datos. **updatePermalink**: Dado un objeto del dominio de tipo **Permalink**, actualizar su fecha de uso en la base de datos. **generateLink**: Método expuesto públicamente a los clientes. Dada una URL, crea y almacena un *permalink* a la misma y devuelve el identificador del enlace generado. **retrieveLink**: Método expuesto públicamente a los clientes. Dado el identificador de un *permalink*, lee de la base de datos el enlace correspondiente y responde con la URL originalmente asociada.

Nombre de la clase
Permalink
Descripción
Clase encargada de representar permalinks
Responsabilidades
- Permitir al sistema operar con *permalinks* como objetos del dominio. - Proveer mecanismos para transformar los objetos traídos de base de datos en objetos del dominio y viceversa.
Atributos Propuestos
_id: Identificador único del enlace dentro de la colección de MongoDB **[96]**. **longUrl**: URL original asociada al *permalink*. **code**: Código único de uso del *permalink*. **date**: Instante con el último uso del *permalink*, inicializado como la fecha de creación de este.
Métodos Propuestos
-

Nombre de la clase
SchemaService
Descripción
Servicio del API que expone operaciones relacionadas con la validación de RDF
Responsabilidades
- Permitir a los clientes solicitar la validación estándar de datos RDF. - Permitir a los clientes solicitar la validación de flujos de datos RDF de *Apache Kafka*.
Atributos Propuestos
verb: Verbo de acción con el que se invoca al servicio desplegado tras el API. Toma el valor **schema**.
Métodos Propuestos
validateStream: Método expuesto públicamente a los clientes. Recibe peticiones de validación de flujos a través de *WebSockets*. Responde con nuevos mensajes *WebSockets* con los resultados de la validación mientras esta transcurre.

Nombre de la clase
StreamValidation
Descripción
Utilidades estáticas empleadas por el servicio de validación para operar con flujos
Responsabilidades
- Procesar la petición de validación de flujos de los usuarios y emitir el flujo de respuesta de mensajes *WebSockets* que finalmente usará el servicio.

- Manejar los errores que surjan a lo largo de la validación y elaborar las respuestas adecuadas para cada uno.
Atributos Propuestos
-
Métodos Propuestos
mkValidationStream: Dado un mensaje *WebSockets* recibido desde el cliente con la solicitud de validación de flujos, procesar la solicitud para fabricar el flujo de resultados convertidos en mensajes *WebSockets* para ser respondidos. **mkErrorFrame**: Dado un error transcurrido durante la creación del flujo de validación (**mkValidationStream**), genera un mensaje *WebSockets* informando al cliente del error e incluyendo información de este. **mkClosingFrame**: Dado un error transcurrido durante la creación del flujo de validación (**mkValidationStream**), genera el mensaje *WebSockets* final que aborte la conexión con el cliente con un resumen del error y un código de estado específico.

Nombre de la clase
StreamOperations
Descripción
Utilidades estáticas encargadas de invocar la biblioteca de validación de flujos
Responsabilidades
- Extraer la configuración de la validación de los datos contenidos en la petición de validación del cliente. - Invocar la biblioteca de validación de flujos con los parámetros indicados por el usuario para generar el flujo de resultados sin adulterar.
Atributos Propuestos
timeout: Límite de tiempo impuesto en el servidor para esperar antes de abortar una validación de un flujo que no emite resultados.
Métodos Propuestos
getValidationStream: Dada la configuración extraída de la petición del cliente, invoca la biblioteca de validación de flujos y devuelve el flujo con los resultados tal y como se emiten, para que puedan ser procesados posteriormente. **mkKafkaExtractor**: Crea un objeto extractor de la biblioteca de validación de flujos ya configurado para leer un *stream* de *Apache Kafka* de acuerdo con los ajustes contenidos en la petición del cliente. genera un mensaje *WebSockets* informando al cliente del error e incluyendo información de este. **mkValidator**: Crea un objeto validador de la biblioteca de validación de flujos ya configurado de acuerdo con los ajustes contenidos en la petición del cliente.

Nombre de la clase
ValidationConfiguration
Descripción
Clase encargada de representar los ajustes aceptados por el servidor para una validación de flujos
Responsabilidades
- Contener los ajustes del validador especificados por el cliente. - Contener los ajustes del flujo de entrada *Kafka* especificados por el cliente. - Contener los ajustes del extractor especificados por el cliente.
Atributos Propuestos
streamConfiguration: Objeto con los ajustes relativos al flujo de *Apache Kafka* que se quiere consumir. **extractorConfiguration**: Objeto con los ajustes relativos al comportamiento del extractor de la biblioteca de validación que va a consumir los datos del flujo. **validatorConfiguration**: Objeto con los ajustes relativos al comportamiento y esquema empleados por el validador de la biblioteca de validación que va a operar los datos del flujo.
Métodos Propuestos
-

Nombre de la clase
StreamConfiguration
Descripción
Clase encargada de representar los ajustes aceptados por el servidor para configurar el origen del flujo de datos de entrada
Responsabilidades
- Contener los ajustes del flujo de entrada *Kafka* especificados por el cliente.
Atributos Propuestos
server: IP o dominio donde se emite el origen del flujo de datos. **port**: Opcionalmente, puerto dentro de **server** desde donde se emite el flujo. Por defecto toma el valor predeterminado de *Kafka*: **9092**. **topic**: *Topic* del flujo de *Kafka* donde se encuentran los datos que se quieren consumir. **groupId**: Opcionalmente, identificador bajo el cual se consumirán datos del productor de *Kafka*. Por defecto tomará un valor basado en el nombre de la aplicación.
Métodos Propuestos
-

Nombre de la clase
ExtractorConfiguration
Descripción

Clase encargada de representar los ajustes aceptados por el servidor para configurar el extractor de la biblioteca de validación que procesará el flujo de entrada
Responsabilidades
- Contener los ajustes del extractor especificados por el cliente.
Atributos Propuestos
format: Formato o sintaxis RDF en que el extractor espera que lleguen los elementos. **inference**: Inferencia que el extractor aplicará sobre los elementos. **concurrentItems**: Opcionalmente, el número de máximo elementos que se desea que el extractor procese en paralelo.
Métodos Propuestos
-

Nombre de la clase
ValidatorConfiguration
Descripción
Clase encargada de representar los ajustes aceptados por el servidor para configurar el validador de la biblioteca de validación que procesará el flujo de entrada
Responsabilidades
- Contener los ajustes del validador especificados por el cliente.
Atributos Propuestos
schema: Objeto de la biblioteca *SHaclEX* representando la *Shape Expression* contra la que el validador comprobará los ítems recibidos. **trigger**: Objeto de la biblioteca *SHaclEX* representando el *ShapeMap* contra el que el validador comprobará los ítems recibidos. **haltOnInvalid**: Opcionalmente, si el validador debe abortar la validación en caso de que un ítem no valide correctamente. **haltOnErrored**: Opcionalmente, si el validador debe abortar la validación en caso de que un ítem produzca un error inesperado al ser procesado para su validación.
Métodos Propuestos
-

4.6.3 Módulo Cliente

4.6.3.1 *Diagrama de Clases*

En el caso del cliente, se parte de un proyecto web basado en *React*. Al igual que con el API, se omiten los detalles del sistema original en aras de planificar las partes que va a extender este proyecto.

A diferencia de los otros módulos, en los que el diagrama de clases intenta emular la estructura de clases final del sistema, en este caso se planifica el diagrama en base a los componentes web que se van a crear y la información que van a contener.

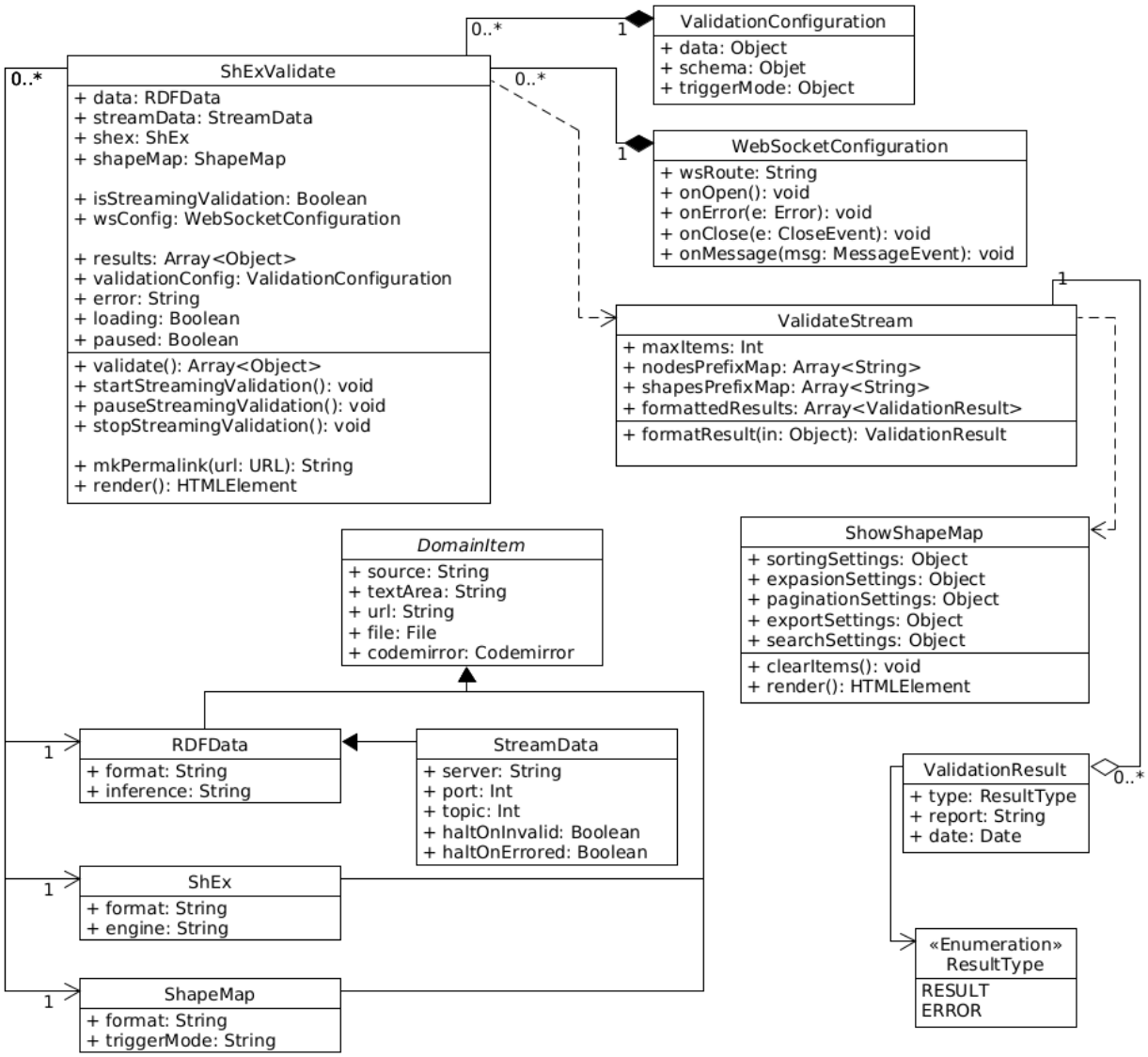


Ilustración 45. Diagrama de clases de análisis - Cliente

4.6.3.2 *Descripción de Clases*

Nombre de la clase
ShExValidate
Descripción
Componente principal de la interfaz de validación de datos RDF mediante Shape *Expressions*, tanto estándar como de flujos
Responsabilidades
- Contener los datos empleados en la validación. - Contener el estado de la validación. - Contener la configuración de la validación. - Contener los resultados de la validación. - Ofrecer mecanismos a los usuarios para introducir los datos de la validación, así como manejarla y cotejar los resultados desde la interfaz gráfica.
Atributos Propuestos
data: Datos RDF con los que opera el sistema en una validación estándar. **streamData**: Datos RDF con los que opera el sistema en una validación de flujos. **shex**: Esquema *Shape Expressions* con el que opera el sistema en la validación. **shapeMap**: *ShapeMap* con el que opera el sistema en la validación. **validationConfig**: Configuración de la validación que se envía al servidor en base a los demás atributos. **wsConfig**: Para validaciones de flujos, configuración de la conexión *WebSockets* con el servidor. **results**: Listado de resultados de la validación recibidos desde el servidor. **isStreamingValidation**: Indicador de si la validación es estándar o de flujos, empleado para operar la interfaz y la lógica del programa en función de si es verdadero o falso. **error**: Opcionalmente, el error que se ha producido en el proceso de validación. **loading**: Para cualquier validación, si el sistema se encuentra contactando con el servidor. Para a ser falso cuando se recibe una respuesta del servidor. **paused**: Para validaciones de flujos, verdadero si la validación se halla pausada y falso si se encuentra en curso.
Métodos Propuestos
validate: Para una validación estándar, invocar el API del servidor con la configuración provista y recibir los resultados. **startStreamingValidation**: Para una validación de flujos, enviar al servidor la petición de validación con la configuración provista mediante *WebSockets*. **pauseStreamingValidation**: Para una validación de flujos, pausar la recepción de resultados. **stopStreamingValidation**: Para una validación de flujos, cerra la conexión con el servidor y finalizar la validación. **mkPermalink**: Dada la URL que representa la operación en curso, solicitar al servidor un *permalink* asociado a esta. **render**: Método principal de la clase *React* para renderizar el componente.

Nombre de la clase
DomainItem
Descripción
Abstracción común a todos los elementos operados por el cliente web e introducidos por los usuarios
Responsabilidades
- Ofrecer una plantilla común con los campos compartidos por los datos operados por el cliente y sus usuarios.
Atributos Propuestos
source: Fuente de la que provienen los datos introducidos por el usuario (texto, URL, archivo…) **textArea**: Datos introducidos por el usuario en formato textual. **url**: URL de residencia de los datos introducida por el usuario. **file**: Archivo contenedor de los datos subido por el usuario. **codeMirror**: Editor de texto *CodeMirror* del interfaz contenedor de la información en **textArea**.
Métodos Propuestos
-

Nombre de la clase
RDFData
Descripción
Subclase del dominio representando las propiedades de un elemento RDF operado por el cliente
Responsabilidades
-
Atributos Propuestos
format: Formato/sintaxis del RDF. **inference**: Inferencia a aplicar sobre los datos en su procesamiento.
Métodos Propuestos
-

Nombre de la clase
StreamData
Descripción
Subclase del dominio dentro de **RDFData** representando las propiedades de los datos RDF operados en una validación de flujos
Responsabilidades
-
Atributos Propuestos
server: Servidor de origen de los datos.

port: Puerto, dentro de **server**, de origen de los datos. **topic**: *Topic* de *Kafka* donde se encuentran los datos. **haltOnInvalid**: Si los datos deben dejar de consumirse en caso de haber un elemento inválido. **haltOnErrored**: Si los datos deben dejar de consumirse en case de haber un elemento causante de un error de ejecución.
Métodos Propuestos
-

Nombre de la clase
ShEx
Descripción
Subclase del dominio representando las propiedades de un esquema *Shape Expressions* operado por el cliente
Responsabilidades
-
Atributos Propuestos
format: Formato/sintaxis del esquema. **engine**: Mecanismo de validación empleado por el esquema.
Métodos Propuestos
-

Nombre de la clase
ShapeMap
Descripción
Subclase del dominio representando las propiedades de un *ShapeMap* operado por el cliente
Responsabilidades
-
Atributos Propuestos
format: Formato/sintaxis del *ShapeMap*. **triggerMode**: Mecanismo de validación empleado por el *ShapeMap*.
Métodos Propuestos
-

Nombre de la clase
ValidationConfiguration
Descripción
Clase contenedora de los ajustes de validación enviados al servidor
Responsabilidades
- Definir los ajustes de una operación de validación.

- Convertir los datos del dominio a un objeto equivalente con toda la información necesaria para que el servidor realice una validación.
Atributos Propuestos
data: Objeto JavaScript con las propiedades y contenido de los datos RDF empleados en la validación serializados a tipos simples operables mediante JSON. **schema**: Objeto JavaScript con las propiedades y contenido del esquema empleado en la validación serializados a tipos simples operables mediante JSON. **triggerMode**: Objeto JavaScript con las propiedades y contenido del *ShapeMap* empleado en la validación serializado a tipos simples operables mediante JSON.
Métodos Propuestos
-

Nombre de la clase
WebSocketConfiguration
Descripción
Clase contenedora de los ajustes de conexión con el servidor en una validación de flujos
Responsabilidades
- Definir los ajustes de la conexión *WebSockets* con el servidor en una operación de validación de flujos. - Definir los comportamientos del cliente ante cada uno de los eventos generados por la conexión con el servidor.
Atributos Propuestos
wsRoute: Ruta del servidor en la que se encuentra escuchando conexiones *WebSockets*.
Métodos Propuestos
onOpen: Función a ejecutar al abrir una conexión con éxito con el servidor. **onError**: Función a ejecutar en el evento de un error en la conexión con el servidor. **onClose**: Función a ejecutar al cierre de la conexión con el servidor. **onMessage**: Función a ejecutar al recibir un mensaje del servidor.

Nombre de la clase
ValidateStream
Descripción
Componente secundario del interfaz empleado por el componente padre para representar el estado y resultados de una validación de flujos
Responsabilidades
- Contener los resultados de la validación, formateados para adaptarlos a las necesidades del cliente. - Ofrecer mecanismos a los usuarios para visualizar los resultados de la validación.
Atributos Propuestos

maxItems: Número máximo de resultados que el navegador mostrará en la interfaz. Una vez superado, se empezará a descartar resultados por orden de llegada. **nodesPrefixMap**: Listado de prefijos empleados en los datos RDF validados, empleados para formatear los resultados y mostrarlos de forma más amigable. **shapesPrefixMap**: Listado de prefijos empleados en el esquema de validación, empleados para formatear los resultados y mostrarlos de forma más amigable. **formattedResults**: Conjunto de resultados de la validación, transformados a través de **formatResult**.
Métodos Propuestos
formatResult: Dado un objeto con los resultados crudos devueltos por el servidor, lo transforma para tener la estructura y formato adecuados para su representación visual. **render**: Método principal de la clase *React* para renderizar el componente.

Nombre de la clase
ValidationResult
Descripción
Objeto del dominio con la información necesaria para representar los resultados de la validación de flujos del servidor dentro del cliente
Responsabilidades
-
Atributos Propuestos
type: Tipo de resultado o respuesta recibida del servidor. **report**: Para validaciones completadas, resultado de la validación en formato JSON. **date**: Objeto fecha con el momento de validación del elemento representado en el resultado.
Métodos Propuestos
-

Nombre de la clase
ResultType
Descripción
Enumero de los distintos tipos de respuestas que puede emitir el servidor durante una validación de flujos
Responsabilidades
- Ayudar al cliente a distinguir ante qué tipo de respuesta del servidor se encuentra
Atributos Propuestos
RESULT: El mensaje recibido del servidor contiene el resultado de una validación completada. **ERROR**: El mensaje recibido del servidor contiene información detallada de un error.
Métodos Propuestos
-

Nombre de la clase
ShowShapeMap
Descripción
Componente secundario del interfaz empleado para representar y operar gráficamente los resultados finales de una validación de flujos
Responsabilidades
- Representar los resultados de una validación de forma amigable. - Permitir operar sobre los resultados de la validación de forma interactiva.
Atributos Propuestos
sortingSettings: Ajustes para la reordenación de los resultados. **expansionSettings**: Ajustes para la expansión/contracción de los detalles de los resultados. **paginationSettings**: Ajustes para la paginación de los resultados. **exportSettings**: Ajustes para la exportación de los resultados a formato CSV. **searchSettings**: Ajustes para la búsqueda de resultados.
Métodos Propuestos
sortItems: Altera el orden en el que los resultados están gráficamente representados. **expandItem**: Dado un resultado concreto, muestra u oculta el resultado completo. **exportItems**: Genera el archive CSV para la descarga de los resultados representados. **searchItems**: Dada una cadena de texto, filtra entre ellos resultados aquellos que la contengan en alguno de sus campos. **clearItems:** Elimina de la interfaz gráfica todos los resultados de la validación recibidos hasta ese momento. **render**: Método principal de la clase *React* para renderizar el componente.

4.7 ASI 8: DEFINICIÓN DE INTERFACES DE USUARIO

Para esta sección se definen las interfaces de usuario presentes en el sistema final. Es importate subrayar que, de todos los módulos del proyecto, sólo el cliente web cuenta con una interfaz de usuario como tal. Además, la herramienta de validación de flujos se va a integrar en una página ya existente.

A continuación, se muestran las interfaces de usuario para la validación de la prueba de concepto elaborada para el cliente de *RDFShape*, ampliando y refactorizando los componentes ya existentes en el sistema original.

4.7.1 Descripción de la Interfaz

Para la validación de flujos, la aplicación web tendrá un tipo de pantalla similar al resto de herramientas con la que va a coexistir, manteniendo la coherencia con el resto interfaces.

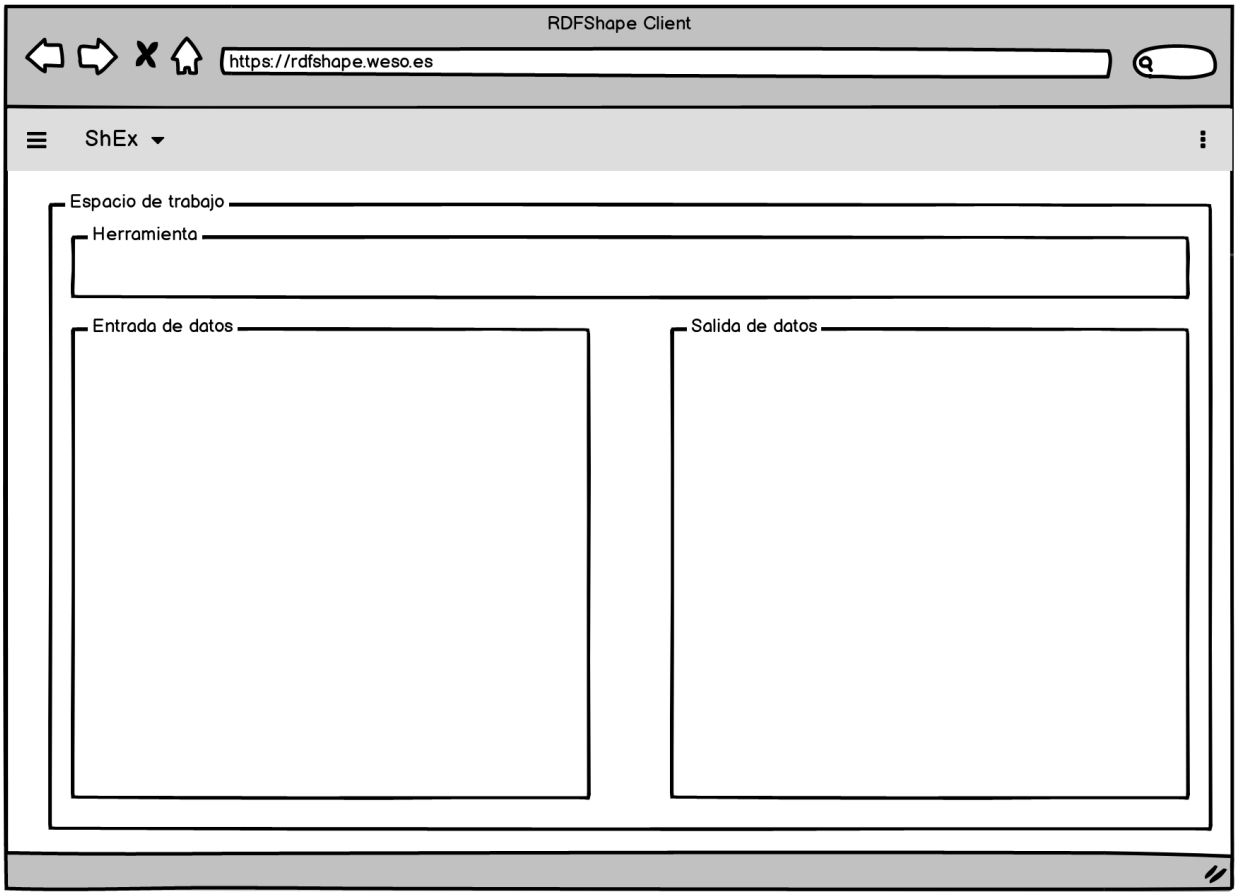


Ilustración 46. Interfaz de usuario - Descripción

Tal y como se observa en la imagen superior, tratamos con una interfaz web limitada a una única herramienta o "pantalla" dentro de una aplicación mayor, de ahí la aparición de elementos base

como la barra de navegación. Dentro del espacio de trabajo disponible, se distinguen tres subespacios:

- **Herramienta**: Se trata del espacio designado al título de la herramienta que se está utilizando, de forma coherente con el resto de las páginas del cliente. Así mismo, en este espacio se incluirá una referencia para obtener ayuda.
- **Entrada de datos**: Dentro del área principal se distinguen dos columnas de trabajo. Esta primera columna, al lateral izquierdo, está destinada a recibir las entradas del usuario mediante elementos de formulario web y representarlas de forma acorde.
- **Salida de datos**: Esta área de trabajo representa los contenidos fabricados como sistema en respuesta a las entradas del usuario. Para esta herramienta de la web, contendrá los resultados de la validación y los elementos necesarios para que el usuario interactúe con ellos.

Así mismo, el sistema se valdrá de mecanismos adicionales para comunicar información puntual al usuario, como consejos de ayuda o notificaciones de generación de *permalinks* que en ningún caso serán modales y restringirán las acciones del usuario. Para el caso de los *permalinks* y similares, se usarán mensajes emergentes con la siguiente apariencia:

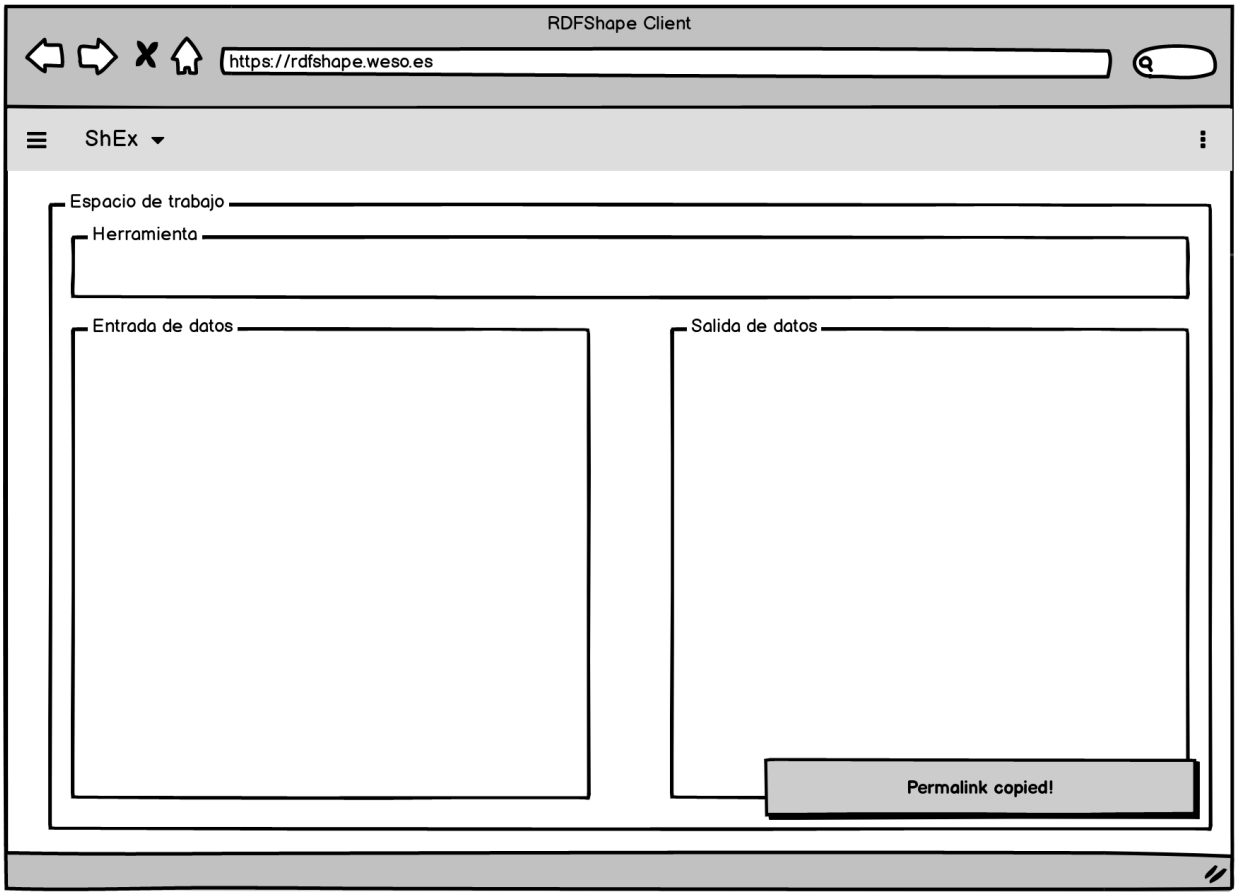


Ilustración 47. Interfaz de usuario - Mensajes emergentes

4.7.2 Definición del Aspecto de la Interfaz

Como se ha mencionado anteriormente, las nuevas interfaces se desarrollan en base a lo ya construido en el cliente *RDFShape* y buscando integrarse con el resto de la aplicación. Adicionalmente, la validación de flujos se implementa dentro de la página ya existente de validación con *Shape Expressions*.

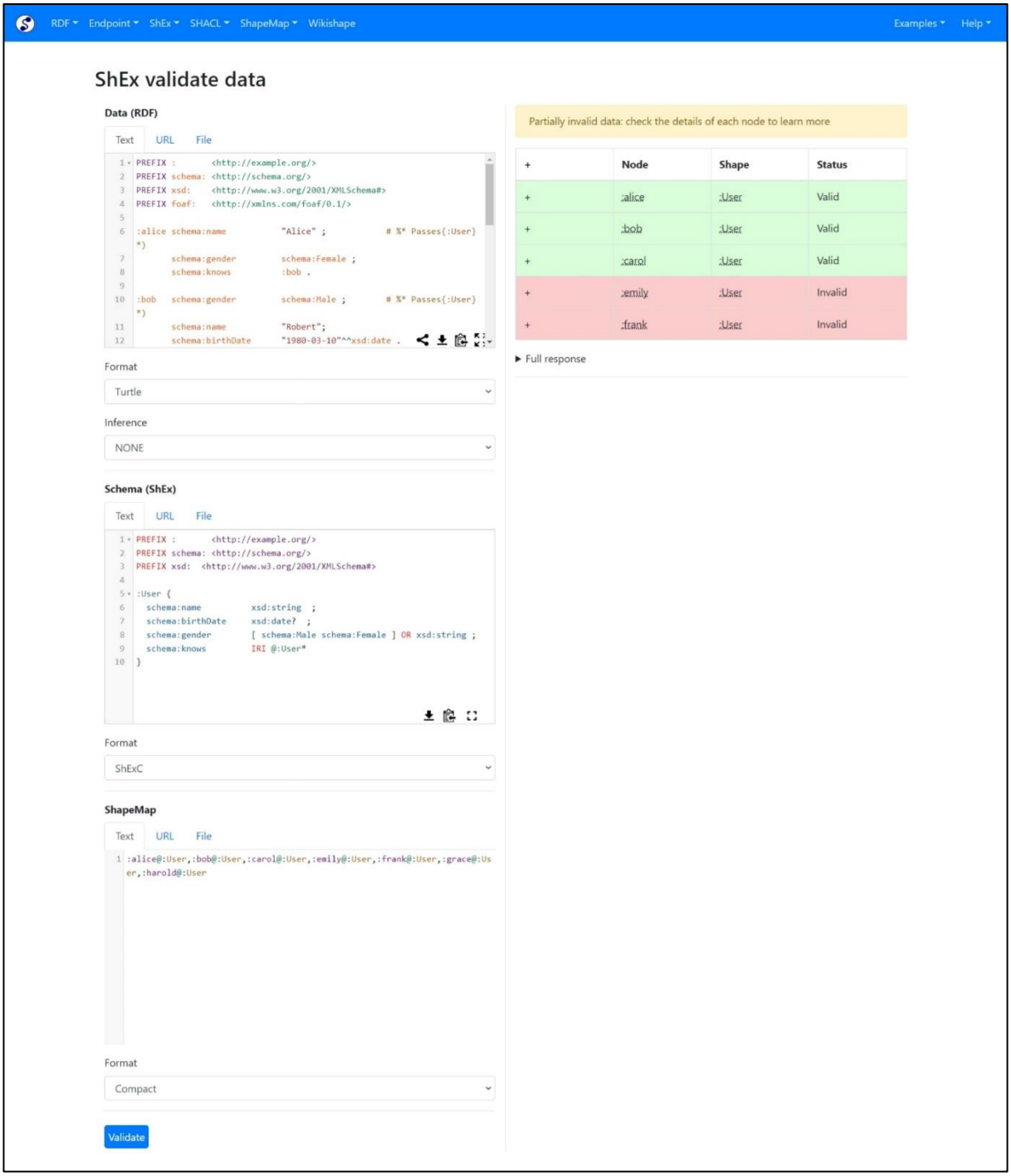


Ilustración 48. Interfaz de usuario - Interfaz existente en RDFShape

Toda la interfaz está pensada para la validación de flujos y sus distintas variantes se adaptan al estado dinámico de esta.

4.7.2.1 *Entrada de datos (pre-validación)*

La pantalla inicial que encontrará el usuario antes de iniciar cualquier validación tendrá un aspecto similar a la interfaz ya existente en el sistema, salvo por la inclusión de un nuevo mecanismo de entrada de datos RDF (mediante flujos/*stream*), representado como una nueva pestaña junto a las opciones ya existentes. Al seleccionar está opción aparecerá un formulario para que el usuario pueda configurar la validación de flujos. Otro cambio menor será la inclusión de un acceso a la ayuda de la herramienta al lado del nombre de esta.

Puesto que no hay resultados, el área de resultados no mostrará más que un mensaje informativo.

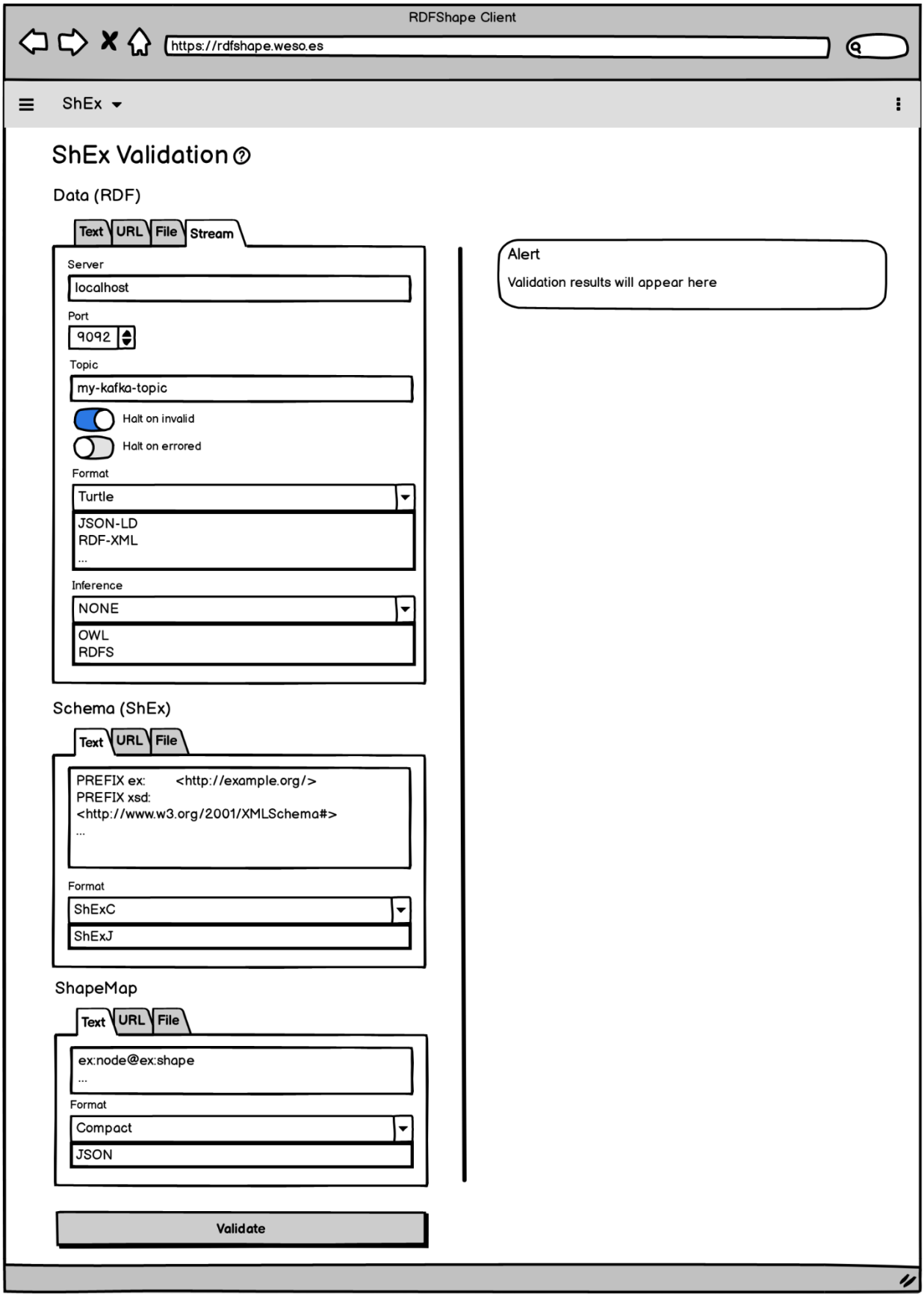


Ilustración 49. Interfaz de usuario - Entrada de datos

4.7.2.2 *Envío de datos (inicialización de validación)*

Al iniciar una validación de datos, el sistema sufrirá los siguientes cambios:

- Área de entradas:
    - Se bloquea el botón de validar para impedir que el usuario emita más solicitudes de validación. Esto se mantendrá así hasta que se pause la validación en curso o suceda un error.
    - Se mantiene libre el formulario de entrada de datos, tal que el usuario puede seguir alterando las entradas en preparación de una futura validación. Esto se mantendrá así en todo momento, siendo el usuario libre de editar sus entradas antes, durante y después de una validación.
- Área de resultados:
    - Se modifica el mensaje del área de resultados para mostrar que se ha recibido la solicitud de validación. Hasta la llegada de resultados, se mantendrá este mensaje.
    - Se muestra el botón de *permalink*, puesto que las opciones suministradas por el usuario son válidas. Al margen de los resultados o errores que puedan suceder en la validación, la operación es apta para almacenarse.

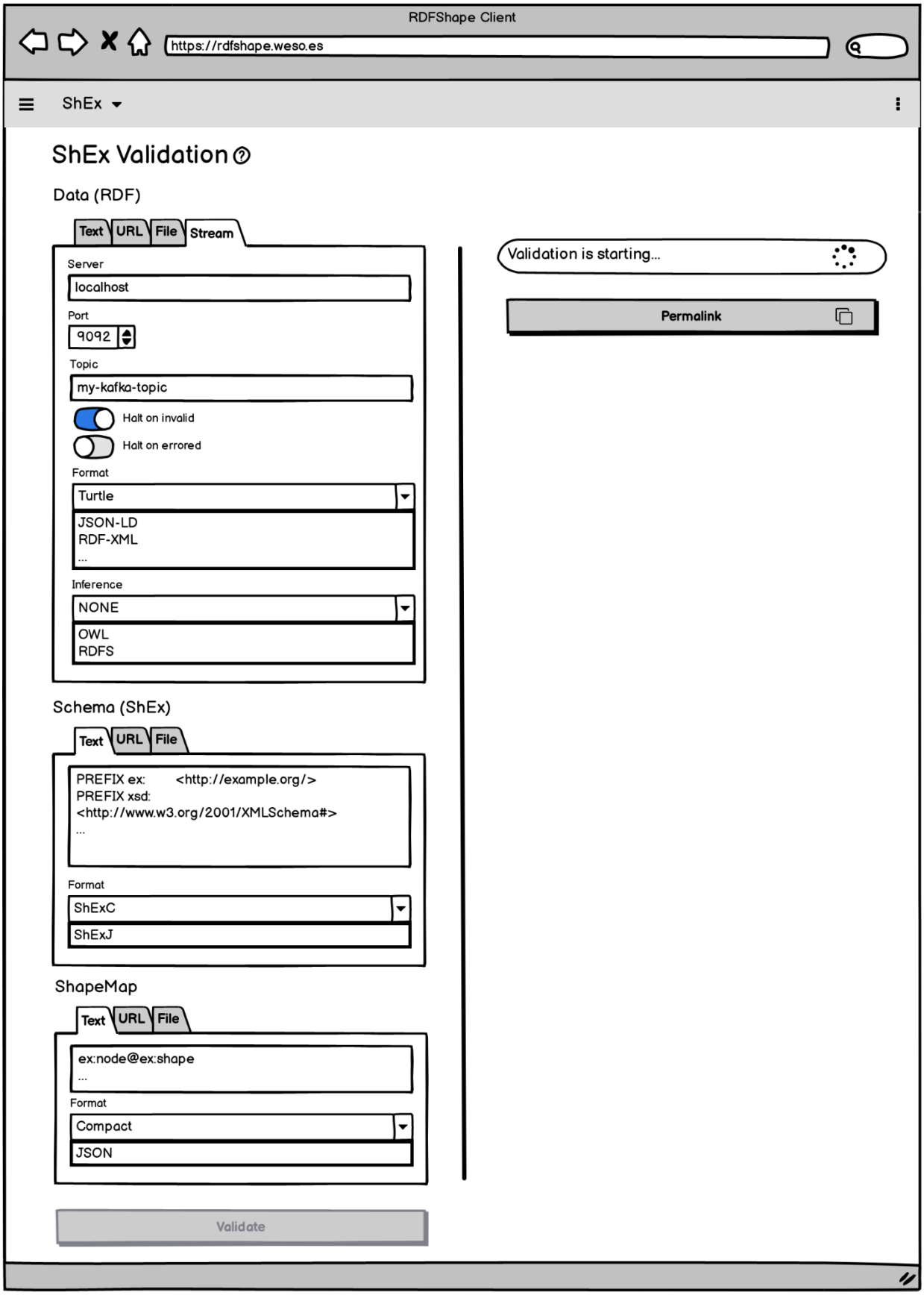


Ilustración 50. Interfaz de usuario - Envío de datos

4.7.2.3 *Controles y listado de resultados (validación en curso)*

En el momento que se reciben resultados del servidor, se considera que la validación se encuentra en curso y la interfaz sufre los siguientes cambios:

- Área de resultados:
    - Se modifica el mensaje del área de resultados para mostrar que la validación está transcurriendo en tiempo real. Hasta que se pause o cancele manualmente o por errores, se mantendrá este mensaje.
    - En el momento que se tienen resultados que mostrar, se muestra la tabla de resultados con las opciones pertinentes para operar sobre ellos, incluyendo:
        - Pausar/continuar.
        - Eliminación de los resultados actuales.
        - Descarga de los resultados actuales.
        - Reordenación de los resultados actuales.
        - Expansión de la información de un resultado concreto.
        - Paginación para navegar entre resultados.

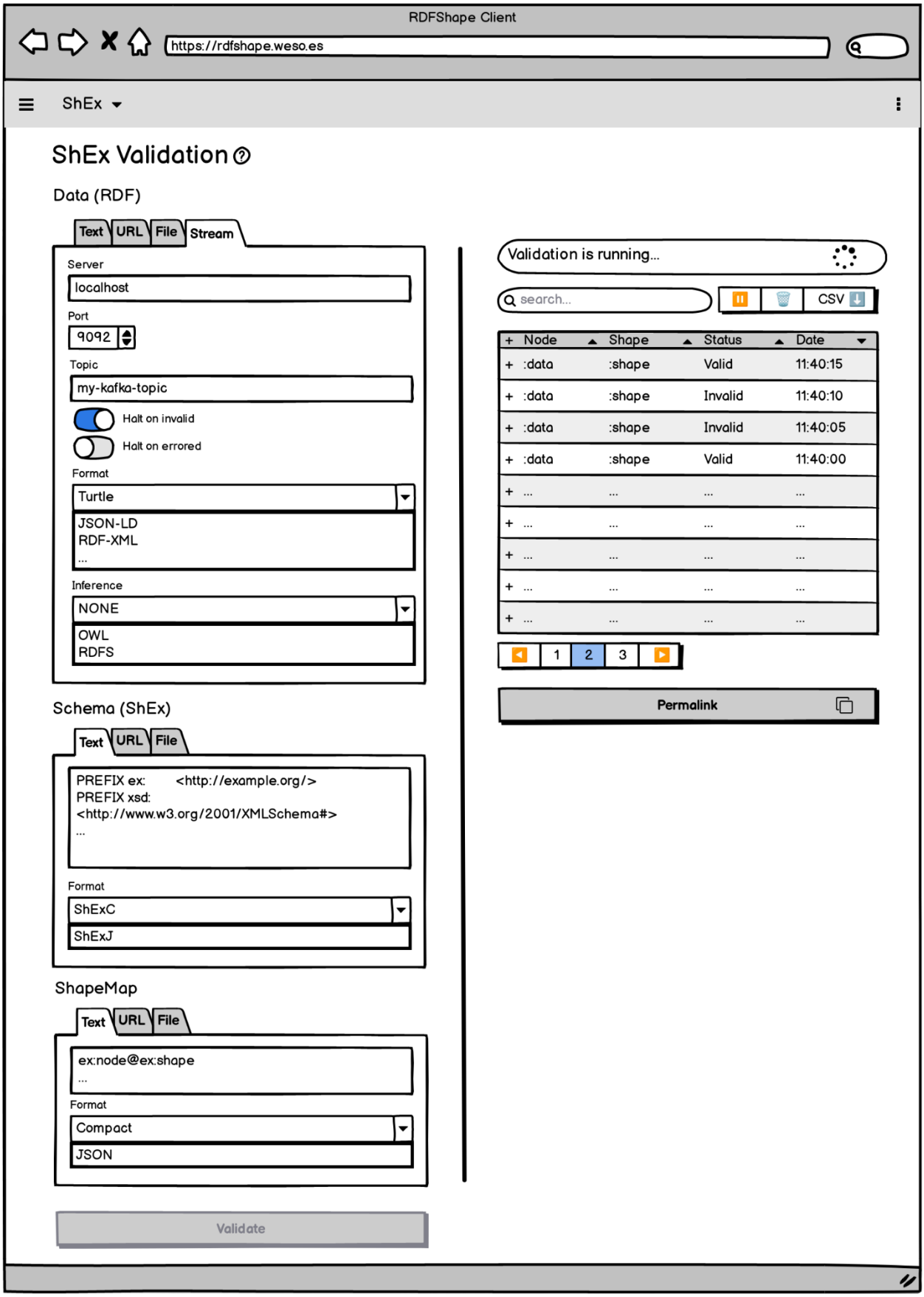


Ilustración 51. Interfaz de usuario - Controles y resultados (en curso)

4.7.2.4 *Controles y listado de resultados (validación en pausa)*

Una vez se pausa una validación en curso, se dejan de recibir resultados del servidor y la interfaz sufre los siguientes cambios:

- Área de entradas:
    - Se desbloquea el botón de validar de forma que el usuario pueda invocar una validación nueva, descartando los datos de la anteriormente en curso. Esto se mantendrá así hasta que se inicie una nueva validación.
- Área de resultados:
    - Se modifica el mensaje del área de resultados para mostrar que la validación está pausada. Hasta que se reanudo o se inicie una nueva validación, se mantendrá este mensaje.
    - En los controles de la validación, se reemplazará la opción de pausa por la opción de reanudar. Esto se mantendrá así hasta que el usuario use el botón para reanudar la validación, momento en que la opción de reanudar se sustituirá por la opción de pausa nuevamente.
    - Adicionalmente, al pausar el sistema es un buen momento para expandir los resultados en detalle.

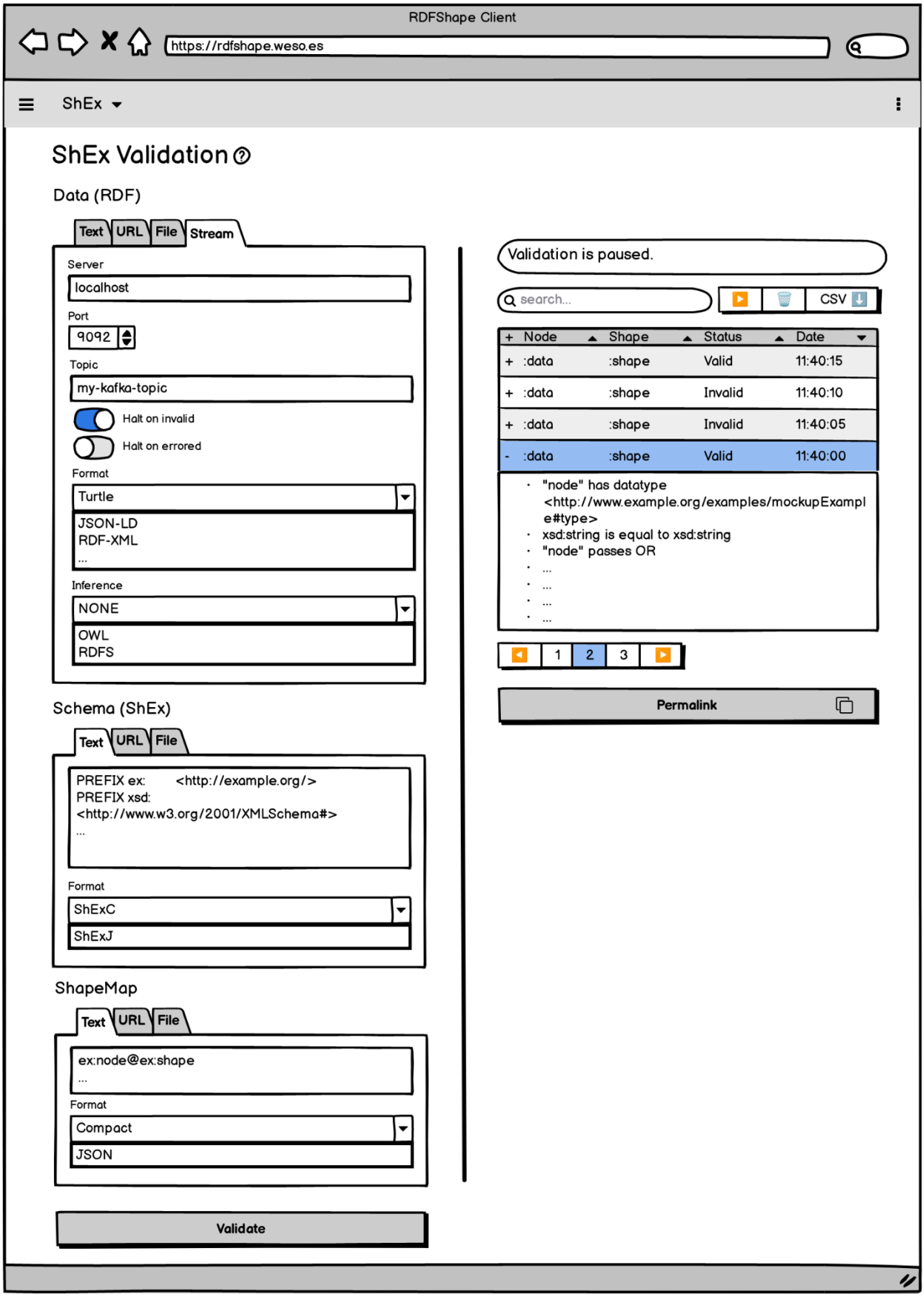


Ilustración 52. Interfaz de usuario - Controles y resultados (pausa)

4.7.2.5 *Error en el proceso (validación abortada)*

En el momento en que se recibe un error del servidor, la validación queda cancelada y el sistema muestra información del error tal que se dan los siguientes cambios:

- Área de entradas:
    - Se desbloquea el botón de validar de forma que el usuario pueda invocar una validación nueva, descartando los datos de la anteriormente en curso. Esto se mantendrá así hasta que se inicie una nueva validación.
- Área de resultados:
    - Se modifica el mensaje del área de resultados para mostrar el error de la validación e información adicional de forma desplegable. Hasta que se inicie una nueva validación, se mantendrá la información del error.
    - Se desactivan los controles de pausa/reanudar pues la validación ha terminado.
    - Se mantendrán intactos los resultados del área de validación presentes en el momento del error. En caso de que el error se deba a un ítem inválido, el último resultado en el sistema será este ítem problemático.

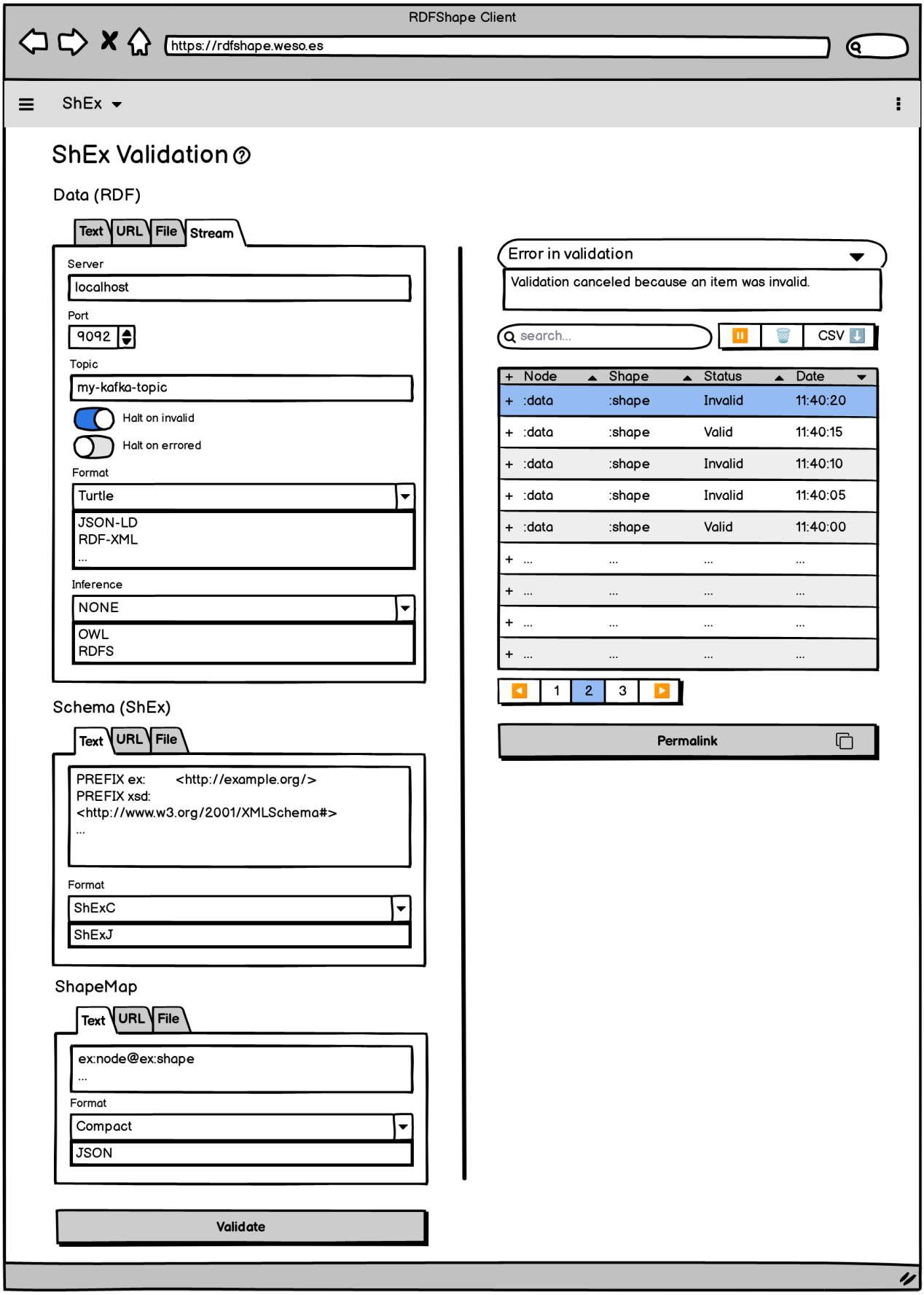


Ilustración 53. Interfaz de usuario - Error en el proceso

4.7.3 Descripción del Comportamiento de la Interfaz

El comportamiento de la interfaz de validación de flujos debe ser coherente con el resto de las herramientas ofrecidas por *RDFShape*, tanto en aspecto como comportamiento, y cumplir los siguientes estándares:

- **Manejo de validaciones**
    - El usuario no será capaz de iniciar una validación si alguno de los datos necesarios para arrancarla está vacío. La interfaz permitirá invocar la validación pero se mostrará un mensaje de error.
    - El usuario no será capaz de invocar una validación mientras haya otra en curso y no esté pausada. El sistema deshabilitará el botón iniciador de la acción según lo especificado en *4.7.2-Definición del Aspecto de la Interfaz*.
- **Manejo de errores**
    - Para errores devueltos por el servidor y, por tanto, con información adicional, el sistema mostrará un resumen de la causa del error junto con la opción de desplegar los detalles de este, según lo especificado en *4.7.2.5-Error en el proceso (validación abortada)*.
- **Sistema de ayuda**
    - El sistema recibirá las peticiones de ayuda cuando el usuario interactúe con el botón situado junto al nombre de la herramienta. En este momento, el sistema mostrará una *tooltip* con texto explicativo.
- **Generación de *permalinks***
    - En el momento en que se solicite la generación de un *permalink*, el botón iniciador de la acción sufrirá los siguientes cambios:
        - Estará desactivado hasta que se reciba una respuesta del servidor.
        - Estará acompañado de un *spinner* o elemento que manifieste que el sistema está operando hasta que se reciba una respuesta del servidor.
    - Cuando se complete la generación del enlace, el sistema lo depositará en el portapapeles del usuario y le informará a través de un mensaje emergente similar al visto en *Ilustración 47. Interfaz de usuario - Mensajes emergentes*.

Adicionalmente, las respuestas del sistema a las interacciones con el usuario y el servidor seguirán el siguiente esquema de colores:

- **Azul**: Mensaje informativo neutro, el sistema está iniciándose.
- **Verde**: Mensaje informativo exitoso, el sistema está trabajando.
- **Amarillo**: Mensaje de advertencia, el sistema está trabajando o se ha detenido pero ha podido operar parcialmente.
- **Rojo**: Mensaje de error, el sistema no ha podido comenzar o terminar su ejecución correctamente.

4.7.4 Diagrama de Navegabilidad

A continuación se muestra el mapa de navegación entre las interfaces previamente descritas. Nótese que las pantallas previas son distintos estados en los componentes de una misma página del cliente web, no diferentes pantallas.

Se muestra el diagrama de navegabilidad en función de cómo el sistema interactúa con el usuario y el servidor:

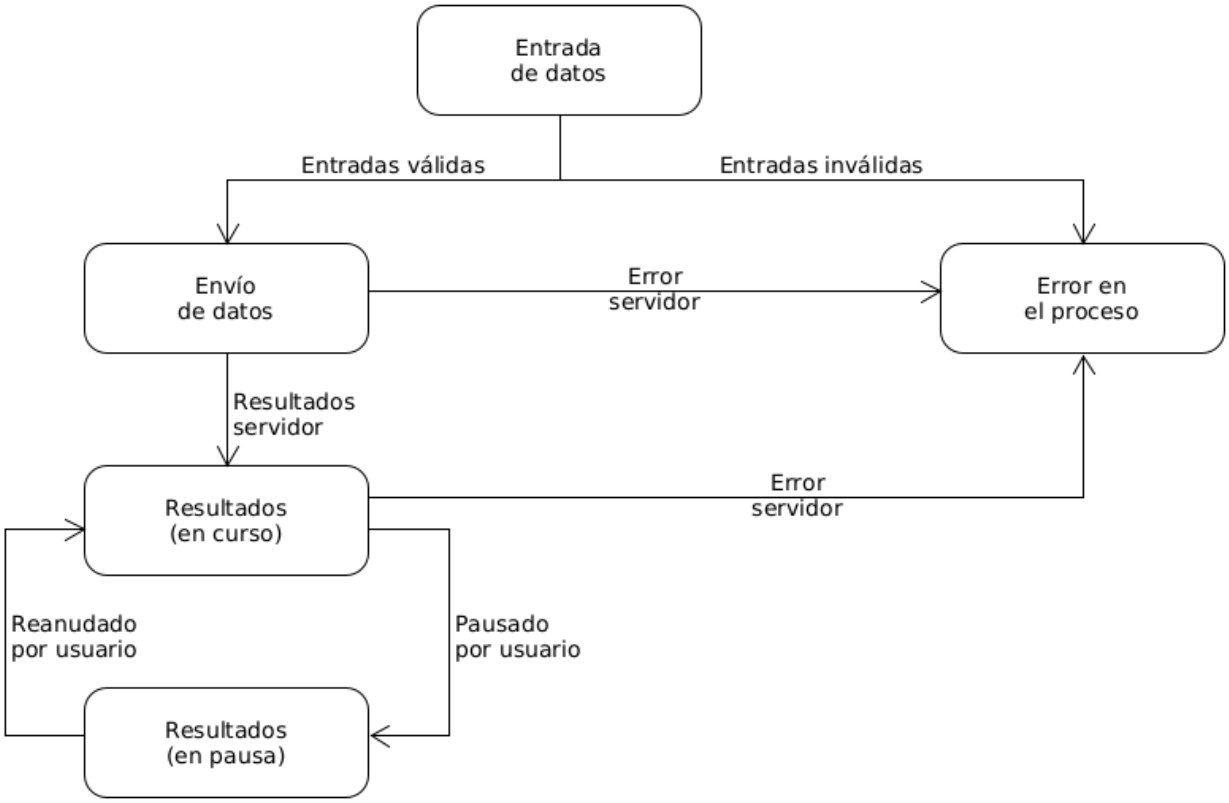


Ilustración 54. Diagrama de navegabilidad del cliente web

4.8 ASI 10: ESPECIFICACIÓN DEL PLAN DE PRUEBAS

Para comprobar que el comportamiento del sistema es el adecuado es necesaria una serie de *tests*. Nótese que el sistema es multimodular y con un alcance muy amplio. Sin embargo, el plan de pruebas estará fuertemente centrado en el nuevo módulo validador mediante pruebas unitarias junto con otro tipo de pruebas para confirmar su integración en el ecosistema *RDFShape*.

Se van a desarrollar los siguientes tipos de pruebas:

- **Pruebas unitarias**
    - Destinadas a comprobar de forma pormenorizada el funcionamiento del nuevo módulo de validación de flujos para varios formatos y tipos de flujos de entrada.
    - Para el resto de los módulos refactorizados y dependencias asociadas, se estima que las pruebas unitarias originalmente presentes en esos sistemas con suficientes.
- **Pruebas de integración**
    - Destinadas a comprobar la integración entre los propios subsistemas y de estos con agentes externos (usuarios y servidores de flujo de datos).
- **Pruebas de usabilidad**
    - Destinadas a comprobar que el nivel de usabilidad del cliente web cumple los requisitos del cliente.
    - Se elaborarán cuestionarios de usabilidad en la sección de diseño (ver *5.5.3-Pruebas de Usabilidad*), una vez se defina el plan de pruebas con la información final del producto.
- **Pruebas de accesibilidad**
    - Destinadas a comprobar que el nivel de accesibilidad del cliente web cumple los requisitos del cliente.
    - Se recurrirá a una herramienta externa para auditar el nivel de accesibilidad en la sección de diseño (ver *5.5.4-Pruebas de Accesibilidad*), una vez se defina el plan de pruebas con la información final del producto.
- **Pruebas de rendimiento**
    - Destinadas a comprobar el rendimiento de la biblioteca en escenarios de alto rendimiento y paralelización.
    - Se someterá a la biblioteca de validación a cargas similares variando el modo de ejecución (monohilo contra multihilo) para medir los tiempos de trabajo en igualdad de condiciones.

A continuación, se define el plan de pruebas a ejecutar en base a los casos de uso y escenarios definidos en *4.5-ASI 4: Análisis de los Casos de Uso*.

4.8.1 Pruebas Unitarias

Como se ha comentado anteriormente, las pruebas unitarias desarrolladas se centrarán en la nueva biblioteca de validación de flujos.

4.8.1.1 *Pruebas de Validación*

Se realizarán varias pruebas para verificar que *SHaclEX*, la herramienta de validación utilizada por el sistema, se comporta de la forma esperada cuando funciona dentro de la biblioteca de validación. Se crean varios datos RDF junto con sus equivalentes en distintas sintaxis y se validan contra un esquema simulando un flujo de entrada de un único elemento.

Pruebas de validación	
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *Turtle*.	El sistema procesa el flujo y emite un único resultado con el estado VALID.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *JSON-LD*.	El sistema procesa el flujo y emite un único resultado con el estado VALID.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *RDF-XML*.	El sistema procesa el flujo y emite un único resultado con el estado VALID.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *Turtle*.	El sistema procesa el flujo y emite un único resultado con el estado INVALID.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *JSON-LD*.	El sistema procesa el flujo y emite un único resultado con el estado INVALID.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *RDF-XML*.	El sistema procesa el flujo y emite un único resultado con el estado INVALID.

Tabla 68. Pruebas de validación – Comet (análisis)

4.8.1.2 *Pruebas de extracción*

Se verificará que el sistema de extractores diseñado para Comet funciona correctamente. Se creará un grupo de datos RDF de tamaño arbitrario que será ofrecido a un validador a través de los distintos extractores implementados por el sistema base. El resultado de la validación será el mismo independientemente del extractor que haya obtenido los datos RDF, a excepción del extractor de *Kafka*, que producirá un error al no tener un servidor que consumir.

Pruebas de extracción	
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de listas.	El sistema procesa el flujo y emite tantos resultados como elementos contiene el flujo de entrada.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de ficheros.	El sistema procesa el flujo y emite tantos resultados como elementos contiene el flujo de entrada.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de *Kafka*.	El sistema es incapaz de procesar el flujo y emite un error.

Tabla 69. Pruebas de extracción – Comet (análisis)

4.8.1.3 *Pruebas de parada*

Se comprobará a través de pruebas que el validador de flujos se detiene cuando se dan ciertos errores o si está configurado para hacerlo bajo ciertas circunstancias.

Para las condiciones de parada del validador, se someterá a validaciones contra datos RDF inválidos o directamente mal redactados y se verificará que el validador ignora los problemas o emite los errores adecuados para cada contexto. Para abortar la validación por inactividad del flujo de entrada, se configurará el validador con el tiempo mínimo de espera, de forma que se detenga por inactividad al instante.

Pruebas de parada	
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF inválido con el validador configurado para ignorar los ítems inválidos.	El sistema procesa el flujo al completo emitiendo los resultados de validación correspondientes.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF inválido con el validador configurado para detenerse ante ítems inválidos.	El sistema detiene la validación tras procesar el elemento inválido. El sistema lanza la excepción asociada a elementos inválidos.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF mal redactado con el validador configurado para ignorar los ítems erróneos.	El sistema procesa el flujo al completo, no emitiendo ningún resultado para los ítems erróneos.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF mal redactado con el validador configurado para detenerse ante ítems erróneos.	El sistema detiene la validación tras procesar el elemento erróneo. El sistema no emite resultados y lanza la excepción asociada a elementos erróneos.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF con el validador configurado para abortar la validación si no se leen datos del flujo instantáneamente.	El sistema no llega a procesar ningún elemento y lanza la excepción asociada a inactividad del flujo tras inicializarse.

Tabla 70. Pruebas de parada – Comet (análisis)

4.8.2 Pruebas de Integración

Las pruebas de integración abarcan a todos los módulos del proyecto en sus interacciones entre ellos y con el exterior. Se dividen en función del módulo del sistema con el que guarden mayor relación para facilitar la lectura del listado de *tests*.

4.8.2.1 *Módulo de Validación de Flujos*

Las pruebas de integración relacionadas con la biblioteca de validación de flujos se basan en repetir los principios de las pruebas de validación unitarias (ver *4.8.1.1-Pruebas de Validación*), pero esta vez incluyendo un flujo de datos de entrada real procedente de un servidor *Apache Kafka* accesible a través de red (local e internet). Así mismo, se trabaja con datos RDF en formato *Turtle* por simplicidad.

Pruebas de integración	
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF procedente de un flujo de datos *Kafka* no disponible o inaccesible.	El sistema no llega a procesar ningún elemento y lanza la excepción asociada a inactividad del flujo transcurrido el período por defecto.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF válido procedente de un flujo de datos *Kafka* ejecutándose localmente.	El sistema procesa el flujo y emite un único resultado con el estado VALID.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF inválido procedente de un flujo de datos *Kafka* ejecutándose localmente.	El sistema procesa el flujo y emite un único resultado con el estado INVALID.
Prueba	**Resultado Esperado**
Validar mediante *ShEx* un conjunto de RDF válido procedente de un flujo de datos *Kafka* accesible a través de internet.	El sistema procesa el flujo y emite un único resultado con el estado VALID.
Prueba	**Resultado Esperado**

Validar mediante *ShEx* un conjunto de RDF inválido procedente de un flujo de datos *Kafka* accesible a través de internet.	El sistema procesa el flujo y emite un único resultado con el estado INVALID.

Tabla 71. Pruebas de integración – Comet (análisis)

4.8.2.2 *Módulo API*

La integración del módulo API con el resto de los agentes y subsistemas se basa en el manejo que hace de las peticiones, tanto *HTTP* como *WebSockets*, provenientes de clientes externos. Los *tests* de integración se centran en las nuevas funcionalidades de validación de flujos así como el sistema de registros.

Se empleará el programa *Postman* **[97]** para el envío de peticiones configurables programáticamente al API y observar su comportamiento.

Pruebas de integración	
Prueba	**Resultado Esperado**
Lanzar una petición contra una ruta inexistente del API.	El sistema devuelve un código de error 404 (*Not Found*).
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos en formato distinto de JSON.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el servidor de origen de los datos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el *topic* de origen de los datos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.

especificar el puerto de origen de los datos.	El sistema trata de inicializar la validación utilizando el puerto por defecto 9092.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el grupo de consumidor de los datos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación utilizando el identificador de grupo por defecto.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el formato de los datos RDF por consumir.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar la inferencia a aplicar sobre los datos por consumir.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación sin aplicar ninguna inferencia.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el esquema *ShEx* contra el que validar los datos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos especificando un esquema *ShEx* inválido.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el *ShapeMap* contra el que validar los datos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.

especificando un *ShapeMap* inválido.	El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el comportamiento del validador ante ítems inválidos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación ignorando los valores inválidos.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos sin especificar el comportamiento del validador ante ítems erróneos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación ignorando los valores erróneos o fallidos.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos contra una combinación de servidor/*topic* inexistente.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación y pasado el período de inactividad del flujo aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente válidos y el validador configurado para no detenerse.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación y envía los resultados a través de *WebSockets* según son recibidos y procesados. El sistema registra el proceso de validación en el sistema de registros del sistema de ficheros.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos con todos los campos pero con datos de tipo incorrecto.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación con todos los campos pero con datos fuera de los rangos permitidos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente informando del problema.

Prueba	Resultado Esperado
Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente válidos y el validador configurado para detenerse ante elementos inválidos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación y envía los resultados a través de *WebSockets* según son recibidos y procesados. El sistema aborta la conexión con el cliente ante el primer resultado inválido informando del problema.
Prueba	**Resultado Esperado**
Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente erróneos y el validador configurado para detenerse ante elementos erróneos.	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema trata de inicializar la validación y envía los resultados a través de *WebSockets* según son recibidos y procesados. El sistema aborta la conexión con el cliente ante el primer elemento incorrecto informando del problema.
Prueba	**Resultado Esperado**
Detener, desde el cliente, la conexión con el servidor durante una validación.	El sistema cierra la conexión, desconecta del flujo de datos y detiene la validación del flujo junto con los recursos dedicados a esta.

Tabla 72. Pruebas de integración – API (análisis)

4.8.2.3 *Módulo Cliente*

El factor diferenciador del cliente es la interacción que tiene con los usuarios finales del sistema a través de una interfaz, traduciendo las acciones de los usuarios en mensajes hacia el API y viceversa. En este caso, los *tests* y los resultados esperados se especifican en términos de qué modificaciones sufre el estado interno del sistema y, por tanto, la interfaz.

Pruebas de integración	
Prueba	**Resultado Esperado**
Invocar una validación de flujos sin rellenar el servidor de origen.	El sistema no contacta con el servidor y muestra un mensaje de error en el área de resultados informando del problema.
Prueba	**Resultado Esperado**
Invocar una validación de flujos sin rellenar el puerto de origen.	El sistema no contacta con el servidor y muestra un mensaje de error en el área de resultados informando del problema.
Prueba	**Resultado Esperado**

Invocar una validación de flujos sin rellenar el *topic* de origen.	El sistema no contacta con el servidor y muestra un mensaje de error en el área de resultados informando del problema.
Prueba	**Resultado Esperado**
Invocar una validación de flujos sin rellenar el esquema *ShEx* a emplear.	El sistema no contacta con el servidor y muestra un mensaje de error en el área de resultados informando del problema.
Prueba	**Resultado Esperado**
Invocar una validación de flujos sin rellenar el *ShapeMap* a emplear.	El sistema no contacta con el servidor y muestra un mensaje de error en el área de resultados informando del problema.
Prueba	**Resultado Esperado**
Invocar una validación de flujos contra un esquema *ShEx* mal formateado.	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. El sistema recibe un error del servidor y los muestra en el área de resultados.
Prueba	**Resultado Esperado**
Invocar una validación de flujos contra un *ShapeMap* mal formateado.	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. El sistema recibe un error del servidor y lo muestra en el área de resultados.
Prueba	**Resultado Esperado**
Invocar una validación de flujos válida contra una combinación de servidor/*topic* inexistente o no disponible.	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. Pasado el tiempo de inactividad fijado en el servidor, el sistema recibe un error del servidor y lo muestra en el área de resultados.
Prueba	**Resultado Esperado**
Invocar una validación de flujos válida contra un flujo existente.	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. Recibido el primer resultado, el sistema actualiza la interfaz para mostrar el listado de resultados y las operaciones disponibles sobre estos. Si se recibe un error del servidor, el sistema se asegura de cerrar la conexión con el servidor y actualizar la interfaz para mostrar el error

	sobre los resultados disponibles así como permitir solicitar una nueva validación.
Prueba	**Resultado Esperado**
Pausar una validación de flujos en curso.	El sistema cierra la conexión *WebSockets* con el servidor, actualizando la interfaz para: - Informar de la pausa de la validación. - Permitir reanudar la validación. - Permitir solicitar una nueva validación.
Prueba	**Resultado Esperado**
Reanudar una validación de flujos en pausa.	El sistema reabre la conexión *WebSockets* con el servidor, actualizando la interfaz para: - Informar de que la validación está en curso. - Incluir todos los resultados que se han perdido durante la pausa del sistema en el listado. - Impedir solicitar una nueva validación.
Prueba	**Resultado Esperado**
Buscar entre los resultados de una validación en curso con un término no vacío (1+ caracteres).	El sistema actualiza el listado de resultados mostrando sólo aquellos cuyo contenido textual en alguno de sus campos coincide con el término de búsqueda.
Prueba	**Resultado Esperado**
Buscar entre los resultados de una validación en curso un término vacío (0 caracteres).	El sistema actualiza el listado de resultados mostrando todos los resultados.
Prueba	**Resultado Esperado**
Limpiar los resultados de una validación en curso.	El sistema actualiza el listado de resultados, mostrando un listado vacío hasta la llegada de nuevos elementos.
Prueba	**Resultado Esperado**
Exportar los resultados de una validación en curso.	El sistema genera un documento CSV con todos los campos de todos los resultados de validación en el listado y lanza el diálogo por defecto del navegador para visualizar/descargar el documento.
Prueba	**Resultado Esperado**
Ordenar los resultados de una validación en curso en base a un campo nuevo.	El sistema reordena el listado de resultados de forma ascendente en base al contenido textual del campo objetivo.
Prueba	**Resultado Esperado**
Ordenar los resultados de una validación en curso en	El sistema reordena el listado de resultados de forma contraria a cómo se encontrase ordenado el contenido en base al contenido textual del campo objetivo.

base a un campo ya ordenado.	
Prueba	**Resultado Esperado**
Generar un *permalink* a una operación de validación.	El sistema contacta con el sistema de generación de enlaces del servidor y actualizará la interfaz hasta obtener una respuesta para: - Mostrar que la generación del enlace está en curso. - Impedir la solicitud de un nuevo enlace. El sistema coloca en el portapapeles del usuario el nuevo enlace a la validación y muestra un mensaje informativo emergente de forma temporal.
Prueba	**Resultado Esperado**
Consumir un *permalink* a una operación de validación.	El sistema muestra una pantalla informativa al usuario comunicando que está procesando el enlace. El sistema contactará con el sistema de generación de enlaces del servidor y redirige al usuario al enlace objetivo automáticamente. El sistema rellena los datos de la validación y la inicia de forma automática.

Tabla 73. Pruebas de integración – Cliente (análisis)

4.8.3 Pruebas de Usabilidad

Las pruebas de usabilidad del cliente web girarán en torno a la elaboración de cuestionarios ofrecidos a usuarios reales y la evaluación de sus respuestas. Los usuarios objetivo de los cuestionarios serán los alumnos de la asignatura de Web Semántica del máster, puesto que para la fecha en que se imparte la asignatura se prevé tener una demo funcional del sistema que puedan utilizar en el transcurso de las clases.

Pruebas de usabilidad	
Prueba	**Resultado Esperado**
Encuestar a los alumnos del máster con cuestionarios de usabilidad del sistema.	Los usuarios están mayormente satisfechos con el sistema.

Tabla 74. Pruebas de usabilidad - Cliente

4.8.4 Pruebas de Accesibilidad

Debido a los requisitos del sistema, es necesario alcanzar un nivel de accesibilidad AA (*WCAG*) en las interfaces web del cliente. Se empleará la herramienta externa *AChecker* sobre las pantallas de validación de flujos sistema para verificar el cumplimiento de los mínimos de accesibilidad marcados.

Pruebas de accesibilidad	
Prueba	**Resultado Esperado**
Auditoría de accesibilidad de la aplicación web.	El sistema cumple con el nivel de accesibilidad requerido.

Tabla 75. Pruebas de accesibilidad – Cliente (análisis)

4.8.5 Pruebas de Rendimiento

Las pruebas de rendimiento están enfocadas en la efectividad de la biblioteca de validación de flujos al enfrentarse a tareas pesadas. Puesto que no se tiene control directo sobre el rendimiento del mecanismo de validación (*SHaclEX*) ni del entorno de despliegue final del sistema (infraestructura de WESO), los *tests* de rendimiento medirán la mejora relativa de aspectos sobre los que sí se ha podido influir en el desarrollo: el nivel de paralelización de la carga de trabajo.

Se emplearán flujos de datos alojados en memoria con una gran cantidad de ítems de contenido pseudoaleatorio y se medirán los tiempos de ejecución del sistema en función a varios escenarios de paralelización:

Pruebas de rendimiento	
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio numeroso en un hilo de ejecución (monohilo).	El sistema completa la validación en un tiempo determinado.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio numeroso en varios hilos de ejecución (multihilo).	El sistema completa la validación en un tiempo menor que en monohilo.

Tabla 76. Pruebas de rendimiento – Comet (análisis)

Capítulo 5 Diseño del Sistema de Información

Fase de Desarrollo

DSI

5.1 DSI 3: DISEÑO DE CASOS DE USO REALES

A continuación, se refinan los casos de uso más relevantes de los especificados en el capítulo de análisis (ver *4.2.3-Especificación de Casos de Uso*) junto con diagramas de secuencia explicativos.

Para cada caso de uso, sus diagramas reflejarán el sistema en su versión de diseño, una iteración posterior al análisis incluyendo las clases y métodos candidatos al sistema final.

5.1.1 Usuario Biblioteca

5.1.1.1 *Invocar una validación de flujos*

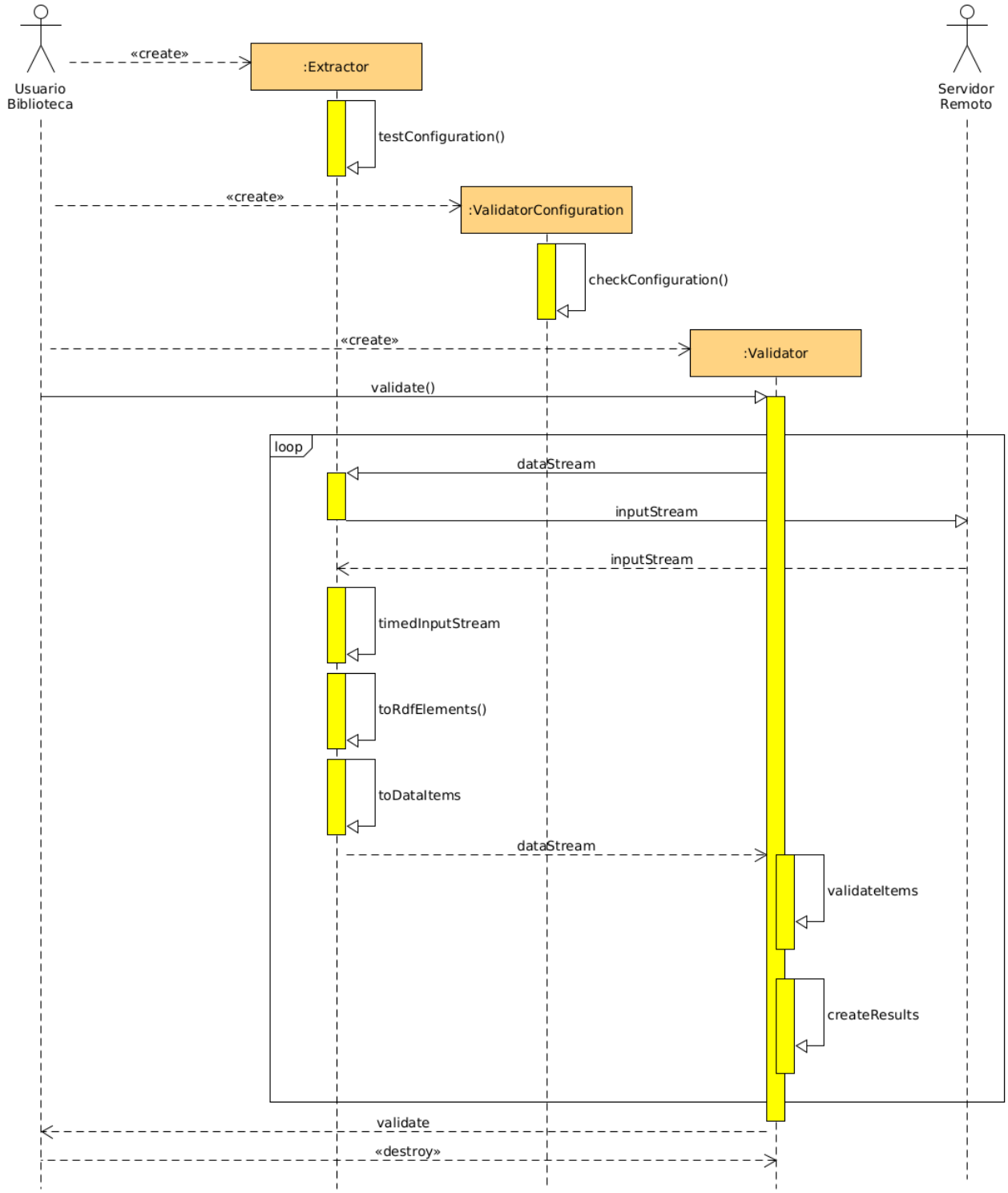


Ilustración 55. Diagrama de secuencia - Invocar una validación de flujos (biblioteca)

1. El informático o **Usuario Biblioteca** creará, programáticamente:
    a. Un **Extractor**: Configurado para acceder a un flujo de datos. En el momento de instanciación, el extractor valida su configuración y aborta la ejecución si es inválida.
    b. Un **Validator**: Configurado de forma externa mediante un objeto **ValidatorConfiguration** creado también manualmente. Al igual que con el **Extractor**, la configuración del validador es comprobada en el momento de creación.
2. El usuario invocará el método **validate** del **Validador**, iniciando el bucle principal del sistema en que el **Validador** pide nuevos datos (**dataStream**) al **Extractor**, que a su vez los solicita del **Servidor Remoto** (**inputStream**).
3. El **Extractor** recoge los datos llegados del **Servidor Remoto** y los somete a varias transformaciones antes de transferirlos al **Validador**:
    a. **timedInputStream**: Añade un temporizador a **inputStream** de forma que se detenga cuando se alcance un límite de tiempo configurado sin recibir contenido.
    b. **toRdfItems**: Convierte los contenidos de **timedInputStream**, que pueden ser instancias de cualquier tipo, a objetos RDF del dominio del programa. Para esto emplea una función de deserialización suministrada en la configuración.
    c. **toDataItems**: Transforma los resultados de **toRdfItems** en los objetos finales procesables por el validador, que contienen RDF válido o un error en caso de haberse deserializado RDF inválido.
4. El **Validador** procesa el flujo recibido del **Extractor**, primero validando cada elemento (**validateItems**) y finalmente generando los objetos-resultado (**createResults**). Los resultados de validación son retornados al usuario que los invocó para poder utilizarlos o seguir procesándolos.

5.1.2 Usuario API

5.1.2.1 *Invocar una validación de flujos*

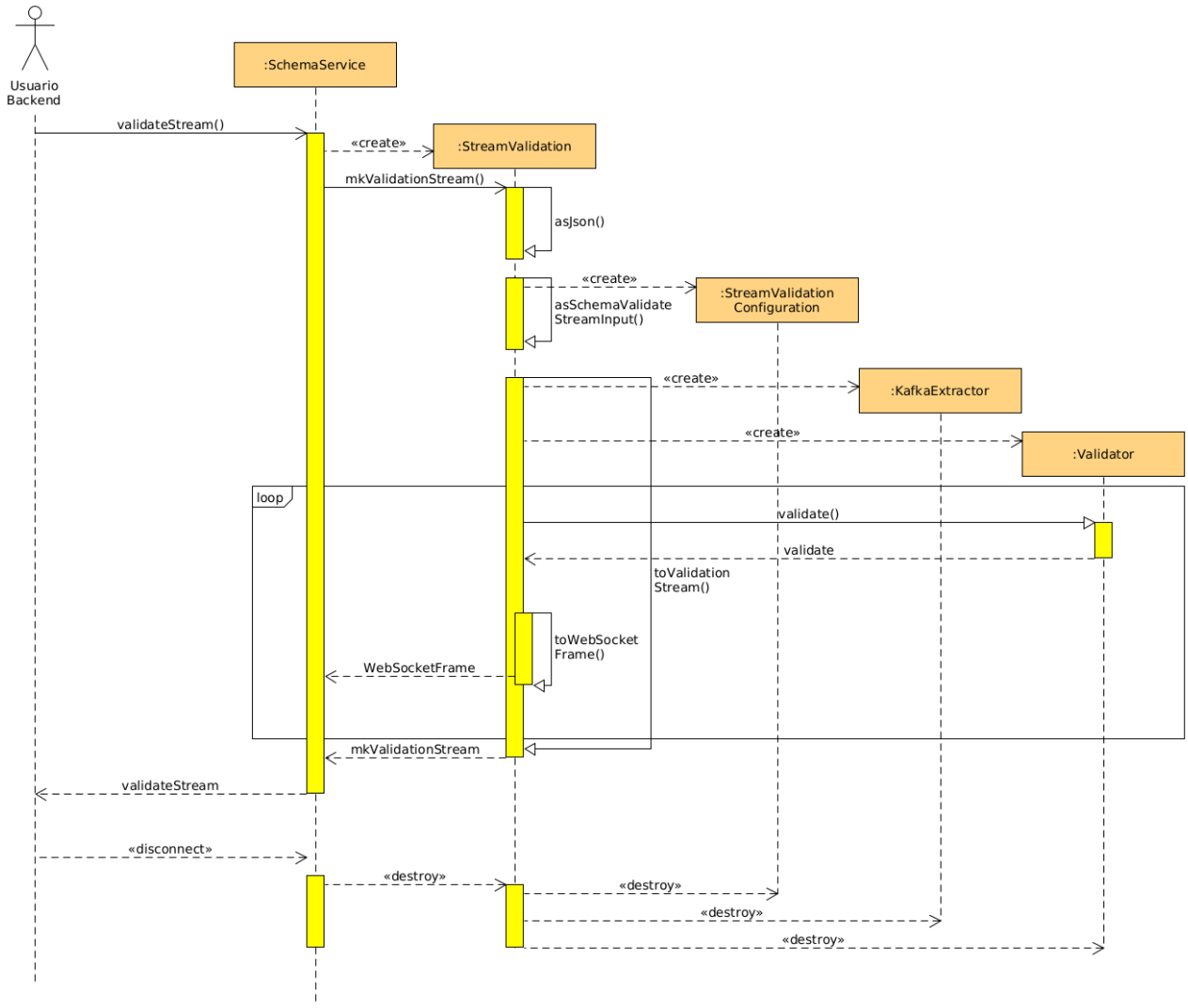


Ilustración 56. Diagrama de secuencia - Invocar una validación de flujos (API)

1. El usuario del sistema o **Usuario Backend** contactará con el API y realizará una petición al backend (**validateStream**), concretamente al **SchemaService** que reside en el servidor y se encarga de las validaciones.
2. Al detectar la petición de una validación de flujos, el **SchemaService** ejecutará la lógica en **mkValidationStream** para instanciar una **StreamValidation** que retornará el flujo de salida de validaciones.
3. La nueva **StreamValidation** recibirá los datos enviados por el usuario y los someterá a varias transformaciones para acabar devolviendo el flujo de validaciones eventualmente:
    a. **asJson**: Leer la petición de validación del usuario en formato JSON. Fuera del escenario principal, el servidor abortará la conexión si los datos recibidos no son legibles como JSON.

b. **asSchemaValidateStreamInput**: Leer los contenidos del JSON recibido por el usuario en busca de una configuración de validación. Si todo es correcto, la **StreamValidation** instanciará y almacenará una nueva **StreamValidationConfiguration** con la información, en caso contrario el servidor abortará la conexión.
c. **toValidationStream**: Crea un **KafkaExtractor** y un **Validator** de la biblioteca de validación de flujos. Se inicia el bucle del sistema ya visto anteriormente (*5.1.1.1-Invocar una validación de flujos*), con el añadido de que cada resultado salido del bucle es convertido en una respuesta *WebSockets* para el cliente (**toWebSocketFrame**).

4. Finalmente, en el momento que el usuario decide cerrar la conexión con el servidor, el **SchemaService** cancela toda la actividad relativa a la petición de validación y todos los elementos relacionados son destruidos.

Cabe destacar que, para los casos de uso adicionales del cliente web *Pausar una validación de flujos* y *Reanudar una validación de flujos*, el diagrama de secuencia sería similar al presente:

- Al momento de pausar una validación de flujos, el cliente web simplemente se desconecta temporalmente del servidor, propiciando el paso 4 explicado anteriormente.
- Así mismo, para reanudar una validación, el cliente envía una nueva petición de validación con la misma configuración que la anterior, con el cual la secuencia de eventos sería la misma, comenzando desde el principio.

5.1.2.2 *Generar permalinks (cliente)*

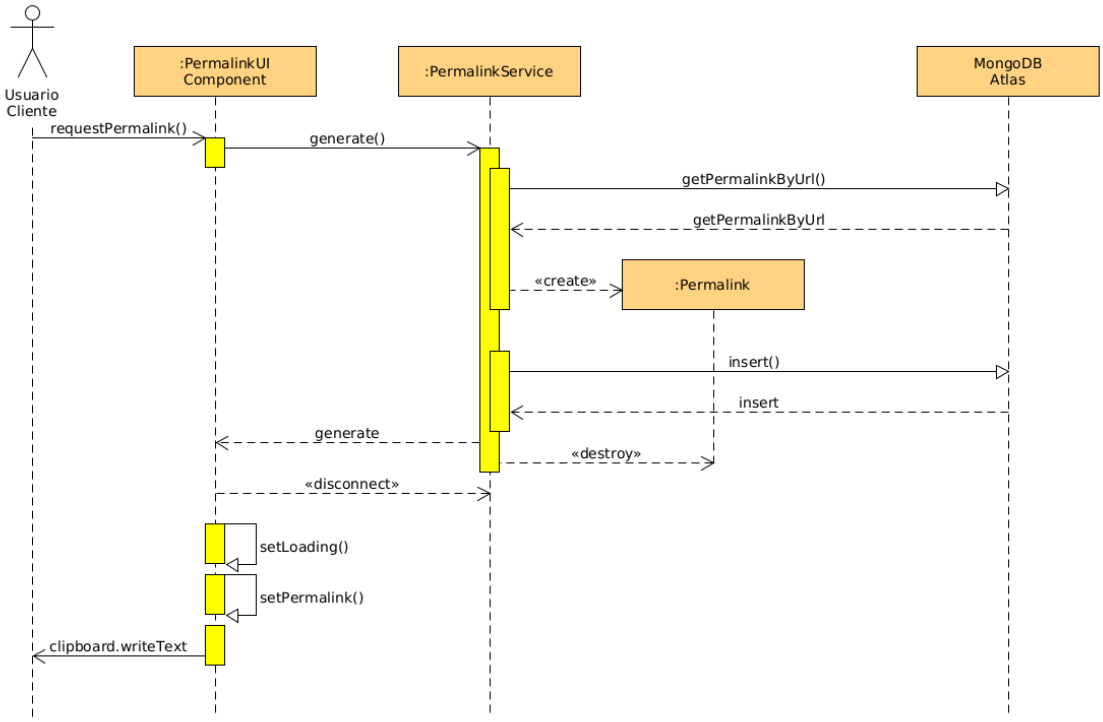


Ilustración 57. Diagrama de secuencia - Generar permalinks (cliente)

1. El usuario de la interfaz web del sistema, **Usuario Cliente**, interactúa con el componente React con la lógica de negocio para contactar con el servidor: **PermalinkUIComponent**.
2. El **PermalinkUIComponent** lanza una petición HTTP contra el servicio del API encargado de gestionar los enlaces: **PermalinkService**. La petición se hace de forma asíncrona y no detiene el comportamiento del sistema ni consume recursos hasta que el servidor responde.
3. En primer lugar, el **PermalinkService** contacta con la base de datos (**MongoDBAtlas**) para comprobar si ya existe un enlace para la URL solicitada (**getPermalinkByUrl**):
    a. Si lo hay, recibe la respuesta de la base y construye un objeto **Permalink** con el contenido de la respuesta.
    b. Si no lo hay, genera un nuevo objeto **Permalink** en memoria.
4. Posteriormente, el **PermalinkService** conecta con la base de datos para insertar (**insert**) el nuevo **Permalink** en base de datos, con su fecha de uso actualizada si ya existía y procede a:
    a. Devolver el código del enlace al componente de la aplicación web que inició la petición.
    b. Destruir de memoria la instancia de **Permalink** utilizada, ahora que está en base de datos y la petición del cliente ha concluido.
5. El **PermalinkUIComponent** cierra la conexión con el servidor y actualiza su estado (**setLoading**, **setPermalink**) para, finalmente, depositar en manos del usuario el enlace generado mediante las APIs del navegador (**clipboard.writeText**).

5.1.2.3 *Consumir permalinks (cliente)*

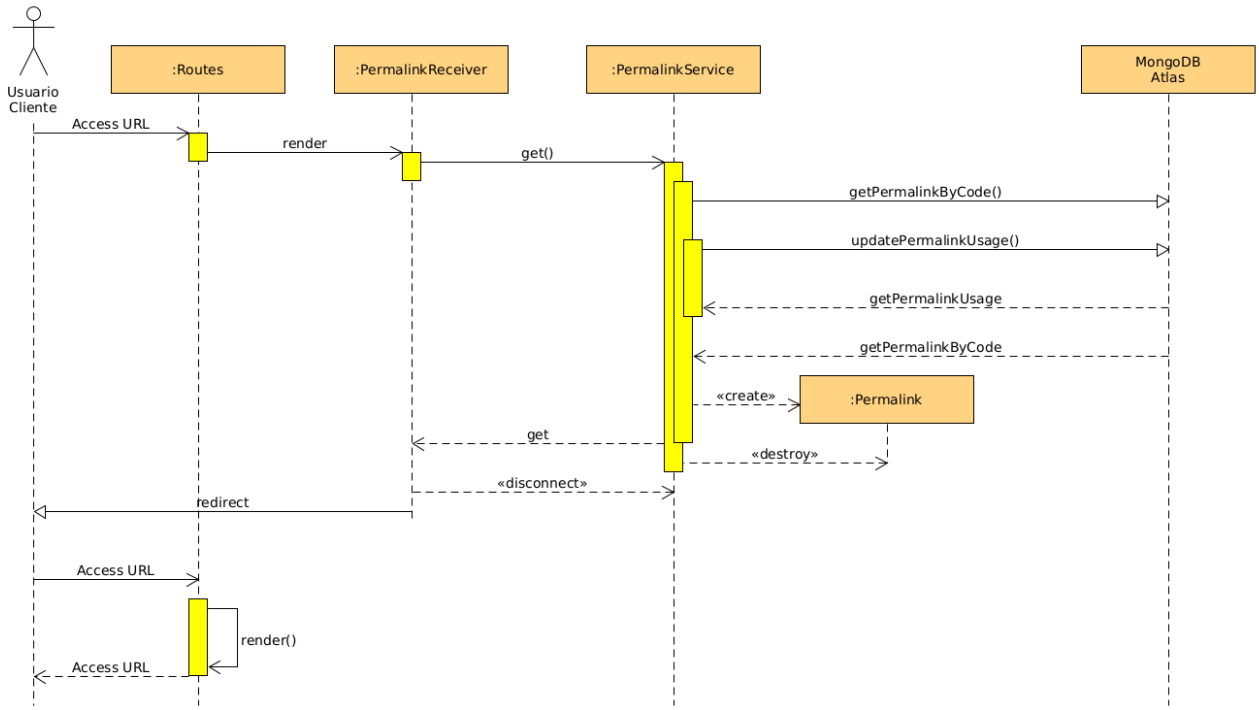


Ilustración 58. Diagrama de secuencia - Consumir permalinks (cliente)

1. El usuario de la aplicación web, **Usuario Cliente**, realiza una petición HTTP contra la aplicación React (**Access URL**), que es manejada por el componente **Routes**, encargado de renderizar los componentes solicitados.
2. **Routes** renderiza la pantalla encargada de recibir al usuario mientras se le redirige al *permalink* solicitado: **PermalinkReceiver**.
3. **PermalinkReceiver** lanza una petición HTTP contra el servicio del API encargado de gestionar los enlaces: **PermalinkService** de manera asíncrona.
4. El **PermalinkService** contacta con la base de datos (**MongoDBAtlas**) para comprobar si ya existe un enlace con el código solicitado (**getPermalinkByCode**).
    a. Si lo hay, recibe la respuesta de la base y construye un objeto **Permalink** con el contenido de la respuesta. Además, el sistema aprovechará la conexión con la base de datos para actualizar la fecha de uso del enlace solicitado (**updatePermalinkUsage**).
    b. Si no lo hay (escenario alternativo), el servidor devolverá un error.
5. Finalmente, el **PermalinkService** devuelve al componente de la aplicación web **PermalinkReceiver** la URL objetivo asociada al *permalink* y destruye de memoria la instancia de **Permalink** utilizada, ahora que la petición del cliente ha concluido.
6. El **PermalinkReceiver** redirige al navegador del **Usuario Cliente** a la URL objetivo.
7. El usuario vuelve a hacer una segunda petición HTTP contra la aplicación web, esta vez a la operación objetivo. De nuevo, el componente **Routes** procesa la petición y carga y dibuja en pantalla los componentes pertinentes (**render**).

5.2 DSI 4: DISEÑO DE CLASES

A continuación, se adjuntan los diagramas de clases propuesto en la fase de diseño del sistema y, en consecuencia, candidatos a formar parte de la implementación final del sistema. Como ya se hizo con la sección homóloga del capítulo de análisis (ver *4.6-ASI 5: Análisis de Clases*), el diseño de clases se dividirá en base a los módulos que conforman el sistema completo.

En esta ocasión los diagramas de diseño incluirán el máximo nivel de detalle representable en el formato de esta memoria, así como los nombres de atributos y métodos definitivos.

5.2.1 Módulo Validador (Comet)

5.2.1.1 *Diagrama de Clases*

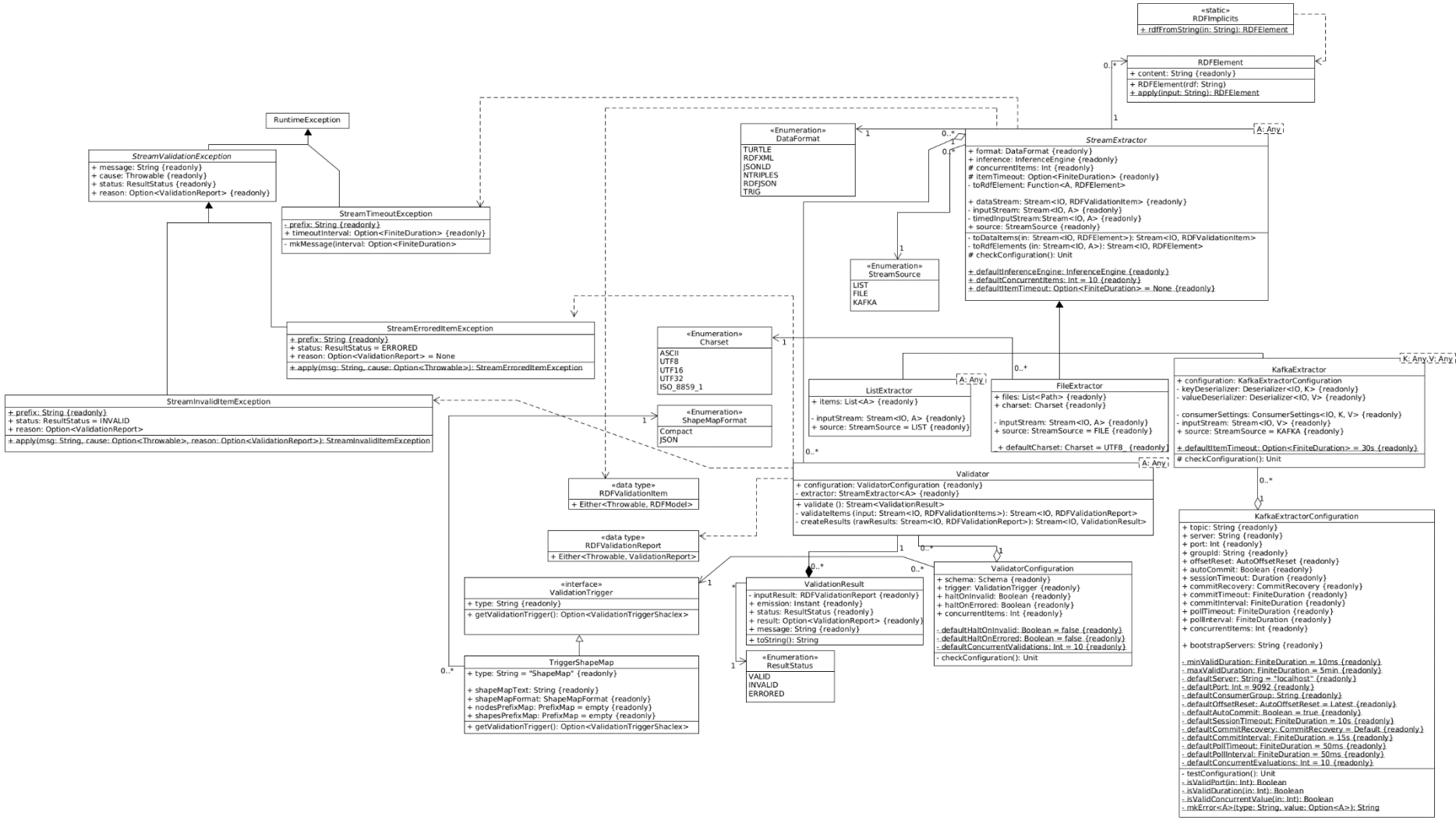


Ilustración 59. Diagrama de clases de diseño - Comet

5.2.1.2 *Descripción de Clases*

Los principales cambios respecto a la etapa de análisis son:

- Creación de la interfaz **ValidationTrigger** y su implementación **TriggerShapeMap**, adaptadores para operar con los *ValidationTriggers* provistos por *SHaclEX* desde una interfaz más cómoda para el validador.
- Adición de los tipos personalizados **RDFValidationItem** y **RDFValidationReport**, ambos envoltorios de otros tipos de datos manejados durante el procesamiento del flujo. Pueden contener un error o bien datos RDF o el resultado de una validación, respectivamente. En el caso de contener un error, este se "arrastra" durante todo el procesamiento del flujo hasta que el validador lo incluye en el resultado o excepción pertinente.
- Inclusión de campos estáticos con valores por defecto para los extractores y validadores, de manera que los usuarios no tengan que suministrar muchos de los valores que el sistema puede ofrecer de manera predeterminada.

5.2.2 Módulo API

5.2.2.1 *Diagrama de Clases*

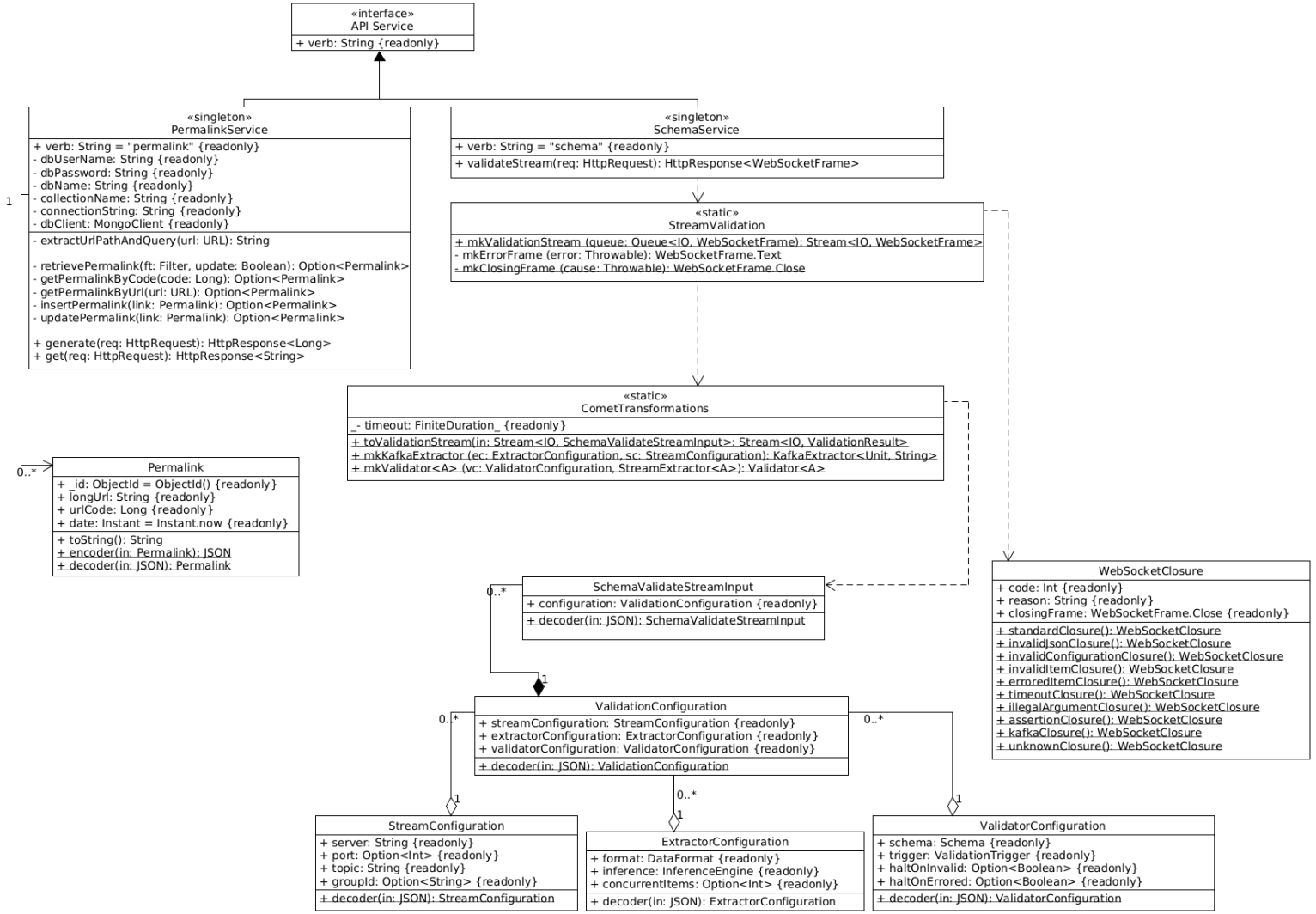


Ilustración 60. Diagrama de clases de diseño - API

5.2.2.2 *Descripción de Clases*

Los principales cambios respecto a la etapa de análisis son:

- Creación de varios métodos **encoder**/**decoder** en las clases que operan con objetos del dominio, con el fin de (de)serializar los datos en formato JSON (inteligible por los clientes) a objetos (inteligibles por la aplicación).
- Creación de la clase **WebSocketClosure** con utilidades para crear los mensajes de cierre de conexión que se envían a los clientes automáticamente para varios errores comunes. Así mismo, también hay mayor concreción en el tipo de mensajes *WebSockets* que generará cada método: *WebSocketFrame.Text* o *WebSocketFrame.Closure*.
- Creación de la clase **SchemaValidateStreamInput**: una clase intermedia para deserializar la configuración de la validación recibida por el cliente en **ValidationConfiguration**s.

5.2.3 Módulo Cliente

5.2.3.1 *Diagrama de Clases*

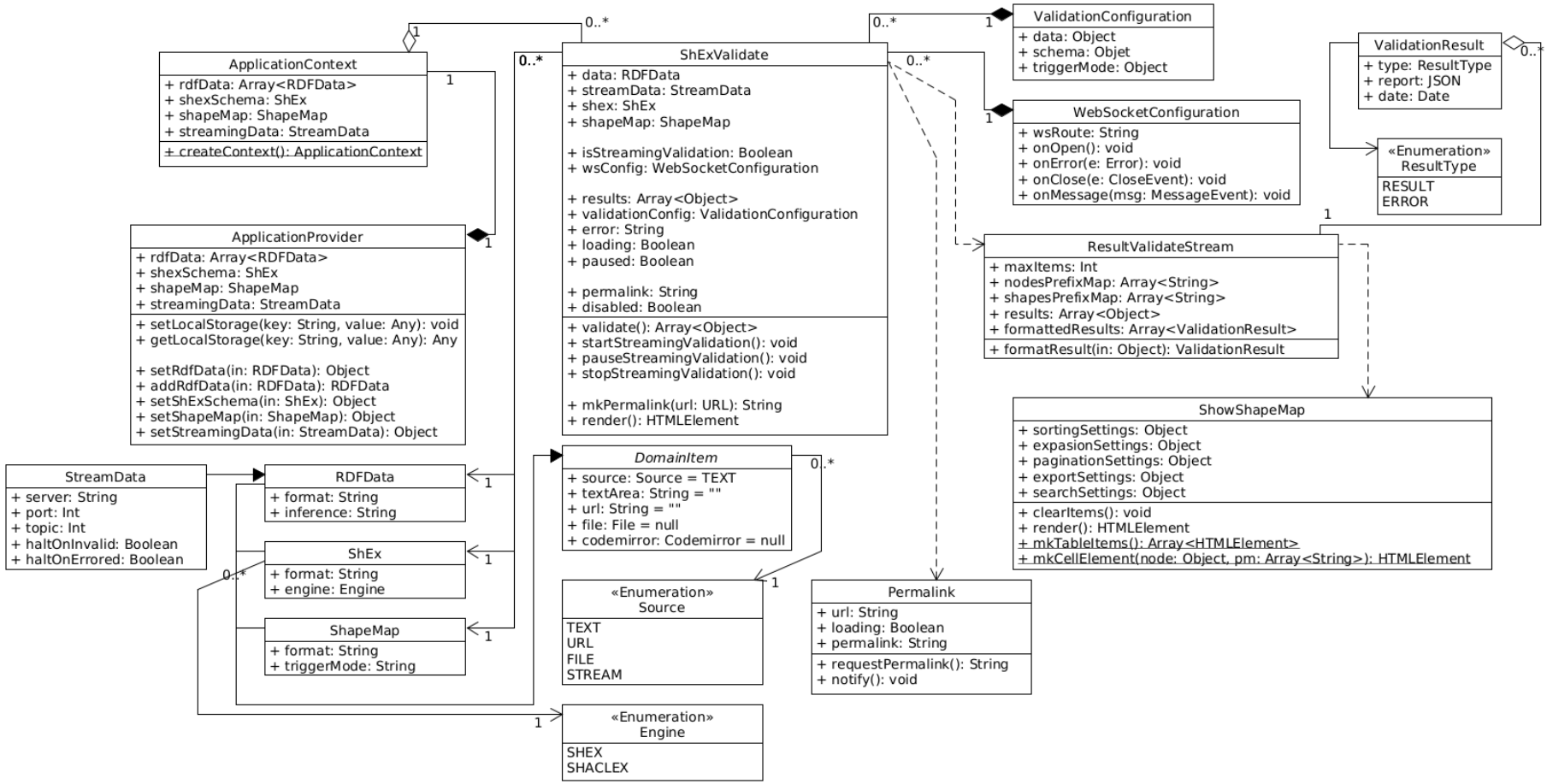


Ilustración 61. Diagrama de clases de diseño - Cliente

5.2.3.2 *Descripción de Clases*

Los principales cambios respecto a la etapa de análisis son:

- Adición del contexto (**ApplicationContext** y **ApplicationProvider**), funcionalidad de React que se va a emplear para almacenar datos localmente y mejorar la usabilidad del cliente.
- Inclusión de **Permalink**, representando un nuevo componente de la interfaz de usuario con identidad propia y la lógica de negocio para en manejos de enlaces permanentes.
- Nuevas funciones dentro de **ShowShapeMap**, encargadas de crear los elementos de la interfaz que finalmente contendrán los resultados en base a los datos de sus componentes padre.
- Enumeraciones para limitar los valores de la fuente de los datos (**Source**) o tipo de esquema (**Engine**).

5.3 DSI 5: DISEÑO DE LA ARQUITECTURA DE MÓDULOS DEL SISTEMA

Como se ha detallado en secciones anteriores del documento, el sistema final está dividido en tres módulos interconectados. A lo largo de este apartado, se presentará la arquitectura planteada para cada uno de los módulos.

5.3.1 DSI 5.1 Diseño de Módulos del Sistema

Para el diseño de cada módulo, se desarrollarán los diagramas de paquetes y de despliegue, acotados al módulo pertinente, con la finalidad de obtener una visión arquitectónica del sistema a alto nivel.

5.3.1.1 *Módulo Validador (Comet)*

El módulo validador se ha desarrollado según el diagrama de clases visto en *5.2.1-Módulo Validador (Comet)*. Es una biblioteca de código que sigue el modelo autodenominado como *extractor-validador*.

Construir el módulo implica el consumo de flujos de datos siguiendo el paradigma funcional y el marco de trabajo proporcionado por *Fs2*, que se resume en definir un origen del flujo de datos y someterlo a varias transformaciones, en este caso hasta que el flujo de entrada se ha convertido en un flujo de resultados que ofrecer a los clientes de la biblioteca. El modelo *extractor-validador* se compone de:

- **Extractores**: Los extractores del módulo son clases que siguen una misma interfaz proporcionada por la biblioteca. Estos bloques de código pueden ser configurados para contactar con una fuente de datos y a continuación, leer e interpretar los datos del origen hasta generar un flujo de RDF listo para alimentar un validador.
- **Validadores**: El núcleo de la biblioteca. Un validador es configurado con un esquema *Shape Expressions* y una serie de condiciones de parada. A continuación, recibe un flujo de datos RDF desde un extractor y lo convierte en un flujo de resultados. Cabe destacar que el extractor de origen de los datos se encuentra totalmente abstraído, de manera que la labor del validador es independiente al origen de los datos y los extractores puedan intercambiarse de forma transparente.

5.3.1.1.1 Patrones de diseño

Dado que el módulo validador es el único del conjunto que se ha desarrollado partiendo de cero, se ha planificado el uso de los siguientes patrones de diseño en la construcción de este:

5.3.1.1.1.1 Strategy

El patrón *strategy* o *policy* tiene la finalidad de, dada una familia de algoritmos, encapsular la funcionalidad de cada uno dentro de su propia "estrategia" (normalmente su propia clase de código) y hacer cada estrategia intercambiable respecto al resto. De esta manera, el cliente que desee utilizar

un algoritmo de la familia puede decidir cuál de ellos emplear de forma dinámica y sin afectar a su implementación.

Para la biblioteca de validación de flujos, la familia de algoritmos son los extractores de datos, puesto que cada uno supone una estrategia que el validador empleará para obtener la información del flujo de entrada. En este caso, todos los extractores implementan una interfaz común *StreamExtractor*, para la que deben especificar su flujo de datos de origen y cómo obtenerlo.

Cada validador contendrá un extractor de tipo *StreamExtractor*, cuya funcionalidad invocará cuando necesite nuevos datos. De esta forma, la lógica de negocio que se ejecutará en el programa dependerá de la implementación o "estrategia" de extractor proporcionada al validador por el cliente.

Finalmente, cabe destacar que este patrón de diseño es el que va a permitir el desarrollo y ampliación de la biblioteca, puesto que facilita a los usuarios una interfaz común a partir de la cual desarrollar sus propios extractores para cualquier flujo de origen. El desarrollo del proyecto plantea implementar un extractor de datos de *Apache Kafka*, así como para listas y ficheros a modo de ejemplos complementarios.

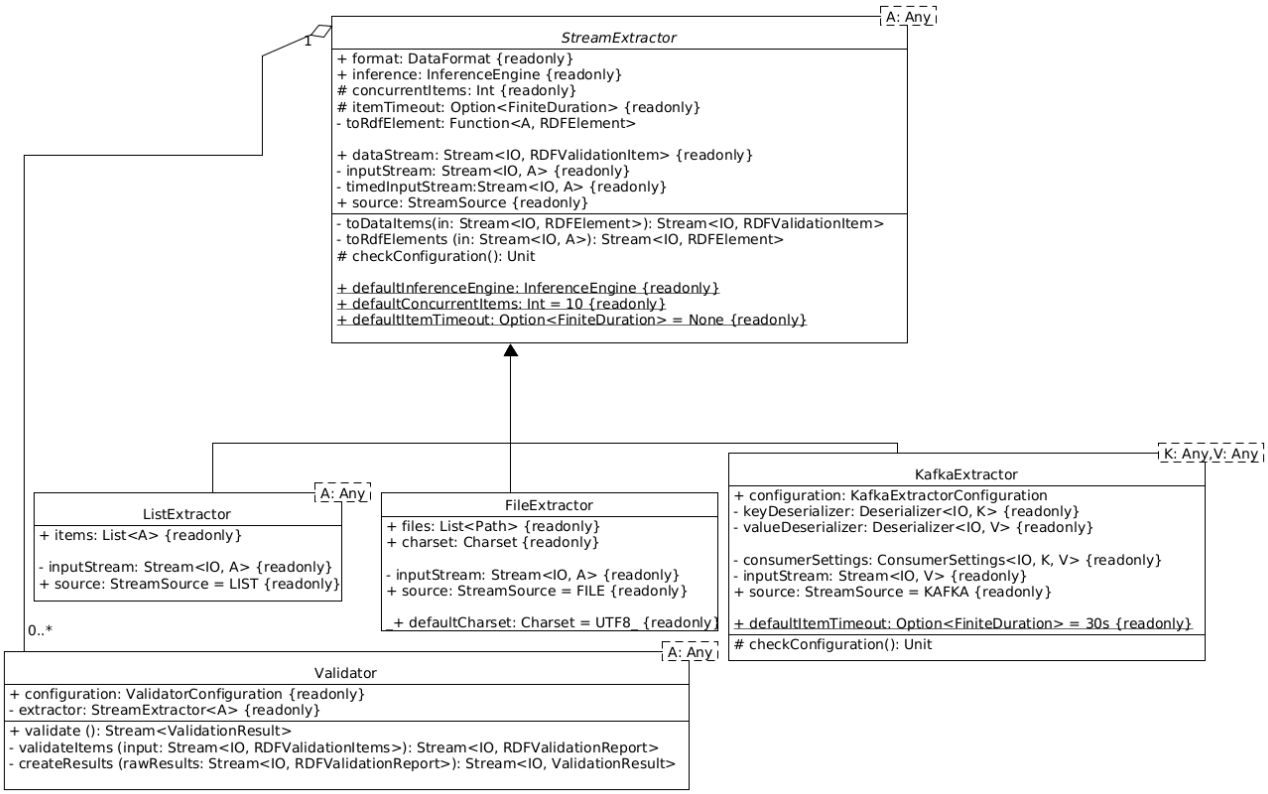


Ilustración 62. Patrones de diseño - Strategy (Comet)

5.3.1.1.1.2 Adapter

La finalidad del patrón *adapter* es convertir una interfaz o API expuesta por un módulo de terceros sobre el cual no tenemos control a otra interfaz apropiada para su consumo por nuestra aplicación. Este patrón permite compaginar la lógica de negocio de varias clases a priori incompatibles.

En el caso concreto de este subsistema, se empleado el patrón *adapter* para manejar de forma más práctica los objetos del dominio (esquemas *ShEx* y *ShapeMaps*) expuestos por la biblioteca de validación *SHaclEX* y hacerlos compatibles con los que espera recibir la biblioteca de validación de flujos.

En concreto, la biblioteca *Comet* pone facilidades a sus usuarios para definir un nuevo *ShapeMap* dado el texto que contiene y su sintaxis (clase *TriggerShapeMap*). Internamente y de forma invisible a los clientes, los objetos *TriggerShapeMap* usan la información que contienen para crear su equivalente en *SHaclEX*, que finamente será utilizado en las validaciones. Todas las clases adaptadoras implementan la interfaz *ValidationTrigger*, diseñada de cara a futuras ampliaciones.

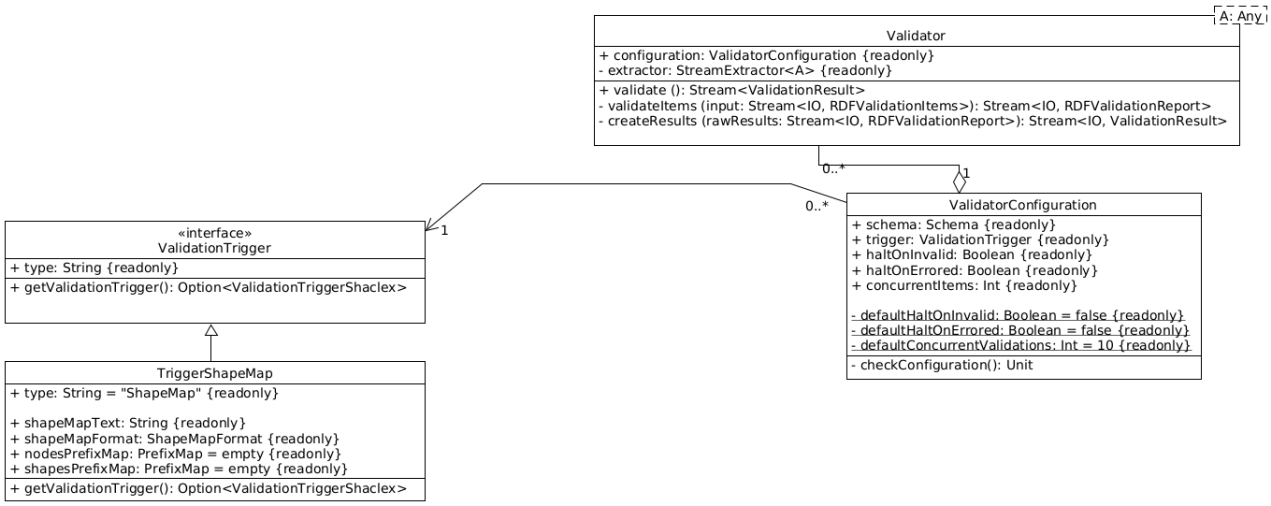


Ilustración 63. Patrones de diseño - Adapter (Comet)

5.3.1.1.1.3 Monad

Puede considerarse como patrón de diseño el uso de mónadas, una forma de trabajo propia de la programación funcional. Las mónadas sirven para operar con funciones puras teniendo en cuenta sus posibles efectos colaterales. De esta manera, permiten combinar varias funciones y sus resultados para construir un resultado final de forma segura **[98]**.

Una mónada debe proporcionar:

- Un tipo “envoltorio” (*wrapping type*), que contendrá los resultados de cada función así como el resultado final hasta el momento de la interpretación de este, que se retrasará puesto que puede generar efectos secundarios.
- Dos operaciones: una para envolver un valor dentro del tipo de la mónada y otra para componer dos valores envueltos de la mónada en un único valor resultado de operar ambos.

En el caso concreto del programa, se empleará el framework de programación funcional *cats-effect*, que proporciona la mónada *IO*, que se puede ver presente a lo largo del diagrama de clases del sistema. Esta mónada ofrece un tipo "*IO*" para envolver toda clase de operaciones que puedan suponer un efecto secundario en el código y los mecanismos para combinar, paralelizar y coordinar estas operaciones.

5.3.1.1.2 Diagrama de paquetes

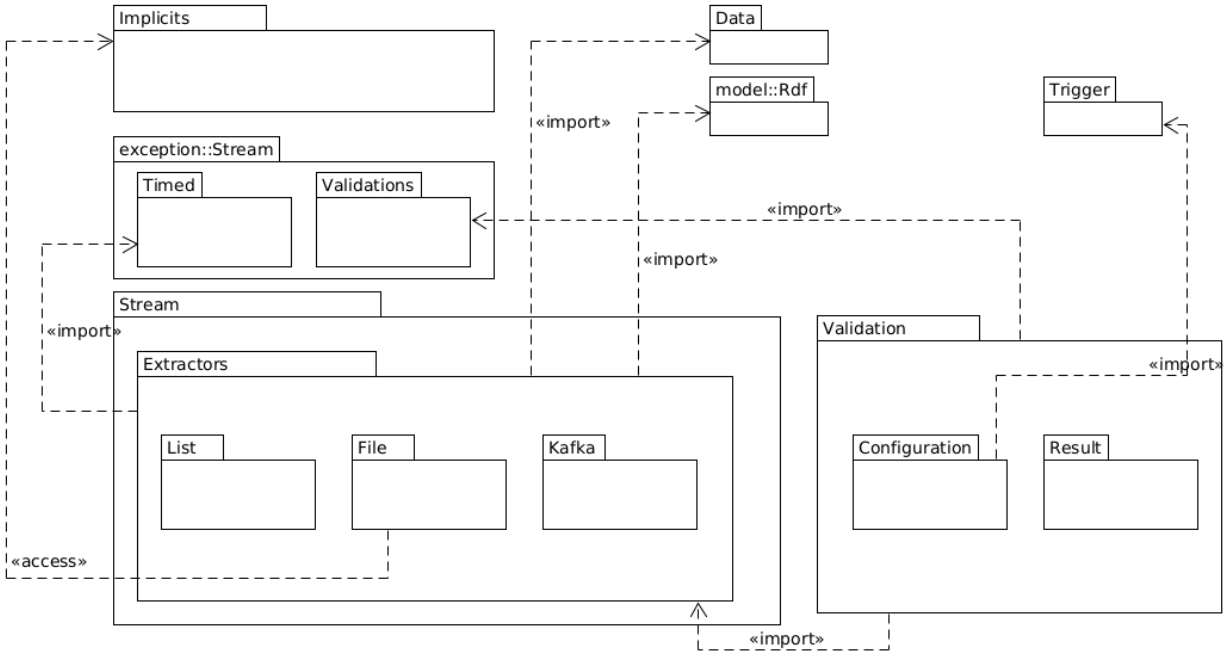


Ilustración 64. Diagrama de paquetes - Comet

5.3.1.1.2.1 Validation

Contiene la clase principal de la biblioteca: el validador, y los paquetes relacionados:

- **Configuration**: Clases encargadas de representar la configuración de un validador. En la configuración se definen el esquema y *ShapeMap* (*trigger*) de la validación.
- **Results**: Clases para la representación de los resultados de cada validación y su estado final.

Importa la funcionalidad del paquete **Extractors** de forma que el validador pueda acceder a los extractores y consumir elementos.

5.3.1.1.2.2 Stream

Clases y funciones para la conexión con las fuentes del flujo, la recepción de los datos y su transformación inicial en instancias **RDF** del dominio. Contiene la interfaz general para la creación de extractores.

- **Extractors**: Paquete donde se ubican las distintas implementaciones de extractores que viene con el sistema base. Cada implementación cuenta con su paquete propio.

5.3.1.1.2.3 Exception::Stream

Paquete con las utilidades para el manejo de errores en el sistema mediante excepciones personalizadas. Se divide en sub-paquetes en función del tipo de error al que se referencia.

- **Timed**: Excepciones personalizadas para el manejo de eventos relacionados con la inactividad del flujo de datos de entrada. Su funcionalidad es empleada desde **Extractors** para definir el período máximo de inactividad que permitirá el extractor.
- **Validations**: Excepciones personalizadas para el manejo de errores surgidos durante la validación del flujo de entrada. Es empleada desde el validador en **Validation** para emitir los errores pertinentes cuando sucedan.

5.3.1.1.2.4 Trigger

Utilidades empleadas para adaptar los *triggers* de *SHaclEX* a los empleados desde la biblioteca de validación de forma transparente al validador.

5.3.1.1.2.5 Data

Tipos de datos RDF aceptados por la aplicación. Accedidos desde **Extractors** de cara a definir el tipo de datos que el extractor espera recibir.

5.3.1.1.2.5.1 model::RDF

Datos RDF representados en el dominio de la aplicación tras ser deserializados por un extractor y antes de ser procesados como entidades *SHaclEX* por el validador.

5.3.1.1.2.6 Implicits

Serie de utilidades puestas a disposición del compilador de forma implícita, lo que en *Scala* implica no tener que ser pasadas como parámetros manualmente. Contiene implementaciones para deserializar cadenas de texto en objetos RDF, que serán usadas por el extractor de ficheros, así como por aquellos que lo requieran.

5.3.1.1.3 Diagrama de despliegue

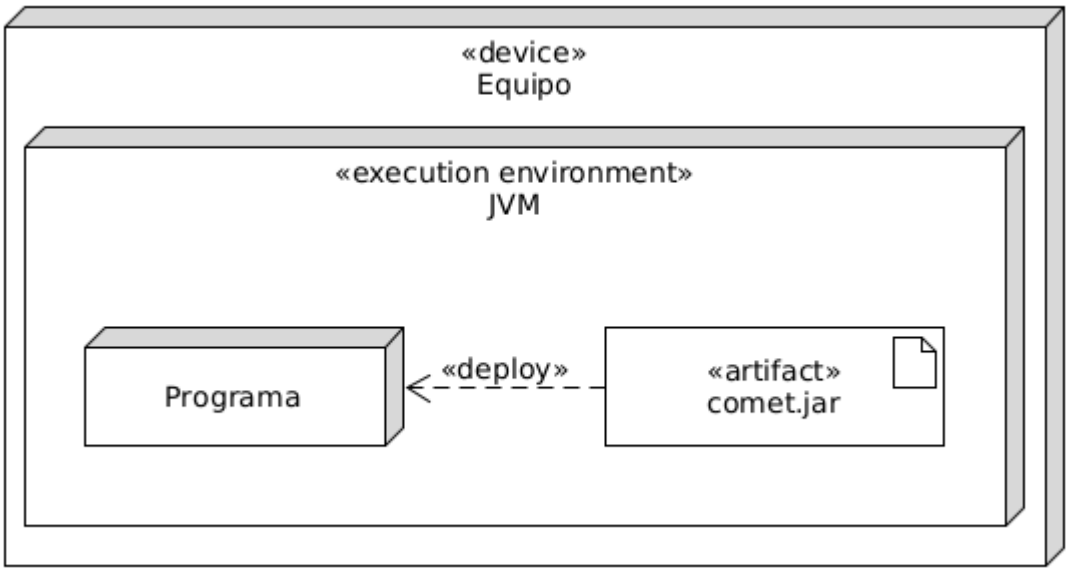


Ilustración 65. Diagrama de despliegue - Comet

Al ser una biblioteca de código, el despliegue como tal de este subsistema está supeditado a ser incluido como dependencia en un proyecto de terceros.

5.3.1.2 *Módulo API*

Como se especificó en *4.4.1.2-Subsistema API*, este módulo está construido para ser desplegado en un servidor web y ofrecer un API REST y un servicio de mensajería *WebSockets*.

El API REST es ampliada respecto a la ya existente para permitir la solicitud de validaciones de flujos y la gestión de *permalinks* a los clientes. Para esto, la arquitectura empleada en el servidor está constituida en torno al marco funcional ofrecido por *http4s*, el *framework* de desarrollo de esta parte del sistema. Si bien la terminología y estructura de clases difiere de otras tecnologías más comunes (véase: *Spring*, *Faces*, etc.), las capas en que se cimenta la arquitectura son similares:

1. **Presentación**: Capa de entrada para todos los clientes del API que implementa, a través de diversos servicios, todas las rutas disponibles. Así mismo, la capa de presentación tiene las siguientes funciones:
    a. Definir las rutas del sistema mediante el *DSL* proporcionado por *http4s*.
    b. Verificar, mediante una serie de funciones decodificadoras, que la información recibida en las solicitudes del cliente es válida y convertirla en objetos del dominio para ser empleados en la capa de servicio.
    c. Invocar a la capa de servicio y construir las respuestas a los clientes con los datos retornados por esta.
2. **Servicio**: Lógica de negocio del sistema. En este caso, contiene las clases del dominio y las funcionalidades necesarias para cada una de las operaciones expuestas por el API:
    a. Contactar con un servidor externo para procesar y crear los resultados de una validación de flujos.
    b. Manejar los *permalinks* como objetos del dominio e invocar a la capa de persistencia cuando sea necesario almacenarlos o recuperarlos de base de datos.
3. **Persistencia**: Implementa el acceso a base de datos y todas las peticiones y/o consultas requeridas en el funcionamiento del sistema.

5.3.1.2.1 Diagrama de paquetes

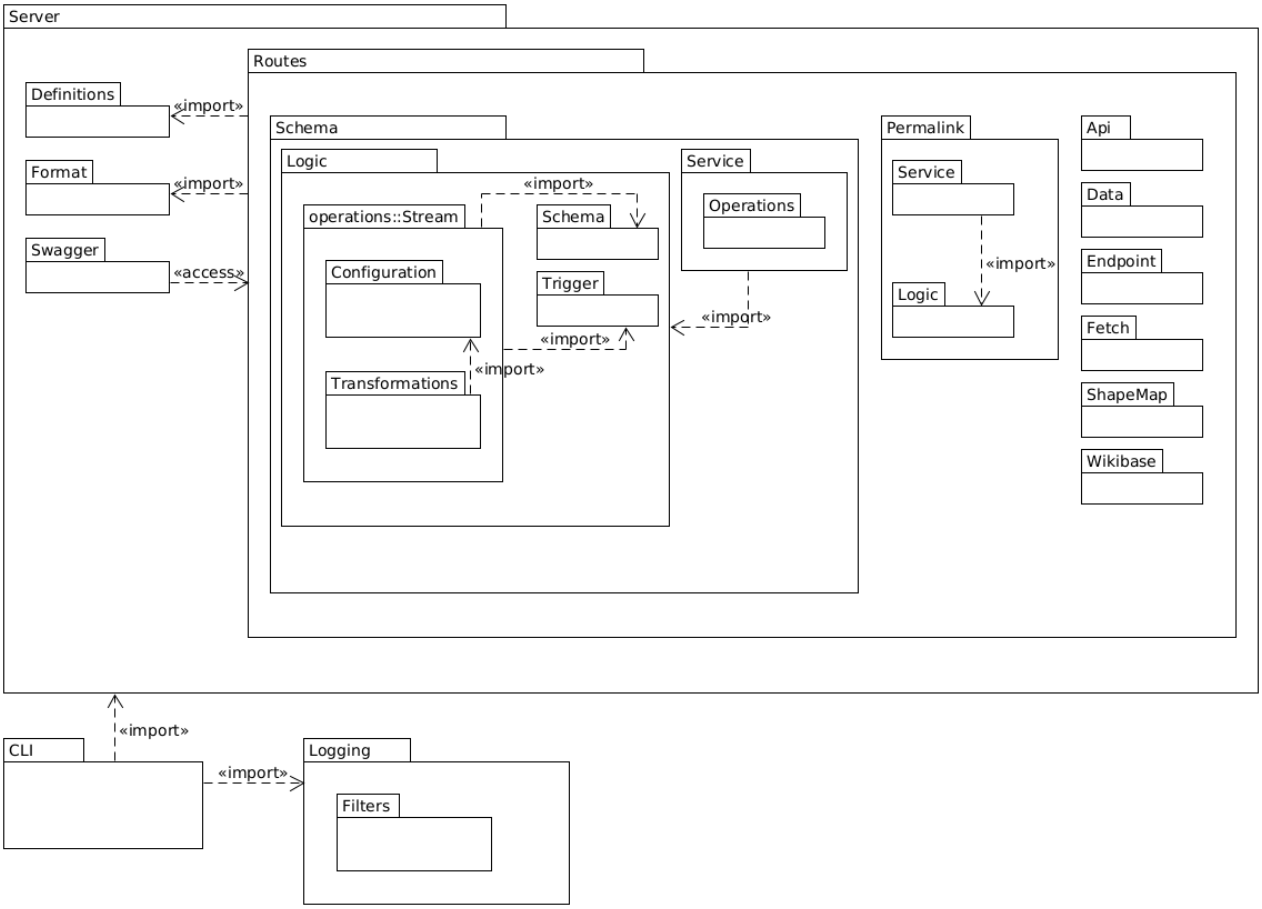


Ilustración 66. Diagrama de paquetes - API

5.3.1.2.1.1 CLI

El API es desplegada en un equipo a través de un ejecutable y acepta varios argumentos introducidos a través de línea de comandos (puerto expuesto, verbosidad de los *logs* en consola, etc.). Este paquete contiene las utilidades para definir, leer y validar los parámetros introducidos desde la terminal, así como configurar e inicializar el servidor.

5.3.1.2.1.2 Logging

Clases para la configuración del sistema de registros de la aplicación. Para la ejecución del sistema desde línea de comandos, permite adaptar el gestor de *logs* a los parámetros del usuario en base filtros personalizados (**Filters**).

5.3.1.2.1.3 Server

Dentro del API, es el subsistema principal encargado de ejecutar el servidor HTTP (y *WebSockets*) de la aplicación. Contiene varias utilidades generales:

- **Definitions**: Definiciones o constantes globales de utilidad para los servicios.

- **Format**: Clases para definir todo tipo de formatos (RDF, esquemas, etc.) con los que opera el servidor, así como para manipularlos (de)codificarlos entre JSON y objetos del dominio.
- **Swagger**: Utilidades estáticas definidas para ayudar a que la herramienta *Rho* pueda generar la documentación del API automáticamente.

5.3.1.2.1.3.1 Routes

Paquete con todas las rutas del API, incluyendo su lógica de presentación, negocio y persistencia dividida por servicios.

5.3.1.2.1.3.1.1 Schema

Clases y funcionalidades del servicio de validación de datos RDF expuesto por el API, ampliado para soportar la validación de flujos.

- **Service**: Rutas expuestas por el servicio y decodificación de los mensajes recibidos del cliente.
- **Logic**: Lógica de negocio, con las definiciones de configuración para una validación de flujos y las transformaciones para pasar de un flujo de entrada RDF a un flujo de salida *WebSockets*. También contiene los paquetes **Schema** y **Trigger**, adaptadores para facilitar la validación con SHaclEX.

5.3.1.2.1.3.1.2 Permalink

Clases y funcionalidades del servicio de manejo de enlaces expuesto por el API.

- **Service**: Rutas expuestas por el servicio y decodificación de los mensajes recibidos del cliente.
- **Logic**: Clases del dominio para representar enlaces y contactar con la base de datos.

5.3.1.2.1.3.1.3 Otras

El resto de las rutas del sistema están representadas pero no se entra a más nivel de detalle puesto que no son relevantes para el proyecto. Sus rutas serán documentadas (*Open API Specification*), así como su código (*Scaladoc*).

5.3.1.2.2 Diagrama de despliegue

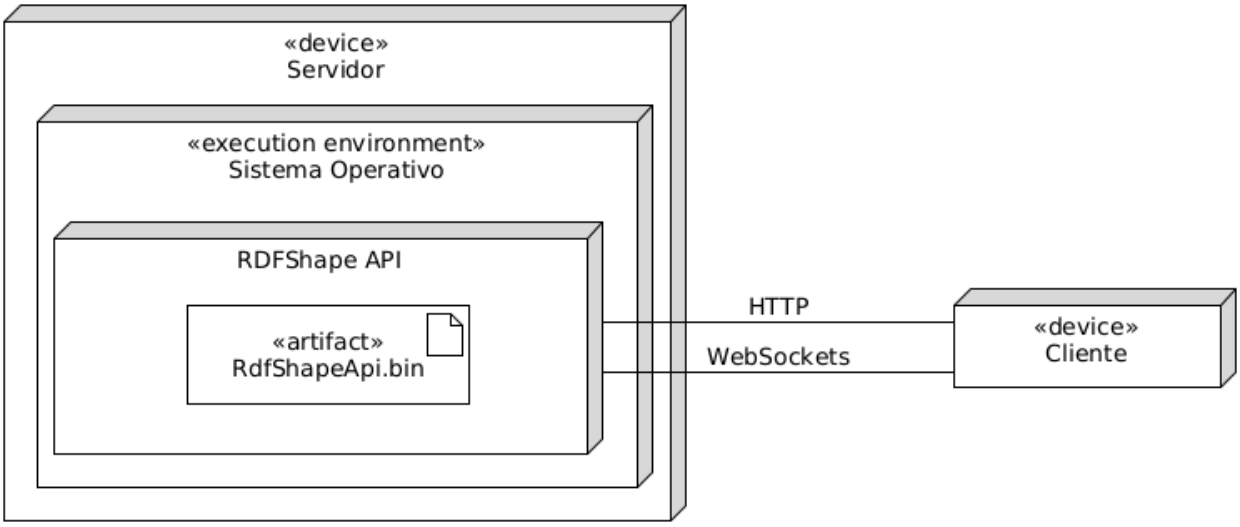


Ilustración 67. Diagrama de despliegue - API

El módulo API es construido a un ejecutable binario independiente del servidor objetivo en que se vaya a desplegar. Así mismo, empleará la biblioteca de validación de flujos como una dependencia y los protocolos HTTP y *WebSockets* para comunicarse con sus clientes.

5.3.1.3 *Módulo Cliente*

El módulo cliente consta de una aplicación web desarrollada en *React* y codificada en *JavaScript*.

5.3.1.3.1 Diagrama de paquetes

Pese a no usarse un lenguaje tipado que soporte estructuras de clases e interfaces, a lo largo de la memoria se desarrollado la arquitectura de clases esperada para el cliente basándose en los componentes *React* que se van a ampliar o construir para el sistema. El mismo razonamiento aplica para elaborar el diagrama de paquetes a alto nivel del módulo.

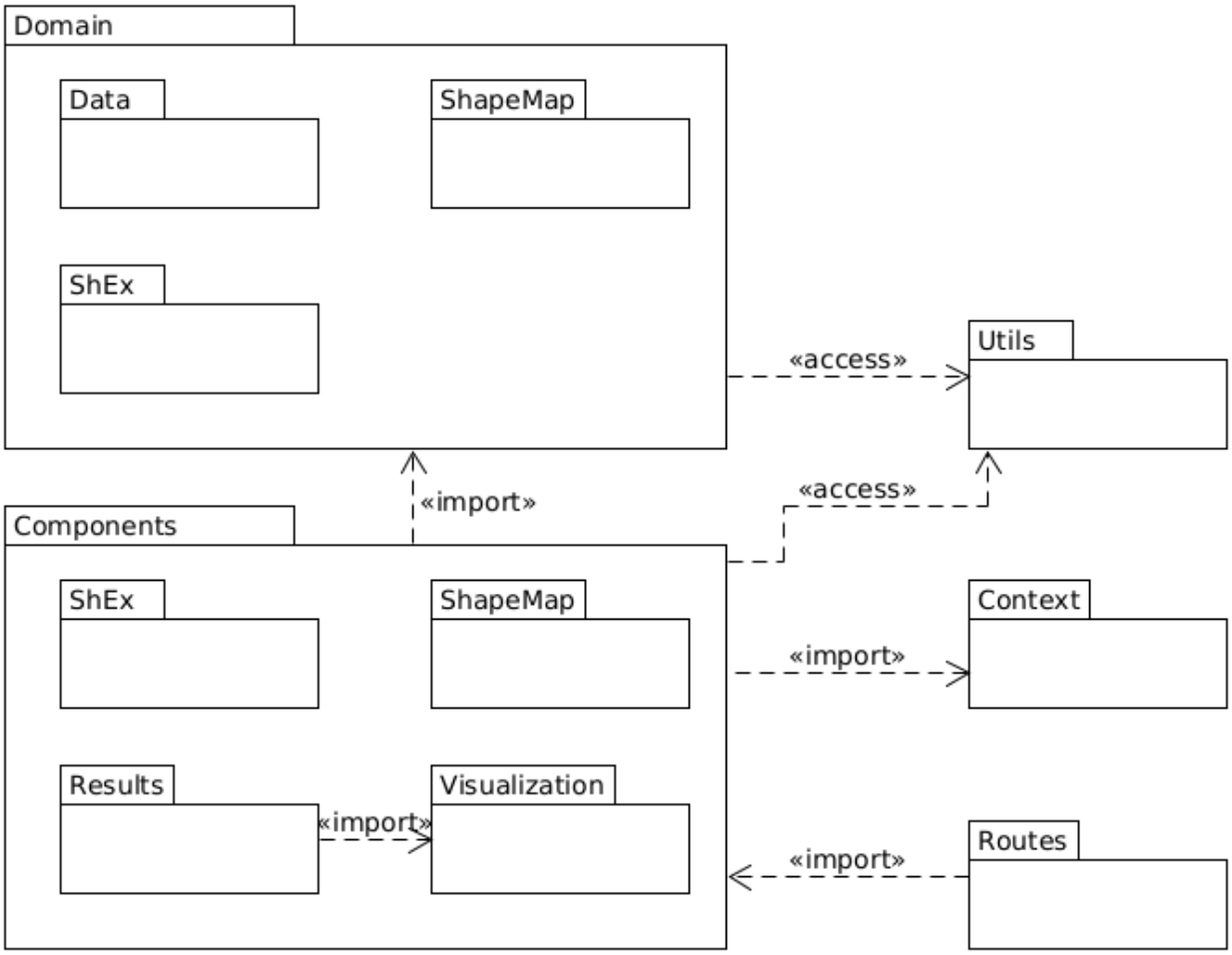


Ilustración 68. Diagrama de paquetes - Cliente

5.3.1.3.1.1 Routes

Al igual que con las rutas del API, el componente *Routes* es el punto de entrada de todas las peticiones de los usuarios de la aplicación web. En función de la dirección solicitada, el componente se encargará de renderizar el resto de los componentes por debajo de él.

5.3.1.3.1.2 Context

Funcionalidades del contexto de *React*, desarrollado para la aplicación web con la finalidad de almacenar el estado del cliente mientras el usuario navega entre páginas. Los componentes principales de cada herramienta pueden acceder al contexto para inicializarse con el último estado guardado.

5.3.1.3.1.3 Domain

Contiene las principales "clases" del dominio de la aplicación, empleadas para encapsular datos, esquemas, *ShapeMaps*, etc.

5.3.1.3.1.4 Components

Componentes *React* encargados de dibujar la interfaz gráfica final de la aplicación web. Se encargan de gestionar las interacciones con el usuario así como de mostrar de forma amigable y accesible los datos del dominio que contienen en su interior.

5.3.1.3.1.5 Utils

Utilidades generales para para operar con datos, contactar con el API y otras constantes u operaciones simples accedidas frecuentemente.

5.3.1.3.2 Diagrama de despliegue

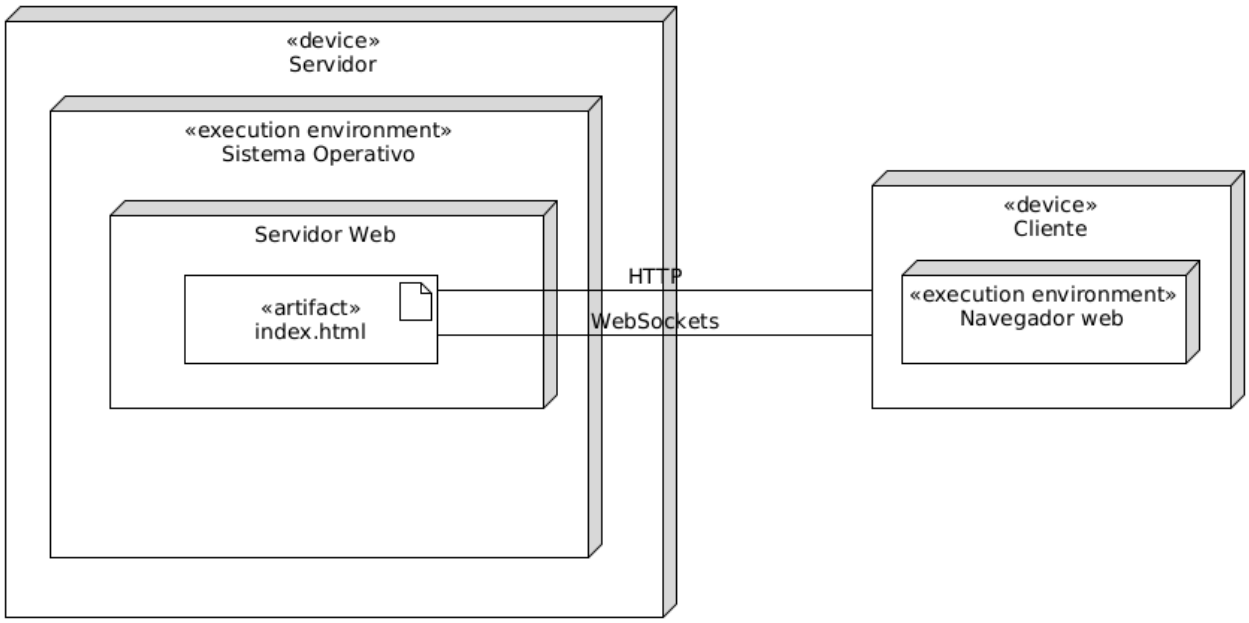


Ilustración 69. Diagrama de despliegue - Cliente

El módulo cliente se despliega como una aplicación web conformada por un conjunto de ficheros *HTML*, *CSS* y *JavaScript* producidos por *React* tras un proceso de compilación. Dichos ficheros son posteriormente alojados y servidos por un servidor de páginas web (*Apache*, *NGINX*, etc.) que los pone a disposición de los usuarios.

5.3.2 DSI 5.2 Diseño de Comunicaciones entre Módulos

Para esta sección, se definen definitivamente los mecanismos de comunicación entre los módulos anteriores. Dicha comunicación se está fuertemente basada en las interfaces descritas en la fase de análisis (ver *4.4.2-Descripción de los Interfaces entre Subsistemas*). Adicionalmente, se adjunta el diagrama de despliegue global para todo el sistema final tal y como se ha planeado para la infraestructura del cliente WESO:

- Todos los módulos de *RDFShape* comparten un mismo entorno de ejecución. Nótese que esto no es una restricción necesaria para el despliegue del sistema, puesto que los módulos interactúan a través de *HTTP*, pudiendo comunicarse a través de internet en lugar de la red local, de darse el caso.
- El sistema al completo es desplegado mediante *Docker* a través de tecnología de contenedores, siguiendo las directrices redactadas para la arquitectura del sistema (ver *2.2.3.2.1.1-Arquitectura final del despliegue*).

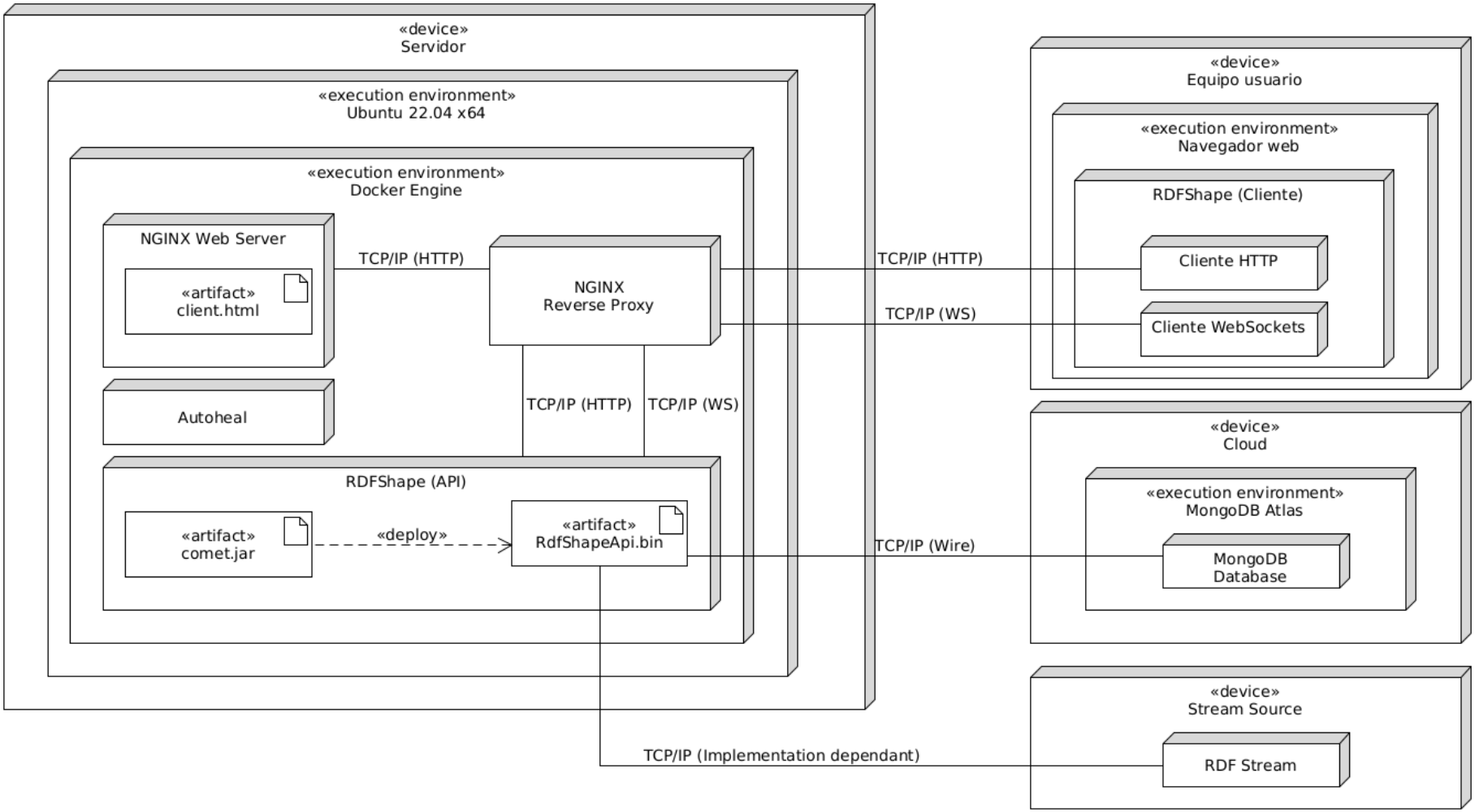


Ilustración 70. Diagrama de despliegue - Sistema

5.3.3 DSI 5.3 Revisión de la Interfaz de Usuario

Durante la fase de diseño, la interfaz de usuario no atraviesa apenas cambios respecto a la descrita en el capítulo de análisis (ver *4.7-ASI 8: Definición de Interfaces de Usuario*). Este se debe, principalmente, a lo acotado de las interfaces de usuario del sistema final, que se limitan a una pantalla dentro de una aplicación web ya existente.

Para el resto de la sección, se definen con mayor nivel de detalle que en el análisis algunos de los aspectos de las interfaces candidatas al sistema final.

5.3.3.1 *Descripción de las Interfaces Finales*

5.3.3.1.1 Datos Obligatorios y Opcionales

La única vía para los usuarios de introducir datos en el sistema gráficamente es a través del formulario presente en el cliente web. Para este formulario se solicitará la siguiente información, según su opcionalidad (o no):

- **Datos RDF**:
  - Servidor (**obligatorio**).
  - Puerto (**obligatorio**).
  - Topic (**obligatorio**).
  - Detener ante inválidos (**obligatorio**, no acepta una entrada vacía).
  - Detener ante errores (**obligatorio**, no acepta una entrada vacía).
  - Formato (**obligatorio**, no acepta una entrada vacía).
  - Inferencia (**obligatorio**, no acepta una entrada vacía).
- **Esquema de validación *ShEx***
  - Contenido (**obligatorio**, sea por texto, URL o fichero).
  - Formato (**obligatorio**, no acepta una entrada vacía).
- ***ShapeMap***
  - Contenido (**obligatorio**, sea por texto, URL o fichero).
  - Formato (**obligatorio**, no acepta una entrada vacía).

5.3.3.1.2 Diseño de los Mensajes del Sistema

Los mensajes que el sistema puede emitir pueden estar relacionados con el transcurso de la validación o no. En cualquier caso, los mensajes contemplados para el cliente nunca implican la redirección a otras páginas y siempre seguirán el código de colores especificado en *4.7.3-Descripción del Comportamiento de la Interfaz*.

5.3.3.1.2.1 Mensajes de validación

Cuando el sistema necesite proporcionar información al usuario relacionada con la validación, incluyendo el estado de la validación de flujos y los errores que la hayan interrumpido, lo hará a través de alertas estáticas.

Dichas alertas se presentarán siempre en la parte superior (más prominente) del área de resultados y serán permanentes, es decir, no desaparecerán salvo que el usuario interactúe con el sistema y modifique su estado. Adicionalmente, dichas alertas presentarán un título descriptivo que podrá ir acompañado de un cuerpo desplegable si el mensaje incluye información adicional.

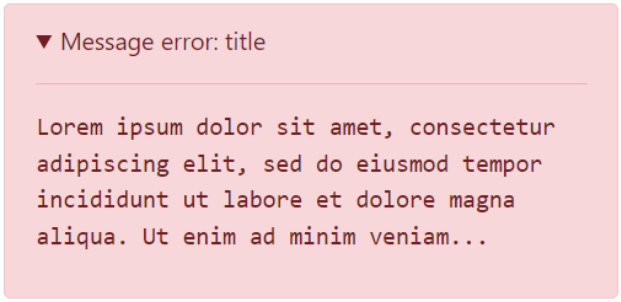


Ilustración 71. Diseño de mensajes del sistema - Error de validación

5.3.3.1.2.2 Notificaciones

En los casos en que el sistema deba notificar al usuario de otro tipo de informaciones secundarias, como es el caso de la generación de *permalinks*, lo hará mediante mensajes de tipo *Toast*. Estos mensajes se mostrarán en la parte inferior derecha de la interfaz y serán volátiles, lo que implica que desaparecerán automáticamente pasado un tiempo.

Ilustración 72. Diseño de mensajes del sistema - Notificaciones "Toast"

5.4 DSI 6: DISEÑO FÍSICO DE DATOS

En este apartado se describe la integración del sistema de gestión de bases de datos empleado en el sistema para administrar los *permalinks* generados y consumidos a través del módulo API, así como el diagrama entidad-relación de la estructura de la base de datos.

5.4.1 Descripción del SGBD Usado

El SGBD utilizado en el sistema en *MongoDB*, tal y como se especifica en la definición inicial de la arquitectura del sistema (ver *2.2.5-Bases de datos: MongoDB (Atlas)*). De igual manera, se describen las propiedades del SGBD durante el estudio tecnológico de alternativas (ver *2.1.5.1-MongoDB*).

MongoDB destaca por ser un gestor de bases de datos no relacionales gratuito y de código abierto. Si bien el sistema permite configurar una instancia de *MongoDB* propia que utilizar como base de datos, la aplicación recurrirá por defecto a una instancia en la nube proporcionada por *MongoDB Atlas*, el sistema *Software as a Service* (*SaaS*) creado por la compañía.

El tipo de clúster contratado en *Atlas* es *M0* (gratuito) y la versión de *MongoDB* contratado es, en el momento de redacción de la memoria, la **5.0.9**, con posibilidad de actualización a nuevas versiones de la línea 5 **[99]**.

En resumen, las principales características del SGBD empleado son:

- **SGBD**
  MongoDB.
- **Versión**
  5.0.9 (con posibilidad de actualización dentro de la rama 5).
- **Entorno de ejecución**
  Nube (*MongoDB Atlas* – Clúster *M0*) por defecto, con la posibilidad de especificar un entorno personalizado.
- **Plan de *backup* de los datos**
  Ninguno, se considera que la información almacenada no es imprescindible ni sensible.
  Se permite a los usuarios del sistema desarrollar sus propios planes de recuperación en sus despliegues personalizados.
- **Plan de mantenimiento y purgado de los datos**
  El despliegue en la nube contará con un disparador (*trigger*) que se ejecutará semanalmente y se encargará de eliminar de la base de datos las entradas a enlaces cuya última fecha de creación/uso sea hace más de un año. Al eliminar *permalinks* en desuso, se pretende mantener el espacio y costes de la base de datos bajo control afectando al menor número de usuarios posible.
- **Permisos de acceso**
  El despliegue en la nube contará con dos roles de acceso:

- Usuario administrador: Rol con permisos de administración sobre la base de datos, destinado a tareas de gestión y mantenimiento.
- RdfShapeUser: Rol con permisos para buscar e insertar entradas en la base de datos, creado para su uso por la aplicación *RDFShape* en la generación y consumo de enlaces. Dicho rol será el empleado desde el módulo API de forma predeterminada.

5.4.2 Integración del SGBD en Nuestro Sistema

Para la integración y manipulación de la base de datos desde el sistema se ha empleado la biblioteca *mongo4cats*, construida sobre el driver oficial de *MongoDB* para *Java* con el objetivo de poder operar contra una base de datos programáticamente desde *Scala* manteniendo el paradigma de programación funcional y abstrayéndose de la sintaxis del lenguaje de consultas de *MongoDB* **[100]**.

Adicionalmente, *mongo4cats* permite definir funciones codificadoras y decodificadoras (*códecs*) para manipular las entradas de la base de datos como objetos del dominio de la aplicación. Así mismo, permite la creación de varios filtros desde el propio código, de forma que las consultas se realizan mediante los métodos expuestos por el API de la biblioteca y no mediante *MongoDB Query Language*.

5.4.3 Diagrama E-R

La siguiente ilustración muestra el diagrama entidad-relación del sistema. Destaca su simplicidad, debida a las escasas necesidades de persistencia del sistema.

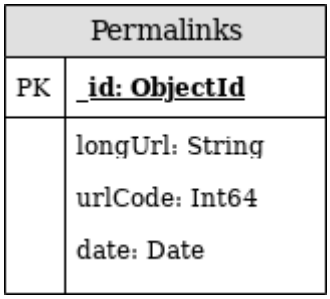


Ilustración 73. Diagrama entidad-relación

5.5 DSI 10: ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS

Partiendo del plan de pruebas definido en la etapa de análisis (ver *4.8-ASI 10: Especificación del Plan de Pruebas*) y refinándolo con el conocimiento adquirido durante la fase de desarrollo, se proceden a describir las pruebas que serán ejecutadas sobre el sistema final.

En el caso de que algunas pruebas no sean superadas, las causas del fallo se analizarán de forma individualizada para mejorar y corregir el comportamiento del sistema (ver secciones a partir de *6.3-CSI 3: Ejecución de las Pruebas Unitarias*).

5.5.1 Pruebas Unitarias

Se replica el listado de pruebas del plan inicial (ver *4.8.1-Pruebas Unitarias*), definiendo las entradas que se van a suministrar al sistema y concretando la respuesta esperada, ahora que se conoce el diseño de clases de este. Para la creación y ejecución de los *tests* en la biblioteca de validación se utilizará la popular biblioteca *ScalaTest* **[101]** junto con la integración *cats-effect-testing* **[102]**.

5.5.1.1 *Entradas*

Para la ejecución de las pruebas unitarias es necesario crear constantemente conjuntos de datos RDF junto con los esquemas contra los que validarlos. Para este propósito se definirán una serie de utilidades para generar los datos y esquemas necesarios para los *tests*, siguiendo una plantilla de ejemplo que simula datos emitidos por sensores de calor y esquemas que comprueben su validez.

5.5.1.1.1 Datos RDF

Se generarán de forma automatizada mediante código datos RDF siguiendo el siguiente patrón:

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:      <http://example.org/> .

ex:reading a ex:sensorReading ;
          ex:readingDatetime "2022-06-05T10:00:00"^^xsd:dateTime ;
          ex:readingTemperature "18"^^xsd:decimal ;
          ex:status "OK" .
```

Ilustración 74. Plantilla RDF para pruebas unitarias

Si bien el ejemplo superior se encuentra en sintaxis *Turtle*, el sistema permitirá generar los datos equivalentes en las distintas sintaxis soportadas. Así mismo, se permitirá alterar los valores de la fecha de lectura y temperatura leída para crear datos válidos o inválidos bajo demanda.

5.5.1.1.2 Esquemas *Shape Expressions*

De forma análoga a cómo se generan los datos RDF de los *tests*, se generarán los siguientes esquemas *ShEx* para validar los datos de prueba. El sistema permitirá generar el esquema equivalente en sintaxis *ShExJ*.

```
PREFIX ex:       <http://example.org/>
PREFIX xsd:  <http://www.w3.org/2001/XMLSchema#>

# Filters of a valid sensor reading
ex:ValidReading {
  ex:readingDatetime     xsd:dateTime  ; # Has a VALID timestamp
  ex:readingTemperature xsd:decimal MININCLUSIVE 18 MAXINCLUSIVE 20 + ;
# 1+ readings in range 18-20
  ex:status [ "OK" "RUNNING" ] # Status must be one of
}
```

Ilustración 75. Plantilla ShEx para pruebas unitarias

5.5.1.2 *Pruebas de Validación*

Pruebas de validación		
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *Turtle*.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: Turtle. - Temperatura: 19. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un único resultado con: - Status: VALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *JSON-LD*.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: JSON-LD. - Temperatura: 19. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un único resultado con: - Status: VALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**

Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *RDF-XML*.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: RDF-XML. - Temperatura: 19. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un único resultado con: - Status: VALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *Turtle*.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: Turtle. - Temperatura: -5. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un único resultado con: - Status: INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *JSON-LD*.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: JSON-LD. - Temperatura: -5. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un único resultado con: - Status: INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *RDF-XML*.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: RDF-XML. - Temperatura: -5. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un único resultado con: - Status: INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.

Tabla 77. Pruebas de validación – Comet (diseño)

5.5.1.3 *Pruebas de Extracción*

Pruebas de extracción		
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de listas.	Datos RDF: se ajusta un conjunto de 5 datos plantilla y se colocan en una lista de *Strings* tal que: - Sintaxis: Turtle. - Temperatura: aleatoria. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un total de 5 resultados con: - Status: VALID/INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: incremental para cada resultado emitido, partiendo del momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de ficheros.	Datos RDF: se ajusta un conjunto de 5 datos plantilla y se coloca su contenido en ficheros UTF-8 individuales tal que: - Sintaxis: Turtle. - Temperatura: aleatoria. Esquema de validación: se emplea el esquema plantilla.	El sistema procesa el flujo y emite un total de 5 resultados con: - Status: VALID/INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: incremental para cada resultado emitido, partiendo del momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de *Kafka*.	Datos RDF: se define un origen de datos en un servidor *Kafka* tal que: - Origen: “server:9092”. - Topic: “topic”. Esquema de validación: se emplea el esquema plantilla.	El sistema es incapaz de localizar el origen del flujo y lanza una excepción derivada de la clase *KafkaException*.

Tabla 78. Pruebas de extracción – Comet (diseño)

5.5.1.4 *Pruebas de Parada*

Pruebas de parada		
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF inválido con el validador configurado para ignorar los ítems inválidos.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: Turtle. - Temperatura: -5. Esquema de validación: se emplea el esquema plantilla. Validador: configurado para no detenerse independientemente de los resultados.	El sistema procesa el flujo y emite un único resultado con: - Status: INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF inválido con el validador configurado para detenerse ante ítems inválidos.	Datos RDF: se ajustan los datos plantilla tal que: - Sintaxis: Turtle. - Temperatura: -5. Esquema de validación: se emplea el esquema plantilla. Validador: configurado para detenerse ante un resultado inválido.	El sistema lanza una excepción *StreamInvalidItemException*, que contiene el informe de validación generado por *SHaclEX* para el ítem inválido.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF mal redactado con el validador configurado para ignorar los ítems erróneos.	Datos RDF: cadena de caracteres que no representa RDF "*Wrong data*". Esquema de validación: se emplea el esquema plantilla. Validador: configurado para no detenerse independientemente de los resultados.	El sistema procesa el flujo al completo, no emitiendo ningún resultado.
Prueba	**Entradas**	**Respuesta**

Validar mediante *ShEx* un conjunto de RDF mal redactado con el validador configurado para detenerse ante ítems erróneos.	Datos RDF: cadena de caracteres que no representa RDF "*Wrong data*". Esquema de validación: se emplea el esquema plantilla. Validador: configurado para detenerse ante un error de procesamiento del RDF o esquema.	El sistema lanza una excepción *StreamErroredItemException*, que contiene como causa la excepción original emitida por SHaclEX al intentar *parsear* los datos.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF con el validador configurado para abortar la validación si no se leen datos del flujo instantáneamente.	Datos RDF: cadena de caracteres que no representa RDF "*Wrong data*". Esquema de validación: se emplea el esquema plantilla. Validador: configurado para detenerse si el flujo de entrada no emite elementos en 1 microsegundo.	El sistema no llega a procesar ningún elemento y lanza una Exception *StreamTimeoutException* tras inicializarse.

Tabla 79. Pruebas de parada – Comet (diseño)

5.5.2 Pruebas de Integración

Se sigue el mismo procedimiento que para las pruebas unitarias. Por definición, son varios los módulos y agentes del sistema que se comunican entre sí durante las pruebas de integración. Cabe destacar que, salvo que la naturaleza de la prueba indique lo contrario, los *tests* se ejecutarán en una sola máquina (*localhost*) encargada de ejecutar los módulos e interactuar con todos los agentes implicados.

Además, para probar la validación contra un flujo real, se pondrá en marcha un productor de *Apache Kafka* que producirá datos RDF basados en la plantilla de las pruebas unitarias. Así mismo, el esquema de validación también será el mismo que el empleado en el apartado anterior, salvo que se especifique lo contrario.

5.5.2.1 *Módulo de Validación de Flujos*

Pruebas de integración		
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF procedente de un flujo de datos *Kafka* no disponible o inaccesible.	Datos RDF: se define un origen de datos en un servidor *Kafka* tal que: - Origen: "localhost:9092". - Topic: "topic". - Período de inactividad máximo: 30 segundos. Flujo de entrada: detenido.	El sistema contacta con el origen pero es incapaz de localizar el flujo en el origen indicado y lanza una excepción *StreamTimeoutException* una vez transcurren 30 segundos.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF válido procedente de un flujo de datos *Kafka* ejecutándose localmente.	Datos RDF: se define un origen de datos en un servidor *Kafka* tal que: - Origen: "localhost:9092". - Topic: "rdf". - Período de inactividad máximo: 30 segundos. Flujo de entrada: ejecutándose localmente.	El sistema procesa el flujo y emite un único resultado con: - Status: VALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF inválido procedente de un flujo de datos *Kafka* ejecutándose localmente.	Datos RDF: se define un origen de datos en un servidor *Kafka* tal que: - Origen: "localhost:9092". - Topic: "rdf". - Período de inactividad máximo: 30 segundos. Flujo de entrada: ejecutándose localmente.	El sistema procesa el flujo y emite un único resultado con: - Status: INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF válido procedente de un flujo de datos *Kafka*	Datos RDF: se define un origen de datos en un servidor *Kafka* tal que: - Origen: "{IP-nube}:9092". - Topic: "rdf".	El sistema procesa el flujo y emite un único resultado con: - Status: VALID. - Resultado: está definido y contiene el informe de

accesible a través de internet.	- Período de inactividad máximo: 30 segundos. Flujo de entrada: ejecutándose en una instancia AWS en la nube.	validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.
Prueba	**Entradas**	**Respuesta**
Validar mediante *ShEx* un conjunto de RDF inválido procedente de un flujo de datos *Kafka* accesible a través de internet.	Datos RDF: se define un origen de datos en un servidor *Kafka* tal que: - Origen: "{IP-nube}:9092". - Topic: "rdf". - Período de inactividad máximo: 30 segundos. Flujo de entrada: ejecutándose en una instancia AWS en la nube.	El sistema procesa el flujo y emite un único resultado con: - Status: INVALID. - Resultado: está definido y contiene el informe de validación generado por *SHaclEX*. - Emisión: momento de ejecución de la prueba.

Tabla 80. Pruebas de integración - Comet (diseño)

5.5.2.2 *Módulo API*

Pruebas de integración		
Prueba	**Entradas**	**Respuesta**
Lanzar una petición contra una ruta inexistente del API.	Ruta petición: */nonExistant*	El sistema devuelve un código de error 404 (*Not Found*).
Prueba	**Entradas**	**Respuesta**
Enviar una petición de validación de flujos en formato distinto de JSON.	Ruta petición: */validate/stream* Contenido de la petición: "Hi!"	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente con el código de error 3000 informando del problema.
Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos sin especificar el servidor de origen de los datos.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.

<table>
<tr><td></td><td><pre>
  "schema": {…},
  "triggerMode": {…},
 "extractor": {
  "data": {
   "format": "Turtle",
   "inference": "None"
  }
 },
 "stream": {
  "port": 9092,
  "topic": "rdf",
  "groupId": "test"
 }
 }
}
</pre></td><td></td></tr>
<tr><th>Prueba</th><th>Entradas</th><th>Respuesta</th></tr>
<tr><td>Enviar una petición de validación de flujos sin especificar el topic de origen de los datos.</td><td>Ruta petición: /validate/stream
Contenido de la petición: JSON incompleto:
<pre>
{
 "configuration": {
  "validator": {
   "haltOnInvalid": false,
   "haltOnErrored": false,
   "schema": {…},
   "triggerMode": {…},
  "extractor": {
   "data": {
    "format": "Turtle",
    "inference": "None"
   }
  },
  "stream": {
   "server": "localhost",
   "port": 9092,
   "groupId": "test"
  }
 }
}
</pre></td><td>El sistema recibe la petición e inicia un diálogo WebSockets con el cliente.
El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.</td></tr>
<tr><th>Prueba</th><th>Entradas</th><th>Respuesta</th></tr>
</table>

Enviar una petición de validación de flujos sin especificar el puerto de origen de los datos.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {
	"data": {
	"format": "Turtle",
	"inference": "None"
	}
	},
	"stream": {
	"server": "localhost",
	"topic": "rdf",
	"groupId": "test"
	}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación utilizando el puerto por defecto 9092.

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos sin especificar el grupo de consumidor de los datos.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {
	"data": {
	"format": "Turtle",
	"inference": "None"
	}
	},
	"stream": {
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación utilizando el identificador de grupo por defecto "*comet*-{version}".

	"server": "localhost", "topic": "rdf", "port": 9092 } } }	
Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos sin especificar el formato de los datos RDF por consumir.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {
	"data": {
	"inference": "None"
	}
	},
	"stream": {…}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos sin especificar la inferencia a aplicar sobre los datos por consumir.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {
	"data": {
	"format": "Turtle"
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación sin aplicar ninguna inferencia.

	}, "stream": {…} } }	
Prueba	**Entradas**	**Respuesta**
Enviar una petición de validación de flujos sin especificar el esquema *ShEx* contra el que validar los datos.	Ruta petición: */validate/stream* Contenido de la petición: JSON incompleto: { "configuration": { "validator": { "haltOnInvalid": false, "haltOnErrored": false, "triggerMode": {…}, "extractor": {…}, "stream": {…} } }	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.
Prueba	**Entradas**	**Respuesta**
Enviar una petición de validación de flujos especificando un esquema *ShEx* inválido.	Ruta petición: */validate/stream* Contenido de la petición: JSON completo: { "configuration": { "validator": { "haltOnInvalid": false, "haltOnErrored": false, "schema": { "content": "Invalid string!", "engine": "ShEx", "format": "ShExC", "source": "byText" }, "triggerMode": {…}, "extractor": {…}, "stream": {…} } }	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente. El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.
Prueba	**Entradas**	**Respuesta**
Enviar una petición de validación de flujos	Ruta petición: */validate/stream*	El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.

sin especificar el *ShapeMap* contra el que validar los datos.	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"extractor": {…},
	"stream": {…}
	}
	}
		El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.

Prueba	**Entradas**	**Resultado Esperado**

Enviar una petición de validación de flujos especificando un *ShapeMap* inválido.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON completo:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {
	"shape-map": {
	"content": "Invalid string",
	"format": "Compact",
	"source": "byText"
	},
	"type": "ShapeMap"
	}
	},
	"extractor": {…},
	"stream": {…}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema aborta la conexión con el cliente con el código de error 3001 informando del problema.

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos sin especificar el comportamiento del	Ruta petición: */validate/stream*
	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación ignorando los valores inválidos.

validador ante ítems inválidos.	"validator": {
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {…},
	"stream": {…}
	}
	}	

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos sin especificar el comportamiento del validador ante ítems erróneos.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON incompleto:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {…},
	"stream": {…}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación ignorando los valores erróneos o fallidos.

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos contra una combinación de servidor/*topic* inexistente.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON completo:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {…},
	"stream": {
	"server": localhost,
	"port": 9092,
	"topic": "nope",
	"groupId": "test"
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación y pasado el período de inactividad del flujo aborta la conexión con el cliente con el código de error 3004 informando del problema.

	} **Prueba**	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente válidos y el validador configurado para no detenerse.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON completo:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {
	"content": "…",
	"engine": "ShEx",
	"format": "ShExC",
	"source": "byText"
	},
	"triggerMode": {
	"shape-map": {
	"content": "",
	"format": "Compact",
	"source": "byText"
	},
	"type": "ShapeMap"
	}
	},
	"extractor": {
	"data": {
	"format": "Turtle",
	"inference": "None"
	}
	},
	"stream": {
	"server": "localhost",
	"port": 9092,
	"topic": "rdf",
	"groupId": "test"
	}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación y envía los resultados a través de *WebSockets* según son recibidos y procesados. Para cada resultado, el sistema envía un mensaje JSON con la estructura:
		
		{
		"type": "result",
		"content": {
		"valid": {true/false},
		"status": {status},
		"message": {mensaje según el estado},
		"instant": {fecha de generación},
		"report": {resultado de SHaclEX}
		}
		}
		
		El sistema registra el proceso de validación en el sistema de registros del sistema de ficheros.

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos con todos los campos pero con datos de tipo incorrecto.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON completo con datos incorrectos:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {…},
	"stream": {
	"server": localhost,
	"port": "bad port value",
	"topic": "nope",
	"groupId": "test"
	}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema aborta la conexión con el cliente con el código de error 3005 informando del problema.

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación con todos los campos pero con datos fuera de los rangos permitidos.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON completo con datos incorrectos:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {…},
	"stream": {
	"server": localhost,
	"port": -50,
	"topic": "nope",
	"groupId": "test"
	}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema aborta la conexión con el cliente con el código de error 3006 informando del problema.

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente válidos y el validador configurado para detenerse ante elementos inválidos.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON completo:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": true,
	"haltOnErrored": false,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {…},
	"stream": {…}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación y envía los resultados a través de *WebSockets* según son recibidos y procesados.
		
		El sistema aborta la conexión con el cliente ante el primer resultado inválido con el código de error 3002 informando del problema.
		El mensaje de error del servidor sigue la siguiente estructura:
		{
		"type": "error",
		"content": {
		"message": {resumen del error},
		"reason": {resultado de SHaclEX}
		}
		}

Prueba	**Entradas**	**Respuesta**

Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente erróneos y el validador configurado para detenerse ante elementos erróneos.	Ruta petición: */validate/stream*
	Contenido de la petición: JSON completo:
	{
	"configuration": {
	"validator": {
	"haltOnInvalid": false,
	"haltOnErrored": true,
	"schema": {…},
	"triggerMode": {…},
	"extractor": {…},
	"stream": {…}
	}
	}
		El sistema recibe la petición e inicia un diálogo *WebSockets* con el cliente.
		El sistema trata de inicializar la validación y envía los resultados a través de *WebSockets* según son recibidos y procesados.
		
		El sistema aborta la conexión con el cliente ante el primer elemento incorrecto con el código de error 3003 informando del problema. El mensaje de error del servidor sigue la siguiente estructura:
		{
		"type": "error",
		"content": {
		"message": {resumen del error},
		}
		}

Prueba	Entradas	Respuesta
Detener, desde el cliente, la conexión con el servidor durante una validación.	Cierre de la conexión *WebSockets*.	El sistema cierra la conexión, desconecta del flujo de datos y detiene la validación del flujo junto con los recursos dedicados a esta.

Tabla 81. Pruebas de integración – API (diseño)

5.5.2.3 *Módulo Cliente*

Para las pruebas realizadas desde la interfaz del cliente, se mantendrá operativo el servidor de *Kafka* empleado

Pruebas de integración		
Prueba	**Entradas**	**Respuesta**
Invocar una validación de flujos sin rellenar el servidor de origen.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: (vacío) - Port: 9092 - Topic: “rdf” - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - Esquema plantilla *ShapeMap*: - *ShapeMap* plantilla.	El sistema no contacta con el servidor y muestra una alerta de error en el área de resultados informando del problema.
Prueba	**Entradas**	**Respuesta**

Invocar una validación de flujos sin rellenar el puerto de origen.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: “localhost” - Port: {vacío} - Topic: “rdf” - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - Esquema plantilla *ShapeMap*: - *ShapeMap* plantilla.	El sistema no contacta con el servidor y muestra una alerta de error en el área de resultados informando del problema.
Prueba	**Entradas**	**Respuesta**
Invocar una validación de flujos sin rellenar el *topic* de origen.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: “localhost” - Port: 9092 - Topic: {vacío} - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - Esquema plantilla *ShapeMap*: - *ShapeMap* plantilla.	El sistema no contacta con el servidor y muestra una alerta de error en el área de resultados informando del problema.
Prueba	**Entradas**	**Respuesta**

Invocar una validación de flujos sin rellenar el esquema *ShEx* a emplear.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: “localhost” - Port: 9092 - Topic: “rdf” - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - {vacío} *ShapeMap*: - *ShapeMap* plantilla.	El sistema no contacta con el servidor y muestra una alerta de error en el área de resultados informando del problema.
Prueba	**Entradas**	**Respuesta**
Invocar una validación de flujos sin rellenar el *ShapeMap* a emplear.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: “localhost” - Port: 9092 - Topic: “rdf” - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - Esquema plantilla. *ShapeMap*: - {vacío}	El sistema no contacta con el servidor y muestra una alerta de error en el área de resultados informando del problema.
Prueba	**Entradas**	**Respuesta**

Invocar una validación de flujos contra un esquema *ShEx* mal formateado.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: "localhost" - Port: 9092 - Topic: "rdf" - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - "Invalid schema!". *ShapeMap*: - *ShapeMap* plantilla.	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. El sistema recibe un error del servidor y muestra una alerta en el área de resultados que puede desplegarse para ver el detalle del fallo.
Prueba	**Entradas**	**Respuesta**
Invocar una validación de flujos contra un *ShapeMap* mal formateado.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: "localhost" - Port: 9092 - Topic: "rdf" - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - Esquema plantilla. *ShapeMap*: - "Invalid shapeMap".	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. El sistema recibe un error del servidor y muestra una alerta en el área de resultados que puede desplegarse para ver el detalle del fallo.
Prueba	**Entradas**	**Respuesta**

Invocar una validación de flujos válida contra una combinación de servidor/*topic* inexistente o no disponible.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: "localhost" - Port: 9092 - Topic: "nope" - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - Esquema plantilla. *ShapeMap*: - *ShapeMap* plantilla.	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. Pasado el tiempo de inactividad fijado en el servidor, el sistema recibe un error del servidor y muestra una alerta en el área de resultados que puede desplegarse para ver el detalle del fallo.
Prueba	**Entradas**	**Respuesta**
Invocar una validación de flujos válida contra un flujo existente.	Se rellena el formulario del cliente web con los siguientes datos: Origen de datos: - Server: "localhost" - Port: 9092 - Topic: "rdf" - Halt invalid: false - Halt errored: false - Format: Turtle - Inference: None Esquema de validación: - Esquema plantilla. *ShapeMap*: - *ShapeMap* plantilla.	El sistema contacta y abre una conexión *WebSockets* con el servidor, actualizando la interfaz para informar del inicio de una validación e impedir solicitar una nueva. Recibido el primer resultado, el sistema: - Actualiza la alerta con el estado de la validación. - Muestra la tabla-listado de resultados con estos ordenados por defecto por fecha descendente, así como disponibles: pausa, buscar, exportar, vaciar. Si se recibe un error del servidor, el sistema se asegura de cerrar la conexión con el servidor y actualizar la interfaz para mostrar el error mediante una alerta desplegable sobre los resultados disponibles, así como permitir solicitar una nueva validación.
Prueba	**Entradas**	**Respuesta**

Pausar una validación de flujos en curso.	Se interactúa con el botón de pausa mientras una validación tiene lugar.	El sistema cierra la conexión *WebSockets* con el servidor, actualizando la interfaz para: - Informar de la pausa de la validación mediante una alerta sobre los resultados disponibles. - Permitir reanudar la validación. - Permitir solicitar una nueva validación, activando el botón de subida del formulario.
Prueba	**Entradas**	**Respuesta**
Reanudar una validación de flujos en pausa.	Se interactúa con el botón de reanudar mientras una validación está en pausa.	El sistema reabre la conexión *WebSockets* con el servidor, actualizando la interfaz para: - Informar de que la validación está en curso mediante una alerta sobre los resultados disponibles. - Incluir todos los resultados que se han perdido durante la pausa del sistema en el listado. - Impedir solicitar una nueva validación, desactivando el botón de subida del formulario.
Prueba	**Entradas**	**Respuesta**
Buscar entre los resultados de una validación en curso con un término no vacío (1+ caracteres).	Se rellena el campo de búsqueda con el término: “reading”.	El sistema actualiza el listado de resultados mostrando sólo aquellos cuyo contenido textual en alguno de sus campos coincide con el término de búsqueda.
Prueba	**Entradas**	**Respuesta**
Buscar entre los resultados de una validación en curso un término vacío (0 caracteres).	Se rellena el campo de búsqueda con el término: “” (vacío).	El sistema actualiza el listado de resultados mostrando todos los resultados.
Prueba	**Entradas**	**Respuesta**
Limpiar los resultados de una validación en curso.	Se interactúa con el botón de vaciar resultados mientras una validación está activa.	El sistema actualiza el listado de resultados, mostrando un listado vacío hasta la llegada de nuevos elementos.
Prueba	**Entradas**	**Respuesta**

Exportar los resultados de una validación en curso.	Se interactúa con el botón de exportar resultados mientras una validación está activa.	El sistema genera un documento CSV con todos los campos de todos los resultados de validación en el listado y lanza el diálogo por defecto del navegador para visualizar/descargar el documento.
Prueba	**Entradas**	**Respuesta**
Ordenar los resultados de una validación en curso en base a un campo nuevo.	Se interactúa con los botones de reordenación en el título de la tabla de resultados para el campo “Status”.	El sistema reordena el listado de resultados de forma descendente en torno al campo “Status”, mostrando primero las entradas “valid” y posteriormente las “invalid”.
Prueba	**Entradas**	**Respuesta**
Ordenar los resultados de una validación en curso en base a un campo ya ordenado.	Tras los pasos del test anterior, se interactúa con los botones de reordenación en el título de la tabla de resultados para el campo “Status”.	El sistema reordena el listado de resultados de forma ascendente en torno al campo “Status”, mostrando primero las entradas “invalid” y posteriormente las “valid”.
Prueba	**Entradas**	**Respuesta**
Generar un *permalink* a una operación de validación.	Se interactúa con el botón de generación de enlaces mientras una validación está activa.	El sistema contacta con el sistema de generación de enlaces del servidor y actualiza la interfaz hasta obtener una respuesta para: - Mostrar que la generación del enlace está en curso mediante un elemento dinámico tipo *spinner*. - Impedir la solicitud de un nuevo enlace, desactivando el botón de solicitud. El sistema coloca en el portapapeles del usuario el nuevo enlace a la validación y mostrará un mensaje informativo tipo *toast*.
Prueba	**Entradas**	**Respuesta**
Consumir un *permalink* a una operación de validación.	Se introduce en la barra de direcciones del navegador un enlace generado previamente por el sistema.	El sistema muestra una pantalla informativa al usuario comunicando que está procesando el enlace. El sistema contacta con el sistema de generación de enlaces del servidor y

	redirige al usuario al enlace objetivo automáticamente. El sistema rellena los datos de la validación y la inicia de forma automática.

Tabla 82. Pruebas de integración – Cliente (diseño)

5.5.3 Pruebas de Usabilidad

Se seguirá el plan de pruebas especificado en la fase de análisis (ver *4.8.3-Pruebas de Usabilidad*).

5.5.3.1 *Diseño de Cuestionarios*

Se diseñan dos tipos de cuestionarios, uno centrado en la usabilidad y accesibilidad percibidas por los usuarios finales del cliente web y otro breve destinado al supervisor y responsable de las pruebas.

En este contexto, los usuarios finales serán los alumnos del máster, mientras que el responsable de las pruebas será el tutor del proyecto, que les formará y acompañará durante el uso de la herramienta.

5.5.3.1.1 Cuestionario de Evaluación

La participación en los cuestionarios de evaluación del sistema será totalmente opcional, por tanto se priorizan dos objetivos para hacerlo más atractivo:

- El cuestionario debe ser fácilmente accesible y realizable de forma digital.
- El cuestionario debe poder ser completado en torno a los 5 minutos.

El cuestionario de evaluación será distribuido a los alumnos de forma online mediante la utilidad *Microsoft Forms*, que facilita la elaboración y distribución de cuestionarios, así como la recolección de las respuestas.

A continuación se adjunta, en formato tabla y por secciones, el modelo de cuestionario elaborado.

5.5.3.1.1.1 Preguntas de carácter general y del contexto del sistema

Del 1 al 5, ¿cuánta experiencia previa considera que posee en el dominio de Web Semántica, así como la manipulación y validación de RDF? *Si es alumno del Máster en Ingeniería Web, cíñase a la experiencia previa a cursar la asignatura de Web Semántica*				
1	2	3	4	5
¿Ha usado alguna vez herramientas web similares (funcionalidad, apariencia, etc.) a RDFShape?				

1. Sí, he empleado software similar para trabajar con RDF, Shapes y otras tecnologías semánticas. 2. Sí, empleo regularmente otras herramientas similares pero especializadas en otros dominios. 3. Sí, aunque sólo utilizo este tipo de herramientas puntualmente. 4. No, nunca.
¿Qué busca principalmente en un programa?
1. Que sea fácil de usar. 2. Que sea intuitivo. 3. Que sea rápido. 4. Que tenga todas las funciones necesarias.

Tabla 83. Cuestionario de evaluación - Preguntas generales

5.5.3.1.1.2 Preguntas sobre la funcionalidad del sistema

Por favor valore, del 1 al 5, las siguientes cuestiones sobre las funcionalidades ofrecidas por *RDFShape*:

¿Funciona cada tarea como usted espera?				
1 4.	2	3	4	5
¿Cree que el tiempo de respuesta de la aplicación es razonable?				
1 5.	2	3	4	5
¿Le han resultado útiles las visualizaciones generadas por la aplicación a la hora de comprender o compartir resultados?				
1 3.	2	3	4	5
¿Le ha resultado útil el sistema de permalinks de la aplicación para guardar o compartir resultados?				
1	2	3	4	5

Tabla 84. Cuestionario de evaluación - Preguntas de funcionalidad

5.5.3.1.1.3 Preguntas sobre la usabilidad del sistema

Por favor valore, del 1 al 5, las siguientes cuestiones sobre la facilidad de uso de *RDFShape*:

¿Le resulta sencillo el uso de la aplicación en general?				
1 5.	2	3	4	5
¿Sabe en todo momento dónde está dentro de la aplicación?				
1 6.	2	3	4	5
¿Cree que el programa está bien estructurado?				

1 4.	2	3	4	5
¿Cree que el diseño de las pantallas es claro y atractivo?				
1	2	3	4	5
¿Existe ayuda para las funciones en caso de que tenga dudas?				
1 6.	2	3	4	5
¿Le son útiles los mensajes informativos y de error emitidos por la aplicación en el desarrollo de sus funciones?				
1 7.	2	3	4	5
¿Le es sencillo operar con los datos según se va desplazando entre pantallas?				
1 5.	2	3	4	5

Tabla 85. Cuestionario de evaluación - Preguntas de usabilidad

5.5.3.1.1.4 Preguntas sobre la accesibilidad del sistema

Por favor valore, del 1 al 5, las siguientes cuestiones sobre la accesibilidad de *RDFShape*:

¿El tipo y tamaño de letra son adecuados?				
1 7.	2	3	4	5
¿Los iconos e imágenes usados son adecuados?				
1 8.	2	3	4	5
¿Los colores empleados son adecuados?				
1 6.	2	3	4	5

Tabla 86. Cuestionario de evaluación - Preguntas de accesibilidad

5.5.3.1.1.5 Observaciones

Opcionalmente, cualquier otra observación o sugerencia

Tabla 87. Cuestionario de evaluación - Observaciones

5.5.3.1.2 Cuestionario para el responsable de las pruebas y actividades guiadas

En tanto que la aplicación web será empleada a lo largo de la asignatura de Web Semántica para introducir todo tipo de conceptos a los alumnos, queda en manos del tutor/responsable guiar a los alumnos a través de todas las herramientas de la aplicación web.

El *feedback* aportado por el responsable de las pruebas será relevante para ayudar a comprender la interacción de los usuarios con el sistema. Se realizará siguiendo un formato de respuesta libre para cada aspecto tenido en cuenta.

Aspecto observado	Notas y observaciones
El usuario entiende rápidamente el funcionamiento de la herramienta.	
Tiempo empleado por el usuario para rellenar los datos necesarios para invocar las operaciones de la herramienta.	
Errores cometidos por el usuario en el desarrollo de las actividades (por ejemplo: el usuario se desubica, no sabe navegar entre resultados, etc.)	

Tabla 88. Cuestionario para el responsable de las pruebas de usabilidad

5.5.4 Pruebas de Accesibilidad

Se parte del plan de pruebas especificado en la fase de análisis (ver *4.8.4-Pruebas de Accesibilidad*) y se detalla:

Pruebas de accesibilidad	
Prueba	**Resultado Esperado**
Escaneo de accesibilidad de la página de validación del cliente web mediante *AChecker*.	El sistema cumple con el nivel de accesibilidad *AA* del *WCAG*.

Tabla 89. Pruebas de accesibilidad – Cliente (diseño)

5.5.5 Pruebas de Rendimiento

La biblioteca de validación de flujos planea ofrecer una interfaz de alto rendimiento para todo tipo de flujos. En consecuencia, el aspecto principal que se va a tener en cuenta para determinar las capacidades del sistema es el tiempo que le toma ejecutar tareas intensas en las que se validan muchos elementos. Para la ejecución de las tareas se tienen en cuenta las siguientes consideraciones:

- Los datos introducidos en el validador serán elementos RDF pseudoaleatorios, generados en base al mismo algoritmo y plantilla empleados en las pruebas unitarias.
- El tiempo de ejecución será medido programáticamente tomando la medida del tiempo tanto antes como después de ejecutar la validación del conjunto de datos completo.
- Se empleará el extractor de datos de listas, de manera que los datos RDF empleados en las pruebas estén inmediatamente accesibles en la memoria del equipo. De esta manera, se pretende centrarse puramente en el rendimiento de la fase de validación y evitar

interferencias relativas a la velocidad de la red o del sistema de almacenamiento del equipo, como ocurriría con los extractores de *Kafka* y de ficheros, respectivamente.

Se detallan las pruebas concretas a realizar en base a las premisas del plan de pruebas de rendimiento de análisis (ver *4.8.5-Pruebas de Rendimiento*):

Pruebas de rendimiento	
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 4000 elementos en un hilo de ejecución (monohilo).	El sistema completa la validación en un tiempo determinado.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 4000 elementos en 2 hilos de ejecución.	El sistema completa la validación en un tiempo menor que en su equivalente monohilo.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 4000 elementos en 4 hilos de ejecución.	El sistema completa la validación en un tiempo menor que en su equivalente monohilo.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 10000 elementos en un hilo de ejecución (monohilo).	El sistema completa la validación en un tiempo determinado.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 10000 elementos en 2 hilos de ejecución.	El sistema completa la validación en un tiempo menor que en su equivalente monohilo.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 10000 elementos en 4 hilos de ejecución.	El sistema completa la validación en un tiempo menor que en su equivalente monohilo.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 30000 elementos en un hilo de ejecución (monohilo).	El sistema completa la validación en un tiempo determinado.

Prueba	Resultado Esperado
Validación de un flujo de datos pseudoaleatorio de 30000 elementos en 2 hilos de ejecución.	El sistema completa la validación en un tiempo menor que en su equivalente monohilo.
Prueba	**Resultado Esperado**
Validación de un flujo de datos pseudoaleatorio de 30000 elementos en 4 hilos de ejecución.	El sistema completa la validación en un tiempo menor que en su equivalente monohilo.

Tabla 90. Pruebas de rendimiento – Comet (diseño)

Capítulo 6 Construcción del Sistema de Información

Fase de Desarrollo

CSI

6.1 CSI 1: PREPARACIÓN DEL ENTORNO DE GENERACIÓN Y CONSTRUCCIÓN

En esta sección se describen una serie de aspectos relativos a la implementación del software desarrollado.

6.1.1 Estándares y normas seguidos

6.1.1.1 *Scala Style Guide*

La *Scala Style Guide* son un conjunto de guías de estilo para la programación de código en Scala. Incluyen recomendaciones sobre la indentación, nomenclatura, sintaxis de tipos, declaraciones o redacción de documentación *Scaladoc* y comparten muchos aspectos con las *Java Code Conventions*.

Estos estándares se han aplicado al código y documentación de los módulos validador y API, adaptándolos al contexto de cada situación.

Pueden consultarse en el siguiente enlace: https://docs.scala-lang.org/style/.

6.1.1.2 *Programación funcional*

La programación funcional es un paradigma de programación donde los programas de construyen aplicando y componiendo funciones, que son combinadas entre sí de manera modular para dar el resultado final. Sus principales características son la trata a las funciones como ciudadanos de primera clase, así como su variante de programación funcional "pura", en que las funciones deben ser deterministas tal que una expresión siempre produzca el mismo resultado para las mismas entradas.

A lo largo del desarrollo de los módulos validador y API, se han utilizado las funciones del lenguaje Scala, así como el framework *cats-effect* y su ecosistema de bibliotecas para desarrollar un código híbrido entre la programación funcional y la orientada a objetos, con el objetivo de mantener la coherencia con el código previo del cliente y aprovechar los beneficios de un código mayoritariamente "puro".

El documento de referencia para el uso de programación funcional en Scala puede encontrarse en el siguiente enlace: https://docs.scala-lang.org/scala3/book/fp-intro.html.

6.1.1.3 *Web Content Accessibility Guidelines (WCAG) 2.0*

Las *WCAG* son un conjunto de estándares internacionales publicadas por el W3C con la finalidad de hacer los contenidos web más accesible a las personas con discapacidad, incluyendo la información

base (textos, imágenes, sonidos y otro multimedia) y la estructura y presentación del contenido. Para lograr este objetivo, las *WCAG* definen cuatro pilares para una web **[103]**:

1. Perceptible: alternativas de texto y multimedia, uso del color y contraste, etc.
2. Operable: Accesibilidad con teclado y otros mecanismos de entrada del usuario, tiempos de entrada y de respuesta, navegabilidad, etc.
3. Inteligible: Legible, predecible y con asistencia al usuario.
4. Robusta: Compatible con todo tipo de clientes, uso de formatos *parseables* por los sistemas que los interpretarán, alertas de estado, etc.

Las guías del *WCAG* se han aplicado en la ampliación del cliente web de *RDFShape*. Puede encontrarse más información en el siguiente enlace: https://www.w3.org/WAI/standards-guidelines/wcag/.

6.1.2 Lenguajes de programación

6.1.2.1 *Scala*

Scala es un lenguaje de programación tipado de propósito general, que soporta los paradigmas de programación orientada a objetos y de programación funcional, así como enfoques híbridos entre ambos. Además destaca por su sistema de inferencia de tipos y su compilador inteligente con un sistema de valores implícitos.

La primera versión del lenguaje fue publicada en 2004 por Martin Odersky, de la Escuela Politécnica General de Luisiana, con el objetivo de proporcionar una plataforma compatible con el popular lenguaje *Java* pero corrigiendo varias de las críticas dirigidas a este (rendimiento, verbosidad, etc.). Actualmente se encuentra en su versión 3, y aunque no goza de la misma popularidad que sus contrapartes *Java* o *Kotlin*, los programas desarrollados en *Scala* son muy versátiles puesto que su código compila a *bytecode* ejecutable en la *Java Virtual Machine* (*JVM*) y por tanto compatible con todo tipo de sistemas.

Scala se ha utilizado en el desarrollo de los módulos validador y API, tal que:

- Biblioteca de validación: Se ha empleado *Scala* 3.1.2, si bien se ha prescindido del uso de sintaxis específica de *Scala* 3, con el objetivo de compilar la biblioteca tanto para Scala 2 como para Scala 3 y ampliar el número de usuarios potenciales.
- *RDFShape* API: Se ha empleado *Scala* 2.13.8, la versión más madura de *Scala* 2 hasta la fecha, para ampliar el desarrollo ya existente en *Scala* 2.

La página oficial del lenguaje de programación *Scala* se encuentra en el siguiente enlace: https://scala-lang.org/.

Ilustración 76. Logo de Scala

6.1.2.2 *HTML*

HTML (abreviado de *Hypertext Markup Language*) es el lenguaje de marcado con el que se estructura y define el contenido de las páginas web. Es un estándar internacional aún vigente, siendo la última versión la 5, ideado para diseñar documentos que van a ser mostrados en un navegador web.

Este lenguaje de marcado se ha utilizado para el desarrollo de las pantallas de la aplicación web del módulo cliente.

La página del W3C dedicada a *HTML* puede encontrarse en el siguiente enlace: https://www.w3.org/html/.

Ilustración 77. Logo de HTML 5

6.1.2.3 *CSS*

CSS (abreviado de *Cascading Style Sheets*) es un lenguaje de estilos creado para describir cómo deben presentarse los contenidos de un documento HTML de forma independiente al contenido. Es un estándar internacional vivo, con una sintaxis propia basada en normas de estilo y selectores que indican sobre qué elementos del documento se han de aplicar las normas.

Este lenguaje de estilado se ha empleado en el diseño de las pantallas de la aplicación web del módulo cliente.

La página del W3C dedicada a *CSS* puede encontrarse en el siguiente enlace: https://www.w3.org/Style/CSS/.

Ilustración 78. Logo de CSS

6.1.2.4 *JavaScript*

JavaScript o *JS* es un lenguaje de programación de alto nivel que sigue el estándar *ECMAScript*, diseñado para garantizar la interoperabilidad entre distintas páginas web y navegadores. Es un lenguaje dinámico, de tipado débil y que admite los paradigmas de programación imperativa y orientada a objetos.

Habitualmente, y en el contexto del presente proyecto, *JavaScript* se ejecuta en el ordenador del cliente de un servidor web, tal que el navegador es el encargado de descargar y posteriormente ejecutar el código de forma interpretada. Por otra parte, existen tecnologías como *Node.js* que permiten ejecutar JavaScript para construir aplicaciones web del lado del servidor.

La página del W3C dedicada a *JavaScript* puede encontrarse en el siguiente enlace: https://www.w3.org/standards/webdesign/script.

Ilustración 79. Logo de JavaScript

6.1.2.4.1 JavaScript XML (JSX)

JavaScript XML, conocido como *JSX*, es una extensión sobre la sintaxis base de *JavaScript* que ofrece la posibilidad de combinar etiquetas *HTML/XML* con expresiones JavaScript para definir interfaces de usuario de forma declarativa cuando se trabaja con el *framework React*.

JavaScript y su variante *JSX* han sido empleadas para el desarrollo de la funcionalidad y presentación de las herramientas de validación de flujos del módulo cliente.

La documentación oficial de *React* sobre *JSX* puede encontrarse en el siguiente enlace: https://reactjs.org/docs/introducing-jsx.html.

6.1.3 Herramientas y programas usados para el desarrollo

6.1.3.1 *Entornos de trabajo y desarrollo*

6.1.3.1.1 Git

Git es un software de control de versiones de código abierto desarrollado originalmente por *Linus Torvalds* en 2005 para el desarrollo del kernel de Linux. Permite almacenar, consultar y recuperar un registro de cambios ramificado que puede sincronizarse con un servidor externo al equipo de desarrollo. Es el software más popular de su clase.

Git se ha utilizado en su versión 2.34 para la supervisar la construcción de todos los módulos del proyecto.

El sitio oficial y de descarga de la herramienta se encuentra en el siguiente enlace: https://git-scm.com/downloads.

Ilustración 80. Logo de Git

6.1.3.1.1.1 GitHub

GitHub es una empresa propiedad de *Microsoft* que comercializa el producto del mismo nombre. *GitHub* ofrece un servicio de alojamiento en la nube especializado en código software versionado mediante *Git*, junto con otras funcionalidades propias destinadas la gestión de incidencias (*issues*, *pull requests*, documentación (*GitHub Pages*, *wikis*) e integración continua (*GitHub Actions*), entre otras.

Los servicios de *GitHub* incluyen varios planes gratuitos y de pago y herramientas para la gestión de organizaciones. Para el desarrollo del proyecto, se ha hecho uso de las herramientas gratuitas de *GitHub* y de repositorios tanto personales como dispuestos por la organización de WESO en la plataforma.

El sitio oficial de *GitHub* se encuentra en el siguiente enlace: https://github.com/.

La organización WESO en *GitHub* se encuentra en el siguiente enlace: https://github.com/weso/.

6.1.3.1.1.1 GitHub Actions

Uno de los servicios que ofrece *GitHub* aparte del almacenamiento y gestión de versiones de código en *GitHub Actions*, un mecanismo para crear flujos de trabajo para la integración y despliegue continuos de las aplicaciones.

La herramienta permite configurar una serie de tareas que se ejecutarán cuando se den unos eventos específicos en el repositorio del proyecto. Dichas tareas podrán operar sobre el código del proyecto e interactuar con otros sistemas en función de las necesidades del programador.

Durante el proyecto, se ha empleado para automatizar varios procesos de cada módulo, tal que:

- Biblioteca de validación:
  - Cuando se efectúa un nuevo commit contra el repositorio:
    - Compilación y ejecución de las pruebas unitarias del sistema en *Scala* 2 y *Scala* 3.
    - Compilación y publicación en *GitHub Pages* de la página web del proyecto.
  - Cuando se crea una nueva versión (*tag*) de la aplicación:
    - Publicación de la biblioteca en el repositorio de dependencias de *Sonatype*, anotada con la nueva versión.
- *RDFShape* API:
  - Cuando se efectúa un nuevo commit contra el repositorio:
    - Compilación y ejecución de las pruebas unitarias del sistema.
    - Compilación y publicación en *GitHub Pages* de la página web del proyecto.
    - Construcción y subida al *GitHub Container Registry* de la imagen *Docker* del sistema.
- *RDFShape* Client:
  - Cuando se efectúa un nuevo commit contra el repositorio:
    - Compilación y ejecución de las pruebas unitarias del sistema.
    - Construcción y subida al *GitHub Container Registry* de la imagen *Docker* del sistema.

Se puede encontrar más información del servicio en el siguiente enlace: https://github.com/features/actions.

6.1.3.1.1.1.2 GitHub Container Registry

GitHub Container Registry es un servicio de alojamiento de imágenes *Docker* (y otras variantes que cumplan el estándar *OCI*). Este servicio se ha utilizado para subir y alojar públicamente las imágenes *Docker* ligadas a los módulos API y Cliente del sistema.

Se puede encontrar más información del servicio en el siguiente enlace: https://docs.github.com/en/packages.

6.1.3.1.1.1.3 GitHub Pages

GitHub Pages en un servicio de alojamiento de páginas web ofrecido directamente por *GitHub*. Permite configurar un dominio de despliegue, así como utilizar un subdominio proporcionado por *GitHub* supeditado al usuario y repositorio.

Se ha empleado *GitHub Pages* para alojar las páginas descriptivas de los módulos validador y API de proyecto, esta última estando alojada bajo el dominio de WESO (https://www.weso.es/).

Se puede encontrar más información del servicio en el siguiente enlace: https://pages.github.com/.

6.1.3.1.2 IntelliJ IDEA

IntelliJ IDEA es uno de la serie de *IDEs* desarrollados por la compañía checa *Jetbrains*. Ofrece un entorno de desarrollo multiplataforma especializado en el desarrollo de aplicaciones con *Java*, Kotlin y *Groovy*, si bien permite el desarrollo con *Scala* y *SBT* a través de plugins.

Al igual que cualquier *IDE*, ofrece varias herramientas de ayuda al programador, como coloración de sintaxis, advertencias y errores de compilación en tiempo real, configuraciones de ejecución, soporte al control de versiones, etc. Es un entorno de trabajo de pago, si bien cuenta con planes gratuitos para estudiantes y otras organizaciones.

IDEA se ha empleado en su versión 2021.3 para el desarrollo de los módulos validador y API del proyecto.

El sitio oficial de descarga y compra del programa se encuentra en el siguiente enlace: https://www.jetbrains.com/idea/.

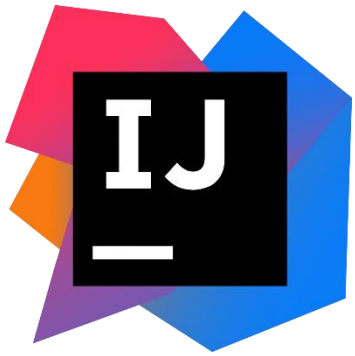


Ilustración 81. Logo de IntelliJ IDEA

6.1.3.1.3 Visual Studio Code

VSCode es un entorno de programación desarrollado por *Microsoft* con una base de código abierto. El programa soporta el desarrollo en prácticamente cualquier lenguaje y tecnología mediante el uso de extensiones para ampliar sus capacidades. Además, es gratuito multiplataforma puesto que está construido con la herramienta *Electron*.

La versión 1.64 de *VSCode* se ha utilizado en el desarrollo del módulo cliente.

El sitio oficial de descarga del programa se encuentra en el siguiente enlace: https://code.visualstudio.com/.

Ilustración 82. Logo de VSCode

6.1.3.1.4 SBT

SBT (abreviado de *Scala Build Tool*) es un popular entorno de construcción de proyectos y manejo de dependencias, similar a *Maven* o *Gradle* y frecuentemente utilizado en proyectos *Scala* para definir todo tipo de aspectos del proyecto, como los mecanismos de compilación, generación de documentación, submódulos del proyecto, dependencias utilizadas por el programa, etc. El funcionamiento base de *SBT* puede ser ampliado a través de *plugins* para obtener nuevas funcionalidades.

La versión 1.6 de *SBT* ha sido utilizado en los proyectos software asociados a los módulos validador y API.

El sitio web oficial y tutoriales de la herramienta se encuentran en el siguiente enlace: https://www.scala-sbt.org/.

Ilustración 83. Logo de SBT

6.1.3.1.5 Postman

Postman es una plataforma colaborativa para construir y probar el funcionamiento de todo tipo de servicios web: REST, GraphQL, SOAP, etc.

Ofrece un entorno de trabajo, tanto web como de escritorio, especializado para actuar como cliente HTTP y *WebSockets* y personalizar las peticiones enviadas así como analizar las respuestas recibidas del servidor objetivo.

Se ha empleado la versión 9.14 del programa para las pruebas de integración sobre el módulo API.

La página web oficial de la plataforma se encuentra en el siguiente enlace: https://www.postman.com/.

Ilustración 84. Logo de Postman

6.1.3.2 *Frameworks y bibliotecas*

6.1.3.2.1 SHaclEX

SHaclEX es una biblioteca de código desarrollada por WESO para la validación de datos RDF contra *Shape Expressions*. Esta herramienta, descrita en el estudio de viabilidad (ver *Capítulo 3-Estudio de Viabilidad del Sistema*), ha sido utilizada en su versión 0.2.2 para el desarrollo de la biblioteca de validación de flujos para validar cada ítem del flujo de entrada individualmente.

El sitio oficial de la biblioteca puede encontrarse en el siguiente enlace: http://www.weso.es/shaclex/.

6.1.3.2.2 Cats-effect

Framework para trabajar programar en *Scala* siguiendo el paradigma de programación funcional. El marco de trabajo, más detallado en el estudio arquitectónico del sistema (ver *2.1.1.1.1-Cats Effect*), ha sido empleada en su versión 3.3 para los módulos validador y API, por su compatibilidad con el ecosistema previo del cliente y las ventajas que aporta.

El sitio oficial del *framework* puede encontrarse en el siguiente enlace: https://typelevel.org/cats-effect/.

6.1.3.2.3 Fs2

Biblioteca del ecosistema *Typelevel* especializada en el procesamiento de flujos de manera abstraíble, funcional y de alto rendimiento. Esta biblioteca, definida en el estudio arquitectónico del sistema (ver *2.1.1.3.1-Fs2*), ha sido empleada en su versión 3.2 para el módulo validador, dada su compatibilidad con el ecosistema del cliente y su alto rendimiento y versatilidad.

El sitio oficial de la biblioteca puede encontrarse en el siguiente enlace: https://fs2.io/#/.

6.1.3.2.3.1 Fs2-kafka

Fs2-kafka expone todas las funcionalidades del cliente oficial de *Apache Kafka* para *Java* a través de un API compatible y diseñada para trabajar con *Fs2*. Esta biblioteca, definida en el estudio arquitectónico del sistema (ver *2.1.1.3.1-Fs2*), ha sido empleada en su versión 3.0 para implementar el extractor de flujos de *Kafka* presente de base en el módulo validador, dada su compatibilidad con *Fs2* y a fin de evitar manipular el API de *Kafka* para *Java* (pensado para el paradigma orientado a objetos) directamente.

El sitio oficial de la biblioteca puede encontrarse en el siguiente enlace: https://fd4s.github.io/fs2-kafka/.

6.1.3.2.4 Http4s

Http4s (ver en la arquitectura del sistema: *2.1.2.1.1.1-http4s*) es el *framework* empleado inicialmente por el cliente en el API REST de *RDFShape*, y el utilizado para ampliar el *API* e incluir la funcionalidad de servidor *WebSockets*. Para el proyecto, se ha utilizado la versión 1.0.

El sitio oficial del *framework* puede encontrarse en el siguiente enlace: https://http4s.org/.

6.1.3.2.5 Circe

Circe es una biblioteca para la manipulación de JSON en *Scala* y dentro el ecosistema de *cats-effect*. *Circe* ofrece varias utilidades para leer JSON desde *Strings*, así como crear, manipular y exportar programáticamente objetos JSON en memoria.

Esta herramienta hace un fuerte uso codificadores y decodificadores, funciones con la capacidad de convertir objetos JSON en objetos del dominio de la aplicación y viceversa, los cuales le permiten trabajar con *http4s* para procesar y validar las peticiones de los clientes.

El sitio oficial de la biblioteca puede encontrarse en el siguiente enlace: https://circe.github.io/circe/.

Ilustración 85. Logo de Circe

6.1.3.2.6 Rho

Rho (ver en la arquitectura del sistema: *2.1.2.1.2.1*-Rho (ρ)) es una biblioteca que amplía las capacidades de *http4s*. Esta herramienta se ha empleado en su versión 0.23 en el módulo API con la finalidad de ayudar a validar las entradas de los usuarios, así como generar de manera automatizada el fichero de documentación del API REST para su posterior publicación en *SwaggerHub*.

Para las rutas del API y objetos del dominio que *Rho* no es capaz de procesar automáticamente, se crean definiciones personalizadas en un paquete *swagger*.

El código de la biblioteca puede encontrarse en el siguiente enlace: https://github.com/http4s/rho.

6.1.3.2.7 Mongo4cats

Mongo4cats (ver en diseño físico de los datos: *5.4.2-Integración del SGBD en Nuestro Sistema*) es la biblioteca empleada en el módulo API para conectar el sistema con la base de datos *MongoDB* encargada de almacenar los enlaces de los usuarios. La versión empleada en el desarrollo ha sido la 0.4.5.

El sitio oficial de la biblioteca puede encontrarse en el siguiente enlace: https://kirill5k.github.io/mongo4cats/.

6.1.3.2.8 Logback

Logback (ver en la arquitectura del sistema: *2.2.2.1.3-Sistema de logs: Logback + scala-logging*), junto con *scala-logging* es el sistema empleado para tomar los registros de la aplicación.

El módulo API cuenta con varios ajustes destinados a configurar la toma de *logs*:

- http4s: Se activa un middleware incluido con el *framework* para comunicarse con *Logback* y registrar datos de las peticiones recibidas por la aplicación y las respuestas generadas.
- CLI: Se crea un nuevo parámetro en la interfaz de línea de comandos: *-v*, *--verbose*. El parámetro se puede usar acumuladamente para definir qué nivel de mensajes queremos que el sistema muestre en la terminal.

Así mismo, el sistema cuenta con un fichero de configuración *logback.xml* configurado con varios *appenders* o sistemas de escritura de registros:

- *ConsoleAppender*: Encargado de mostrar los mensajes de *log* en consola. EL nivel de los mensajes mostrados depende de un filtro configurado al inicio del sistema en función de los parámetros con los que se ejecute el programa.
- *RollingFileAppender*: Encargado de escribir los mensajes de *log* en ficheros y de rotarlos y comprimirlos según una política de nombres y tamaño máximo. Para evitar saturar el disco duro del servidor, sólo los mensajes de gravedad *INFO* o mayor son tenidos en cuenta.
- *AsyncAppender*: Ayuda a ejecutar la escritura a ficheros del *appender* anterior de forma asíncrona para mejorar el rendimiento del sistema.

Los registros del sistema se ubicarán en una carpeta en la ubicación */logs* relativa al ejecutable del programa y el programa comprobará cada minuto si hay cambios en la configuración para aplicarlos sin necesidad de reiniciarse.

El sitio oficial de las bibliotecas puede encontrarse en los siguientes enlaces:

- Logback: https://logback.qos.ch/.
- scala-logging: https://github.com/lightbend/scala-logging.

6.1.3.2.9 Plugins SBT

6.1.3.2.9.1 Mdoc + Docusaurus

Mdoc (ver en la arquitectura del sistema: *2.1.3.3.2.1-mdoc*) y *Docusaurus* (ver en la arquitectura del sistema: *2.1.3.3.2.2-Docusaurus*) son las herramientas utilizadas para documentar el funcionamiento de los módulos validador y API a través de documentos en formato *Markdown* e insertarlos en una página web eficiente y responsiva.

Para las páginas desarrolladas con *Docusaurus* se crea una carpeta *website* en la raíz de cada proyecto, destinada a contener el subproyecto de la web, inicializado a través de *NodeJS*. A partir de ahí, se modifican los ficheros clave de configuración *docusaurus.config.js* y *sidebars.js* para adaptar el comportamiento y esquema de colores de la web a las necesidades de cada módulo.

Ambas herramientas se han empleado a través del *plugin* disponible para *SBT* para *Mdoc* en su versión 2.2.21.

El sitio oficial de las herramientas puede encontrarse en los siguientes enlaces:

- *Mdoc*: https://scalameta.org/mdoc/.
- *Docusaurus*: https://docusaurus.io/docs.

6.1.3.2.9.2 SBT CI Release

Este es un *plugin* para automatizar la publicación de binarios en repositorios de dependencias. En este proyecto se ha empleado para, una vez configurada una cuenta en *Sonatype*, publicar las nuevas versiones de la biblioteca de validación de flujos automáticamente desde el sistema de integración

continua. Una ventaja adicional de este *plugin* es que se encarga de la publicación del sistema en todas las versiones para las que está configurado para compilar, en este caso *Scala* 2 y *Scala* 3.

El código del *plugin* puede encontrarse en el siguiente enlace: https://github.com/sbt/sbt-ci-release.

6.1.3.2.9.3 Buildinfo

BuildInfo es un *plugin* que expande las capacidades de introspección del código, generando automáticamente una clase con valores estáticos accesibles al resto del programa y metadatos de este, por ejemplo: nombre de la aplicación, versión de la aplicación, versión de *Scala*, versión de *SBT*, etc.

El *plugin* ha sido utilizado en el API para mostrar información del módulo (nombre y versión) desde el menú de ayuda de la interfaz de comandos, así como en la biblioteca de validación para definir el nombre del consumidor de *Kafka* en base al nombre y versión del sistema cuando no se ha especificado ninguno.

El código del *plugin* puede encontrarse en el siguiente enlace: https://github.com/sbt/sbt-buildinfo.

6.1.3.2.10 React

React (ver en la arquitectura del sistema: *2.1.2.2.1.1-React*) ha sido el *framework* empleado en su versión 16.14 para la lógica e interfaces del módulo cliente.

Se ha hecho uso especialmente del API de contexto de *React*, con la capacidad de almacenar el estado de la aplicación de forma global y compartida entre algunos de los componentes de la jerarquía. Sincronizando los datos de la aplicación almacenados en el contexto con las APIs por defecto de los navegadores (*localStorage*, *sessionStorage*) se ha construido un mecanismo capaz de mantener los datos que los usuarios están operando entre las distintas pantallas del sistema e incluso entre sesiones de navegación, aumentando la usabilidad del cliente en base a los requisitos del sistema.

El sitio oficial del *framework* puede encontrarse en el siguiente enlace: https://reactjs.org/.

6.1.3.2.10.1 Axios

Axios (ver en la sección de arquitectura: *2.2.2.2.1-Construcción del Cliente: React + Bootstrap*) es el popular módulo empleado como cliente *HTTP* del cliente web, en línea con el código existente previamente a la refactorización del módulo. La versión empleada ha sido la 0.21.4.

El sitio oficial del módulo puede encontrarse en el siguiente enlace: https://axios-http.com/.

6.1.3.2.10.2 CodeMirror, YATE, YASHE

CodeMirror, *YATE* y *YASHE* (ver en la sección de arquitectura: *2.2.2.2.1-Construcción del Cliente: React + Bootstrap*) son el conjunto de módulos empleados para poner a disposición de los usuarios un método adecuado de entrada de datos RDF y esquemas *ShEx* en el sistema, en línea con el código existente previamente a la refactorización del módulo. Las versiones empleadas han sido las 5.65.2, 0.10.0 y 1.3.22, respectivamente.

El sitio oficial de los módulos puede encontrarse en los siguientes enlaces:

- CodeMirror: https://codemirror.net/.
- YATE: https://perfectkb.github.io/yate/.
- YASHE: https://www.weso.es/YASHE/.

6.1.3.2.10.3 React useWebSocket

React useWebSocket (ver en la sección de arquitectura: *2.2.2.2.1-Construcción del Cliente: React + Bootstrap*) es el cliente *WebSockets* empleado en el módulo cliente para comunicarse con el servidor. El módulo usa por debajo las APIs nativas del navegador, pero las integra como una función *hook* más de *React*. La versión del módulo empleada ha sido la 3.0.0.

El módulo puede encontrarse en el siguiente enlace: https://www.npmjs.com/package/react-use-websocket.

6.1.3.2.10.4 React Bootstrap, React Bootstrap Table Next

React Bootstrap y sus módulos acompañantes (ver en la sección de arquitectura: *2.2.2.2.1-Construcción del Cliente: React + Bootstrap*) han sido los módulos empleados para facilitar el trabajo de construir interfaces visualmente atractivas a la par que accesibles.

Las versiones empleadas han sido la 1.6.4 y 3.3.5, respectivamente. Así mismo, se han empleado otros módulos compatibles para ofrecer las funcionalidades de paginación, búsqueda y exportación de los resultados: *react-bootstrap-table2-paginator* (2.1.2), *react-bootstrap-table2-toolkit* (2.1.3) y *react-bootstrap-typeahead* (3.4.0).

El sitio oficial de los módulos puede encontrarse en los siguientes enlaces:

- React Bootstrap: https://react-bootstrap.github.io/.
- React Bootstrap Table Next: https://react-bootstrap-table.github.io/react-bootstrap-table2/.

6.1.3.3 *Herramientas auxiliares*

6.1.3.3.1 UMLet

UMLet es un programa construido en *Java* y de código abierto para la creación de todo tipo de diagramas UML. Cuenta tanto con una versión en línea conocida como *UMLetino* como con un programa de escritorio.

Se ha empleado el programa de escritorio en su versión 14.3 para crear los diagramas de clases, de secuencia, de flujo, de casos de uso, de paquetes y de despliegue expuestos a lo largo de la memoria.

El sitio oficial de la herramienta puede encontrarse en el siguiente enlace: https://www.umlet.com/.

Ilustración 86. Logo de UMLet

6.1.3.3.2 Diagrams.net

Diagrams.net, conocido popularmente como *draw.io*, es una herramienta gratuita de código abierto que permite la creación de todo tipo de diagramas desde una interfaz web y exportarlos en formatos XML o de imagen. Ofrece otros planes de pago enfocados a usuarios profesionales y equipos de trabajo.

Se ha empleado la herramienta para la creación de los diagramas entidad-relación presentes en la memoria.

El sitio oficial de la herramienta puede encontrarse en el siguiente enlace: https://app.diagrams.net/.

Ilustración 87. Logo de diagrams.net

6.1.3.3.3 Balsamiq Wireframes

Balsamiq es una herramienta profesional para el prototipado de interfaces de usuario. Ofrece tanto programas de escritorio como un entorno de trabajo en línea, así como varios planes de pago enfocados a equipos profesionales.

Se ha empleado la herramienta en su versión web para diseñar los bocetos y derivados de las interfaces de usuario durante la etapa de análisis (ver *4.7-ASI 8: Definición de Interfaces de Usuario*).

El sitio oficial de la herramienta puede encontrarse en el siguiente enlace: https://balsamiq.com/wireframes/.

Ilustración 88. Logo de Balsamiq Wireframes

6.1.3.3.4 Microsoft Office

Se han empleado los siguientes programas de la suite de ofimática *Microsoft Office* a través del servicio de suscripción *Office 365*.

El sitio oficial de la herramienta puede encontrarse en el siguiente enlace: https://www.microsoft.com/microsoft-365.

6.1.3.3.4.1 Word

Procesador de textos empleado en la redacción de la memoria.

6.1.3.3.4.2 Excel

Procesador de hojas de cálculo empleado en la creación del presupuesto del proyecto.

6.1.3.3.5 Microsoft Project

Project es el programa de gestión y planificación de proyectos desarrollado por *Microsoft* y empleado en el desarrollo de la planificación temporal del trabajo, incluida la gestión de las tareas y sus dependencias y los recursos dedicados a estas. El software se ha usado en su versión *Professional 2021*.

El sitio oficial de la herramienta puede encontrarse en el siguiente enlace: https://www.microsoft.com/microsoft-365/project/project-management-software.

6.1.3.3.6 ACheker

AChecker es una herramienta de evaluación de accesibilidad web diseñada en torno a las especificaciones del *WCAG* con el propósito de permitir a los desarrolladores web diagnosticar y corregir los problemas de accesibilidad de un proyecto. *ACheker* se encuentra disponible en línea y su uso es gratuito.

El sitio oficial de la herramienta puede encontrarse en el siguiente enlace: https://achecker.achecks.ca/checker/index.php.

6.1.3.3.7 GIMP

GIMP (abreviado de *GNU Image Manipulation Program*) es un programa gratuito, multiplataforma y de código abierto para la manipulación y edición de imágenes. El programa ha sido empleado en su versión 2.10.30 para la elaboración de los logos de la biblioteca de validación y del cliente web, así como para otros retoques menores en las imágenes de los sitios web de los módulos validador y API.

El sitio oficial del programa puede encontrarse en el siguiente enlace: https://www.gimp.org/.

Ilustración 89. Logo de GIMP

6.1.4 Aspectos de seguridad

En esta sección se explican brevemente las medidas de seguridad implementadas en la construcción del sistema. Cabe destacar que el sistema no trata con datos personales de ningún tipo, por lo que cuando se hace referencia a seguridad del sistema nos estamos refiriendo a:

- Verificar la validez de los datos introducidos por los usuarios en el sistema en cualquiera de sus formas (biblioteca de validación, API o cliente).
- Asegurar que el sistema maneja los errores, tanto aquellos esperados como los que no, de forma segura, sin comprometer el funcionamiento de la aplicación (paradas, *crashes*, etc.) y sin filtrar información sensible de las tecnologías utilizadas.
- Cifrar las conexiones de los usuarios con los módulos web que conforman *RDFShape*, y poner en marcha los mecanismos que procedan para evitar la carga y/o ejecución de recursos externos en el cliente web.

	SI	NO	N/A	Observaciones
Se hace validación en el cliente y en el servidor	×			Cliente: se verifica que todos los datos necesarios para una validación están presentes en la petición. Servidor: se validan los datos recibidos del cliente antes de ejecutar ninguna lógica de negocio.
Se han usado validadores nativos de un *framework*	×			Servidor: se emplean los decodificadores de *http4s* y *Rho* para convertir las entradas de los usuarios en objetos del dominio, validándolas en el proceso.
Se han validado las entradas JSON y XML con esquemas adecuados	×			-
Se hacen conversiones de tipo adecuadas con captura estricta de excepciones a las entradas leídas como *string*	×			Se verifica, entre otros, que los datos y esquemas suministrados como *strings* están bien formados.
Se comprueba el rango de los valores de entrada numéricos	×			Se verifica, primero en el servidor y luego al invocar la biblioteca de validación, que los datos están en rangos adecuados.
Se comprueba la longitud mínima y máxima para *strings*	×			Se verifica la longitud de los esquemas y *ShapeMaps* para prevenir la creación

				de enlaces a operaciones que usan *strings* demasiado grandes.
Se comprueba el conjunto de valores admitidos para entradas que solo puedan poseer un número finito de valores de forma estricta			×	-
Se usan expresiones regulares para validar cualquier otro tipo de dato estructurado que no encaje en las categorías anteriores			×	-
Se usan listas blancas de entradas admitidas, salvo para bloqueo de clientes conocidos			×	-
Si hay entradas de texto libre Unicode, se normalizan a una codificación común	×			-
Se usan bibliotecas de codificación de entradas anti XSS	×			-
Solo se permiten insertar datos en lugares considerados como seguros, usando codificación de datos adecuada a cada lugar	×			-
Si se admiten entradas en formato HTML, se usa una biblioteca para "limpiar" su contenido			×	-
Se usa el *flag HTTPOnly*			×	No se utilizan *cookies*.
Se implementa CSP	×			-
Se usan frameworks de desarrollo modernos y actualizados	×			-
Se toman medidas para validar los archivos que el usuario pueda subir	×			Recibidos e interpretados en el servidor como texto. Nunca son almacenados de forma permanente ni ejecutados en el servidor.

Se toman medidas para validar las direcciones de e-mail que vayan a usarse			×	No hay registro de usuarios.

Tabla 91. Medidas de seguridad tomadas en el código del sistema

Para consultar los procedimientos de seguridad adicionales relativos al despliegue del sistema, consultar el *Capítulo 7-Implantación y Aceptación del Sistema*.

6.2 CSI 2: GENERACIÓN DEL CÓDIGO DE LOS COMPONENTES Y PROCEDIMIENTOS

El objetivo de esta sección es complementar las explicaciones desarrolladas en los capítulos de análisis (ver *4.6-ASI 5: Análisis de Clases*) y desarrollo (ver *5.2-DSI 4: Diseño de Clases*) para las clases del sistema. Con esta finalidad se enlaza, para cada módulo, a la documentación de código disponible así como a su repositorio público, donde se puede encontrar cada subsistema en su versión más actualizada:

- Módulo de validación de flujos – *Comet*:
  - Repositorio: https://github.com/ulitol97/comet.
  - *Scaladoc*: https://ulitol97.github.io/comet/scaladoc/org/ragna/comet.
- Módulo API – *RDFShape API*
  - Repositorio: https://github.com/weso/rdfshape-api.
  - *Scaladoc*: https://www.weso.es/rdfshape-api/api/es/weso/rdfshape/.
  - *SwaggerHub*: https://app.swaggerhub.com/apis-docs/weso/RDFShape/.
- Módulo cliente – *RDFShape Client*
  - Repositorio: https://github.com/weso/rdfshape-client.

6.3 CSI 3: EJECUCIÓN DE LAS PRUEBAS UNITARIAS

En esta sección, se describen los resultados de las pruebas unitarias especificadas en el capítulo de diseño (ver *5.5.1*-Pruebas Unitarias).

6.3.1.1 *Pruebas de Validación*

Pruebas de validación		
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *Turtle*.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *JSON-LD*.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF válido en sintaxis *RDF-XML*.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *Turtle*.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *JSON-LD*.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF inválido en sintaxis *RDF-XML*.	✓	N/A

Tabla 92. Pruebas de validación – Comet (resultados)

6.3.1.2 *Pruebas de Extracción*

Pruebas de extracción		
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de listas.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de ficheros.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF proveniente del extractor de *Kafka*.	✗	El sistema queda esperando la recepción de datos incluso sin poder contactar con el origen del flujo y espera hasta emitir un *timeout*. Se ajusta el comportamiento del sistema para emitir la excepción lanzada por el consumidor de *Kafka* si no puede conectar con el servidor.

Tabla 93. Pruebas de extracción – Comet (resultados)

6.3.1.3 *Pruebas de Parada*

Pruebas de parada		
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF inválido con el validador configurado para ignorar los ítems inválidos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF inválido con el	✓	N/A

validador configurado para detenerse ante ítems inválidos.		
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF mal redactado con el validador configurado para ignorar los ítems erróneos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF mal redactado con el validador configurado para detenerse ante ítems erróneos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF con el validador configurado para abortar la validación si no se leen datos del flujo instantáneamente.	✓	N/A

Tabla 94. Pruebas de parada – Comet (resultados)

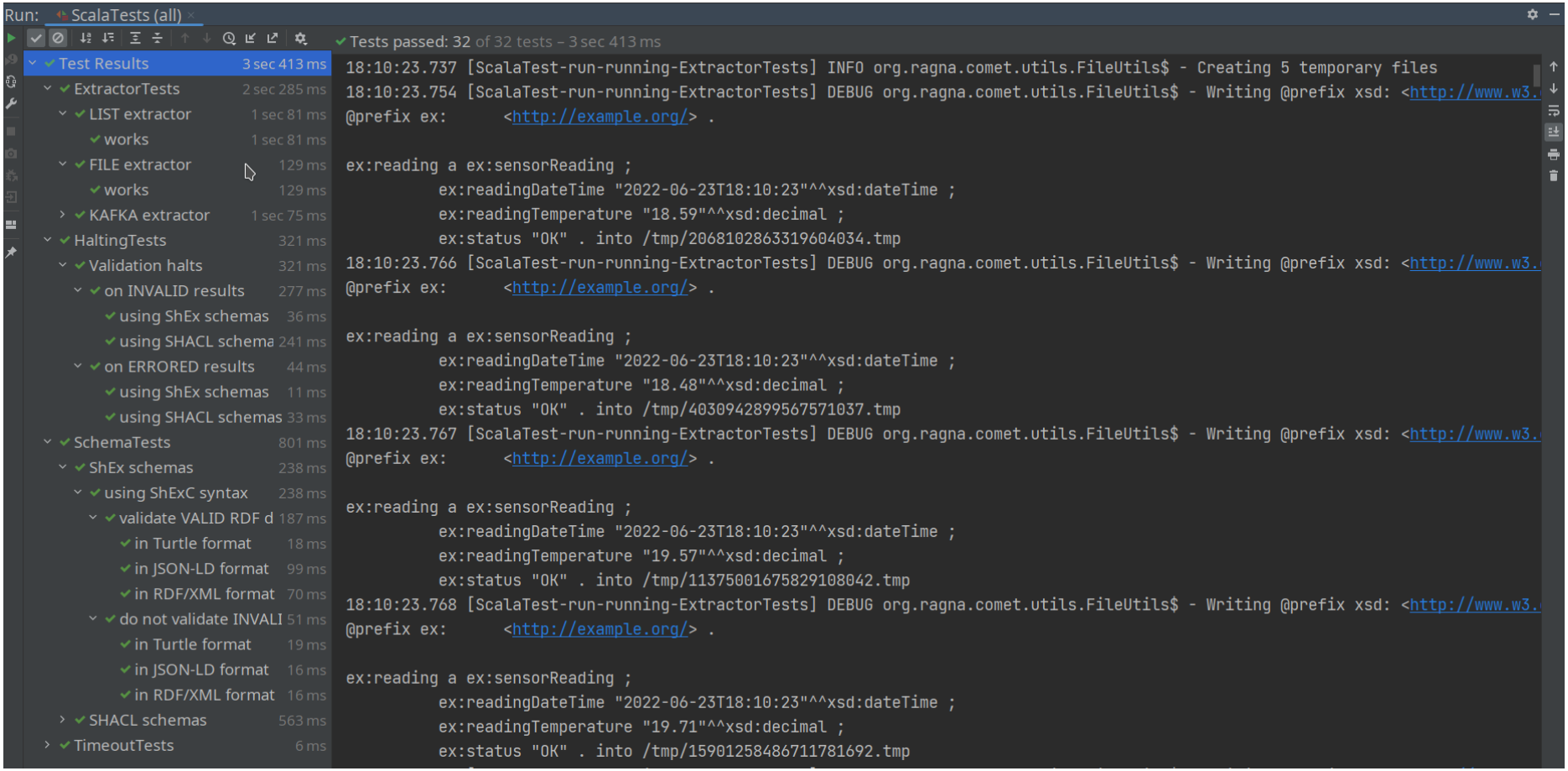


Ilustración 90. Pruebas unitarias - Resultados finales

6.4 CSI 4: EJECUCIÓN DE LAS PRUEBAS DE INTEGRACIÓN

En esta sección, se describen los resultados de las pruebas de integración especificadas en el capítulo de diseño (ver *5.5.2-Pruebas de Integración*).

6.4.1.1 *Módulo de Validación de Flujos*

Pruebas de integración		
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF procedente de un flujo de datos *Kafka* no disponible o inaccesible.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF válido procedente de un flujo de datos *Kafka* ejecutándose localmente.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF inválido procedente de un flujo de datos *Kafka* ejecutándose localmente.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Validar mediante *ShEx* un conjunto de RDF válido procedente de un flujo de datos *Kafka* accesible a través de internet.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**

Validar mediante *ShEx* un conjunto de RDF inválido procedente de un flujo de datos *Kafka* accesible a través de internet.	✓	N/A

Tabla 95. Pruebas de integración - Comet (resultados)

6.4.1.2 *Módulo API*

Pruebas de integración		
Prueba	**Resultado**	**Acciones correctivas**
Lanzar una petición contra una ruta inexistente del API.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos en formato distinto de JSON.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el servidor de origen de los datos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el *topic* de origen de los datos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el puerto de origen de los datos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**

Enviar una petición de validación de flujos sin especificar el grupo de consumidor de los datos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el formato de los datos RDF por consumir.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar la inferencia a aplicar sobre los datos por consumir.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el esquema *ShEx* contra el que validar los datos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos especificando un esquema *ShEx* inválido.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el *ShapeMap* contra el que validar los datos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos	✓	N/A

especificando un *ShapeMap* inválido.		
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el comportamiento del validador ante ítems inválidos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos sin especificar el comportamiento del validador ante ítems erróneos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos contra una combinación de servidor/*topic* inexistente.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente válidos y el validador configurado para no detenerse.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos con todos los campos pero con datos de tipo incorrecto.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación con todos	✓	N/A

los campos pero con datos fuera de los rangos permitidos.		
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente válidos y el validador configurado para detenerse ante elementos inválidos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Enviar una petición de validación de flujos válida contra un flujo de datos aleatoriamente erróneos y el validador configurado para detenerse ante elementos erróneos.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Detener, desde el cliente, la conexión con el servidor durante una validación.	✓	N/A

Tabla 96. Pruebas de integración – API (resultados)

6.4.1.3 *Módulo Cliente*

Para las pruebas realizadas desde la interfaz del cliente, se mantendrá operativo el servidor de *Kafka* empleado

Pruebas de integración		
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos sin rellenar el servidor de origen.	✓	N/A

Prueba	Resultado	Acciones correctivas
Invocar una validación de flujos sin rellenar el puerto de origen.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos sin rellenar el *topic* de origen.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos sin rellenar el esquema *ShEx* a emplear.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos sin rellenar el *ShapeMap* a emplear.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos contra un esquema *ShEx* mal formateado.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos contra un *ShapeMap* mal formateado.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos válida contra una combinación de servidor/*topic* inexistente o no disponible.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Invocar una validación de flujos válida contra un flujo existente.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**

Pausar una validación de flujos en curso.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Reanudar una validación de flujos en pausa.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Buscar entre los resultados de una validación en curso con un término no vacío (1+ caracteres).	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Buscar entre los resultados de una validación en curso un término vacío (0 caracteres).	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Limpiar los resultados de una validación en curso.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Exportar los resultados de una validación en curso.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Ordenar los resultados de una validación en curso en base a un campo nuevo.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Ordenar los resultados de una validación en curso en base a un campo ya ordenado.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**

Generar un *permalink* a una operación de validación.	✓	N/A
Prueba	**Resultado**	**Acciones correctivas**
Consumir un *permalink* a una operación de validación.	✓	N/A

Tabla 97. Pruebas de integración – Cliente (resultados)

6.5 CSI 5: EJECUCIÓN DE LAS PRUEBAS DEL SISTEMA

6.5.1 Prueba de Usabilidad

A continuación, se adjuntan los resultados de los cuestionarios planteados en la etapa de diseño para la usabilidad del módulo cliente (ver *5.5.3-Pruebas de Usabilidad*).

6.5.1.1 *Cuestionarios de evaluación*

Los cuestionarios de evaluación del sistema fueron cumplimentados por cuatro personas, todos ellos con el perfil informático que se presupone a los alumnos del máster en ingeniería web. Así mismo, cabe destacar que el tiempo promedio en completar el cuestionario fue, según *Microsoft Forms*, de tres minutos y cincuenta segundos (3:50), demostrando que se cumplió el objetivo de no ocupar más de 5 minutos del tiempo de los sujetos.

A continuación se adjuntan los resultados de los cuestionarios junto con el porcentaje de los votos totales recibidos por cada opción:

6.5.1.1.1 Preguntas de carácter general y del contexto del sistema

Del 1 al 5, ¿cuánta experiencia previa considera que posee en el dominio de Web Semántica, así como la manipulación y validación de RDF? *Si es alumno del Máster en Ingeniería Web, cíñase a la experiencia previa a cursar la asignatura de Web Semántica*	
1	0 %
2	25 %
3	50 %
4	25 %
5	0 %
¿Ha usado alguna vez herramientas web similares (funcionalidad, apariencia, etc.) a RDFShape?	
Sí, he empleado software similar para trabajar con RDF, Shapes y otras tecnologías semánticas.	25 %
Sí, empleo regularmente otras herramientas similares pero especializadas en otros dominios.	25 %
Sí, aunque sólo utilizo este tipo de herramientas puntualmente.	50 %
No, nunca	
¿Qué busca principalmente en un programa?	
Que sea fácil de usar.	0 %
Que sea intuitivo.	0 %
Que sea rápido.	0 %

Que tenga todas las funciones necesarias.	100 %

Tabla 98. Cuestionario de evaluación - Preguntas generales (respuestas)

6.5.1.1.2 Preguntas sobre la funcionalidad del sistema

Por favor valore, del 1 al 5, las siguientes cuestiones sobre las funcionalidades ofrecidas por *RDFShape*:

¿Funciona cada tarea como usted espera?	
1	0 %
2	0 %
3	0 %
4	75 %
5	25 %
¿Cree que el tiempo de respuesta de la aplicación es razonable?	
1	25 %
2	25 %
3	50 %
4	0 %
5	0 %
¿Le han resultado útiles las visualizaciones generadas por la aplicación a la hora de comprender o compartir resultados?	
1	0 %
2	0 %
3	0 %
4	25 %
5	75 %
¿Le ha resultado útil el sistema de permalinks de la aplicación para guardar o compartir resultados?	
1	0 %
2	0 %
3	0 %
4	25 %
5	75 %

Tabla 99. Cuestionario de evaluación - Preguntas de funcionalidad (respuestas)

6.5.1.1.3 Preguntas sobre la usabilidad del sistema

Por favor valore, del 1 al 5, las siguientes cuestiones sobre la facilidad de uso de *RDFShape*:

¿Le resulta sencillo el uso de la aplicación en general?	
1	0 %
2	0 %
3	25 %

4	25 %
5	50 %
¿Sabe en todo momento dónde está dentro de la aplicación?	
1	0 %
2	0 %
3	25 %
4	0 %
5	75 %
¿Cree que el programa está bien estructurado?	
1	0 %
2	0 %
3	0 %
4	75 %
5	25 %
¿Cree que el diseño de las pantallas es claro y atractivo?	
1	0 %
2	0 %
3	0 %
4	100 %
5	0 %
¿Existe ayuda para las funciones en caso de que tenga dudas?	
1	0 %
2	0 %
3	25 %
4	25 %
5	50 %
¿Le son útiles los mensajes informativos y de error emitidos por la aplicación en el desarrollo de sus funciones?	
1	0 %
2	0 %
3	0 %
4	100 %
5	0 %
¿Le es sencillo operar con los datos según se va desplazando entre pantallas?	
1	0 %
2	0 %
3	0 %
4	50 %
5	50 %

Tabla 100. Cuestionario de evaluación - Preguntas de usabilidad (respuestas)

6.5.1.1.4 Preguntas sobre la accesibilidad del sistema

Por favor valore, del 1 al 5, las siguientes cuestiones sobre la accesibilidad de *RDFShape*:

¿El tipo y tamaño de letra son adecuados?	
1	0 %
2	0 %
3	0 %
4	75 %
5	25 %
¿Los iconos e imágenes usados son adecuados?	
1	0 %
2	0 %
3	0 %
4	75 %
5	25 %
¿Los colores empleados son adecuados?	
1	0 %
2	0 %
3	0 %
4	50 %
5	50 %

Tabla 101. Cuestionario de evaluación - Preguntas de accesibilidad (respuestas)

6.5.1.1.5 Observaciones

No se recibieron ningunos comentarios adicionales por parte de los alumnos.

Opcionalmente, cualquier otra observación o sugerencia
N/A

Tabla 102. Cuestionario de evaluación - Observaciones (respuestas)

6.5.1.2 *Cuestionario para el responsable de las pruebas*

Aspecto observado	**Notas y observaciones**
El usuario entiende rápidamente el funcionamiento de la herramienta.	Los usuarios comprenden rápidamente el *modus operandi* de la web y no presentan problemas para navegar por ella.

Tiempo empleado por el usuario para rellenar los datos necesarios para invocar las operaciones de la herramienta.	Los usuarios se toman su tiempo para redactar las entradas de la herramienta, especialmente si escriben RDF o *Shape Expressions* a mano. La mayoría de ellos toma el hábito de usar los ejemplos disponibles y modificarlos según les convenga. La dificultad para rellenar los campos no está relacionada con la forma en que está construido el sitio web.
Errores cometidos por el usuario en el desarrollo de las actividades (por ejemplo: el usuario se desubica, no sabe navegar entre resultados, etc.)	La mayoría de los usuarios tienen claro cómo navegar en el formulario de entrada de datos y entre los distintos resultados. Algunos tienen dificultades para darse cuenta de si la herramienta lanza un error o empieza a emitir resultados, al quedar los resultados tan "lejos" del botón de enviar.

Tabla 103. Cuestionario para el responsable de las pruebas de usabilidad (respuestas)

6.5.1.3 *Análisis de los resultados*

Los resultados en general son positivos, especialmente si tenemos en cuenta que la aplicación web sobre la que se ha preguntado a los usuarios es el módulo al que se le ha dedicado menos tiempo del deseado puesto que no es el foco del proyecto sino un escaparate de la funcionalidad desarrollada para el *back-end*.

Se dan por buenos los resultados recibidos en los cuestionarios de evaluación y, atendiendo a las sugerencias del supervisor, se implementa un nuevo comportamiento por el cual el sistema se desplaza automáticamente a la sección de resultados después de que el usuario invoque una operación en el servidor.

6.5.2 Pruebas de Accesibilidad

Como se ha mencionado en el diseño del plan de pruebas (ver *5.5.4-Pruebas de Accesibilidad*), se emplea la herramienta automática *AChecker* sobre las direcciones URL en las que se encuentra desplegado el módulo cliente durante la etapa de pruebas.

Los objetivos son verificar que las páginas seleccionadas cumplen el nivel *AA* del *WCAG 2.0*. El resto de las advertencias y problemas potenciales detectados se resolverán de ser factible en el plazo de tiempo disponible. En el caso contrario, se tendrán en cuenta de cara a futuras ampliaciones del sistema.

6.5.2.1 *Evaluación con AChecker*

Se evalúan todas las pantallas que se han refactorizado o ampliado en el transcurso del proyecto, si bien el foco debe estar sobre la pantalla de validación mediante *Shape Expressions*.

6.5.2.1.1 Pantalla de inicio del sistema

Resultados:

- Errores: 0.
- Advertencias y problemas potenciales: 23.

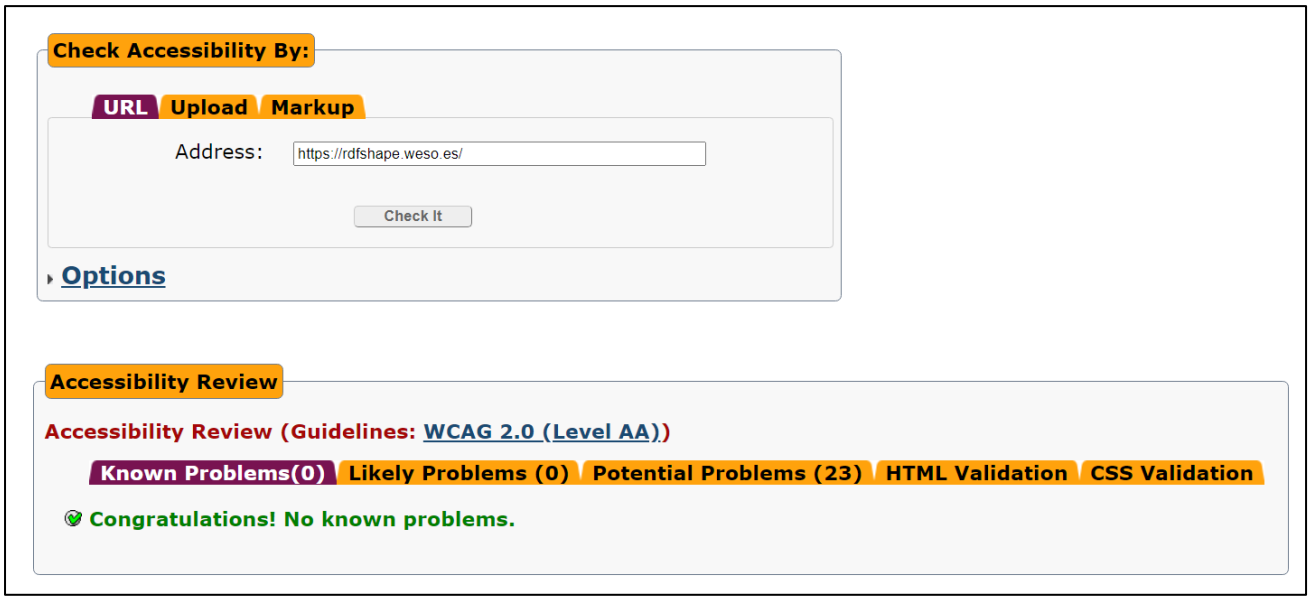


Ilustración 91. Pruebas de accesibilidad - Pantalla de inicio

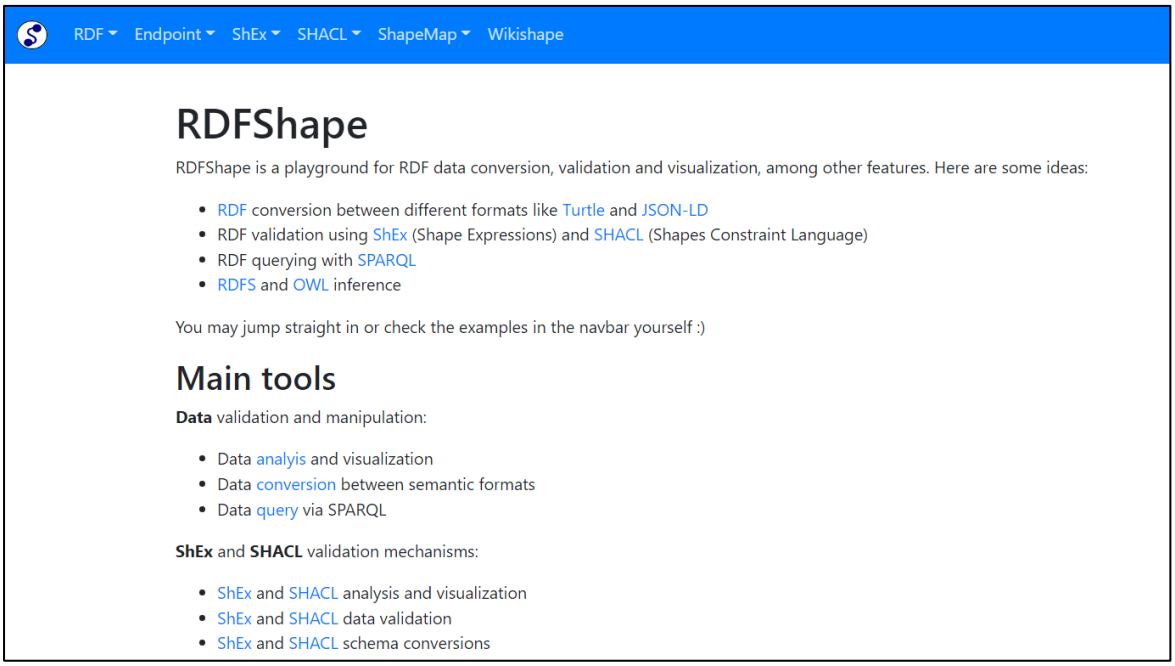


Ilustración 92. Pantalla de inicio del sistema

6.5.2.1.2 Pantalla de ejecución de consultas SPARQL

Resultados:

- Errores: 0.
- Advertencias y problemas potenciales: 23.

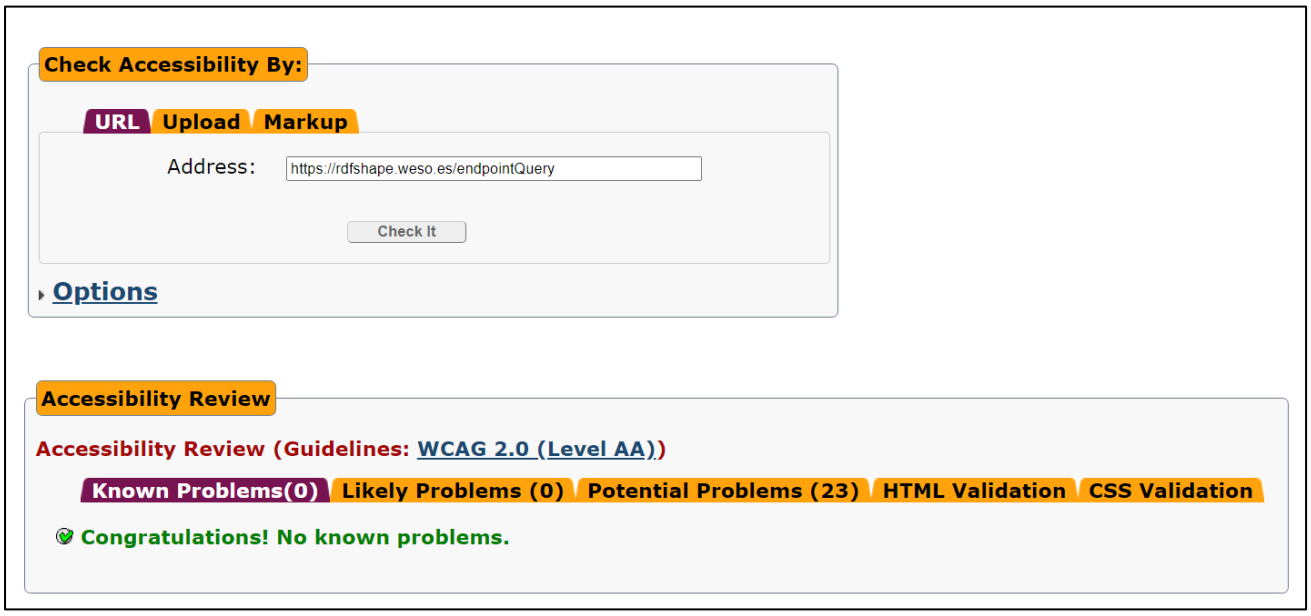


Ilustración 93. Pruebas de accesibilidad - Pantalla de consultas SPARQL

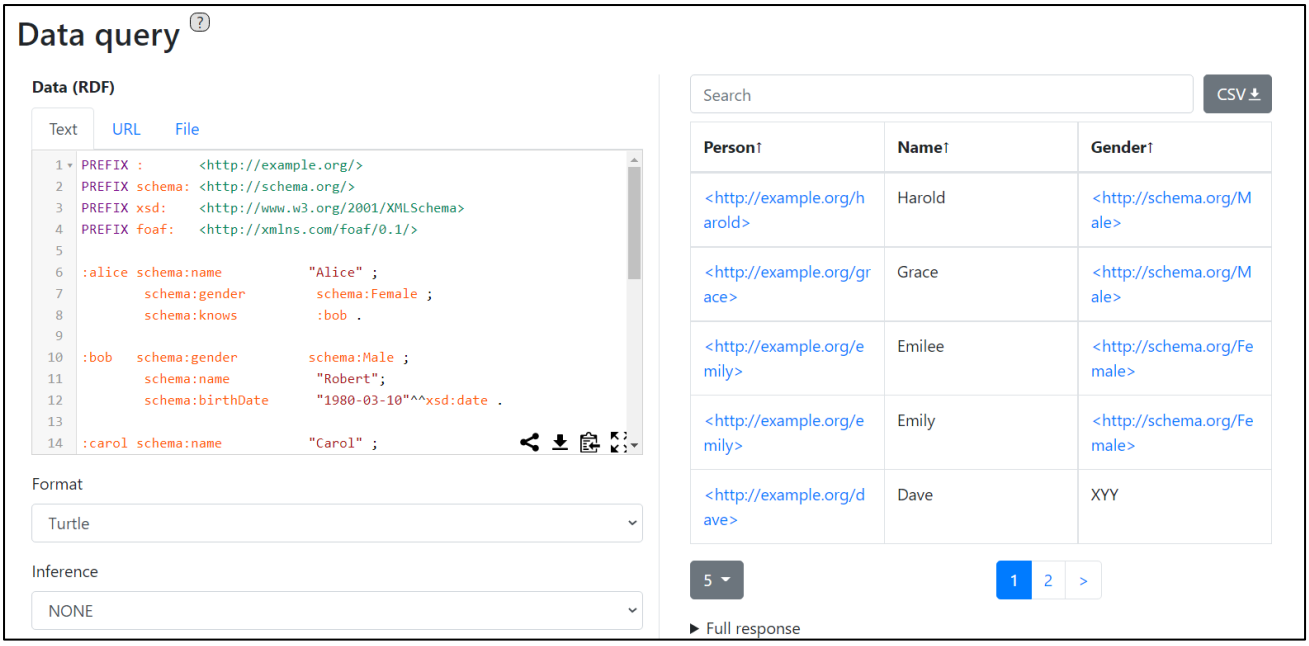


Ilustración 94. Pantalla de ejecución de consultas SPARQL

6.5.2.1.3 Pantalla de análisis y visualización de RDF

Resultados:

- Errores: 0.
- Advertencias y problemas potenciales: 23.

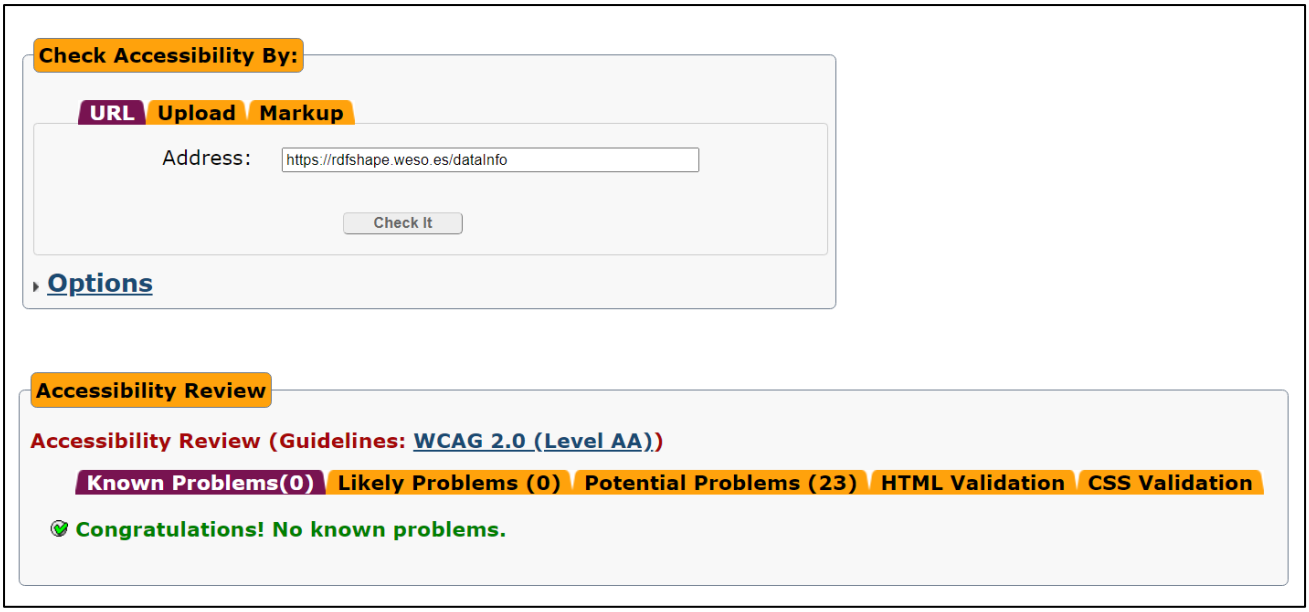


Ilustración 95. Pruebas de accesibilidad - Pantalla de análisis RDF

Ilustración 96. Pantalla de análisis y visualización de RDF

6.5.2.1.4 Pantalla de análisis y visualización de *Shape Expressions*

Resultados:

- Errores: 0.
- Advertencias y problemas potenciales: 23.

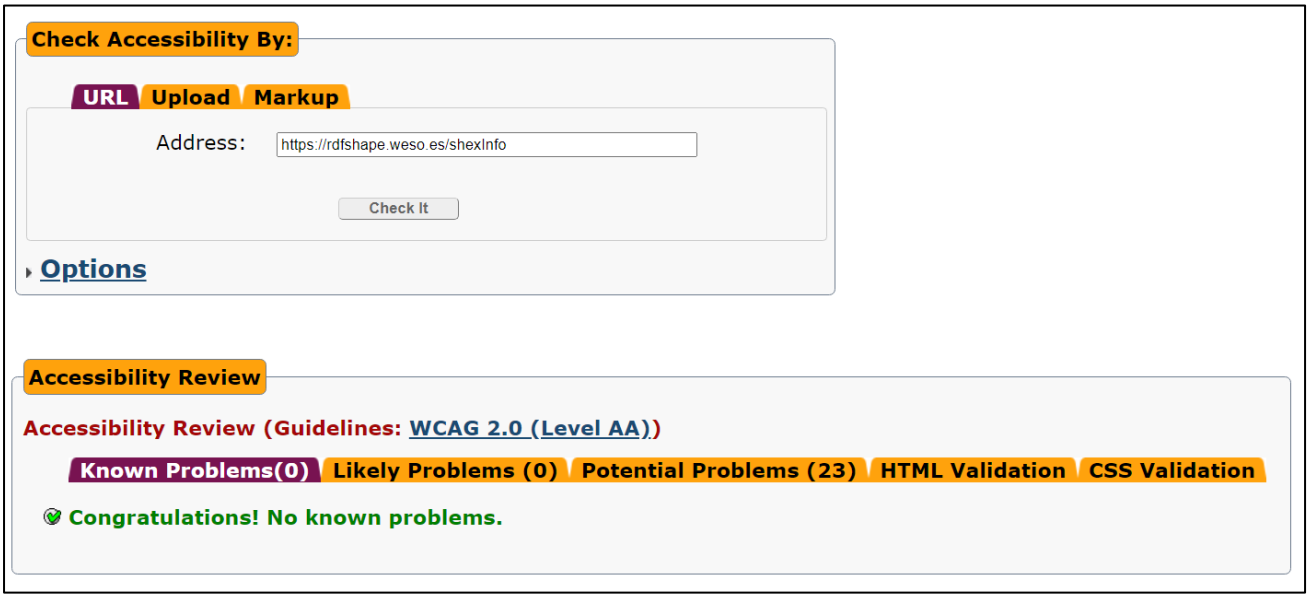


Ilustración 97. Pruebas de accesibilidad - Pantalla de análisis ShEx

Ilustración 98. Pantalla de análisis y visualización de ShEx

6.5.2.1.5 Pantalla de validación de datos RDF mediante *Shape Expressions*

Resultados:

- Errores: 0.
- Advertencias y problemas potenciales: 23.

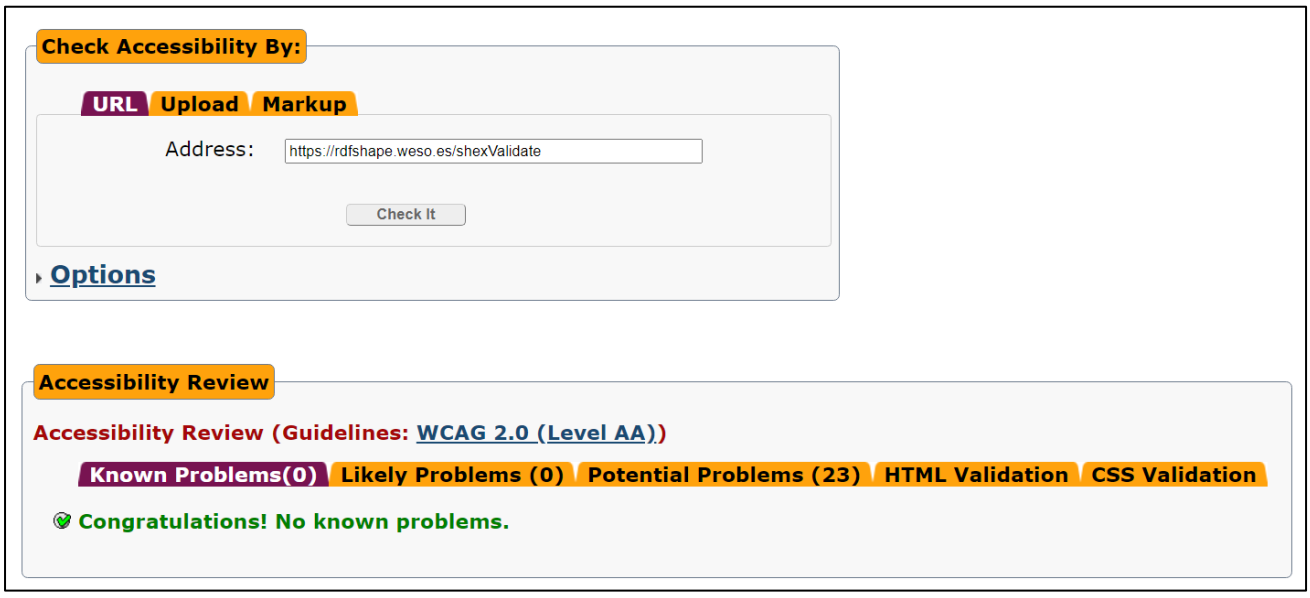


Ilustración 99. Pruebas de accesibilidad - Pantalla de validación ShEx

Ilustración 100. Pantalla de validación de RDF mediante ShEx

6.5.2.2 *Análisis de resultados*

Todas las páginas analizadas superan el análisis de accesibilidad con éxito, por lo que se dan los resultados por satisfactorios.

Finalmente, se observa que los mismos problemas potencias son advertidos por la herramienta en cada una de las páginas, lo que implica que son incidencias ligadas al HTML base de la web. Estos problemas son leves y mayormente relacionados con la dirección del texto y otras sugerencias menores.

6.5.3 Pruebas de rendimiento

Para el plan de pruebas descrito en el capítulo de diseño (ver *5.5.5-Pruebas de Rendimiento*), se ejecuta cada uno de los *tests* cinco veces y se hace la media de tiempos para determinar el resultado final.

6.5.3.1 *Equipo de pruebas*

Es necesario especificar las características principales del equipo en que se van a llevar a cabo las pruebas de rendimiento:

- Modelo: MSI GE 75 Raider 10SF.
- Sistema Operativo: Ubuntu 22.04 (LTS).
- Procesador: Intel Core i7-10750H (x64, 6 núcleos, 12 hilos (*Hyper Threading*)).
- Memoria RAM: 32GB (DDR4, 2666MHz).

La información anterior se ha obtenido a través del programa *inxi* **[104]**.

6.5.3.2 *Ejecución de las pruebas*

Se ejecuta cada caso planteado dentro de una *suite* de pruebas, de forma que sea el propio sistema quien mida el tiempo dedicado a cada *test*.

Ítems	Hilos	T1 (s)	T2 (s)	T3 (s)	T4 (s)	T5 (s)	T promedio (s)	Mejora vs. Monohilo (%)
4000	1	7,626	8,354	7,025	8,539	8,329	7,975	N/A
4000	2	4,623	4,682	4,594	4,787	4,852	4,708	40,97%
4000	4	3,586	3,559	4,153	4,033	3,914	3,849	51,73%
10000	1	12,621	12,586	12,119	13,904	13,463	12,939	N/A
10000	2	8,452	8,519	9,152	9,077	9,143	8,869	31,46%
10000	4	8,399	7,688	7,962	8,130	8,034	8,043	37,84%
30000	1	26,931	25,251	24,123	24,959	24,694	25,192	N/A
30000	2	24,358	23,506	23,642	23,855	23,622	23,797	5,54%
30000	4	24,509	22,829	22,729	22,888	23,780	23,347	7,32%

Tabla 104. Pruebas de rendimiento - Resultados

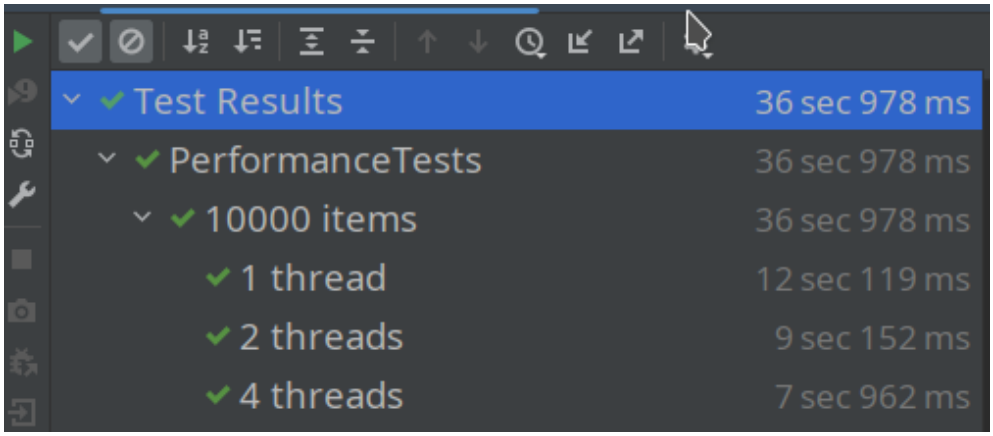


Ilustración 101. Resultados de las pruebas de rendimiento en IntelliJ IDEA (10000 ítems, tiempo nº3)

6.5.4 Análisis de los resultados

Se verifica que el sistema mejora sustancialmente su rendimiento según se utilizan más hilos de ejecución, con lo que se dan las pruebas de rendimiento por superadas.

Es importante subrayar que las mejoras de rendimiento obtenidas por la paralelización del proceso no son ilimitadas, como puede contemplarse al ver los porcentajes de mejora decrecer según

aumenta la carga de trabajo. Es más, el sistema intentará hacer un uso inteligente de los recursos disponibles pero solicitar más hilos de los razonables puede llegar a penalizar el rendimiento en función del sistema en que se ejecute la validación.

6.6 CSI 6: ELABORACIÓN DE LOS MANUALES DE USUARIO

6.6.1 Manual de Instalación

6.6.1.1 *Módulo validador (Comet)*

El módulo validador consiste de la biblioteca de validación de flujos de nombre *Comet*, desarrollada en *Scala*. Instalar la biblioteca es sinónimo de obtener sus binarios compilados y añadirlos como dependencia a un proyecto.

Existen dos métodos principales de instalación de *Comet*:

1. Gestor de dependencias (recomendado):
   Dado que los binarios de la biblioteca se encuentran disponibles públicamente en los repositorios de *Sonatype*, la opción de instalación más sencilla es añadir la biblioteca a las dependencias de un proyecto y dejar que sea el gestor de dependencias quien se encarga de descargar y añadir al *classpath* el código del módulo.
2. Instalación manual:
   Alternativamente, y dado que el proyecto es de código abierto, es posible descargar el código fuente, compilarlo y añadir los binarios generados como dependencia del proyecto manualmente.

6.6.1.1.1 Prerrequisitos

Para emplear la biblioteca de validación es indispensable tener instalado el *Java Runtime Environment* en su versión 11 o superior. Para la instalación manual será necesario contar, además, con *Scala* y *SBT* instalados en el equipo que vaya a compilar el módulo:

- Java Development Kit (JDK): Versión 11 o superior.
- Scala: Versión *2.13.x* (compilación para Scala 2) o versión *3.1.x* o superior (compilación para *Scala* 3).
- SBT: Versión *1.5* o superior.

Se recomienda seguir las guías oficiales para la instalación de *Scala* y *SBT*, que pueden encontrarse en el siguiente enlace: https://www.scala-lang.org/download/.

6.6.1.1.2 Instalación

6.6.1.1.2.1 Gestor de dependencias

Para descargar e instalar *Comet* como dependencia en un proyecto *Scala* que utilice *SBT*, es necesario añadir la siguiente dependencia en el fichero *build.sbt*:

```
libraryDependencies += "io.github.ulitol97" %% "comet" % "0.1.7"
```

Código 1. Adición de Comet como dependencia

Si por algún motivo *SBT* no es capaz de resolver y descargar la dependencia, es necesario añadir la siguiente configuración al fichero *build.sbt* especificando *Sonatype* como posible origen de los binarios:

```
resolvers ++= Seq(
 Opts.resolver.sonatypeSnapshots,
   Opts.resolver.sonatypeReleases
 )
```

Código 2. Adición de Sonatype al gestor de dependencias

Para la descarga e instalación mediante otros gestores de dependencias (*Maven*, *Gradle*, etc.), dirigirse a la página de publicación en *Sonatype* y seleccionar la versión y gestor de dependencias deseado: https://search.maven.org/artifact/io.github.ulitol97/comet_3.

Las instrucciones de instalación completas y actualizadas pueden encontrarse en el sitio web desarrollado para el proyecto: https://ulitol97.github.io/comet/docs/installation/instructions.

6.6.1.1.2.2 Manual

Para compilar y utilizar la biblioteca *Comet* manualmente es necesario seguir los siguientes pasos:

1. Descargar el código fuente del módulo de su repositorio en *GitHub*: https://github.com/ulitol97/comet.
2. Dirigirse a la carpeta del proyecto y, desde una terminal, lanzar *SBT*. Si se tiene *SBT* instalado y accesible, el comando de lanzamiento será "*sbt*". *SBT* cargará el proyecto y descargará las dependencias necesarias.
3. Configurar, a través del fichero *build.sbt* la configuración para el empaquetado y exportación del proyecto. La guía oficial se puede consultar aquí: https://www.scala-sbt.org/1.x/docs/Howto-Package.html.
4. Una vez todo esté listo, ejecutar el comando *comet/package* desde la consola de *SBT* para generar el *.jar* del módulo principal del proyecto.
5. Buscar el fichero *comet_{versión}.jar* dentro del directorio *{raíz_proyecto}/api/target*.
6. Añadir el *.jar* como dependencia en el proyecto en que se quiera integrar.

6.6.1.2 *Módulo API*

El módulo API es un programa que, una vez ejecutado en el servidor en que se quiere desplegar, escucha y atiende peticiones HTTP recibidas de los clientes.

La instalación del sistema en el servidor objetivo se puede hacer de dos maneras:

1. Instalación manual: Implica descargar el código del proyecto y ejecutarlo de una de las siguientes maneras, configurando su comportamiento mediante argumentos de la terminal:
    a. Ejecutar el módulo desde SBT.
    b. Generar un ejecutable del módulo y ejecutarlo desde el sistema operativo del servidor.
2. Instalación mediante *Docker* (recomendado): Construir o descargar la imagen *Docker* del sistema para posteriormente ejecutarla a través de *Docker* como un contenedor, configurando el comportamiento del sistema a través de variables de entorno.

Las instrucciones de instalación completas y actualizadas pueden encontrarse en el sitio web desarrollado para el proyecto: https://www.weso.es/rdfshape-api/docs/api-deployment/deployment_overview/.

6.6.1.2.1 Prerrequisitos

Para ejecutar el API en cualquiera de las alternativas manuales es preciso tener instalado el *Java Runtime Environment* en su versión 11 o superior. Para ejecutar o construir el módulo será necesario, además, contar con *Scala* y *SBT* instalados en el equipo que vaya a compilar el módulo:

- Java Development Kit (JDK): Versión 11 o superior.
- Scala: Versión *2.13.x*.
- SBT: Versión *1.5* o superior.

Se recomienda seguir las guías oficiales para la instalación de *Scala* y *SBT*, que pueden encontrarse en el siguiente enlace: https://www.scala-lang.org/download/.

Si se desea construir o ejecutar la imagen *Docker* del módulo, tan sólo es necesario contar con *Docker* instalado en el equipo que vaya a realizar dichas tareas. Las guías oficiales de instalación y uso de *Docker* se encuentran en el siguiente enlace: https://docs.docker.com/engine/install/.

6.6.1.2.2 Instalación

6.6.1.2.2.1 Manual

6.6.1.2.2.1.1 Modo interactivo con SBT

Para descargar y dejar el módulo API listo para su ejecución con SBT son necesarios los siguientes pasos:

1. Descargar el código fuente del módulo de su repositorio en *GitHub*: https://github.com/weso/rdfshape-api.
2. Dirigirse a la carpeta del proyecto y, desde una terminal, lanzar *SBT*. Si se tiene *SBT* instalado y accesible, el comando de lanzamiento será "*sbt*". *SBT* cargará el proyecto y descargará las dependencias necesarias.

6.6.1.2.2.1.2 *Modo ejecutable binario*

Para generar un único ejecutable del programa y depositarlo en el sistema de archivos del servidor es necesario:

1. Descargar el código fuente y lanzar la consola de *SBT*, como en caso anterior.
2. Desde la consola de *SBT*, ejecutar el comando "*Universal/packageBin*". El sistema generará un fichero comprimido con el ejecutable y todas sus dependencias y lo depositará en la carpeta *{raíz_proyecto}/target/universal*.

6.6.1.2.2.2 Docker

Para obtener una copia de la imagen *Docker* del módulo, es posible:

- Descargarla desde *GitHub Container Registry*, donde está disponible en https://github.com/weso/rdfshape-api/pkgs/container/rdfshape-api. Las etiquetas de la imagen disponibles son:
  - ***stable***: Versiones estables subidas manualmente al repositorio.
  - ***<número_hash>***: Versiones creadas y subidas automáticamente por el sistema de integración continua, cuyo nombre es el *hash* del *commit* que las originó.

```
docker pull ghcr.io/weso/rdfshape-api:stable
```

Código 3. Descarga de la imagen Docker del módulo API

- Construirla localmente a partir del *Dockerfile* proporcionado con el código del sistema:
  - Descargar el código fuente del módulo desde su repositorio público.
  - Inspeccionar el fichero *Dockerfile* adjunto y realizar las modificaciones que se deseen.
  - Una vez listo, construir la imagen *Docker* personalizada del sistema. Desde la carpeta raíz del proyecto:

```
docker build . -t <nombre_imagen>:<etiqueta_imagen>
```

Código 4. Construcción de la imagen Docker del módulo API

6.6.1.3 *Módulo Cliente*

El módulo Cliente no es más que una aplicación web que contacta con el módulo API e interpreta sus respuestas y las complementa con visualizaciones y otras funcionalidades. La instalación de este subsistema puede entenderse desde dos perspectivas:

- Perspectiva del usuario final: Los usuarios finales del cliente no necesitan realizar ningún tipo de instalación, tan sólo acceder a la dirección de despliegue del módulo a través de un navegador web.

- Perspectiva del administrador del servidor: Para poder servir la página web a los usuarios finales, los administradores del servidor deben descargar y poder ejecutar el sistema. Existen dos maneras de servir la aplicación web desde el servidor objetivo:
    - Instalación manual: Descargar el código fuente del proyecto y ejecutarlo como un proyecto de *Node.js* a través de la utilidad *npm*.
    - Instalación mediante *Docker* (recomendado): Construir o descargar la imagen *Docker* del sistema para posteriormente ejecutarla a través de *Docker* como un contenedor, configurando el comportamiento del sistema a través de variables de entorno.

Las instrucciones de instalación completas y actualizadas pueden encontrarse en la *wiki* del proyecto: https://github.com/weso/rdfshape-client/wiki/Client-Deployment.

6.6.1.3.1 Prerrequisitos

Para acceder y hacer uso de la aplicación es necesario contar con un navegador web instalado en el equipo.

Por otra parte, para poder servir la aplicación web es necesario, para la instalación manual:

- *NodeJS*: Versión 12 o superior.
- *npm*: Versión 7 o superior.

Se recomienda seguir las guías oficiales para la instalación de *NodeJS* y *npm*, que pueden encontrarse en https://nodejs.org/en/download/ y https://docs.npmjs.com/downloading-and-installing-node-js-and-npm, respectivamente.

Si se desea construir o ejecutar la imagen *Docker* del módulo, tan sólo es necesario contar con *Docker* instalado en el equipo que vaya a realizar dichas tareas. Las guías oficiales de instalación y uso de *Docker* se encuentran en el siguiente enlace: https://docs.docker.com/engine/install/.

6.6.1.3.2 Instalación

6.6.1.3.2.1 Manual

Para descargar y dejar la aplicación web lista para su ejecución con *npm* son necesarios los siguientes pasos:

1. Descargar el código fuente del módulo de su repositorio en *GitHub*: https://github.com/weso/rdfshape-client.
2. Dirigirse a la carpeta del proyecto y, desde una terminal instalar las dependencias con "*npm install*".

6.6.1.3.2.2 Docker

Para obtener una copia de la imagen *Docker* del módulo, es posible:

- Descargarla desde *GitHub Container Registry*, donde está disponible en https://github.com/weso/rdfshape-api/pkgs/container/rdfshape-api. Las etiquetas de la imagen disponibles son:
    - ***stable***: Versiones estables subidas manualmente al repositorio.
    - ***<número_hash>***: Versiones creadas y subidas automáticamente por el sistema de integración continua, cuyo nombre es el *hash* del *commit* que las originó.

```
docker pull ghcr.io/weso/rdfshape-client:stable
```

Código 5. Descarga de la imagen Docker del módulo Cliente

- Construirla localmente a partir del *Dockerfile* proporcionado con el código del sistema:
    - Descargar el código fuente del módulo desde su repositorio público.
    - Inspeccionar el fichero *Dockerfile* adjunto y realizar las modificaciones que se deseen.
    - Una vez listo, construir la imagen *Docker* personalizada del sistema. Desde la carpeta raíz del proyecto:

```
docker build . -t <nombre_imagen>:<etiqueta_imagen>
```

Código 6. Construcción de la imagen Docker del módulo Cliente

6.6.2 Manual de Ejecución

6.6.2.1 *Módulo validador (Comet)*

La biblioteca de validación se ejecuta a través del código de los programas de terceros que la utilizan.

6.6.2.2 *Módulo API*

Las siguientes guías ejecución pueden encontrarse también en el sitio web del módulo:

- Ejecución manual: https://www.weso.es/rdfshape-api/docs/api-deployment/deployment_manual/.
- Ejecución con *Docker*: https://www.weso.es/rdfshape-api/docs/api-deployment/deployment_docker/.

6.6.2.2.1 Ejecución manual

6.6.2.2.1.1 Modo interactivo con SBT

Una vez seguidos los pasos del manual de instalación (ver *6.6.1.2.2.1.1-Modo interactivo con SBT*) y arrancada un terminal *SBT* desde la carpeta raíz del proyecto, ejecutar el sistema a través del comando "*run*" seguido de los argumentos deseados.

```
sbt run --port 8080 -vvv
```

Código 7. Comando de ejemplo de ejecución del módulo API desde SBT

En este caso, el sistema de ejecutará con la identidad y permisos del usuario que haya lanzado la consola de *SBT*. Así mismo, para detener el sistema en funcionamiento será necesario interrumpir el proceso (*Ctrl+C*) o bien finalizar el proceso que ejecuta *SBT*.

6.6.2.2.1.2 Modo ejecutable binario

Con el ejecutable generado en los pasos del manual de instalación (ver *6.6.1.2.2.1.2-Modo ejecutable binario*), lanzar el programa a través de la línea de comandos del equipo.

```
./rdfshape --https --silent
```

Código 8. Comando de ejemplo de ejecución del módulo API desde el ejecutable

Salvo que se especifique lo contrario, el programa se lanzará en primer plano en la terminal en nombre del usuario dueño de la sesión y será posible pararlo enviando una señal de terminación (*Ctrl+C*) o finalizando el proceso que ejecuta la terminal.

6.6.2.2.1.3 Interfaz de línea de comandos

Sea cual sea el modo de ejecución manual, el sistema posee una interfaz de línea de comandos para configurar su comportamiento en el momento de inicialización:

- ***--https***
  Intenta servir el API a través de *HTTPS*, buscando los certificados según lo especificado en las variables de entorno. El valor por defecto es ***false***.
- ***-p, --port <arg>***
  Puerto en que el API escuchará en busca de peticiones entrantes. Los valores deben estar en el rango 1-65535. El valor por defecto es ***8080***.
- ***-s, --silent***
  Activa el modo silencioso, en el que no se imprime ninguna información en pantalla durante la ejecución del sistema. El valor por defecto es ***false***.
- ***-t, --stream-timeout <arg>***
  Número entero especificando los segundos que el servidor debe esperar a recibir datos de un servidor externo durante una validación de flujos antes de abortar la validación por inactividad. El valor por defecto es ***40***.
- ***-v, --verbose***
  Muestra información adicional durante la ejecución del sistema (usar de forma acumulativa para obtener mayor nivel de detalle, tal que: *-vvv*). Por defecto, el sistema sólo mostrará mensajes de *log* con el nivel ***ERROR***.
- ***--version***
  Muestra en pantalla la versión del programa y finaliza la ejecución.
- ***--help***
  Muestra en pantalla el menú de ayuda del programa y finaliza la ejecución.

Los argumentos aceptados por el sistema pueden consultarse, junto con su descripción, lanzando el módulo con el parámetro *--help* así como en el sitio web: https://www.weso.es/rdfshape-api/docs/api-usage/usage_cli/.

6.6.2.2.1.4 Configuración HTTPS

Para cualquiera de los métodos de ejecución manuales, es posible configurar el sistema para servir el API a través de *HTTPS*. Para ello, es necesario contar con un certificado válido y almacenado en el *Java Keystore* del sistema, así como especificar las siguientes variables de entorno:

- **KEYSTORE_PATH**: Ubicación del *keystore* con el certificado.
- **KEYSTORE_PASSWORD**: Contraseña protegiendo el acceso al *keystore* (dejar en blanco si no hay ninguna).
- **KEYMANAGER_PASSWORD**: Contraseña protegiendo el certificado (dejar en blando si no hay ninguna).

Si se inicia el sistema con el argumento "*–https*", este tratará de buscar y utilizar el certificado en base a la información de las variables anteriores.

6.6.2.2.2 Docker

6.6.2.2.2.1 Ejecución

Para ejecutar el módulo a través de su imagen *Docker*, es necesario seguir el manual de instalación (ver *6.6.1.2.2.2-Docker*) y crear y lanzar un nuevo contenedor. Se recomienda:

- Dar un nombre identificativo al contenedor (*--name*).
- Ejecutar el contenedor en modo *detached* (*-d*), tal que funcione en segundo plano.
- Mapear los puertos del contenedor (*-p*) a los puertos del servidor en que se quiera ofrecer la aplicación. El API estará expuesta por defecto en el puerto 8080 del contenedor, siendo el caso de uso más común ofrecerla en el puerto 80 (*HTTP*) o 443 (*HTTPS*) del servidor.

```
docker run --name api -d -p 80:8080 ghcr.io/weso/rdfshape-api:stable
```

Código 9. Ejecución del módulo API a través de Docker (línea de comandos)

Dentro del contenedor, la aplicación de ejecutará en nombre de un usuario no privilegiado "*rdfshape*" para prevenir riesgos de seguridad. En cualquier caso, se recomienda no ejecutar *Docker* como superusuario. Se puede encontrar más información al respecto en los tutoriales de *Docker*: https://docs.docker.com/engine/install/linux-postinstall/#manage-docker-as-a-non-root-user.

6.6.2.2.2.2 Configuración

Cuando se ejecuta el módulo mediante contenedores no es posible pasar argumentos al ejecutable del API como en las ejecuciones manuales. En este caso, se puede configurar el comportamiento del sistema pasándole variables de entorno en el momento de inicialización:

- ***PORT* (opcional)**: Puerto en que el API estará disponible dentro del contenedor. Por defecto es el puerto 8080.
- ***USE_HTTPS* (opcional)**: Asignar cualquier valor no vacío para que el sistema intente servir el API a través de *HTTPS*, siguiendo los procedimientos vistos para la ejecución manual. Dejar en blanco para que el sistema sirva el API mediante *HTTP*. Por defecto está en blanco.
- ***STREAM_TIMEOUT* (opcional)**: Número entero especificando los segundos que el servidor debe esperar a recibir datos de un servidor externo durante una validación de flujos antes de abortar la validación por inactividad. El valor por defecto es 40 segundos.

6.6.2.2.2.3 Detención

Para detener el sistema, será necesario detener el contenedor que lo está ejecutando. El sistema intentará detenerse grácilmente y pasado unos segundos se forzará su cierre si es necesario. Una vez el contenedor está creado y detenido, es posible relanzarlo de forma que mantenga su configuración y registros internos.

```
docker stop api
```

Código 10. Detención del módulo API a través de Docker (línea de comandos)

```
docker start api
```

Código 11. Relanzamiento del módulo API a través de Docker (línea de comandos)

6.6.2.3 *Módulo Cliente*

Desde la perspectiva del usuario de la aplicación web, sólo es necesario acceder a la dirección en la que está desplegado el sistema.

De cara a iniciar el servidor, las siguientes guías ejecución pueden encontrarse también en la wiki del módulo:

- Ejecución manual: https://github.com/weso/rdfshape-client/wiki/Deploying-RDFShape-Client-(React).
- Ejecución con *Docker*: https://github.com/weso/rdfshape-client/wiki/Deploying-RDFShape-Client-(Docker).

6.6.2.3.1 Ejecución manual

Tras seguir los pasos del manual de instalación (ver *6.6.1.3.2.1-Manual*), es posible ejecutar el sistema en un servidor de pruebas a través de *npm*.

```
npm start
```

Código 12. Comando de ejecución del módulo Cliente desde npm

Por otra parte, y para mejorar la seguridad y el rendimiento de la aplicación, se recomienda construir la aplicación web final. Los ficheros generados se depositarán en la carpeta *{raíz_proyecto}/build*, listos para ser servidos como contenidos estáticos mediante un servidor web como *Apache* o *NGINX*.

```
npm run build
```

Código 13. Generación de la aplicación web React desde npm

6.6.2.3.2 Docker

6.6.2.3.2.1 Ejecución

Para ejecutar el módulo a través de su imagen *Docker*, es necesario seguir el manual de instalación (ver *6.6.1.3.2.2-Docker*) y crear y lanzar un nuevo contenedor. De forma análoga al módulo API, se recomienda:

- Dar un nombre identificativo al contenedor (*--name*).
- Ejecutar el contenedor en modo *detached* (*-d*), tal que funcione en segundo plano.
- Mapear los puertos del contenedor (*-p*) a los puertos del servidor en que se quiera ofrecer la aplicación. El cliente web estará expuesto por defecto en el puerto 80 del contenedor, siendo el caso de uso más común servirlo en el puerto 80 (*HTTP*) o 443 (*HTTPS*) del servidor.

```
docker run --name client -d -p 80:80 ghcr.io/weso/rdfshape-client:stable
```

Código 14. Ejecución del módulo Cliente a través de Docker (línea de comandos)

El contenedor viene equipado con un servidor *NGINX*, en el que se depositan la aplicación *React* ya construida y lista para ser servida. La aplicación de ejecutará en nombre del usuario *nginx* para prevenir riesgos de seguridad. Como se ha comentado para el módulo API y de cualquier manera, no es recomendable ejecutar *Docker* como superusuario.

6.6.2.3.2.2 Configuración

Cuando se ejecuta el módulo mediante contenedores se puede configurar el comportamiento del sistema pasándole variables de entorno en el momento de inicialización:

- ***RDFSHAPE_HOST* (opcional)**: Dirección en la que se encuentra desplegado el API de *RDFShape* con el que intentará conectar el cliente. Contiene por defecto la dirección de despliegue del API en los servidores de WESO: https://api.rdfshape.weso.es.
- ***RDFSHAPE_WS_HOST* (opcional)**: Dirección en la que se encuentra desplegado el servidor *WebSockets* con el que intentará contactar el cliente para la validación de flujos. Contiene por defecto la dirección de despliegue del API en los servidores de WESO: *wss://api.rdfshape.weso.es*.

6.6.2.3.2.3 Detención

Similar a como se ha descrito para el módulo API, es necesario detener el contenedor correspondiente para parar el sistema. Una vez el contenedor está creado y detenido, es posible relanzarlo de forma que mantenga su configuración y registros internos.

```
docker stop client
```

Código 15. Detención del módulo Cliente a través de Docker (línea de comandos)

```
docker start client
```

Código 16. Relanzamiento del módulo Cliente a través de Docker (línea de comandos)

6.6.3 Manual de Usuario

6.6.3.1 *Módulo validador (Comet)*

6.6.3.1.1 Introducción

El siguiente manual está enfocado a los usuarios finales de la biblioteca de validación *Comet*, es decir, aquellos con conocimientos de programación que busquen importar el módulo en un proyecto existente e invocar su funcionalidad. Por tanto, y pese a ser un manual de usuario, tiene un fuerte perfil técnico.

Aquí se cubren los aspectos básicos para iniciar una validación con *Comet* y manejar los errores que puedan surgir durante la misma. Nótese que se asume cierto conocimiento previo por parte del lector en el lenguaje de programación *Scala* y el framework de programación funcional *cats-effect*. Finalmente, se puede encontrar una copia actualizada de este manual en el sitio web del sistema: https://ulitol97.github.io/comet/docs/getting_started/example/.

6.6.3.1.2 Primeros pasos

A continuación, y a modo de "*Hola Mundo*" se muestra un ejemplo simplificado de uso de *Comet* para iniciar una validación de un flujo empleando:

- Un ***ListExtractor***: Contiene una lista de *strings* con contenidos RDF a partir de la cual se formará el flujo de datos que alimentará al validador.
- Un ***Validator***: Configurado con un esquema *ShEx* y una *ShapeMap* contra los que validar los elementos del flujo de entrada.
- Un conjunto de bloques de código encargados de evaluar el flujo de datos mediante operaciones de *Fs2*:
    - ***evalMap***: Imprimir cada resultado en pantalla al final del procesamiento del flujo.
    - ***handleErrorWith***: Definir el comportamiento para el manejo de excepciones.

```
import data.DataFormat.*
import exception.stream.validations.*
import stream.extractors.list.ListExtractor
import validation.Validator
import validation.configuration.ValidatorConfiguration

import cats.effect.*
import fs2.*

import scala.concurrent.duration.*

/**
 * Código de demostración:
 * 1. Crear un esquema de validación (ShEx) y un trigger (ShapeMap) de SHaclEX
 * 2. Crear un extractor para alimentar el validador con una lista predefinida de RDF
 * 3. Crear el validador, ofreciéndole:
 *  - Una configuración con el esquema y trigger de validación
 *  - El extractor del que extraer datos
 * 4. Comenzar la validación
 *
 * @note We pretend to have pre-defined functions that generate RDF data, Schemas,
etc.
 *       to hide the complexity of generating these
 */
object Main extends IOApp.Simple {

  override def run(args: List[String]): IO[ExitCode] = {

    for {
      // 1. Crear un esquema y un trigger:
      schema <- makeSchemaShexIO() // Esquema
      trigger = makeTriggerShex() // Trigger

      // 2. Crear un extractor de listas
      listExtractor = ListExtractor(
        items = makeRdfItems(), // Lista de datos RDF
        format = TURTLE
      )
```

Código 17. Ejemplo de uso de Comet (I)

```
    // 3. Crear el validador
    // 3.1 Configuración del validador
    validatorConfiguration = ValidatorConfiguration(schema, trigger)
    // 3.2 Instancia del validador
    validator = Validator(validatorConfiguration, listExtractor)

    // 4. Comenzar el flujo de validaciones
    app <- validator.validate // Iniciar
      .evalMap(IO.println) // Imprimir cada elemento
      .handleErrorWith { error => // Manejar errores ocurridos
        Stream.eval(
          error match {
            case timeoutErr: StreamTimeoutException => IO.println("timeout!")
            case invalidErr: StreamInvalidItemException => IO.println("invalid!")
            case erroredErr: StreamErroredItemException => IO.println("errored!")
            case other => IO.println("other throwable!")
          }
        )
      }
      .onFinalize(IO.println("Main finalizado")) // Acción final, siempre se ejecuta
      .compile.drain

    } yield app
  }
}
```

Código 18. Ejemplo de uso de Comet (II)

6.6.3.1.3 Validadores

Los **Validators** (ver *Scaladoc*) son el corazón de *Comet*. Un objeto validador consume un flujo de elementos RDF y mapea cada elemento para producir un flujo de resultados de validación.

Para crear una instancia de un validador se necesitan:

1. Una **ValidationConfiguration** (ver *Scaladoc*): Contiene el esquema contra el que el validador comprobará los datos RDF que consuma, así como las condiciones de parada del validador.
2. Un **StreamExtractor** (ver *Scaladoc*): Referencia a la instancia de extractor de la cual el validador obtendrá los datos que va a validar (ver *6.6.3.1.4-Extractores*).
3. Un tipo genérico "**A**": Tipo de datos que el extractor del validador espera recibir.

```
class Validator[A](
  configuration: ValidatorConfiguration,
  private val extractor: StreamExtractor[A]
 )
```

Código 19. Estructura de la clase Validator

6.6.3.1.3.1 Configuración del validador

Los validadores de *Comet* necesitan de un objeto de configuración para poder operar. Para construir una configuración se necesitan los siguientes parámetros:

6.6.3.1.3.1.1 Parámetros obligatorios

- **Schema**
  Esquema, instanciado con el API de SHaclEX, contra el que el validador validará los elementos que consuma.
- **Trigger**
  ShapeMap, instanciado con el API de SHaclEX, que usará el esquema para validar los elementos de entrada.

6.6.3.1.3.1.2 Parámetros opcionales

- **HaltOnInvalid**
  Especifica si el validador debe detenerse y lanzar un error cuando un ítem RDF no supere la validación contra el esquema o si tan sólo debe emitir un resultado con el estado *INVALID* y seguir trabajando. El valor por defecto es **false**.
- **HaltOnErrored**
  Especifica si el validador debe detenerse y lanzar un error cuando suceda un error de ejecución procesando un ítem RDF o si tan sólo debe emitir un resultado con el estado *ERRORED* y seguir trabajando. El valor por defecto es **false**.
- **ConcurrentItems**
  Define el máximo número de elementos que el validador puede validar en paralelo. El valor por defecto es **10**.

6.6.3.1.3.2 Resultados de la validación

Cada ítem procesado por un validador de *Comet* es traducido a una instancia de **ValidationResult** (ver *Scaladoc*) emitida en el flujo de salida del validador. Cada resultado contiene:

- **ResultStatus** (ver *Scaladoc*): Valor enumerado que representa si la validación fue exitosa, no exitosa o abortada a causa de un error.
- **result:** El resultado de la validación del ítem RDF, tal y como ha sido generado por SHaclEX. Es un valor opcional que estará vacío si la validación no se pudo completar.

6.6.3.1.4 Extractores

Si bien un validador debidamente configurado sirve para consumir y validar un flujo de datos RDF, es necesario suministrar un extractor que le sirva un flujo de datos de entrada.

Los extractores de *Comet* son bloques de código capaces de generar un flujo de datos RDF que, finalmente, sirve de entrada a un validador. El extractor implementa el algoritmo que consume un grupo de datos como un flujo de forma transparente al validador, al cual no le importa de qué extractor provengan los datos (de listas, de ficheros, de *Kafka*…) mientras sigan la estructura necesaria.

Comet define la clase abstracta **StreamExtractor** (ver *Scaladoc*), de la que derivan el resto de los extractores.

```
abstract class StreamExtractor[A](
   val format: DataFormat,
   val inference: InferenceEngine = defaultInferenceEngine,
   protected val concurrentItems: Int = defaultConcurrentParsing,
   protected val itemTimeout: Option[FiniteDuration] = defaultIncomingItemsTimeout
 )
 (
   implicit private val toRdfElement: A => RDFElement
 )
```

Código 20. Estructura de la clase StreamExtractor

6.6.3.1.4.1 Configuración del extractor

Al margen de su implementación concreta, todos los extractores definen una serie de parámetros de configuración.

6.6.3.1.4.1.1 Parámetros obligatorios

- **DataFormat**
  Formato **DataFormat** (ver *Scaladoc*), de entre los aceptados por el módulo, de los datos RDF procesados por el extractor. El extractor espera que todos los datos vengan en el mismo formato.

6.6.3.1.4.1.2 Parámetros opcionales

- **Inference**
  Indica la inferencia a aplicar sobre los datos RDF entrantes en el momento de interpretarlos. El valor por defecto es **ninguna**.
- **ItemTimeout**

Opcionalmente, la cantidad de tiempo que el extractor debe esperar sin consumir elementos antes de cerrar el flujo de entrada por inactividad. Esto es especialmente útil para ahorrar recursos del sistema. El valor por defecto es **ningún tiempo límite**.

- **ConcurrentItems**
  Define el máximo número de elementos que el extractor puede leer e interpretar en busca de RDF en paralelo. El valor por defecto es **10**.

6.6.3.1.4.2 Tipo genérico

Al crear un extractor es necesario definir (o dejar que el compilador infiera) un tipo genérico "**A**", donde **A** es el tipo de los datos que el extractor espera recibir.

Esta opción permite una mayor flexibilidad y personalización por parte del emisor de los datos. Por ejemplo, en el caso de estar recibiendo datos RDF de un flujo remoto de *Kafka*, los datos pueden llegar de una de las siguientes maneras:

1. Codificados como *strings* con los datos RDF. Es una alternativa sencilla de manejar pero limitada.
2. Codificados como instancias de una clase personalizada que tiene los contenidos RDF, metadatos, etc. Es una alternativa más compleja pero más completa.

Para no limitar a los extractores a procesar flujos de datos de entrada compuestos de *strings*, se permite especificar la siguiente información:

1. El tipo genérico **A**.
2. La función implícita **toRdfElement**, que permite a *Comet* convertir las instancias entrantes de tipo "**A**" a objetos del tipo **RDFElement** (ver *Scaladoc*) manejados por *Comet*.

```
implicit private val toRdfElement: A => RDFElement
```

Código 21. Firma del método "toRdfElement"

6.6.3.1.4.3 Extractores predefinidos

Comet ofrece algunas funcionalidades de serie, pues viene con algunos extractores ya implementados para su uso.

6.6.3.1.4.3.1 List Extractor

Extractor capaz de generar un flujo de entrada a partir de un listado de elementos en memoria.

6.6.3.1.4.3.2 File Extractor

Extractor capaz de generar un flujo de entrada a partir del contenido de un listado de ficheros:

- Se espera que cada fichero contenga un único conjunto de datos RDF.
- Usa el API de manejo de ficheros de Fs2 (está inspirado en https://fs2.io/#/getstarted/example).

Necesita las siguientes configuraciones adicionales respecto al extractor base:

- **Charset**: Juego de caracteres empleado en los ficheros que se van a leer. El valor por defecto es **UTF-8**.

6.6.3.1.4.3.3 Kafka Extractor

Extractor capaz de generar un flujo de entrada a partir de los contenidos de un *stream* de *Apache Kafka*. Al ser más complejo que los anteriores, requiere de un objeto de configuración **KafkaExtractorConfiguration** (ver *Scaladoc*) que especifica como contactar y operar el flujo de *Kafka*.

Necesita las siguientes configuraciones adicionales respecto al extractor base:

6.6.3.1.4.3.3.1 Parámetros obligatorios

- **Topic**
  Nombre del *topic* dentro del flujo de *Kafka* donde se encuentran los ítems que consumir.

6.6.3.1.4.3.3.2 Parámetros opcionales

Hay una gran cantidad de configuraciones adicionales. Las principales son:

- **Server**
  Servidor de origen del flujo de *Kafka*. El valor por defecto es **localhost**.
- **Port**
  Puerto de origen del flujo de *Kafka* dentro del servidor. El valor por defecto es **9092**.
- **GroupId**
  Identificador bajo el que opera el extractor como consumidor de *Kafka*. El valor por defecto es autogenerado tal que **comet:{versión_app}**

Con la finalidad de mejorar y ampliar las funcionalidades del sistema, es posible extender la clase base de los extractores para crear otros extractores personalizados. Se puede obtener más información al respecto en el manual del programador (ver *6.6.4.1-Módulo validador (Comet)*).

6.6.3.1.5 Manejo de errores y excepciones

Comet implementa una serie de clases de excepción personalizadas para los errores más comunes que pueden ocurrir durante su ejecución.

6.6.3.1.5.1 Errores de configuración

- **IllegalArgumentException:** Lanzada cuando un validador o extractor es configurado con parámetros inválidos.

6.6.3.1.5.2 Errores en la trata de flujos

6.6.3.1.5.2.1 Validación

Ha ocurrido un error en que el resultado de la validación no cumple los requisitos de configuración del validador.

- **StreamInvalidItemException**: Emitida cuando un ítem no supera la validación sin errores y el validador está configurado para detenerse ante ítems inválidos. Contiene el resultado de validación problemático.
- **StreamErroredItemException**: Emitida cuando el procesamiento de un ítem da lugar a un error de ejecución y el validador está configurado para detenerse ante fallos. Contiene el error original como causa.

6.6.3.1.5.2.2 Timing

- **StreamTimeoutException**: Emitida cuando el extractor de datos no pudo consumir ningún elemento nuevo durante un período de tiempo mayor que el permitido.

6.6.3.1.5.3 Otros errores

Otros errores que no han sido considerados en este manual ni en las excepciones creadas para *Comet* pueden ocurrir. Estos errores se pueden manejar de manera similar al resto a través del API de Fs2.

6.6.3.2 *Módulo API*

6.6.3.2.1 Introducción

El módulo API está conformado por el *backend* de la aplicación web *RDFShape*. Una vez desplegado, el módulo ofrece un API REST para validar, transformar y realizar consultas sobre datos RDF apoyándose en otras tecnologías semánticas. Desde la perspectiva de los usuarios finales, tan sólo es necesario contar con un software que haga de cliente *HTTP* y/o *WebSockets* (como *curl* o *Postman*) para invocar operaciones en el servidor y recibir sus resultados.

Este manual de usuario se centra en el consumo de las operaciones del API desarrolladas como parte del proyecto para la validación de flujos. Para obtener más información sobre el uso del API REST y las herramientas que pone a disposición de los clientes, dirigirse a la documentación del API en *SwaggerHub*, elaborada como parte del TFM: https://app.swaggerhub.com/apis-docs/weso/RDFShape.

Una copia actualizada de este manual puede encontrarse, junto con información adicional, en el sitio web del sistema: https://www.weso.es/rdfshape-api/docs/api-usage/streaming/. Cabe destacar que tanto la documentación en línea como el presente manual usarán el despliegue del API en los servidores de WESO (https://api.rdfshape.weso.es/api) a modo de ejemplo.

6.6.3.2.2 Validación de flujos

El API de *RDFShape* expone una utilidad para la validación de flujos de datos provenientes de *Apache Kafka* en lugar de datos RDF aislados y estáticos. Esto se consigue empleando la biblioteca de validación *Comet* junto con la biblioteca de validación *SHaclEX* en el servidor.

6.6.3.2.2.1 Modo de empleo

Al estar trabajando con *streams* de datos, la comunicación con el servidor de *RDFShape* no se realiza a través de *HTTP(S)* como para el resto de las utilidades, sino mediante *WebSockets*. El flujo de trabajo para operar con el *endpoint* de validación de flujos es el siguiente:

1. El cliente intenta abrir una conexión *WebSockets* con el servidor.
2. Si el intento de conexión es exitoso, el servidor queda a la espera de solicitudes del cliente.
3. El cliente puede entones enviar un mensaje al servidor en formato JSON (ver *6.6.3.2.2.2.1- Modelo de peticiones*) solicitando al servidor que realice una validación sobre un flujo de datos RDF.
4. El servidor verifica que la petición del cliente contiene el formato y contenidos correctos y comienza la validación. El servidor enviará el resultado de la validación de cada ítem del flujo al cliente a través de mensajes *WebSockets* individuales.

6.6.3.2.2.2 Modelo de datos

6.6.3.2.2.2.1 Modelo de peticiones

Para que el servidor comience a enviar resultados al cliente, es este último quien debe enviar una petición al servidor en primer lugar. Estas peticiones son mensajes *WebSockets* en formato JSON, que incluyen la siguiente información para el servidor:

- ***Validador***: Configuración del validador empleado contra los ítems del flujo.
  - schema: Esquema *Shape Expressions* contra el que se validarán los elementos del flujo de entrada. Sigue el mismo formato que en el resto de las operaciones del API.
  - triggerMode: *ShapeMap* contra el que se validarán los elementos del flujo de entrada. Sigue el mismo formato que en el resto de las operaciones del API.
  - haltOnInvalid (opcional): Si la validación debe detenerse en cuanto un elemento no supere la validación con éxito. Por defecto es **falso**.
  - haltOnErrored (opcional): Si la validación debe detenerse en cuanto el procesamiento y validación de un elemento produzca un error inesperado. Por defecto es **falso**.
- ***Extractor***: Configuración del extractor empleado por *Comet* para obtener el flujo de ítems de entrada.
  - data: Objeto con las propiedades de los datos contenidos en el flujo de entrada.
    - *format*: Formato de los datos RDF entrantes.
    - *Inference (opcional)*: Inferencia a aplicar sobre los datos RDF entrantes. Por defecto es **ninguna**.
- ***Stream***: Configuración sobre el origen y consumo del flujo de *Apache Kafka*.

- server**:** IP/dominio de origen del flujo de datos.
- Port (opcional): Puerto de origen del flujo de datos. Por defecto es el **9092**.
- topic: *Topic* de *Kafka* en que se encuentran los datos dentro del flujo de entrada.
- groupId (opcional): Identificador bajo el que la aplicación consumirá el flujo de entrada. Por defecto será ***rdfshape:{versión_api}***.

```
{
  "configuration": {
    "validator": {
      "haltOnInvalid": false,
      "haltOnErrored": false,
      "schema": {
        "content": "PREFIX ex: <http://example.org/>\nPREFIX xsd:
<http://www.w3.org/2001/XMLSchema#>\n\n# Filters of a valid sensor
reading\nex:ValidReading {\n  ex:readingDatetime     xsd:dateTime  ; # Has a VALID
timestamp\n  ex:readingTemperature xsd:decimal MININCLUSIVE 18 MAXINCLUSIVE 20
+ ; # 1+ readings in range 18-20\n  ex:status [ \"OK\" \"RUNNING\" ] # Status must be
one of \n}",
        "engine": "ShEx",
        "format": "ShExC",
        "source": "byText"
      },
      "triggerMode": {
        "shape-map": {
          "content": "ex:reading@ex:ValidReading",
          "format": "Compact",
          "source": "byText"
        },
        "type": "ShapeMap"
      }
    },
    "extractor": {
      "data": {
        "format": "Turtle",
        "inference": "None"
      },
    },
    "stream": {
      "server": "localhost",
      "port": 9092,
      "topic": "rdf",
      "groupId": "myGroup"
    }
  }
}
```

Código 22. Ejemplo de petición de validación de flujos

6.6.3.2.2.2.2 Modelo de respuestas

Los resultados emitidos desde el servidor hacia los clientes tendrán la siguiente estructura:

- **type**: Metainformación sobre el resultado informando al cliente del tipo de contenido que hay en el interior. Puede tomar los valores:
  - result: el mensaje contiene el resultado de una validación en formato JSON.
  - error: el mensaje contiene información sobre un error ocurrido en la validación en formato JSON.
- **content:**
  - Para los resultados, contendrá la salida de la validación.
  - Para los errores, contendrá la descripción del error.

```
{
  "type": "result",
  "content": {
    "valid": true,
    "status": "valid",
    "message": "Data validation was successful",
    "instant": "2022-05-05T15:00:57.925050695Z",
    "report": {
      "valid": true,
      "type": "Result",
      "message": "Validated",
      "shapeMap": [ ... ],
      "errors": [ ],
      "nodesPrefixMap": { ... },
      "shapesPrefixMap": { ... }
    }
  }
}
```

Código 23. Ejemplo simplificado de respuesta del servidor para un resultado

6.6.3.2.2.3 Cierre de la validación

El servidor dejará de ejecutar la validación de flujos en el momento en que el cliente que la solicitó se desconecte. En caso de que el cliente mantenga la conexión activa, la validación se seguirá ejecutando salvo si:

1. Un ítem del flujo de validación es inválido o provoca un error y el validador está configurado para detenerse en estos casos.

2. El servidor no recibe datos del flujo de entrada durante un período de tiempo, tras el cual el servidor aborta la validación para ahorrar recursos.

En cualquiera de los casos anteriores el servidor enviará dos mensajes *WebSockets* al cliente antes de cerrar la validación:

1. Un mensaje estándar conteniendo texto en formato JSON detallando el error que ha causado el final de la validación. La estructura de este mensaje será la siguiente:
   - **type**: Tomará el valor "*error*".
   - **content:**
     - message: Resumen del error.
     - reason: Causa detallada del error. Sólo estará disponible cuando el error se deba a un ítem RDF inválido, en cuyo caso el resultado de la validación fallida se incluirá aquí.
2. Un mensaje de cierre, con una breve descripción del motivo del cierre y su código de error asociado:
   - **3000**: La petición de validación de cliente no contiene un JSON bien formado.
   - **3001**: La petición de validación del cliente no contiene una estructura de configuración válida.
   - **3002**: Un ítem del flujo de validación es inválido.
   - **3003**: Ha ocurrido un error durante la validación de un ítem del flujo.
   - **3004**: No se han recibido ítems del flujo de entrada durante un tiempo.
   - **3005**: La petición de validación del cliente contiene valores fuera de los rangos aceptables en la configuración.
   - **3006**: La petición de validación del cliente contiene valores inválidos en la configuración.
   - **3007**: No se ha podido conectar con el flujo de datos de origen.
   - **4999**: Ha ocurrido un error desconocido.

```
{
  "type": "error",
  "content": {
    "message": "StreamInvalidItemException - Stream halted because an item was
invalid",
    "reason": {
      "valid": false,
      "type": "Result",
      "message": "Validated with errors",
      "shapeMap": [ ... ],
      "errors": [ ... ],
      "nodesPrefixMap": { ... },
      "shapesPrefixMap": { ... }
    }
  }
}
```


Código 24. Ejemplo de respuesta del servidor antes de cerrar la conexión

6.6.3.3 *Módulo Cliente*

6.6.3.3.1 Introducción

El público objetivo de este manual son aquellos usuarios que van a hacer uso del cliente web de *RDFShape* para consumir el módulo API mediante una interfaz gráfica a través de un navegador web.

Si bien el foco del proyecto son las validaciones de flujos, se cubrirá el aspecto, modo de empleo y posibles interacciones y notificaciones emitidas por la aplicación al completo para tener todo el contexto de uso. Así mismo, se asume cierto grado de conocimiento por parte del lector en las tecnologías semánticas sobre las que opera el sistema. Finalmente, el manual usará el despliegue del cliente en los servidores de WESO (https://rdfshape.weso.es/) a modo de ejemplo.

6.6.3.3.2 Uso básico de la aplicación web

6.6.3.3.2.1 Pantalla de inicio

La primera pantalla que aparece cuando se accede a la aplicación contiene una serie de textos explicativos sobre el propósito y utilidad del sitio web. Se dan explicaciones sobre las principales herramientas disponibles junto con enlaces de acceso a las mismas.

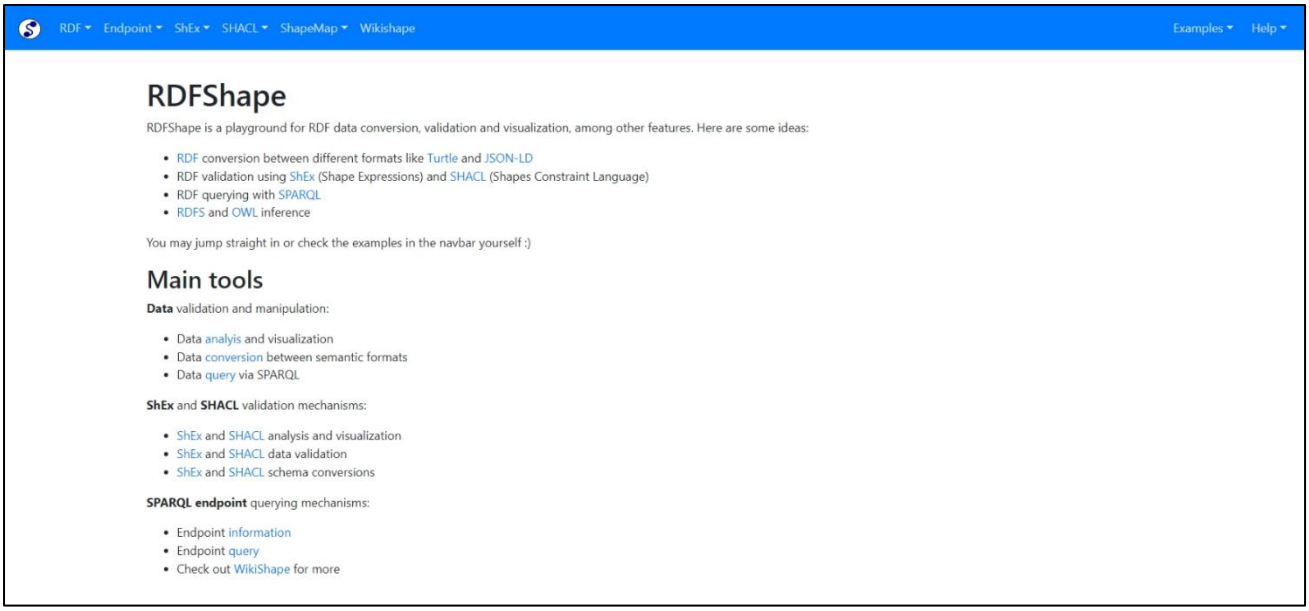


Ilustración 102. Cliente web - Pantalla de inicio

6.6.3.3.2.2 Barra de navegación

En la parte superior de la pantalla siempre estará presente una barra de navegación, para cualquier página a la que se navegue. Esta barra de navegación contiene, en su parte izquierda y de forma categorizada, enlaces de acceso a cada una de las herramientas puestas a disposición de los usuarios, en función de si su foco principal es operar con datos RDF, esquemas *ShEx*/*SHACL*, etc. Por otra parte y a mano derecha, aparecen otros enlaces secundarios con ejemplos de uso de varias herramientas del sistema así como enlaces de ayuda y documentación adicional.

Si es la primera vez que emplea el sistema, es recomendable comenzar por acceder a la ayuda y ejemplos.

Ilustración 103. Cliente web - Barra de navegación

6.6.3.3.2.3 Desplazamiento entre páginas/herramientas

El cliente de *RDFShape* ofrece un compendio de herramientas independientes entre sí. Para desplazarse entre ellas, puede recurrir a los enlaces de la barra de navegación en todo momento.

Para ayudar a moverse entre páginas, el sistema almacena automáticamente la información con la que estaba operando (datos RDF, esquemas, consultas SPARQL, etc.) e intenta cargar dicha

información en la nueva página solicitada para evitar que tenga que introducir los datos de nuevo o copiarlos/pegarlos manualmente.

6.6.3.3.3 Uso de las herramientas de la aplicación

Cada herramienta de entre las disponibles en el portal web consta de una funcionalidad única, por ejemplo: no tienen nada en común la herramienta de análisis de esquemas *Shape Expressions* y la herramienta de consultas SPARQL sobre datos RDF. Sin embargo, todas se han construido sobre unas guías de estilo y componentes comunes y reutilizables, lo que hace que, desde la perspectiva de la interfaz de usuario, todas se utilicen de manera similar.

6.6.3.3.3.1 Estructura de las herramientas

La pantalla de cada herramienta consta de:

- La barra de navegación
  Localizada en la parte superior.
- El título de la herramienta, acompañado del componente de ayuda ("**?**")
  Es posible interactuar con el componente de ayuda para obtener información adicional de uso y posibilidades de la herramienta.
- El cuerpo de la herramienta
  Una estructura en dos columnas que alojan, respectivamente:
  - Los **formularios de entrada** de datos (izquierda).
    - La entrada de datos puede ser de varios tipos: RDF, esquemas, consultas SPARQL, *ShapeMaps*, UML, etc. Se ofrecerán ayudas de sintaxis y autocompletado donde estén disponibles.
    - Para todos los casos se ofrecerá, mediante una interfaz de pestañas, la posibilidad de insertar los datos manualmente, subiendo un fichero o referenciado a una URL que los contenga en texto plano.
    - Bajo el formulario de entrada se encontrará siempre un **botón de acción** para invocar la herramienta en el servidor con los datos introducidos. Dicho botón se desactivará cuando el sistema se encuentre enviando/esperando/recibiendo información del servidor sobre una invocación previa.
  - Los **componentes de salida** de datos o **resultados** (derecha).
    - Un componente principal de resultados. Varía entre herramientas, aunque generalmente consisten en un listado de resultados (consultas, validaciones, etc.) o una salida en formato texto (conversiones, etc.).
    - Un desplegable con la respuesta completa recibida del servidor (para usuarios avanzados).
    - Un botón para generar un *permalink* a la operación ya exitosa.

La siguiente captura de pantalla muestra, utilizando la herramienta de consultas SPARQL, las distintas secciones en que se divide el interfaz de las herramientas web:

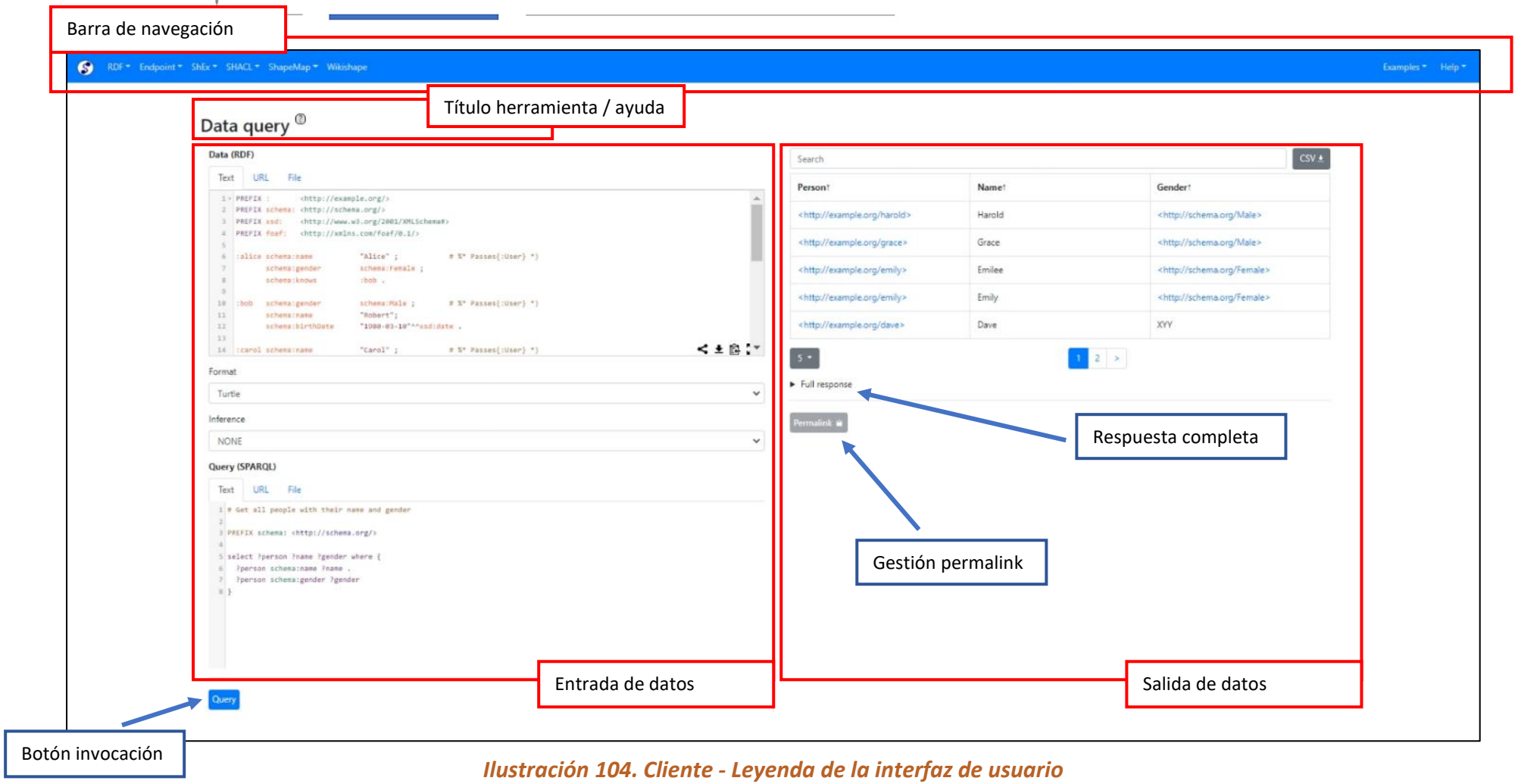


Ilustración 104. Cliente - Leyenda de la interfaz de usuario

6.6.3.3.3.2 Gestión de permalinks

El cliente tiene la capacidad de generar y consumir *permalinks*, es decir, enlaces acortados persistentes, que redirigen a operaciones realizadas previamente.

6.6.3.3.3.2.1 Generación de enlaces

Para generar un *permalink* a una operación deben darse las siguientes precondiciones:

- La operación ha podido completarse sin errores graves, de forma que se ha obtenido una respuesta con resultados desde el servidor. Si no se cumple está condición, el botón de generación de enlaces no se mostrará.
- Los datos de entrada de la operación:
    - No han sido suministrados mediante la subida de ficheros.
    - La longitud del texto (introducido manualmente o por URL de referencia) suministrado no excede cierto límite.

  Si no se cumple una de las dos condiciones previas, el botón de generación de enlaces se mostrará junto con los resultados, pero estará desactivado y mostrará un *tooltip* advirtiendo del problema.

Si se cumplen los supuestos anteriores, el sistema permitirá interactuar con el botón de generación, cuyo comportamiento será el siguiente:

1. El sistema desactivará el botón una vez el usuario ha interactuado con él.
2. Se mostrará un componente dinámico *spinner* mientras el cliente espera la respuesta del servidor con el enlace.
3. Una vez recibido el enlace, el sistema:
    a. Mostrará una notificación *toast* informando al usuario de que el enlace se ha copiado a su portapapeles.
    b. Reactivará el botón de generación. Las subsecuentes interacciones con el botón copiarán el enlace ya generado al portapapeles de nuevo sin invocar al servidor.

Ilustración 107. Cliente - Botón permalink

Ilustración 106. Cliente - Botón permalink (procesando)

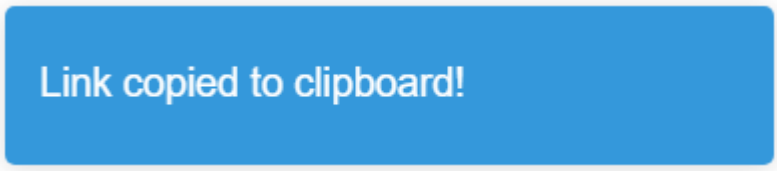


Ilustración 105. Cliente - Notificación permalink adquirido

6.6.3.3.3.2.2 Consumo de enlaces

Cuando se posee un *permalink*, sólo es necesario entrar a él desde un navegador web para acceder a su operación asociada. La aplicación web se comportará de la siguiente manera:

1. Mostrará una pantalla de carga mientras consulta con el servidor la dirección objetivo del *permalink*.
2. Redirigirá al usuario automáticamente a la página de destino del enlace, rellenado automáticamente los campos de la operación e invocándola.

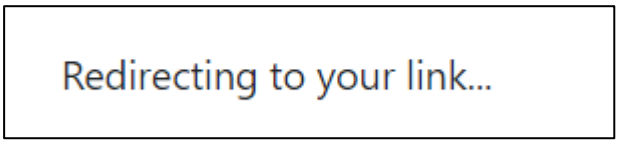


Ilustración 108. Cliente - Mensaje de carga de permalinks

En caso de errores (servidor no disponible, enlace inexistente, etc.), el sistema mostrará un mensaje de error explicativo junto con la opción de regresar al inicio.

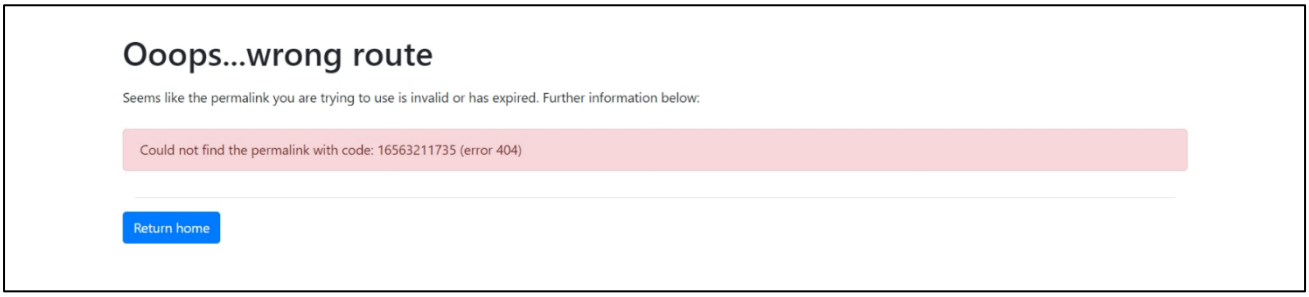


Ilustración 109. Cliente - Pantalla de error de permalinks

6.6.3.3.4 Herramienta de validación de datos

La herramienta de validación de datos mediante *Shape Expressions* es la más relevante de las ofertadas por los módulos API/Cliente: ofrece la posibilidad de validar datos RDF tanto "estáticos" como provenientes de un *stream* de *Apache Kafka* contra un esquema *ShEx*.

6.6.3.3.4.1 Estructura de la herramienta

Grosso modo, la utilidad de validación sigue el mismo patrón de interfaz que el resto de las herramientas (ver *6.6.3.3.3.1-Estructura de las herramientas*).

6.6.3.3.4.1.1 Entrada de datos

El formulario de entrada de datos contiene los siguientes elementos:

1. **Entrada de datos RDF**: Permite introducir, a partir de varios métodos de entrada, los datos RDF que se van a validar. Para cualquier entrada es necesario especificar el formato RDF de los datos y la inferencia a aplicar sobre estos:

- Texto: Los datos se introducen manualmente en una caja de texto del navegador. La biblioteca YATE ofrece sintaxis coloreada y autocompletado de prefijos.
- URL: El usuario introduce una dirección URL en la que residen los datos.
- Fichero: El usuario sube un archivo con los datos.
- *Stream*: En esta opción se especifica el origen de los datos y no los datos en sí. A diferencia del resto de opciones, introducir los datos de esta manera iniciará una validación de flujos. Para esto es necesario definir el servidor, puerto y *topic* del que se emite el flujo de *Kafka*, así como si la validación debe detenerse al encontrarse un ítem inválido o que cause un error.

2. **Entrada de esquema *ShEx***: Permite introducir, a partir de los métodos de entrada de texto, URL y fichero, el esquema *ShEx* contra el que validar los datos. Para la alternativa de texto, la biblioteca YASHE ofrece sintaxis coloreada y autocompletado de prefijos.
3. **Entrada de *ShapeMap***: Permite introducir, a partir de los métodos de entrada de texto, URL y fichero, el *ShapeMap* con el que validar los datos.

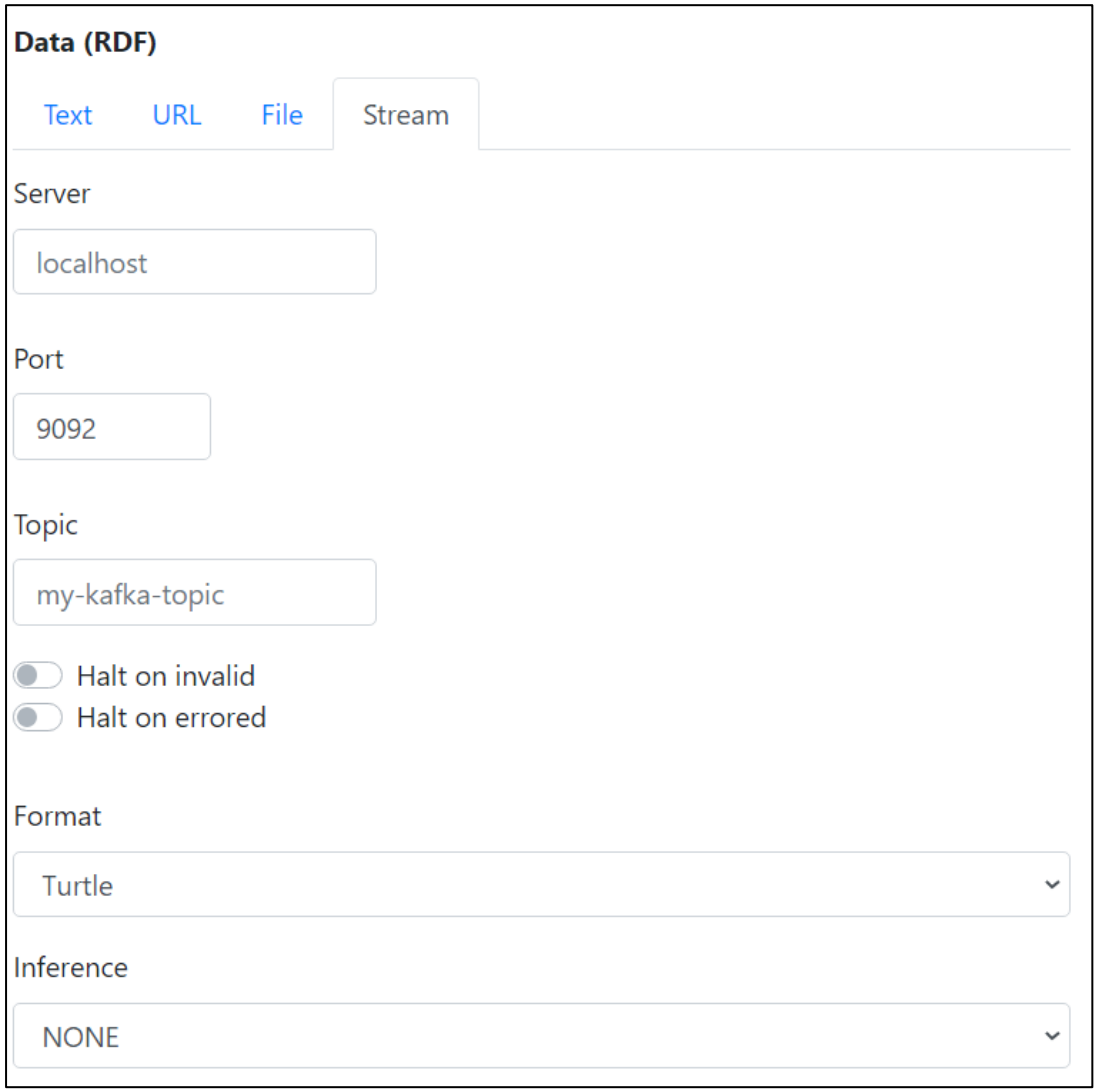


Ilustración 110. Cliente - Formulario de entrada de datos mediante stream

6.6.3.3.4.1.2 Salida de datos

La zona de salida de datos cuenta con los siguientes elementos:

1. **Zona de alertas**
   Espacio situado sobre la tabla de resultados donde el sistema muestra su estado a través de mensajes breves siguiendo el siguiente código de colores:
   - Rojo: la validación no se ha podido realizar a causa de un error en las entradas o en el servidor.
   - Amarillo: la validación se ha completado aunque no todos los elementos la han superado con éxito.
   - Verde: la validación se ha completado con éxito para todos los elementos.

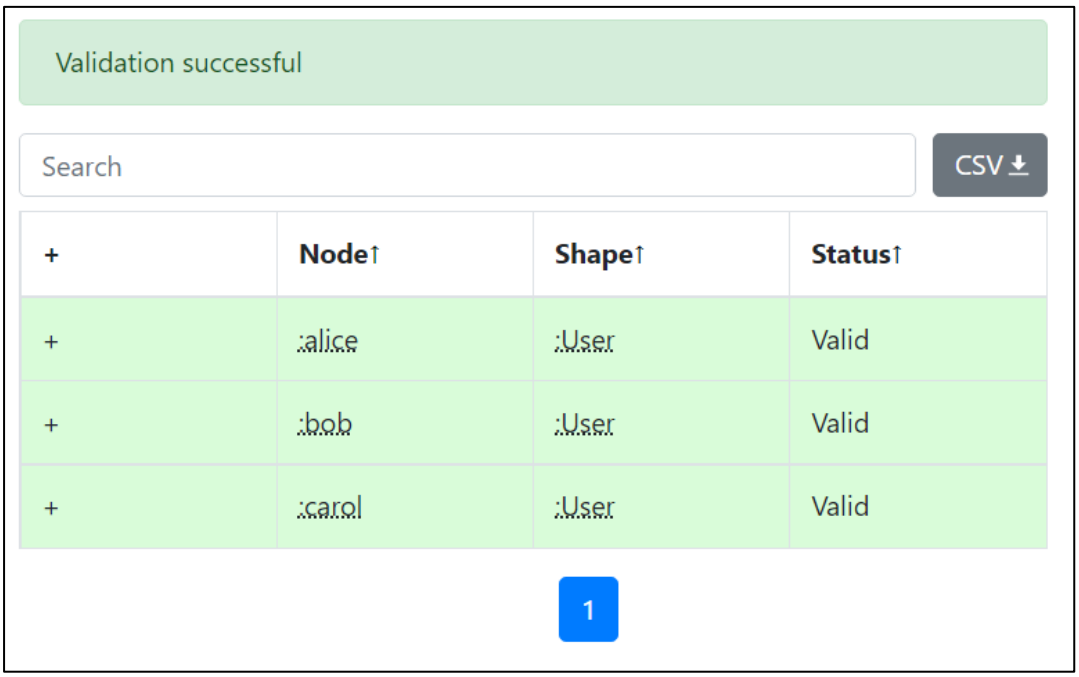


Ilustración 111. Cliente - Resultados de una validación exitosa

2. **Tabla de resultados**
   El componente principal de la herramienta. Está constituido por una tabla donde cada fila representa el resultado de validar un ítem de los datos RDF contra una *shape* del esquema de validación.
   La tabla cuenta con la siguiente información, es posible reordenar los elementos de la tabla en función a uno de estos campos pulsando en el encabezado de su columna asociada:
   - Nodo (columna "Node"): Nodo de los datos RDF validado.
   - Shape (columna "shape"): *Shape*, de las descritas en el esquema, contra la que se ha validado el nodo.
   - Estado (columna "Status"): Estado final de la validación, puede ser válido o inválido.
   - Resultado detallado (columna "+" o "-"): Informe completo de validación con las causas por las que el nodo es válido o no. Se puede desplegar para cada resultado interactuando con la primera columna de la tabla.

Así mismo, los resultados siguen el siguiente código de colores:

- Verde: color de fondo empleado en los resultados válidos.
- Rojo: color de fondo empleado en los resultados inválidos.

Finalmente, la tabla de resultados cuenta con las siguientes utilidades:

- Paginación de resultados: Cuando los resultados exceden cierta cantidad, se dividirán en páginas y se mostrará un componente para desplazarse entre estas.
- Búsqueda de resultados: Sobre la tabla de resultados hay un cuadro de búsqueda. Introducir un término de búsqueda mostrará sólo los resultados que lo incluyan dentro de uno de sus campos.
- Exportación de resultados: Junto al cuadro de búsqueda se muestra un botón para la descarga de los resultados en formato CSV. El botón estará siempre disponible e interactuar con él abrirá el cuadro de diálogo por defecto del navegador para la descarga/apertura del archivo.

3. **Desplegable (respuesta del servidor)**.
4. **Botón de *permalink***.

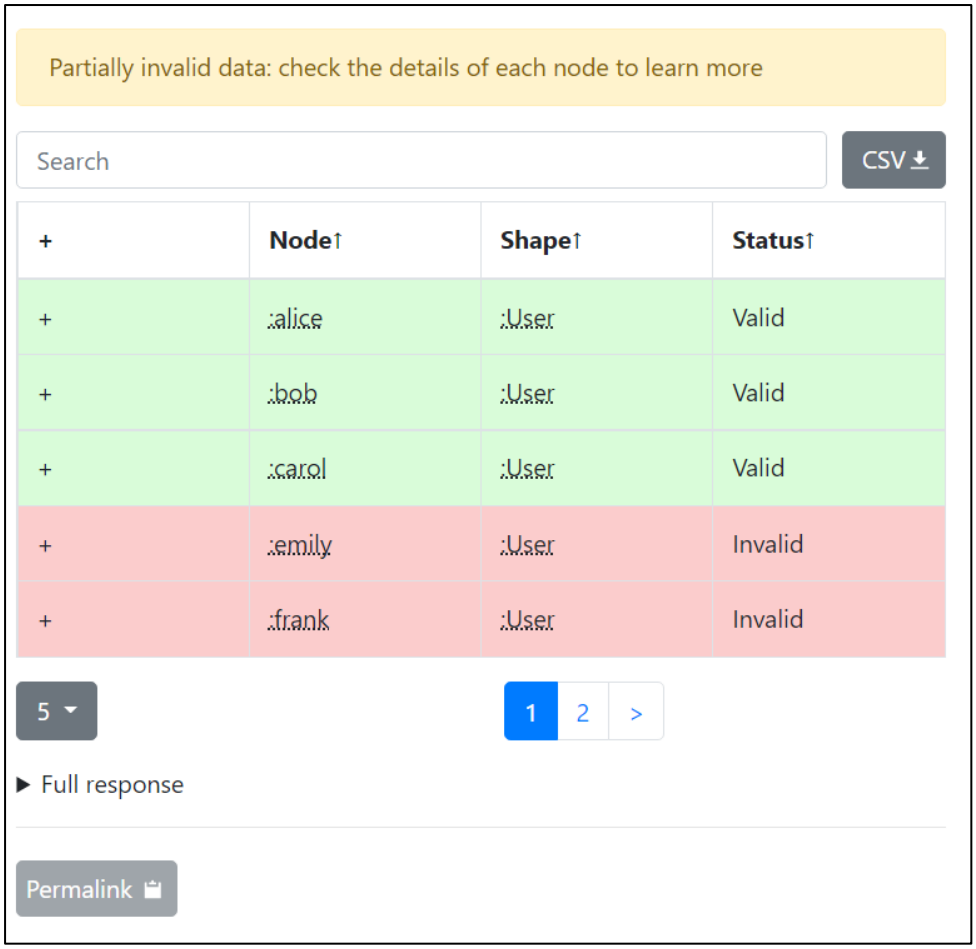


Ilustración 112. Cliente - Tabla de resultados y salidas de una validación

6.6.3.3.4.2 Validación de flujos

Cuando se introducen los datos RDF de entrada como provenientes de un flujo, la herramienta de validación activa una validación de flujos, cuyo comportamiento es muy similar al de una validación estándar aunque con posibilidades adicionales.

6.6.3.3.4.2.1 Entrada de datos

La entrada de datos en el sistema es similar a la de una validación al uso.

6.6.3.3.4.2.2 Salida de datos

La zona de salida de datos sufre las siguientes modificaciones respecto a una validación normal:

1. **Zona de alertas**
   El propósito de la zona de alertas sigue siendo manifestar el estado del sistema. Sin embargo, y puesto que en una validación de flujos el estado final del sistema es indeterminado, las alertas pasan a mostrar el estado de la conexión con el servidor que realiza las validaciones.
   Se emplea el siguiente código de colores:

- Rojo:
    - La validación no se ha podido realizar a causa de un error en las entradas o en el servidor.
    - La validación se ha interrumpido a causa de un error emitido desde el servidor.
- Azul: el cliente está estableciendo contacto con el servidor y a la espera de resultados.
- Verde: la validación está en curso y ya se han recibido resultados.

Así mismo, las alertas van acompañadas de un elemento dinámico tipo *spinner* para reflejar los casos en que el cliente está estableciendo contacto o conversando activamente con el servidor.

Ilustración 113. Cliente - Alerta de inicio de validación de flujos

2. Tabla de resultados

- La tabla incluye una nueva columna "*Date*", que refleja el instante en que se realizó la validación. Los resultados están ordenados de manera predeterminada en orden de fecha descendente (más recientes primero).
- Los contenidos de la tabla se actualizan en tiempo real con la llegada de nuevos resultados.
- La aparición de un error en el servidor aborta la validación pero los resultados emitidos hasta el momento se mantienen en la interfaz.
- Junto al cuadro de búsqueda aparecen dos opciones adicionales:
    - Pausar/Continuar: El botón de pausa permite detener una validación en curso. Una vez accionado, el sistema:
        - Cambiará el botón de pausa por el de continuar.
        - Dejará de recibir nuevos resultados hasta que se decida continuar la validación en curso. Si se decide continuar la validación previa, el sistema intentará procesar todos los ítems que hayan sido emitidos mientras estaba en estado de pausa.
        - Permitirá invocar una nueva validación de flujos con nuevas entradas.
    - Limpiar resultados: Permite descartar todos los resultados presentes en la tabla en el momento en que se activa.

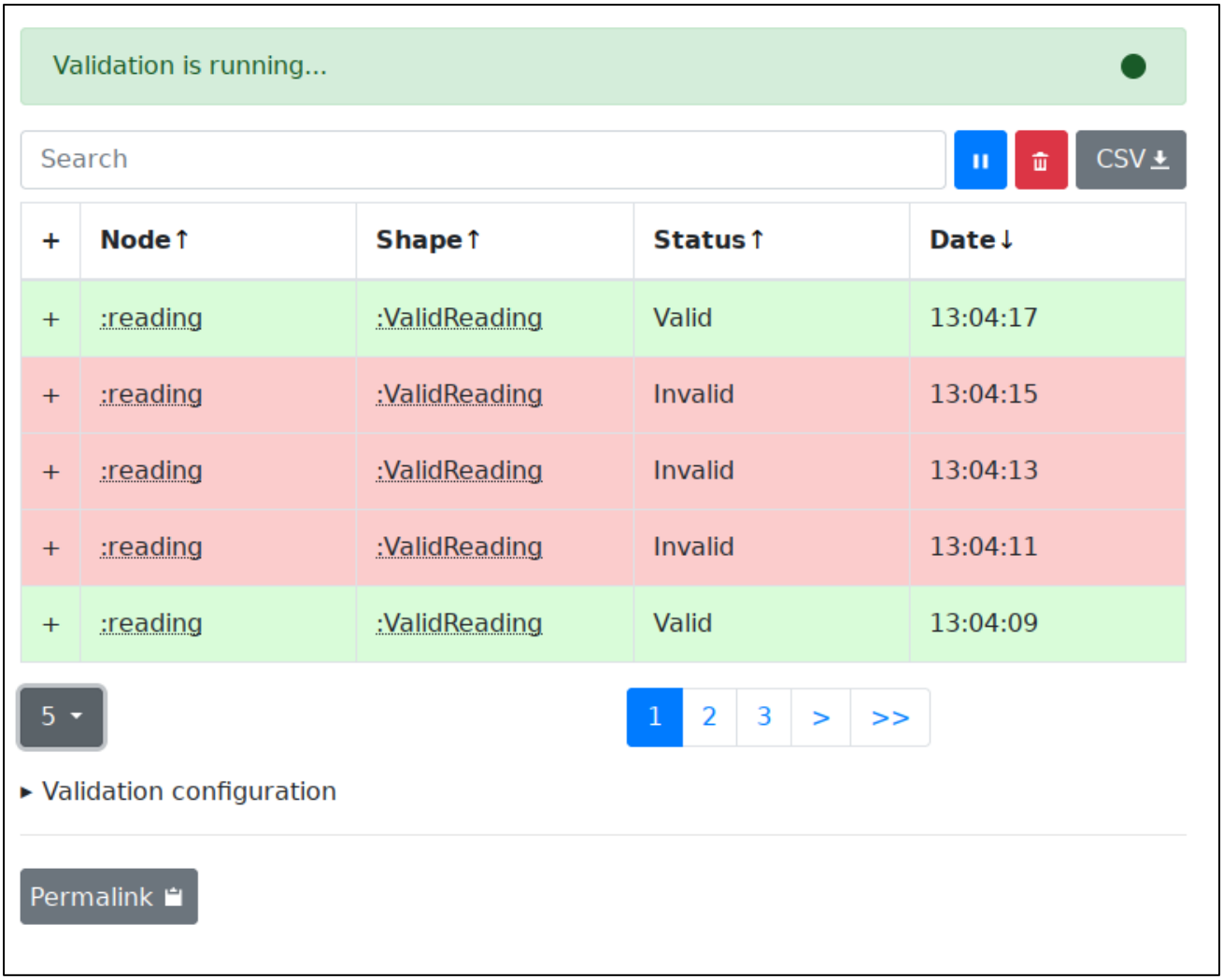


Ilustración 114. Cliente - Resultados y opciones de una validación de flujos en curso

6.6.4 Manual del Programador

6.6.4.1 *Módulo validador (Comet)*

6.6.4.1.1 Introducción

El objetivo de este manual es concretar los aspectos técnicos de la biblioteca de validación de flujos *Comet* de cara a su comprensión y futura modificación o ampliación por parte de otros programadores. Dado que *Comet* es un sistema que ya es de código abierto y pensado para ser extendido, los programadores interesados pueden encontrar información de utilidad del proyecto en:

- Código fuente: https://github.com/ulitol97/comet.
- Sitio web del módulo: https://ulitol97.github.io/comet/.
- Scaladoc del módulo: https://ulitol97.github.io/comet/scaladoc/org/ragna/comet.

El manual hará hincapié en aspectos relevantes o que no se puedan deducir de la lectura de los recursos anteriores.

6.6.4.1.2 Configuración del proyecto

La biblioteca ha sido construida desde su concepción como un proyecto *SBT* en que se distinguen tres módulos:

- root: Módulo raíz de proyecto, con las configuraciones básicas del mismo que el resto de los módulos pueden heredar y sobrescribir, incluyendo:
  - **packagingSettings**: al publicar el proyecto no incluir los ficheros de configuración de *logback* para evitar que interactúen con las configuraciones de *logs* del proyecto que implemente la biblioteca.
  - **scaladocSettings**: opciones lógicas y estéticas para la generación del Scaladoc y su documentación web asociada.
- api: Código con la lógica de la biblioteca expuesta a terceros para su uso. Es el módulo más importante para aquellos interesados en cambiar la funcionalidad del sistema e incluye:
  - **crossScalaVersions**: versiones de *Scala* para las que se quiere compilar y publicar el sistema.
- docs: Módulo destinado a contener la documentación en formato *Markdown* de la que posteriormente saldrá la documentación web. Destacan:
  - **mdocSettings**: opciones para la generación y publicación de documentación con *mdoc*.

Se recomienda inspeccionar estas configuraciones en el propio fichero *build.sbt* y sus comentarios adjuntos.

6.6.4.1.3 Sitio web

El sitio web del proyecto consiste en una página web en *React* generada automáticamente por *Docusaurus*. Es posible automatizar el despliegue de la página web en *GitHub Pages* a través del *plugin* de *mdoc*/*Docusaurus* para SBT.

6.6.4.1.3.1 Edición de la web

Los contenidos de la página web residen en:

- La carpeta ***website***, en la raíz del proyecto, con los archivos de configuración de *Docusaurus* y las páginas y componentes creados en *React*.
- La carpeta ***docs***, en la raíz del proyecto, con los archivos en formato *Markdown* que componen la sección de documentación del sitio web.

6.6.4.1.3.2 Directrices para crear contenido

6.6.4.1.3.2.1 Páginas web

Para crear páginas nuevas, es necesario crear un nuevo fichero *JavaScript* o *TypeScript* con los contenidos de la página dentro de la carpeta ***website/src/pages*** para que *Docusaurus* la reconozca y le asigne una nueva ruta en función del nombre del archivo y su ubicación dentro del árbol de directorios.

6.6.4.1.3.2.2 Documentación web

Para ampliar la documentación en línea, es necesario crear nuevos documentos en formato *Markdown* dentro de la carpeta ***/docs***. Las páginas nuevas serán procesadas por *mdoc* y *Docusaurus*, que les asignará una URL la próxima vez que se genere el sitio web mediante la consola de *SBT* a través de *docs/docusaurusCreateSite*.

6.6.4.1.3.2.3 Scaladoc

Con el fin de publicar el *Scaladoc* como una subsección dentro del sitio web, se configuró el *Scaladoc* para publicarse dentro de la carpeta ***website/static/scaladoc***. Hecho esto, el *Scaladoc* puede navegarse con normalidad accediendo a la ruta ***/scaladoc*** del sitio web.

6.6.4.1.3.3 Problemas en la creación del sitio web

Para poder utilizar *mdoc* junto con *Docusaurus*, se siguió la guía en el siguiente enlace: https://scalameta.org/mdoc/docs/docusaurus.html. Además, fue necesario hacer los siguientes cambios:

- Editar el archivo *package.json* para incluir el script *publish-gh-pages* y poder publicar en *GitHub Pages* desde *GitHub Actions*.
- Modificar el fichero de configuración de *Docusaurus* (ver en línea) con la ubicación de los documentos *Markdown* de los que generar la documentación.

- Modificar el fichero de configuración de la navegación o *sidebar* (ver en línea) para personalizar el índice de la documentación.

6.6.4.1.4 Ampliación del proyecto

6.6.4.1.4.1 Definir nuevos extractores

La funcionalidad de *Comet* está construida para ser extendida con nuevos extractores de flujos de datos. Con ese fin, *Comet* define la clase **StreamExtractor** de la que heredan todos los extractores.

Para implementar un nuevo extractor capaz de alimentar un validador es necesario sobrescribir, como mínimo, todos los miembros no implementados de la clase abstracta que es **StreamExtractor**. Estos son:

1. source: Constante de tipo *string* que identifica el tipo de extractor.
2. inputStream: El flujo de datos de entrada que entra a este extractor, en términos del API de Fs2.

Finalmente, se recomienda visitar y tomar ejemplo de los extractores ya implementados a la hora de crear uno propio.

6.6.4.1.4.1.1 Asertos y restricciones

La clase **StreamExtractor** implementa un método **checkConfiguration** que comprueba que los valores suministrados para el nivel de paralelismo y el tiempo máximo de llegada de nuevos ítems son válidos. Este método se ejecuta en el momento de instanciar un nuevo extractor, impidiendo la creación de un nuevo objeto mal configurado.

Es posible y recomendado extender y/o sobrescribir este método en los extractores que hereden de **StreamExtractor** para adaptarlo a las necesidades y restricciones de su caso de uso particular.

6.6.4.2 *Módulo API*

6.6.4.2.1 Introducción

Este manual pretende guiar y ayudar al programador a entender cómo se ha ampliado y refactorizado el *backend* del sistema *RDFShape* ya existente dentro de WESO. Otras cuestiones relacionadas con la implementación base del API en http4s se dejarán aparte salvo que sea necesario mencionarlas explícitamente.

RDFShape es de código abierto y los programadores interesados pueden encontrar información del proyecto en:

- Código fuente: https://github.com/weso/rdfshape-api.
- Sitio web del módulo: https://www.weso.es/rdfshape-api/.
- Documentación del API: https://app.swaggerhub.com/apis-docs/weso/RDFShape/.

6.6.4.2.2 Validación de flujos

La validación de flujos se ha implementado a través de *WebSockets* para asegurar un canal de comunicación en tiempo real ente el servidor y sus clientes. Para habilitar el uso de *WebSockets* en *http4s* es necesario indicarlo en el momento de construcción del servidor (ver **Server.scala**).

```
val baseServer = BlazeServerBuilder[IO]
  .bindHttp(port, ip)
  .withIdleTimeout(idleTimeout.minutes)
  .withResponseHeaderTimeout(requestTimeout.minute)
  /* WebSocketBuilder was deprecated and must be obtained here, from the
   * server builder */
  .withHttpWebSocketApp((wsBuilder: WebSocketBuilder[IO]) =>
    createApp(client, wsBuilder)
  )
```

Código 25. Habilitando el uso de WebSockets en https4

Hecho esto, ya es posible declarar una ruta del API con el método GET como punto de entrada para una conexión *WebSockets*. Para el sistema, esta ruta es */schema/validate/stream*. *Http4s* trabaja con los mensajes de *WebSockets* como si fueran flujos de datos *Fs2*, por lo que es necesario definir de antemano dos flujos de datos y qué se va a hacer con ellos:

- El flujo de mensajes entrantes de los clientes.
- El flujo de mensajes de salida hacia los clientes.

En este caso, el cliente es quien inicia la validación con un mensaje con la información necesaria, por lo que los mensajes de los clientes de ponen en una cola que luego es consumida por el sistema para generar el flujo de mensajes de salida con las validaciones (ver **SchemaService.scala**).

```
Queue
        .unbounded[IO, Option[WebSocketFrame]]
        .flatMap { queue =>
          /* Stream of messages from the server to the client */
          val toClient: Stream[IO, WebSocketFrame] = mkValidationStream(queue)

          /* Pipe for processing the stream of messages from the client to the
           * server:
           * - Put all incoming client messages into the queue so that the
           * server processes them */
          val fromClient: Pipe[IO, WebSocketFrame, Unit] =
            _.enqueueNoneTerminated(queue)

          webSocketsBuilder.build(toClient, fromClient)
        }
    }
```

Código 26. Cada mensaje WS del cliente se encola para ser procesado en un nuevo flujo de validaciones saliente

Toda la lógica para convertir cada mensaje entrante del cliente en un flujo de validaciones de la biblioteca *Comet* se encuentra en la función **mkValidationStream** (ver StreamValidation.scala). Dada la cantidad y complejidad de las transformaciones sufridas por el flujo, se recomienda visitar el código de la aplicación y los comentarios de lo acompañan. En cualquier caso, la función sigue los siguientes pasos:

0. Tomar el flujo de entrada de peticiones del cliente de la cola en que se encolaron.

```
/* 0. Get input data stream.
  * Poll the queue to see if new messages arrived from the client */
     .fromQueueNoneTerminated(queue)
```

Código 27. Validación de flujos en el servidor - Paso 0

1. Intentar interpretar el contenido del mensaje del cliente como un documento JSON válido.

```
/* 1. WebSocketFrame => JSON.
  * Mapping: attempt to parse JSON out of the client message */
     .map(_.asJson)
```

Código 28. Validación de flujos en el servidor - Paso 1

2. Decodificar el JSON recibido a un objeto del dominio con la configuración de la validación de flujos ya validada.

```
/* 2. JSON => SchemaValidateStreamInput.
      * Mapping: attempt to get the validation settings from the client's JSON
      * message.
      * If decoding errors arise: throw */
     .evalMap(_.as[SchemaValidateStreamInput] match {
       case Left(err)    => IO.raiseError(err)
       case Right(value) => IO.pure(value)
     })
```

Código 29. Validación de flujos en el servidor - Paso 2

3. Crear, a partir de la configuración, un flujo de validaciones invocando al validador de *Comet*.

```
/* 3. Stream[IO, SchemaValidateStreamInput] =>
      * Stream[IO,ValidationResult].
      * Pipe: create a comet validation stream from the parsed configuration.
      * If the validation begins, a stream of results will be returned */
     .through(toValidationStream)
```

Código 30. Validación de flujos en el servidor - Paso 3

4. Transformar cada uno de los resultados emitidos en un mensaje *WebSockets* que enviar de vuelta al cliente.

```
/* 4. ValidationResult => WebSocketFrame.
      * Mapping: convert each validation result to a WebSocketFrame object
      * containing the results in its text */
     .map(_.toWebSocketFrame)
```

Código 31. Validación de flujos en el servidor - Paso 4

5. Manejar cualquier error en el proceso enviando al cliente un mensaje con el error y otro de cierre de conexión.

```
/* 5. Error processing.
      * Catch all known exceptions and generate two frames:
      * 1. A text frame with the detailed error cause.
      * 2. A closing frame with code and reason dependant on the error. */
     .handleErrorWith(err => {
      Stream.evalSeq(IO {
        logger.debug(s"Interrupted streaming validation due to err: $err")
        List(mkErrorFrame(err), mkClosingFrame(err))
      })
```

Código 32. Validación de flujos en el servidor - Paso 5

6.6.4.2.3 Documentación del API

Para la documentación del API se ha hecho uso de la biblioteca *Rho*, cuyo código fuente se encuentra en https://github.com/http4s/rho. La documentación del API consta de dos pasos principales:

1. Redefinir cada una de las rutas del API siguiendo el DSL de *Rho* para definir, desde el código, la descripción de la ruta, sus parámetros de entrada, etc.

```
"Validates RDF data against a given schema" **
    POST / `verb` / "validate" ^ jsonOf[IO, SchemaValidateInput] |>> {
      body: SchemaValidateInput =>
        SchemaValidate
          .schemaValidate(body.data, body.schema, body.triggerMode)
          .flatMap(result => Ok(result.asJson))
          .handleErrorWith(err => InternalServerError(err.getMessage))
    }
```

Código 33. Definición de la ruta de validación de datos RDF con el DSL de Rho

2. Ayudar al sistema a crear las definiciones que no haya podido deducir o identificar a partir del código, así como sobrescribir aquellas que se quiera personalizar. Dentro del API, el paquete ***swagger*** contiene el conjunto de definiciones personalizadas para la documentación del API.

```
val urlModel: Set[Model] = Set(
    ModelImpl(
      id = classOf[URL].getSimpleName,
      `type` = "string".some,
      name = "URL".some,
      description =
        "Representation of a valid URL (including protocol, host, port... as needed)".some,
      isSimple = true,
      example = wikidataUrl.some,
      externalDocs = ExternalDocs(
        "RFC 1738 - Uniform Resource Locators (URL)",
        "https://datatracker.ietf.org/doc/html/rfc1738"
      ).some
    )
```

Código 34. Definición de la documentación a generar para parámetros de entrada de la clase URL

Finalmente, para acceder al fichero de documentación del API en formato JSON o YAML, es necesario acceder a los endpoints ***api/swagger.json*** o ***api/swagger.yml*** del API una vez desplegada. Dicho fichero luego puede integrarse en *SwaggerHub* automáticamente.

6.6.4.2.4 Sistema de logs

6.6.4.2.4.1 Infraestructura

El sistema de registros del módulo API está conformado por dos bibliotecas de código maduras:

1. ***Logback***: El *backend* del sistema de registros. Proporciona varios niveles de registro, un mecanismo de configuración en formato XML y una documentación muy completa.
2. ***Scala-logging***: El frontal del sistema de registros. Reduce la complejidad y verbosidad de integrar logback en el código del sistema a través de un sistema de interfaces y utilidades implícitas. Para poder registrar mensajes desde el código de una clase u objeto, no es necesario crear y dar nombre y contexto a un nuevo objeto *logger*, con que la clase extienda la interfaz *Logging* o *LazyLogging* ya tendrá acceso inmediato a un *logger* creado para la ocasión.

6.6.4.2.4.2 Funcionalidad

La funcionalidad del sistema viene marcada, principalmente, por su configuración en el fichero ***logback.xml*** en la raíz del proyecto.

6.6.4.2.4.2.1 Mensajes de consola

El sistema está configurado para utilizar un **ConsoleAppender** de *Logback* para imprimir mensajes en consola. El nivel de verbosidad de los mensajes se puede configurar desde el parámetro *-v*, *--verbose* de línea de comandos gracias a los siguientes pasos realizados por el sistema:

1. El sistema lee e interpreta los parámetros recibidos a través de la línea de comandos.
2. El sistema mapea el nivel de verbosidad seleccionado por el usuario a uno de los niveles de registros disponibles: *ERROR*, *WARNING*, *INFO*, *DEGUG* (se descarta el nivel *TRACE*).
3. El sistema almacena el nivel seleccionado por el usuario en una propiedad del sistema (ver clase **LoggingManager**).
4. El sistema, que cuenta con un *appender* dedicado a mostrar los registros de consola, lo invoca cada vez que el código quiere registrar un mensaje.
5. El *appender* de consola acude a un filtro personalizado, **VerbosityFilter**, que lee el nivel de verbosidad seleccionado por el usuario y deniega los mensajes de menor relevancia que la seleccionada.

```
<appender name="CONSOLE"
          class="ch.qos.logback.core.ConsoleAppender">
      <!-- Use custom filter -->
      <filter class="es.weso.rdfshape.logging.filters.VerbosityFilter"/>

      <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n
        </pattern>
      </encoder>
   </appender>
```

Código 35. Configuración de logback para el appender de consola

6.6.4.2.4.3 Registro a ficheros

El sistema está configurado para emplear un **RollingFileAppender** de *Logback* encargado de grabar los mensajes emitidos por el sistema con el nivel **INFO** o superior en archivos dentro del sistema de archivos del equipo. Los registros escritos en ficheros:

- Se localizan en una carpeta ***logs*** creada dentro de la ruta de ejecución del sistema. Por tanto, es necesario ejecutar la aplicación con un usuario con permisos de escritura en la ubicación de esta.
- Siguen una política **SizeBasedTriggeringPolicy**, con lo que los ficheros de registro son rotados, comprimidos y descartados según va aumentando el tamaño total de *logs* almacenados.

La ruta de destino de los *logs* así como la periodicidad y condiciones de la rotación de estos se encuentran definidas dentro del archivo *logback.xml*.

```
<appender name="ROLLING"
          class="ch.qos.logback.core.rolling.RollingFileAppender">
      <!-- Destination file -->
      <file>${LOGS_FOLDER}/${LOGS_FILENAME}</file>
      <!-- Level filter, do not log below INFO level (that is DEBUG and TRACE) -->
      <filter class="ch.qos.logback.classic.filter.ThresholdFilter">
        <level>INFO</level>
      </filter>

      <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
        <pattern>%d %level %thread %mdc %logger - %m%n</pattern>
      </encoder>
      <triggeringPolicy
            class="ch.qos.logback.core.rolling.SizeBasedTriggeringPolicy">
        <maxFileSize>50MB</maxFileSize>
      </triggeringPolicy>
      <rollingPolicy
            class="ch.qos.logback.core.rolling.FixedWindowRollingPolicy">
        <fileNamePattern>${LOGS_FOLDER}/${LOGS_FILENAME}.%i.zip
        </fileNamePattern>
        <minIndex>1</minIndex>
        <maxIndex>10</maxIndex>
      </rollingPolicy>
    </appender>
```

Código 36. Configuración de logback para el appender de ficheros

6.6.4.2.5 Configuración del proyecto

El módulo está construido como un proyecto *SBT*. Toda la labor de ampliación y refactorización del proyecto se ha efectuado sobre la configuración base.

Se recomienda inspeccionar el propio fichero *build.sbt* y sus comentarios adjuntos.

6.6.4.2.6 Sitio Web

El sitio web está constituido de forma similar al del módulo validador, por lo que se remite a la sección correspondiente: *6.6.4.1.3-Sitio web*.

6.6.4.3 *Módulo Cliente*

6.6.4.3.1 Introducción

La meta de este manual del programador es orientar a aquellos que busquen entender mejor el funcionamiento del cliente de *RDFShape* en vistas a refactorizarlo y/o extenderlo. La guía está centrada en los aspectos que han sido modificados en el sistema como parte de la elaboración del este proyecto. Lo relacionado con la implementación base del cliente en *React* se dejará aparte salvo que sea necesario mencionarlo.

RDFShape es de código abierto y los programadores interesados pueden encontrar información del proyecto en:

- Código fuente: https://github.com/weso/rdfshape-client.
- Despliegue del cliente: https://rdfshape.weso.es/.

6.6.4.3.2 Configuraciones de despliegue

6.6.4.3.2.1 Variables de entorno

Si bien *Node.js* y *React* permiten definir variables de entorno para los entornos de desarrollo y producción, el valor de estas variables queda fijado en el *JavaScript* de la aplicación final una vez construida. Esto implica que para modificar la configuración del sistema a través de variables de entorno era necesario:

1. Modificar las variables de entorno en cuestión (ficheros ***.env***).
2. Reconstruir la aplicación (despliegue con *Node.js*) o la imagen *Docker* completa (despliegue con contenedores).
3. Relanzar el sistema con el código *JavaScript* modificado.

Con la finalidad de poder configurar el comportamiento del sistema mediante variables de entorno en contenedores *Docker* sin tener que recrear la imagen, se ha ideado el siguiente mecanismo, ejecutado gracias al script ***env.sh*** en la raíz del proyecto:

1. Al lanzar un nuevo contenedor, este ejecuta el script **env.sh**.
2. El script escanea las variables de entorno definidas en los ficheros de *React* (***.env***) y las compara con las variables definidas para el contenedor.
3. El script vuelva los contenidos de las variables de entorno en un fichero *JavaScript* **env-config.js** que contiene un objeto con el par clave-valor de las variables de entorno, donde el valor dado a las variables del contenedor tiene preferencia sobre el valor base del **.env**.
4. La aplicación lee y accede dinámicamente al objeto en **env-config.js** para obtener el valor de las variables de entorno más actualizado.

6.6.4.3.2.2 NGINX

Sin ninguna modificación adicional, *NGINX* no es capaz de servir una aplicación *React* correctamente, puesto que intenta resolver las rutas de la aplicación él mismo en lugar de delegar en *React* para ello. Dado que el despliegue en contenedores se realiza sirviendo la aplicación desde *NGINX*, se ha resultado esta situación sobrescribiendo algunas configuraciones de *NGINX* con las presentes en los ficheros de la carpeta **nginx**, en la raíz del módulo, con los ficheros:

- *default.conf*: Configura el servidor para que, en caso de ser solicitada una ruta desconocida, delegue en *React* para resolverla, gracia a la directiva *try_files*.
- *gzip.conf*: Configuración opcional para mejorar el rendimiento del cliente mediante la compresión de los contenidos con el algoritmo Gzip.

```
server {

  listen 80;

  location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
    try_files $uri $uri/ /index.html;
  }

  error_page   500 502 503 504  /50x.html;

  location = /50x.html {
    root   /usr/share/nginx/html;
  }
}
```

Código 37. Archivo de configuración de NGINX compatible con las rutas de React

6.6.4.3.3 Configuración de la aplicación

6.6.4.3.3.1 Axios

Cada herramienta del cliente hace uso de una ruta diferente del API del *backend*. Como parte de la refactorización del cliente y para reducir la verbosidad y código duplicado, se ha creado una instancia global de *Axios* preconfigurada con la URL base del API (ver **axiosConfig.js**).

```
export const rootApi = environmentConfiguration.apiHost + "/api/";
export const rootWsApi = environmentConfiguration.wsApiHost + "/api/";

// Global axios instance with default settings for API connections
export default axios.create({
  baseURL: rootApi,
});
```

Código 38. Instancia y configuración globales de Axios

El valor de la URL base del *backend* se modificará al suministrado en el momento de despliegue del cliente. Cualquier configuración adicional que se quiera hacer al comportamiento de *Axios* (interceptores, manejo de errores, etc.) podrá introducirse en este punto para que se haga efectiva globalmente.

6.6.4.3.3.2 Contexto de React

La principal mejora de usabilidad introducida en el cliente es la capacidad de mantener los datos con los que el cliente está operando según este se mueve entre páginas y herramientas. Para este fin se ha creado un mecanismo basado en el API de contexto de React y la utilidad nativa *localStorage* de los navegadores web.

El contexto se encarga de almacenar los datos mientras dura la sesión del cliente y de sincronizarlos con el almacenamiento local *localStorage* de forma que persistan entre visitas del usuario. Cuando el usuario vuelve a la página web, el sistema intenta cargar los datos desde *localStorage* y recurre a los datos por defecto en caso contrario. Este mecanismo se puede ver en acción en las funciones **setLocalStorage** y **getLocalStorage** en **ApplicationProvider.js**.

En lo que respecta al contexto, es inicialmente definido en **ApplicationContext.js**, donde se asignan los datos por defecto (vacíos) para los datos aceptados por el contexto.

```
// Initial values in context
export const initialApplicationContext = {
  // Array of data (merge uses 2 units of data and more data compound could be added
in the future)
  // See provider for the function to add new data
  rdfData: [],
  sparqlQuery: InitialQuery,
  sparqlEndpoint: "",
  shexSchema: InitialShex,
  shaclSchema: InitialShacl,
  shapeMap: InitialShapeMap,
  umlData: InitialUML,
  streamingData: InitialDataStream,
};

// Shared context for storing the data the user is operating on
// and using it throughout the application (e.g.: for autifilling input forms when changing
page)
export const ApplicationContext = createContext(initialApplicationContext);
```

Código 39. Código de inicialización del Contexto

Es en el **ApplicationProvider** donde estos datos, junto con otras funciones capaces de operar con ellos, son imbuidas en el componente **Provider** que se monta en el árbol de componentes del sistema (ver App.js).

```
function App() {
  const [appContext, setAppContext] = useState(initialApplicationContext);
  return (
    <Container fluid={true}>
      <ApplicationProvider>
        <Routes />
      </ApplicationProvider>
    </Container>
  );
}

export default App;
```

Código 40. Montaje del ApplicationProvider en el árbol de componentes

Finalmente, una vez todo el contexto ha sido preparado, los distintos componentes de la aplicación son capaces de acceder a él para inicializarse con los datos ya existentes.

```
// Get all required data from state: data, schema, shapemap…
  const {
    rdfData: [ctxData],
    addRdfData,
    shexSchema: ctxShex,
    shapeMap: ctxShapeMap,
    streamingData: ctxStreamingData,
    setStreamingData: setCtxStreamingData,
  } = useContext(ApplicationContext);

  const [data, setData] = useState(ctxData || addRdfData());
  const [shex, setShEx] = useState(ctxShex || InitialShex);
  const [shapeMap, setShapeMap] = useState(ctxShapeMap || InitialShapeMap);
  …
```

Código 41. Inicialización con los datos del contexto en el componente ShExValidate

Escuela de
Ingeniería Informática
Universidad de Oviedo

ASTURIAS
CAMPUS DE EXCELENCIA
INTERNACIONAL
AD FUTURUM

Capítulo 7 Implantación y Aceptación del Sistema

Fase de Desarrollo

IAS

7.1 IAS 1: ESTABLECIMIENTO DEL PLAN DE IMPLANTACIÓN

El proyecto desarrollado tiene el objetivo de formar parte de la infraestructura del grupo de investigación WESO de la Universidad de Oviedo, en lo que respecta a:

- **Código y documentación de los componentes**
  El código de los módulos, así como su documentación asociada (*Scaladoc*, documentación en línea, wikis, etc.) y sitios web desarrollados formarán parte o estarán enlazados desde los propios repositorios de WESO en GitHub y la página web del grupo.
- **Despliegue y publicación de los sistemas**
  El sistema final producto de la puesta en marcha coordinada de todos los módulos desarrollados estará implantado en la estructura de servidores de WESO en la Universidad de Oviedo.

7.1.1 Plan de implantación del sistema

A la hora de preparar el despliegue en el entorno personalizado del cliente, no es suficiente con completar el desarrollo de cada módulo individual y entregarlo junto con sus manuales. En consecuencia, desde el inicio del proyecto se ha esbozado un plan de implantación priorizando la facilidad, portabilidad y automatización del despliegue de la plataforma *RDFShape*, que contiene los módulos Validador, API y Cliente desarrollados.

7.1.1.1 *Infraestructura existente*

El grupo WESO pone a disposición del proyecto un servidor cuya IP pública es **156.35.94.30** con las siguientes características básicas:

- Sistema Operativo: Ubuntu 22.04 LTS (x64).
- Memoria RAM: 10GB.
- Disco/Almacenamiento: 24GB.

Si bien el servidor no es más que una máquina virtual dentro de una infraestructura mayor, se descarta para el período de vida del proyecto alterar los recursos virtualizados disponibles. Así mismo, se permite la libre utilización del dominio **rdfshape.weso.es**, incluida la creación y gestión de subdominios por debajo del mismo.

7.1.1.2 *Dockerización de los servicios*

Desde el inicio, se opta por el uso de contenedores para el despliegue de los servicios, siendo *Docker* la tecnología seleccionada durante el desarrollo. La *Dockerización* de los servicios responde a dos necesidades:

- Tener un mecanismo de despliegue que cumpla los requisitos del cliente.
- Contar con un método estandarizado y documentado para arrancar, detener y monitorizar los servicios del servidor, en sustitución del sistema manual de *scripts* que se emplea previamente al plan de implantación.

Finalmente, el proceso de generación de imágenes *Docker* se integrará en la línea de integración continua de los módulos que los requieran y las imágenes se subirán al registro de *GitHub* tal que estén disponibles públicamente. Las imágenes contarán con la etiqueta *stable* si han sido probadas y subidas a mano por el desarrollador.

7.1.1.2.1 Módulo validador (*Comet*)

Es una biblioteca de código empleada como dependencia en el módulo API, por lo que no requiere de modificaciones en este aspecto.

7.1.1.2.2 Módulo API (*backend RDFShape*)

Se crea el fichero *Dockerfile* que soporte la funcionalidad y opciones de personalización definidas en los manuales de ejecución del módulo (ver *6.6.2.2.2-Docker*). El *Dockerfile* emplea un proceso de varias etapas para construir la imagen final, con la intención de ser más legible a la vez que se reduce considerablemente el tamaño de la imagen final **[105]**:

1. **Etapa de construcción**: Se parte de una imagen con todas las dependencias necesarias para construir el proyecto: *JDK*, *Scala*, *SBT*. Se instalan manualmente el resto de los paquetes necesarios y se construye un único ejecutable binario del módulo a través de la consola de *SBT*.
2. **Etapa de producción**: Se parte de una imagen base más ligera que tan sólo contenga el *JRE* necesario para ejecutar el sistema, prescindiendo de todas las herramientas para su construcción. Se copia el ejecutable del módulo y se crea un usuario no privilegiado encargado de iniciarlo. Así mismo, se definen las variables de entorno por defecto cuyos valores marcarán el comando de arranque del sistema.

El resultado final es una imagen *Docker* de aproximadamente 900MB lista para exponer el API. Se puede consultar el *Dockerfile* utilizado en el siguiente enlace: https://github.com/weso/rdfshape-api/blob/master/Dockerfile.

7.1.1.2.3 Módulo Cliente (*frontend RDFShape*)

Para el cliente, se opta por servir la aplicación web final construida por *React* a través de un servidor *NGINX* ejecutándose sobre *Alpine Linux*, lo que garantiza un sistema mucho más rápido y ligero que servir la aplicación mediante *Node.js*. Se opta por un proceso de construcción de la imagen en dos etapas:

1. **Etapa de construcción**: Se parte de una imagen basada en la versión LTS de *Node.js*. Se copian todos los ficheros de la aplicación, se instalan las dependencias y se construyen los ficheros HTML, CSS y JavaScript de producción.

2. **Etapa de producción**: Se parte de una imagen con *NGINX* preinstalado en la distribución *Alpine*. Se copian los archivos de producción de la aplicación dentro del directorio público de *NGINX* así como otras configuraciones necesarias y se sirve la aplicación a nombre del usuario de *NGINX*.

El resultado final es una imagen *Docker* de aproximadamente 60MB preparada para servir la aplicación web. Se puede consultar el *Dockerfile* utilizado en el siguiente enlace: https://github.com/weso/rdfshape-client/blob/master/Dockerfile.

7.1.1.3 *Automatización y securización del despliegue*

Tener imágenes *Docker* disponibles para cada módulo nos permite inicializarlos de forma sencilla y abstraída de su funcionamiento interno. Sin embargo, las condiciones del despliegue real exigen más que eso, puesto que se necesita:

- Un despliegue de los módulos:
  - Accesible a través del dominio **rdfshape.weso.es** y sus subdominios.
  - Securizado mediante certificados SSL que permitan encriptar los mensajes enviados mediante los protocolos *HTTP* y *WebSockets*.
- Un mecanismo de monitorización de los contenedores que permita su reinicio automático.

Se opta por utilizar imágenes *Docker* adicionales para el manejo de los dominios y certificados SSL de despliegue. Finalmente, y dado el crecimiento de la complejidad del sistema, se emplea *Docker Compose* para gestionar el despliegue coordinado de todos los contenedores. La arquitectura final es la planteada en *2.2.3.2.1.1-Arquitectura final del despliegue*.

El archivo *Docker Compose* elaborado para el despliegue del sistema final puede encontrarse en el siguiente enlace: https://gist.github.com/ulitol97/285086f9a8b1fd51207bfdf1ccc1a185.

7.1.1.4 *Calendario de despliegues*

Sentadas las bases para la implantación del sistema en el servidor del cliente, se define un calendario de despliegues abierto. El alumno contará con total autonomía para desplegar nuevas versiones del sistema en la infraestructura de WESO según vaya avanzando el desarrollo de los módulos y se implemente el plan de implantación, si bien tendrá que cumplir con las siguientes condiciones:

- Garantizar la máxima disponibilidad del sistema *RDFShape*.
- Finalizar la refactorización y ampliación de los módulos API y Cliente para la fecha de inicio de las clases de Web Semántica del máster, momento en que ambos módulos serán utilizados durante las lecciones.

7.2 IAS 7: PREPARACIÓN DEL MANTENIMIENTO DEL SISTEMA

De cara al futuro mantenimiento del sistema por parte de terceros ajenos al alumno, se han elaborado los siguientes procedimientos y manuales, disponibles de informa interna para el grupo WESO.

7.2.1 Manuales de despliegue y mantenimiento

7.2.1.1 *Despliegue del sistema*

7.2.1.1.1 Precondiciones

El equipo donde se vaya a desplegar el sistema debe cumplir las siguientes condiciones:

- El equipo debe tener conexión a internet para descargar los módulos del sistema, así como para ofrecerlos al exterior.
- Puerto 22 abierto para la administración a través de SSH (para servidores remotos).
- Puertos 80 y 443 abiertos para la conexión de los clientes a través de HTTP(S).
- *Docker* y *Docker Compose* deben estar instalados.

7.2.1.1.2 Método

Una vez llevada a cabo la *Dockerización* de los servicios que conforman *RDFShape*, el proceso de despliegue del *stack* API-Cliente se ha automatizado. Para desplegar la aplicación, es necesario seguir estos pasos:

1. Iniciar sesión en la máquina como el usuario *rdfshape*. Si no existe en el servidor, se recomienda crear un nuevo usuario no privilegiado para ejecutar los pasos del despliegue.
2. Verificar que tanto *Docker* como *Docker Compose* están instalados en la máquina.
3. **OPCIONAL.** Verificar que las versiones *stable* de las siguiente imágenes Docker están presentes en la máquina (usar *docker images*). Si no lo están, *Compose* se encargará de descargarlas antes de ejecutarse:
    a. ***RDFShape API***
    b. ***RDFShape Client***
4. Verificar la existencia del fichero ***docker-compose.yml*** en */home/rdfshape*. En caso de no existir o estar desactualizado, este fichero no es más que una versión modificada de la plantilla disponible públicamente para el despliegue, y cuenta con las siguientes mejoras/cambios:
    a. Ya está preparada para usar un *proxy* inverso y certificados SSL proporcionados por **Let's Encrypt** automáticamente.
    b. Incluye el contenedor *autoheal*, que monitoriza al resto y los reinicia en caso de fallo.
    c. Usa la política de reinicio "*always*" en vez de "*unless-stopped*", que pone en marcha los contenedores en cuanto se lanza el daemon de *Docker*.

5. Finalmente, iniciar el sistema:
    a. En el directorio /home/rdfshape: ***docker-compose up -d*** (usar -d para lanzar los servicios en segundo plano).
    b. Desde cualquier directorio: ***docker-compose up -f /home/rdfshape/docker-compose.yml***.

7.2.1.2 *Monitorizar y reiniciar los contendores*

Los contenedores de RDFShape ya tienen una serie de configuraciones básicas para monitorizarlos que se pueden ver en el fichero *docker-compose.yml*:

- Política de reinicio *always*: para relanzarse de nuevo incluso si la máquina se reinicia o el *daemon* de *docker* cae.
- *Healthcheks*: los contenedores se lanzan peticiones HTTP a sí mismos periódicamente para ver si las aplicaciones web en su interior responden o no. En caso de que no, un vistazo a los contenedores con "*docker ps*" los mostrará como *unhealthy*.

En el caso de que las aplicaciones sigan "funcionando" pero de forma errónea (*unhealthy*) a causa de bucles, bugs... *Docker* no puede relanzar los contenedores por sí mismo. Para solventar esto se ha añadido al *stack* de despliegue un nuevo contenedor *autoheal* que monitoriza al resto y los reinicia en caso de que estén *unhealthy* (sin intervención humana). Es importante subrayar que *autoheal* siempre reinicia los contenedores marcados como *unhealthy*, por lo que es de máxima importancia diseñar correctamente los *healthchecks* de los contenedores para evitar falsos positivos o negativos.

En última instancia, siempre es posible devolver todo el sistema a su estado inicial a través de:

1. Detener todos los servicios: ***docker-compose down***.
2. Relanzar todos los servicios: ***docker-compose up -d***.

7.2.1.3 *Actualizar las imágenes Docker stable*

Como se ha descrito anteriormente, las imágenes *Docker* con la etiqueta ***:stable*** son las empleadas en el despliegue. Esta etiqueta se actualiza a mano con imágenes que se han comprobado que funcionan correctamente. El proceso para generar y subir estas imágenes al *GitHub Container Registry* es el siguiente:

1. Estar autenticado en el Container Registry usando un token válido de WESO, más información aquí.
2. Generar la nueva imagen *Docker* de la aplicación en la máquina de desarrollo local.
3. Etiquetar la imagen generada como "***ghcr.io/weso/{NOMBRE_APP}:stable***".
4. Hacer *push* de la imagen al registro.
5. En el próximo despliegue el servidor detectará que hay una nueva versión de las imágenes y las descargará (si no, forzar con *docker-compose pull*) antes de iniciar el sistema.

Por ejemplo, para subir una nueva versión ***:stable*** de *RDFShape API*, el proceso sería:

```
# Loguearse en el registro, necesario la primera vez
echo "token" | docker login ghcr.io -u GITHUB_USERNAME --password-stdin

# Construir la imagen con una etiqueta cualquiera (aquí se emplea “local”)
docker build . -t rdfshape-api:local

# Etiquetar la imagen generada para que esté lista para subir
docker tag {ID_IMAGEN_GENERADA_EN_LOCAL} ghcr.io/weso/rdfshape-api:stable

# Subir la imagen
docker push ghcr.io/weso/rdfshape-api:stable
```

Código 42. Actualización manual de una imagen Docker del sistema

7.2.1.4 *Manejo de certificados SSL*

Tras la automatización del despliegue, la gestión de los certificados SSL de *Let's Encrypt* se hace de forma automática gracias al contenedor *jwilder/nginx-proxy* y su acompañante *nginxproxy/acme-companion*.

Puesto que el proxy de *NGINX* es el único punto de entrada al sistema, sus comunicaciones están aseguradas mediante certificados *SSL*. De esta forma, no es necesario configurar los módulos API o Cliente para usar certificados propios.

7.3 IAS 9-10: Presentación y Aprobación del Sistema y Paso a Producción

El ecosistema *RDFShape* se encuentra en producción desde antes de la realización del proyecto. Actualmente, el sistema final puede encontrarse en las siguientes ubicaciones:

- **API**: https://api.rdfshape.weso.es/.
- **Cliente Web**: https://rdfshape.weso.es/.

Desde su ampliación y refactorización durante el desarrollo del proyecto, el sistema ha sido:

- Empleado por otros grupos de investigación en la línea de Web Semántica.
- Promocionado en el sitio web de *Shape Expressions*, amparado bajo el *W3C*.
    - *RDFShape* puede encontrarse en el listado de herramientas en https://shex.io/.
- Mostrado en charlas y eventos de varias organizaciones:
    - Lotico: *Data Shapes Event.*
      Disponible: http://www.lotico.com/index.php/Data_Shapes.
- Empleado en la asignatura "Web Semántica" del Máster en Ingeniería Web y la asignatura optativa del grado "Sistemas de Información para la Web" como herramienta de introducción y práctica con tecnologías semánticas.

Capítulo 8 Conclusión y Ampliaciones

8.1 CONCLUSIONES

8.1.1 Conclusiones técnicas

Se puede concluir que el proyecto ha sido finalizado cumplimentando todos los requisitos y obteniendo el visto bueno del cliente.

Para el módulo de validación de flujos, se ha desarrollado una biblioteca de código de alto rendimiento y de alcance público cumpliendo una serie de objetivos estratégicos:

- Colaboración con WESO y publicación como código abierto.
- Publicación de la biblioteca en un repositorio público accesible a gestores de dependencias.
- Generación de un sitio web personalizado para el sistema.
- Compilación cruzada para el máximo alcance.

Los módulos API y Cliente, del ecosistema de *RDFShape*, partían de una estructura existente que se ha ampliado no sólo con los objetivos de acoger la validación de flujos, si no para mejorar el funcionamiento y documentación de los sistemas con tecnologías web de vanguardia: automatización de la documentación del API, uso del Contexto de *React*, *WebSockets*, etc. Adicionalmente, y pese a que la usabilidad y accesibilidad del sistema final no eran el foco del proyecto, también se han obtenido reseñas positivas por parte del tutor y los usuarios finales. Aun así, el módulo cliente se beneficiaría de una fuerte refactorización y migración a *TypeScript* antes de introducir cambios adicionales.

Finalmente, se ha ideado un plan de implantación para facilitar un despliegue automatizado, portable, mantenible y seguro para cualquier infraestructura.

En tanto al ámbito de gestión de proyectos, el presupuesto ha sido mayormente certero y no ha sufrido grandes modificaciones entre los puntos de revisión establecidos. Quedan aspectos a mejorar en lo relativo a la gestión de riesgos puesto que, aunque los riesgos han sido identificados correctamente, en general se subestimó la probabilidad de que apareciesen así como el impacto final sobre el cronograma. En sucesivas ocasiones el daño ocasionado por los riesgos se vio corregido con retrasos en la fecha de finalización ya que las reservas temporales no eran suficientes. Afortunadamente, y gracias al inicio temprano del proyecto, se contaba con margen suficiente para atrasar la entrega, pero este no es el *modus operandi* ideal en un ambiente corporativo. En una nota más positiva, considero que estos errores han sido una oportunidad más para seguir aprendiendo y creciendo de cara a la realidad del mundo laboral.

8.1.2 Conclusiones personales

Como puede comprobarse desde el inicio de la memoria, este proyecto abarca un desarrollo no sólo multimodular sino también multidisciplinar que incluye una biblioteca de código, un API y un cliente web, así como la coordinación de todas las partes para el despliegue.

Si bien se partía de una base preestablecida para los dos módulos que constituían *RDFShape*, trabajar en todas las partes que componen un sistema web complejo me ha ofrecido una nueva perspectiva de lo que implica mantener y ampliar un desarrollo:

- La necesidad de una documentación completa pero también clara y concisa, tanto dentro del propio código como en formato más amigable o de manual.
- La dificultad de mantener un ecosistema coherente y cohesionado, especialmente cuando no se quiere comprometer la compatibilidad entre un API y sus clientes oficiales.
- Las posibilidades que ofrecen los sistemas de integración continua, no sólo para construir y ejecutar las pruebas unitarias de las aplicaciones, sino para publicarlas como se ha hecho con la biblioteca o con los sitios web autogenerados para algunos módulos.
- La dificultad y cantidad de tiempo necesario para comprender un sistema desarrollado por terceros antes de poder ampliarlo.

Por otra parte, la mayoría de las complicaciones del proyecto han venido de la mano del paradigma de programación funcional y sus tecnologías asociadas: *Scala*, *cats-effect*, *http4s*, *fs2*, etc. No quiero decir que estas metodologías y herramientas sean superiores o inferiores, pero sí pertenecen a un ecosistema con el que no era familiar y en el que he tenido la posibilidad de formarme. Actualmente, siento que son tecnologías muy poderosas de las que me gustaría hacer uso en futuros proyectos personales y profesionales.

Finalmente, este es el proyecto de mayor envergadura al que me he enfrentado por mi cuenta y contar con el apoyo de WESO así como saber que el código es consultado y utilizado por otros investigadores, profesores y alumnos de mi misma titulación me hace sentir orgulloso.

8.2 AMPLIACIONES

Mantener y ampliar un ecosistema web completo es una tarea complicada, con lo que la necesidad de finalizar el proyecto a tiempo ha hecho que varias metas de alcance se descartasen a lo largo del desarrollo.

A continuación, se listan a modo de potenciales ampliaciones algunas de las ideas descartadas junto con otras adicionales:

- **Módulo de validación de flujos**
    - Capacidad de definir condiciones específicas de detención del validador, más allá de si algún elemento es inválido o si se producen errores.
    - Facilitar la integración con sistemas de alertas por email, SMS, etc. con la finalidad de notificar a los usuarios cuando el validador se detenga o se den ciertas condiciones específicas.
    - Implementar nuevos extractores capaces de procesar flujos provenientes de otras fuentes aparte de *Apache Kafka*.
- **Módulo API**
    - Ampliar los mecanismos de monitorización del API más allá del sistema de *logs* actual:
        - Mediante formatos y tecnologías más amigables, como *Prometheus* y *Grafana*.
        - Añadiendo más métricas monitorizables además de las peticiones recibidas: uso de CPU, memoria, disco, etc.
- **Módulo Cliente**
    - Migración del código a *TypeScript*.
    - Creación de una biblioteca de componentes comunes a otros proyectos web de WESO, como *WikiShape*.
- **Despliegue**
    - Emplear mecanismos más accesibles para la gestión del despliegue que ofrezcan un cuadro de mandos y/o una interfaz gráfica, como *Portainer*.

ANEXOS

A1. PLAN DE GESTIÓN DE RIESGOS

A1.1 Introducción

A.1.1.1 Objetivo

El presente documento define el plan de gestión de riesgos del proyecto "Validación de flujos RDF mediante Shape Expressions".

El objetivo de este plan es describir la metodología a seguir para identificar, categorizar, evaluar y tratar los riesgos del proyecto.

A.1.1.2 Alcance

El plan de aquí descrito contiene la siguiente información aplicable a la gestión de riesgos:

- *Metodología*
- *Herramientas y Tecnologías*
- *Roles y Responsabilidades*
- *Presupuesto de Riesgos*
- *Calendario de Riesgos*
- *Categorías de Riesgo*
- *Definiciones de Probabilidad*
- *Definiciones de Impacto por Objetivos*
- *Matriz de Probabilidad-Impacto*
- *Niveles de Tolerancia*
- *Planes de Contingencia*
- *Formatos de la Documentación*

A.1.1.3 Conceptos generales

Con el objetivo de evitar ambigüedades, se deben definir algunos conceptos utilizados en la gestión de riesgos:

- **Riesgo**
  Es cualquier evento que pueda causar un efecto en el proyecto. Generalmente, el efecto será una amenaza, si bien podría ser una oportunidad. Las primeras deben ser evitadas y las segundas explotadas.
- **Factor de riesgo**
  Los factores de riesgo son circunstancias asociadas con el riesgo de que aumenta la posibilidad de conseguir los efectos descritos por el riesgo. Por lo general, los riesgos son imposibles o

difíciles de medir o controlar directamente y las decisiones se toman controlando los factores de riesgo con el fin de evitar o minimizar (o potenciar) los efectos de riesgo.

- **Indicadores de riesgo**
  Los indicadores de riesgo son elementos asociados al riesgo y/o sus factores, que se puede medir y sirve como información acerca de la posibilidad de que se produzca el efecto del riesgo

A1.2 Metodología

Este plan de gestión de riesgos se aplica a todos los riesgos identificados en la fase de análisis del proyecto, una vez se han identificado la arquitectura y requisitos formales. Los riesgos deben ser conocidos por todas las partes involucradas en el desarrollo del sistema, así como supervisados mediante un plan de auditorías. Esto es consecuencia de que los riesgos pueden evolucionar durante la vida del proyecto, causando pérdidas temporales, económicas e incluso la cancelación de este.

La metodología aplicada en este plan no es derivada de ninguna otra metodología presente en el grupo de investigación WESO, si bien está cimentada sobre fuentes fiables y el propio conocimiento del jefe de proyecto. Dicha metodología se subdivide en dos facetas a concretar a continuación.

A.1.2.1 Metodología General

La metodología general describe, *grosso modo*, el proceso de gestión de riesgos desde el comienzo hasta el cierre de las tareas del proyecto. Este proceso consta de las siguientes fases:

1. Identificación inicial de riesgos

   Primer contacto con los riesgos del proyecto. Se aplican las herramientas y tecnologías disponibles, como la lluvia de ideas, para listar los potenciales riesgos del proyecto. En consecuencia, los riesgos serán tratados de manera proactiva, y nunca reactiva.

2. Elaboración del Plan de Gestión de Riesgos

   Se elabora el plan de gestión inicial, el cual comprende, entre otros: los criterios para la gestión de riesgos, el registro de riesgos y las hojas de riesgos para todos los riesgos priorizados.

3. Monitorización de Riesgos

   Según el proyecto avance, se llevarán a cabo:

   - **Evaluaciones de riesgos**. Con cierta periodicidad, los riesgos han de ser reevaluados a fin de confirmar que no amenazan al proyecto.
   - **Actualizaciones del plan de gestión de riesgos**. Gran cantidad de riesgos sobreviven al avance del proyecto, pero no todos necesariamente. Algunos riesgos tienen un ciclo de vida ligado a una fase concreta del proyecto, mientras que otros pueden aparecer o aumentar su probabilidad con el transcurso del tiempo. Se debe actualizar el plan y el registro de riesgos una vez se den estas circunstancias.

- **Proyecto de actualización del plan**. Se contempla la posibilidad de actualizar el plan de gestión de riesgos, siempre con el fin de evitar o mitigar el impacto de los riesgos según evolucionen.

4. Informe Final de Riesgos

   Antes del cierre del proyecto, se elaborará un informe que muestre la evolución de los riesgos identificados.

A.1.2.2 Metodología de Gestión de Riesgos

Se plantea una metodología de gestión de riesgos que opera a un nivel más técnico que la metodología general y que definimos en seis pasos a aplicar mediante las herramientas y tecnologías disponibles.

1. Planificación de la gestión de riesgos

   Se define cómo realizar las actividades de gestión de los riesgos para el proyecto.

2. Identificación de riesgos

   Se determinan los riesgos que pueden afectar el proyecto y se documentan sus características.

3. Análisis cualitativo de riesgos

   Se priorizan los riesgos para realizar otros análisis o acciones posteriores, evaluando y combinando la probabilidad de ocurrencia con el impacto de dichos riesgos.

4. Análisis cuantitativo de riesgos

   Se analiza numéricamente el efecto de los riesgos identificados sobre los objetivos generales del proyecto.

5. Planificación de las respuestas a los riesgos

   Se desarrollan opciones y acciones para mejorar las oportunidades y reducir las amenazas a los objetivos del proyecto.

6. Monitorización y control de riesgos

   Se implementan planes de respuesta a los riesgos, se rastrean los riesgos identificados, se monitorizan los riesgos residuales, se identifican nuevos riesgos y se evalúa la efectividad del plan de riesgos a través del proyecto.

A1.3 Herramientas y Tecnologías

A continuación, se describen las técnicas contempladas para la gestión de riesgos.

A.1.3.1 Tormenta de Ideas

Esta técnica será utilizada principalmente para la obtención del listado principal de los riesgos. Consiste en la discusión, por parte de los perfiles implicados en el proyecto, sobre los principales objetivos del proyecto y todas las cosas que pueden salir mal.

A.1.3.2 Evaluaciones Periódicas

Regularmente, todos los indicadores serán evaluados de acuerdo con el plan establecido en las hojas de riesgos. Todos estos resultados se discutirán en los eventos marcados por el plan de seguimiento de riesgos.

A.1.3.3 Reuniones

Regularmente, se debe incluir un punto de discusión de riesgos en el orden del día de las reuniones con el fin de tomar decisiones acerca de los riesgos del proyecto y sus efectos.

A.1.3.4 Delphi

En caso de desacuerdo sobre las características de un riesgo, se aplicará el método Delphi para resolver el problema.

A.1.3.5 Diagramas Causa-Efecto

Es posible, debido a la complejidad de las relaciones entre un riesgo y sus factores de riesgo, emplear un diagrama de causa-efecto para obtener una representación gráfica.

A.1.3.6 Análisis DAFO

Se valorará la aplicación del análisis DAFO (Debilidades, Amenazas, Fortalezas y Oportunidades) al contexto del proyecto para la identificación de nuevos riesgos.

A.1.3.7 Documentación Existente

En el supuesto de que existiesen proyectos similares al aquí desarrollado, revisar su documentación para ayudar en la identificación y evaluación de riesgos.

A.1.3.8 Otros

Se usarán herramientas y procesos matemáticos para hacer las evaluaciones cuantitativas, según proceda.

A1.4 Roles y Responsabilidades

Los siguientes roles dentro del personal asignado al proyecto tendrán un papel protagónico en la gestión de riesgos.

A.1.4.1 Jefe de Proyecto

Tendrá las siguientes responsabilidades:

- Ser el responsable y director del proyecto, lo que incluye la gestión de sus riesgos.
- Informar al cliente y tener en consideración sus encargos y preocupaciones.
- Definir el plan de gestión de riesgos.
- Definir el plan de seguimiento de riesgos.
- Velar por el cumplimiento de la metodología de riesgos.
- Aplicar con criterio las herramientas y tecnologías de gestión de riesgos.
- Gestionar el cronograma, presupuesto y personal asignado a la gestión de riesgos.
- Participar activamente en las auditorías de seguimiento de riesgos y definir las tareas de corrección oportunas.

A.1.4.2 Analista y Consultor de Tecnología

Tendrán las siguientes responsabilidades:

- Participar en las tareas de identificación, análisis, priorización y elaboración de estrategias del plan de gestión de riesgos.
- Aportar una perspectiva técnica, así como conocimiento del dominio a las tareas de gestión de riesgos.
- Apoyar al jefe de proyectos y al resto del equipo en las tareas de gestión de riesgos que surjan a lo largo del desarrollo del proyecto.

A.1.4.3 Equipo de Análisis

Conformado por el resto del personal asignado al proyecto, es responsable de apoyar a los roles previamente descritos en el cumplimiento de sus responsabilidades y de notificar cualquier incidencia o sugerencia que surja a lo largo del desarrollo del proyecto.

A1.5 Presupuesto de Riesgos

En la siguiente tabla se muestran las tareas del proyecto dedicadas a la gestión de riesgos junto con su asignación monetaria.

Tarea (ID MPP)	Descripción	Asignación (€)
109	Definición del plan de gestión de riesgos	390,00 €
110	Identificación y registro de riesgos	1.102,00 €
111	Definición del plan de seguimiento de riesgos	130,00 €
125	Sesiones de seguimiento de riesgos	585,00 €
135	Informe final de riesgos	260,00 €
	TOTAL	**2467,00 €**

Presupuesto de riesgos

A1.6 Calendario de Riesgos

En la siguiente tabla se muestran las tareas del proyecto dedicadas a la gestión de riesgos junto con su asignación temporal.

Dado que el plan de riesgos se elabora al arranque del proyecto, las fechas aquí mostradas son las definidas por la planificación inicial y pueden verse levemente alteradas según se realicen correcciones en la planificación.

ID Tarea	Descripción	Duración
109	Definición del plan de gestión de riesgos	17/09/2021 – 20/09/2021
110	Identificación y registro de riesgos	08/12/2021 – 16/12/2021
111	Definición del plan de seguimiento de riesgos	21/09/2021 – 21/09/2021
125	Sesiones de seguimiento de riesgos	21/01/2022 – 01/04/2022
135	Informe final de riesgos	03/05/2022 – 04/05/2022

Calendario de riesgos

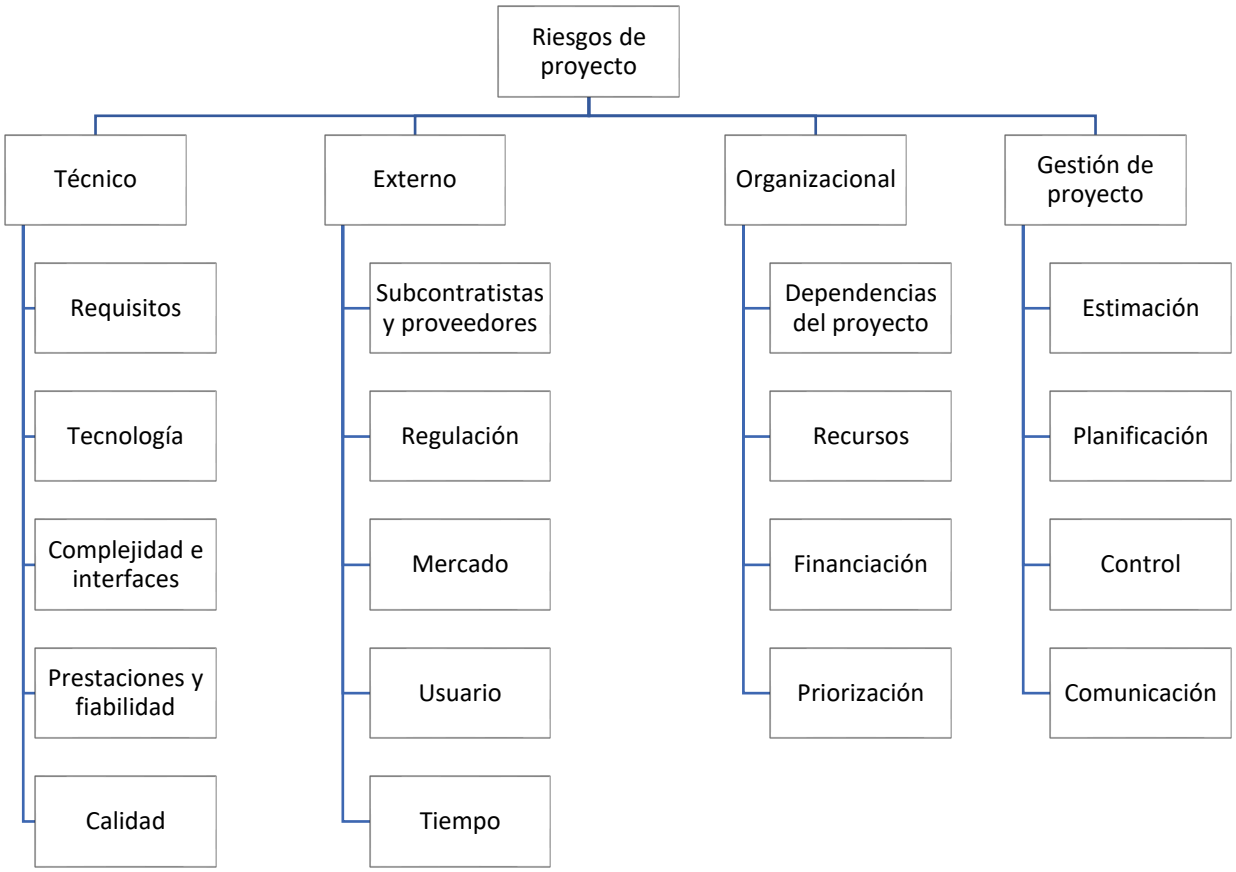
A.1.6.1 Seguimiento de Riesgos

Hay un total de tres sesiones de seguimiento de riesgos planificadas y presupuestadas.

Estas sesiones se encuentran espaciadas de la manera más equidistante posible entre el momento en que se tiene el catálogo inicial de riesgos y el cierre del proyecto. El jefe de proyecto será el encargado de evaluar el estado actual del proyecto frente al estado deseado, reevaluar los riesgos según proceda y establecer correcciones en el plan.

A1.7 Categorías de Riesgo

Los riesgos del proyecto se categorizan en base al siguiente diagrama RBS, o *Risk Breakdown Structure*.



Para nuestro RBS se definen cuatro categorías principales, las cuales se explican como:

- Riesgo técnico: riesgos derivados de especificaciones técnicas y tecnológicas del proyecto.
- Riesgo externo: riesgos causados por factores externos que pueden influir en el proyecto.
- Riesgo organizaciones: riesgos derivados del contexto de la organización responsable del proyecto.
- Riesgo de gestión de proyecto: riesgos derivados de la gestión del proyecto.

Un riesgo puede pertenecer a varias (sub)categorías simultáneamente.

A1.8 Definiciones de Probabilidad

Cuando se trata con riesgos, estos tienen una probabilidad asociada en cada momento en el tiempo. A continuación, se definen las probabilidades empleadas en el presente proyecto y cómo deben interpretarse.

Nombre	Porcentaje equivalente (%)	Descripción
Muy baja	0% – 20%	La probabilidad de que el riesgo se materialice es muy baja
Baja	20% - 40%	La probabilidad de que el riesgo se materialice es baja
Media	40% - 60%	Existe cierta probabilidad de que el riesgo llegue a producirse
Alta	60% - 80%	La probabilidad de que el riesgo se materialice es elevada
Muy alta	80% - 100%	La probabilidad de que el riesgo se materialice es muy elevada

Definiciones de probabilidad de riesgos

A1.9 Definiciones de Impacto por Objetivos

A continuación, se muestran las definiciones de impacto para cada uno de los objetivos del proyecto. Los objetivos del proyecto que se han contemplado son:

- Alcance: tareas y objetivos abarcados por el proyecto
- Calidad: utilidad del proyecto final y cumplimiento de los requisitos
- Coste: presupuesto destinado a la finalización del proyecto
- Cronograma: tiempo destinado a la finalización del proyecto

Así mismo, se elaboran estas definiciones tanto para las amenazas como para las oportunidades del proyecto, a modo de poder interpretar las cifras en ambos casos.

Objetivos	Escalas relativas / numéricas (%)				
	Muy bajo / 5%	Bajo / 15%	Medio / 30%	Alto / 55%	Muy alto / 90%
Alcance	Reducciones de alcance inapreciables	Afectadas áreas menores del alcance	Afectadas áreas importantes del alcance	Reducciones del alcance inaceptables para el cliente	El resultado final del proyecto no es realmente útil
Calidad	Degradación de calidad inapreciable	Aplicaciones muy exigentes se verán afectadas	La reducción de la calidad requiere aceptación del cliente	Reducción de calidad inaceptable para el cliente	El resultado final del proyecto no es realmente útil
Coste	Incremento insignificante del coste	Incremento del coste < 15%	Incremento del coste 15% - 30%	Incremento del coste 30% - 55%	Incremento del coste > 55%
Cronograma	Incremento insignificante de duración	Incremento de duración < 15%	Incremento de duración 15% - 30%	Incremento de duración 30% - 55%	Incremento de duración > 55%

Definiciones de impacto por objetivos – Amenazas

Objetivos	Escalas relativas / numéricas (%)				
	Muy bajo / 5%	Bajo / 15%	Medio / 30%	Alto / 55%	Muy alto / 90%
Alcance	Nuevas funcionalidades poco apreciables	Mejoradas áreas menores del alcance	Mejoradas áreas importantes del alcance	Ampliaciones del alcance que satisfacen al cliente	El resultado final del proyecto es más satisfactorio para el cliente de lo esperado
Calidad	Aumento de calidad inapreciable	Aplicaciones muy exigentes se verán mejoradas	El aumento de calidad es satisfactorio para el cliente	El aumento de calidad es muy satisfactorio para el cliente	El resultado final del proyecto es más satisfactorio para el cliente de lo esperado
Coste	Decremento insignificante del coste	Decremento del coste < 15%	Decremento del coste 15% - 30%	Decremento del coste 30% - 55%	Decremento del coste > 55%

Cronograma	Decremento insignificante de duración	Decremento de duración < 15%	Decremento de duración 15% - 30%	Decremento de duración 30% - 55%	Decremento de duración > 55%

Definiciones de impacto por objetivos – Oportunidades

A1.10 Matriz de Probabilidad-Impacto

La matriz de probabilidad e impacto define los baremos que se emplearán a la hora de cuantificar la importancia de los riesgos y, en consecuencia, priorizarlos. Para este proceso, se atiende a la probabilidad de que el riesgo suceda junto con el impacto que tendría sobre el proyecto en caso de ocurrir.

En la siguiente tabla se representa de forma gráfica la matriz empleada para este proyecto, atendiendo a las definiciones de probabilidad y de impacto de los apartados anteriores:

Probabilidad	**Muy Alta**	***0.90***	0.05	0.14	0.27	0.50	0.81
	Alta	***0.70***	0.04	0.11	0.21	0.39	0.63
	Media	***0.50***	0.03	0.08	0.15	0.28	0.45
	Baja	***0.30***	0.02	0.05	0.09	0.17	0.27
	Muy Baja	***0.10***	0.01	0.02	0.03	0.06	0.09
			0.05	***0.15***	***0.30***	***0.55***	***0.90***
			Muy Bajo	**Bajo**	**Medio**	**Alto**	**Muy alto**
			Impacto				

Matriz de Probabilidad-Impacto

A1.11 Niveles de Tolerancia

El umbral de riesgo se define en una cota de **0,40**. Los riesgos que sobrepasen ese valor serán considerados una verdadera amenaza u oportunidad que prevenir o explotar. Los riesgos con un impacto menor que dicha cota no se considerarán relevantes en un primer término.

A1.12 Planes de Contingencia

A.1.12.1 Planificación

La fecha de finalización estimada para el proyecto es mayo de 2022. El proyecto no debe, bajo ningún concepto, demorarse más allá del 3 de julio de 2022, fecha designada por el cliente para la entrega del proyecto.

A.1.12.2 Presupuesto

El jefe de proyecto se compromete a revisar y corregir el presupuesto estimado según corresponda, comunicando los cambios al cliente. No se tolerará que el presupuesto final supere al presupuesto inicial en más de un 5%.

A1.13 Formatos de la Documentación

Para la documentación de riesgos se tendrán como referencia los siguientes estándares y normas:

- UNE-ISO 31000:2010. Gestión del riesgo.
- UNE-EN 31010:2011. Gestión del riesgo. Técnicas de apreciación del riesgo.

A2. HOJAS DE RIESGOS

Este anexo contiene las hojas de los riesgos identificados en *4.3.2.3-Riesgos*.

A2.1 Hoja de riesgos I

<table>
<tr><td>ID:
1</td><td colspan="6">Nombre:
Inexperiencia del programador en las tecnologías empleadas por el cliente en el módulo API</td></tr>
<tr><td colspan="7">Descripción:
En el programa que se ejecuta en el servidor que monta el API REST para la validación de RDF, el cliente hace uso de tecnologías muy potentes, pero poco populares en el ámbito general, tales como el lenguaje de programación Scala y la metodología de programación funcional con las herramientas del ecosistema Typelevel. La complejidad y el desconocimiento de las bases de estas tecnologías puede lastrar los resultados y demorar las tareas del programador.</td></tr>
<tr><td colspan="7">Categoría(s) de riesgo:
Tecnología (técnico), Calidad (técnico), Estimación (gestión de proyecto)</td></tr>
<tr><td>Estado:
Activo</td><td colspan="6">Causas del Riesgo (Identificación de los factores del riesgo):
- El desarrollador carece de experiencia en los lenguajes de programación exigidos por el cliente para el módulo API.
- El desarrollador carece de experiencia en las tecnologías empleadas por el cliente en el módulo API.
- Escasez de documentación y material didáctico para la formación en las tecnologías afectadas.
- Programa de formación del programador ineficiente o de escasa duración.</td></tr>
<tr><td rowspan="2">Probabilidad</td><td colspan="4">Impacto</td><td rowspan="2">Impacto Total</td><td rowspan="2">Respuestas</td></tr>
<tr><td>Presupuesto</td><td>Planificación</td><td>Alcance</td><td>Calidad</td></tr>
<tr><td>Muy alta</td><td>Medio</td><td>Alto</td><td>Alto</td><td>Alto</td><td>0,50</td><td>Mitigar.
Dentro del equipo de proyecto se contará con un consultor de tecnologías con experiencia previa en el campo que definirá, antes de iniciarse el desarrollo y con</td></tr>
</table>

						el beneplácito del jefe de proyecto, un plan de formación para el programador del sistema.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
28/01/2022	Medio	Medio	Muy bajo	Bajo	0,27	- Llegados a este punto del desarrollo, el programa de formación ya ha sido completado con éxito, sin embargo, el proceso ha llevado una semana más de lo esperado, materializando el riesgo. - Se atrasa una semana el plan de reuniones y auditorías de seguimiento para ajustarse al retraso sufrido. - Se procede a reestimar el tiempo de desarrollo del módulo API, concluyéndose que la estimación inicial ya es suficiente, dado que: o El programa de formación es completado con buenos resultados. o El módulo API necesita ser refactorizado y ampliado, no codificado desde cero, aliviando la carga del programador.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		

07/03/2022	Bajo	Bajo	Muy bajo	Muy bajo	0,14	- Los retrasos en el período de formación ya han sido paliados y replanificados tras la primera revisión. - Los resultados de la formación y estimaciones iniciales están demostrando ser correctas y el módulo API se encuentra en una fase avanzada del desarrollo, según lo previsto.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
08/04/2022	Muy bajo	Muy bajo	Muy bajo	Muy bajo	0,05	- El módulo API ha sido codificado y ha pasado las pruebas en los plazos previstos.

Riesgos derivados de éste:

- Retrasos en la fase inicial del proyecto (formación) y en la fase de construcción del sistema que atrasen la fecha de finalización del proyecto.
- Incremento del presupuesto de desarrollo por las horas de trabajo y el esfuerzo adicional del programador y otros perfiles técnicos.
- Reducción moderada o acusada de la calidad del proyecto final, pudiendo llegar a recortarse el alcance de este.

Riesgo residual:

Retrasos en el desarrollo del módulo API y, por ende, de la fecha de entrega del proyecto. Reducción del nivel de satisfacción del cliente y posible degradación de la calidad del producto final en función de los conocimientos técnicos adquiridos por el programador en el dominio tecnológico del cliente.

Plan de Contingencia:

1. Hacer un estudio de tecnologías como parte de los estudios iniciales del proyecto y previo al desarrollo de los módulos. Si se admite una tecnología dentro del *stack* tecnológico del proyecto está vendrá acompañada de:
    a. Documentación que contenga como mínimo:
        i. Precio de la tecnología
        ii. Licencia de uso de la tecnología
        iii. Listado de alternativas
        iv. Justificación de su uso frente a las alternativas
    b. El beneplácito del jefe de proyecto, el arquitecto de software u otro personal especializado cuando sea pertinente.
2. Contratar personal que tenga experiencia suficiente en el dominio de investigación como consultor de tecnologías para el proyecto.
3. Elaborar y ejecutar un plan de formación del programador como parte de los estudios iniciales del proyecto y previo al desarrollo de los módulos.
    a. Se formará al programador es las tecnologías escogidas para la arquitectura final del sistema y según lo estipulado en el estudio tecnológico.
    b. La formación será impartida por el consultor tecnológico del proyecto.

Presupuesto para contingencias:

2.152,00 € (estudio tecnológico validación RDF)
1.784,00 € (formación back-end)

Total: 3.396,00 €

Planificación temporal de las contingencias:

El estudio tecnológico y la formación deberán planearse y ejecutarse como parte de la fase de estudios iniciales del proyecto, antes de la fase de análisis.

Las hipotéticas correcciones de planificación o presupuesto a aplicar se decidirán en las auditorías de seguimiento y se nutrirán de las reservas de tiempo y dinero reservadas para contingencias.

Comentarios:

N/A

Monitorización (Descripción de la relación de los factores de riesgo y los indicadores y descripción de los planes de monitorización)**:**

Se han identificado los siguientes indicadores asociados a los factores de riesgo:

	Valor del riesgo				
INDICADORES	***Muy alto***	***Alto***	***Medio***	***Bajo***	***Muy Bajo***
Indicador 1: Horas de experiencia demostrable (vida laboral, cursos, etc.) del programador en los lenguajes del módulo API	< 20	< 50	< 100	< 150	< 200
Indicador 2: Horas de experiencia demostrable (vida laboral, cursos, etc.) del programador en las tecnologías del módulo API	< 20	< 50	< 100	< 150	< 200
Indicador 3: Nivel de detalle y fecha de actualización de la documentación del API de las bibliotecas	> 6 meses desde la última actualización	> 4 meses desde la última actualización	> 2 meses desde la última actualización	> 1 mes desde la última actualización	< 1 mes desde la última actualización
Indicador 4: Horas de retraso en la ejecución del plan de formación	> 20	> 15	> 10	> 5	> 2

El riesgo se valorará como el más alto valor de cualquiera de los indicadores.

Indicadores (Descripción de los indicadores, del modo de evaluación y de la consolidación de indicadores si existe)**:**

Indicador 1: Horas de experiencia demostrable (vida laboral, cursos, etc.) del programador en los lenguajes del módulo API	**Evaluación:** Se evalúa al inicio del proyecto, al consolidar el equipo de trabajo
Indicador 2: Horas de experiencia demostrable (vida laboral, cursos, etc.) del programador en las tecnologías del módulo API	**Evaluación:** Se evalúa al inicio del proyecto, al consolidar el equipo de trabajo
Indicador 3: Nivel de detalle y fecha de actualización de la documentación del API de las bibliotecas	**Evaluación:** Se evalúa en el transcurso del estudio tecnológico, por parte del personal asignado

- Existencia de documentación online (wiki, sitio web propio, etc.) de la biblioteca y número de páginas de documentación dedicadas - Tiempo transcurrido desde la última actualización de la documentación	
Indicador 4: Horas de retraso en la ejecución del plan de formación	**Evaluación:** Se evalúa una vez finalizado el plan de formación

A2.2 Hoja de riesgos II

<table>
<tr><td>ID:
2</td><td colspan="5">Nombre:
Errores o "bugs" en las bibliotecas de código de terceros empleadas en el desarrollo del módulo de validación</td></tr>
<tr><td colspan="6">Descripción:
El módulo de validación de streams se desarrolla desde cero e implementa dependencias de código de terceros. Es posible que este código, desarrollado de forma externa al proyecto, tenga limitaciones o fallos que lastren al mismo.</td></tr>
<tr><td colspan="6">Categoría(s) de riesgo:
Tecnología (técnico), Calidad (técnico), Estimación (gestión de proyecto)</td></tr>
<tr><td>Estado:
Activo</td><td colspan="5">Causas del Riesgo (Identificación de los factores del riesgo):
- Uso de bibliotecas de código mantenidas de forma colaborativa por la comunidad.
- Uso de bibliotecas de código obsoletas que han quedado sin mantenimiento.
- Deuda técnica heredada por dependencias de las bibliotecas de código empleadas.</td></tr>
<tr><td rowspan="2">Probabilidad</td><td colspan="4">Impacto</td><td rowspan="2">Impacto Total</td><td rowspan="2">Respuestas</td></tr>
<tr><td>Presupuesto</td><td>Planificación</td><td>Alcance</td><td>Calidad</td></tr>
<tr><td>Media</td><td>Medio</td><td>Alto</td><td>Muy alto</td><td>Muy alto</td><td>0,45</td><td>Mitigar.
Se contará con un consultor de tecnologías con experiencia previa en el campo que decidirá, antes de iniciarse el desarrollo, qué bibliotecas de código se utilizarán atendiendo a los antecedentes disponibles. En dichas decisiones se priorizará el software open-source</td></tr>
</table>

						y se exigirá el beneplácito del analista y el arquitecto del proyecto.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
28/01/2022	Bajo	Medio	Medio	Alto	0,28	- Llegados a este punto del desarrollo, los estudios iniciales del proyecto han concluido y ya se ha seleccionado la alternativa tecnológica y arquitectura finales del proyecto. Empleando las reservas temporales asignadas para el estudio tecnológico, los perfiles designados han encontrado alternativas tecnológicas: o Fiables para la manipulación de streams y el montaje del servidor y cliente web. o En fase de pruebas o en desarrollo para la manipulación y validación de RDF (entorno de investigación). - Se mantiene la planificación en base a los resultados del estudio inicial.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
07/03/2022	Muy bajo	Muy bajo	Muy bajo	Muy bajo	0,03	- En este punto, el módulo de validación de streams ha sido completado con éxito, en línea con la

						planificación. Se han dado errores menores en la tecnología de validación con Shape Expressions escogida, pero han sido solventados y los responsables del código avisados dentro de las reservas de tiempo asignadas. - Se descarta la existencia de errores relevantes en las dependencias del módulo, así como su impacto en el proyecto ahora que está finalizado.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
08/04/2022	Muy bajo	Muy bajo	Muy bajo	Muy bajo	0,03	- El módulo validador está desarrollado y no hay incidencias.

Riesgos derivados de éste:

- Retrasos difíciles de estimar en la fase de construcción del sistema que atrasen la fecha de finalización del proyecto.
- Incremento del presupuesto de desarrollo por las horas de trabajo y el esfuerzo adicional del programador y otros perfiles técnicos.
- Reducción moderada o acusada de la calidad del proyecto final, pudiendo llegar a recortarse el alcance de este.

Riesgo residual:

Retrasos en el desarrollo del módulo validador y, por ende, de la fecha de entrega del proyecto. Reducción del nivel de satisfacción del cliente y posible degradación de la calidad del producto final en función del rendimiento y errores identificados en el entorno tecnológico seleccionado.

Plan de Contingencia:	**Presupuesto para contingencias:**
1. Hacer un estudio de tecnologías procesamiento de streams como parte de los estudios iniciales del proyecto y previo al desarrollo de los módulos. Si se admite una tecnología dentro del *stack* tecnológico del proyecto está vendrá acompañada de: a. Documentación que contenga como mínimo: i. Precio de la tecnología ii. Licencia de uso de la tecnología iii. Listado de alternativas iv. Justificación de su uso frente a las alternativas b. El beneplácito del jefe de proyecto, el arquitecto de software u otro personal especializado cuando sea pertinente. 2. Contratar personal que tenga experiencia suficiente en el dominio de investigación como consultor de tecnologías para el proyecto. La búsqueda y selección de tecnologías debe seguir el siguiente criterio: 1. Buscar bibliotecas compatibles con los lenguajes de programación y entorno tecnológico escogidos para el proyecto. 2. Priorizar la selección de bibliotecas de código abierto, activas (últimos *commits*, actualización del sitio web, etc.) y con una comunidad y antecedentes que avalen su calidad. 3. Valorar la viabilidad de modificar, mejorar o corregir el código fuente de las alternativas en caso de aparición de errores o comportamientos que no se ajusten a los casos de uso del proyecto. 4. En caso de ejecutar los pasos anteriores y no valorarse ninguna alternativa viable, recurrir a opciones de pago.	350,00 € (estudio tecnológico procesamiento streams) Total: 350,00 € **Planificación temporal de las contingencias:** El estudio tecnológico deberá planearse y ejecutarse como parte de la fase de estudios iniciales del proyecto, antes de la fase de análisis. Las hipotéticas correcciones en la selección de tecnologías se decidirán en las auditorías de seguimiento y se nutrirán de las reservas de tiempo y dinero reservadas para contingencias. Así mismo, la corrección de problemas derivados de errores en el código de terceros se realizará dentro de las reservas temporales destinadas a la construcción del sistema.

Comentarios:

N/A

Monitorización (Descripción de la relación de los factores de riesgo y los indicadores y descripción de los planes de monitorización)**:**

Se han identificado los siguientes indicadores asociados a los factores de riesgo:

	Valor del riesgo				
INDICADORES	***Muy alto***	***Alto***	***Medio***	***Bajo***	***Muy Bajo***
Indicador 1: Número de versiones disponibles de la biblioteca	1	< 5	< 10	< 15	< 30
Indicador 2: Número de incidencias (*issues*) activas en el sistema gestor de versiones en que se aloja la biblioteca	> 50	> 30	> 15	> 10	> 5
Indicador 3: Última fecha de modificación (*commit*) del código de la biblioteca	> 6 meses desde la última actualización	> 4 meses desde la última actualización	> 2 meses desde la última actualización	> 1 mes desde la última actualización	< 1 mes desde la última actualización
Indicador 4: Horas destinadas a la corrección de fallos en el código de terceros	> 20	> 15	> 10	> 5	> 2

El riesgo se valorará como el más alto valor de cualquiera de los indicadores.

Indicadores (Descripción de los indicadores, del modo de evaluación y de la consolidación de indicadores si existe)**:**

Indicador 1: Número de versiones disponibles de la biblioteca - Número de versiones (mayores o menores) numeradas de la biblioteca - Un mayor número de versiones refleja la madurez del proyecto	**Evaluación:** Se evalúa en el transcurso del estudio tecnológico, por parte del personal asignado
Indicador 2: Número de incidencias (*issues*) activas en el sistema gestor de versiones en que se aloja la biblioteca - El número de incidencias refleja la cantidad y calidad de la comunidad del proyecto	**Evaluación:** Se evalúa en el transcurso del estudio tecnológico, por parte del personal asignado

- Contrastar la vida media de las incidencias y el número de incidencias cerradas frente a las abiertas	
Indicador 3: Última fecha de modificación (*commit*) del código de la biblioteca - Tiempo transcurrido desde la última actualización del código de la biblioteca	**Evaluación:** Se evalúa en el transcurso del estudio tecnológico, por parte del personal asignado
Indicador 4: Horas destinadas a la corrección de fallos en el código de terceros	**Evaluación:** Se evalúa una vez finalizada la construcción del módulo validador

A2.3 Hoja de riesgos III

<table>
<tr><td>ID:
7</td><td colspan="5">Nombre:
Verificar el funcionamiento del módulo validador de streams supone una complejidad añadida</td></tr>
<tr><td colspan="6">Descripción:
Para verificar el funcionamiento del módulo de validación de streams se requiere de un stream de datos de prueba, implementado en la tecnología que se escoja finalmente y moldeable para realizar pruebas. Esto implica un tiempo de desarrollo y de testing añadido.</td></tr>
<tr><td colspan="6">Categoría(s) de riesgo:
Tecnología (técnico), Complejidad e Interfaces (técnico)</td></tr>
<tr><td>Estado:
Activo</td><td colspan="5">Causas del Riesgo (Identificación de los factores del riesgo):
- No disponer de un mecanismo para regular las características (contenido, periodicidad, origen, etc.) del flujo de datos utilizado para las pruebas del módulo validador.
- Dificultad del despliegue y comunicación con un flujo de datos para las pruebas del módulo validador.</td></tr>
<tr><td rowspan="2">Probabilidad</td><td colspan="4">Impacto</td><td rowspan="2">Impacto Total</td><td rowspan="2">Respuestas</td></tr>
<tr><td>Presupuesto</td><td>Planificación</td><td>Alcance</td><td>Calidad</td></tr>
<tr><td>Alta</td><td>Medio</td><td>Muy alto</td><td>Muy bajo</td><td>Bajo</td><td>0,63</td><td>Aceptar.
Para comprobar el funcionamiento de este módulo es necesario hacer pruebas de integración contra un stream de datos externo. Se destinarán reservas de tiempo al desarrollo de un stream de datos manipulable de forma programática y que el tester pueda tener en cuenta en el diseño del plan de pruebas.</td></tr>
</table>

Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		

28/01/2022	Bajo	Alto	Muy bajo	Bajo	0,39	- Todavía no se ha comenzado la construcción de los módulos del sistema.
						- Por requisitos del cliente, se decide utilizar Apache Kafka como la alternativa tecnológica para el manejo de streams.
						- Dado que Apache Kafka requiere de una complejidad añadida por su uso y despliegue junto con Apache Zookeeper, se decide lo siguiente:
						  ○ Se creará una solución propia para el despliegue de stream de pruebas, utilizando la tecnología de contenedores *Docker* para facilitar el despliegue.
						  ○ Se permitirá la modificación de las características del stream en el momento de la construcción de las imágenes *Docker* y con variables de entorno en los contenedores.
						- Se engrosan las reservas temporales destinadas a crear el stream de pruebas.

Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		

07/03/2022	Bajo	Bajo	Muy bajo	Muy bajo	0,11	- En este punto, el stream de pruebas ha sido completado con éxito, en línea con las modificaciones en objetivos y planificación de la primera revisión.

Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
08/04/2022	Muy bajo	Muy bajo	Muy bajo	Muy bajo	0,04	- El stream de pruebas de ha empleado para los tests del módulo validador sin incidencias.

Riesgos derivados de éste:

- Retrasos difíciles de estimar en la fase de construcción del sistema que atrasen la fecha de finalización del proyecto.
- Incremento del presupuesto de desarrollo por las horas de trabajo y el esfuerzo adicional de los expertos en tecnologías, el programador y otros perfiles técnicos.

Riesgo residual:

Oportunidad de ahorrar tiempo en la fase de pruebas y obtener un mayor nivel de satisfacción del cliente con la metodología de pruebas, su facilidad de despliegue y su reusabilidad.

Retrasos en las pruebas del módulo validador y, por ende, de la fecha de entrega del proyecto.

Plan de Contingencia:	**Presupuesto para contingencias:**
1. Destinar reservas temporales y monetarias adicionales en la fase de construcción del sistema, que sirvan para el desarrollo de los mecanismos de pruebas de integración del módulo validador.	608,00 € (desarrollo del stream de pruebas) Total: 608,00 €

	Planificación temporal de las contingencias: Las hipotéticas correcciones en el desarrollo del stream de pruebas se ejecutarán en la fase de construcción del sistema y se nutrirán de las reservas de dinero reservadas para contingencias.

Comentarios:

N/A

Monitorización (Descripción de la relación de los factores de riesgo y los indicadores y descripción de los planes de monitorización)**:**

Se han identificado los siguientes indicadores asociados a los factores de riesgo:

	Valor del riesgo				
INDICADORES	***Muy alto***	***Alto***	***Medio***	***Bajo***	***Muy Bajo***
Indicador 1: Número de escenarios de prueba desarrollados por el tester	> 10	> 8	> 4	> 1	1
Indicador 2: Horas de retraso en el montaje y despliegue del flujo de datos de pruebas	> 20	> 15	> 10	> 8	> 5

El riesgo se valorará como el más alto valor de cualquiera de los indicadores.

Indicadores (Descripción de los indicadores, del modo de evaluación y de la consolidación de indicadores si existe)**:**

Indicador 1: Número de escenarios de prueba desarrollados por el tester - El número de escenarios de prueba tenidos en cuenta refleja la necesidad de crear un flujo de datos fácilmente modificable	**Evaluación:** Se evalúa en la fase de análisis, una vez se conoce el entorno tecnológico y el plan de pruebas inicial
Indicador 2: Horas de retraso en el montaje y despliegue del flujo de datos de pruebas	**Evaluación:** Se evalúa en el transcurso del estudio tecnológico, por parte del personal asignado

A2.4 Hoja de riesgos IV

ID:	**Nombre:**
11	Demoras en el despliegue del sistema por la complejidad de integrar 3 módulos

Descripción:

El proyecto por desarrollar es multimodular e implica la comunicación entre tres módulos que se ejecutan independientemente: Cliente, API y biblioteca de validación. Además, los streams validados por el servidor provendrán, mayoritariamente, del exterior. Facilitar la correcta comunicación entre módulos y de estos con el exterior una vez desplegados puede causar complicaciones.

Categoría(s) de riesgo:

Prestaciones y Fiabilidad (técnico), Estimación (gestión de proyecto)

Estado:

Activo

Causas del Riesgo (Identificación de los factores del riesgo)**:**

- Dificultades en la comunicación entre los módulos API, Cliente y un stream de datos externo.
- El desarrollador carece de control sobre la infraestructura de red en que se despliega el producto.
- Complicaciones en el despliegue en los entornos de pruebas y producción del cliente.

Probabilidad	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
Alta	Alto	Muy alto	Muy bajo	Bajo	0,63	**Evitar**. Se contará con un arquitecto de software y con un ingeniero de sistemas dentro del equipo de desarrollo para elaborar la arquitectura del sistema y apoyar al programador en el desarrollo, despliegue y creación de manuales. Aparte, se formará al programador en el entorno de despliegue del cliente y se dedicarán

						reservas de tiempo a generar imágenes de cada módulo compatibles con tecnología de contenedores como *Docker*, para conseguir un despliegue sencillo y repetible.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
28/01/2022	Medio	Alto	Muy bajo	Bajo	0,39	- En este punto, ni la codificación de los módulos ni la elaboración de planes de despliegue ha comenzado. - Tras los estudios iniciales, se decide emplear *Docker* como tecnología de contenedores, así como *docker-compose* y un *reverse-proxy NGINX* para el despliegue coordinado y organizado de estos. - Se mantienen las reservas temporales asignadas en la planificación inicial para la construcción de cada módulo, lo cual incluye tareas para crear imágenes Docker de API y cliente. - Tras decidirse las tecnologías de despliegue y reevaluar las reservas de tiempo, se reduce el impacto sobre la planificación.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		

07/03/2022	Medio	Medio	Muy bajo	Muy bajo	0,21	- Todos los módulos han sido codificados según la planificación y se hallan en período de pruebas. - El día 4 de abril se produce una caída en un entorno de pruebas donde se habían desplegado los módulos. El fallo materializa el riesgo 10 del proyecto y revela varios apartados a mejorar en los despliegues actuales del sistema. Se decide: o Ampliar el tiempo dedicado a la generación de imágenes, para poder automatizar la renovación de certificados SSL de los contenedores. o Ampliar el tiempo dedicado a la elaboración de manuales de ejecución y despliegue. o **Puede consultarse el informe *post mortem* del incidente en el anexo *A3-Informe "Post Mortem" RDFShape*.**
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
08/04/2022	Muy bajo	Muy bajo	Muy bajo	Muy bajo	0,04	- Se han realizado las acciones correctivas especificadas en la anterior revisión. - Las pruebas de integración entre módulos se completan con éxito, desestimando el riesgo.
Riesgos derivados de éste:						

- Retrasos en la fase de construcción del sistema (creación de las imágenes/contenedores de los módulos, retraso en los despliegues de pruebas y tests) que atrasen la fecha de finalización del proyecto.
- Incremento del presupuesto de desarrollo por las horas de trabajo y el esfuerzo adicional del programador y el ingeniero de sistemas.

Riesgo residual:

Oportunidad de ahorrar tiempo en la fase de implantación y obtener un mayor nivel de satisfacción del cliente al proveer un mecanismo de despliegue robusto y reutilizable.

Retrasos en los tests y despliegue de los módulos y, por ende, de la fecha de entrega del proyecto. Reducción del nivel de satisfacción del cliente por las demoras.

Plan de Contingencia:

1. Contratar personal que tenga experiencia suficiente en el ámbito de sistemas y despliegues como ingeniero de sistemas del proyecto, que apoyará al programador en la codificación y despliegue de los módulos.
2. Asignar reservas temporales y presupuestarias holgadas dentro de la fase de construcción para elaborar imágenes desplegables como contenedores de los módulos API y cliente.
3. Asignar reservar temporales y presupuestarias dentro de la fase de implantación del sistema para verificar el despliegue del sistema y tomar medidas para automatizar su mantenimiento.

Presupuesto para contingencias:

584,00 € (imágenes desplegables back-end)
518,00 € (imágenes desplegables front-end)
168,00 € (pruebas de implantación)
448,00 € (mantenimiento despliegue)

Total: 1.718,00 €

Planificación temporal de las contingencias:

Las reservas temporales para la construcción de implantación de un entorno de contenedores se asignarán desde la planificación inicial.

Las hipotéticas correcciones de programación o despliegue se llevarán a cabo dentro de sus fases correspondientes y se nutrirán de las reservas de tiempo y dinero reservadas para contingencias.

Comentarios:

N/A

Monitorización (Descripción de la relación de los factores de riesgo y los indicadores y descripción de los planes de monitorización)**:**

Se han identificado los siguientes indicadores asociados a los factores de riesgo:

	Valor del riesgo				
INDICADORES	***Muy alto***	***Alto***	***Medio***	***Bajo***	***Muy Bajo***
Indicador 1: Porcentaje de tests de integración fallidos	> 80%	> 60%	> 40	> 20	0%
Indicador 2: Minutos empleados en el despliegue de la aplicación	> 90	> 60	> 30	> 20	> 10
Indicador 3: Horas de retraso en la fase de implantación del sistema	> 20	> 15	> 10	> 5	> 2

El riesgo se valorará como el más alto valor de cualquiera de los indicadores.

Indicadores (Descripción de los indicadores, del modo de evaluación y de la consolidación de indicadores si existe)**:**

Indicador 1: Porcentaje de tests de integración fallidos - Dado que contamos con un plan de pruebas específico para nuestro proyecto, documentar los tests de integración fallidos nos puede indicar los problemas que tenemos para comunicar los módulos.	**Evaluación:** Se evalúa tras la ejecución de los tests de integración en la fase de pruebas, dentro de la fase de construcción del sistema
Indicador 2: Minutos empleados en el despliegue de la aplicación - Tiempo empleado por el personal del proyecto en los despliegues en los servidores del cliente. - Los despliegues son efectuados por el ingeniero de sistemas, el más cualificado y documentado sobre el proceso de despliegue, junto al programador.	**Evaluación:** Se evalúa a partir de la fase de construcción del sistema, una vez se tienen los módulos codificados y se despliegan para pruebas de integración

- Un mayor tiempo de despliegue, incluso cuando es llevado a cabo por los propios desarrolladores, indica complejidad o dificultad para adaptar el proceso a una infraestructura cambiante.	
Indicador 3: Horas de retraso en la fase de implantación del sistema - Retrasos en la elaboración de los manuales y el proceso de despliegue y *securización* final	**Evaluación:** Se evalúa en la fase de implantación y aceptación, una vez finalizado del despliegue en producción del sistema

A2.5 Hoja de riesgos V

ID: 15	**Nombre:** Retrasos en la planificación del proyecto por causas personales y/o sanitarias
Descripción: Si cualquiera de los perfiles del proyecto queda temporalmente inhabilitado por motivos personales o de salud (estado de pandemia), puede retrasar la finalización de este.	
Categoría(s) de riesgo: Control (gestión de proyecto), Planificación (gestión de proyecto)	
Estado: Activo	**Causas del Riesgo** (Identificación de los factores del riesgo)**:** - Situaciones en el ámbito familiar/personal que resten productividad o incapaciten a los perfiles del proyecto. - Situaciones sanitarias que resten productividad o incapaciten a los perfiles del proyecto. - Accidentes laborales.

Probabilidad	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
Baja	Medio	Alto	Muy bajo	Muy bajo	0,17	**Aceptar**. A la hora de planificar el proyecto, se partirá de horarios de trabajo flexibles para los trabajadores, que incluyan suficientes horas de descanso al día, posibilidad de teletrabajo en caso de necesidad y libertad en los fines de semana. Además, las estimaciones de tiempo iniciales serán holgadas en la medida de lo posible para poder compensar dichos contratiempos. Más allá de estas

						medidas estándar, no se pueden predecir ni prevenir estos incidentes.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
28/01/2022	Medio	Medio	Muy bajo	Muy bajo	0,09	- A lo largo del mes de noviembre, varios perfiles del proyecto notificaron encontrarse en un proceso de mudanza, lo que podría restarles tiempo de trabajo productivo. - Gracias a las medidas de horario flexible y/o teletrabajo, no es necesario reestimar ninguna tarea y se comprueba que los procesos de mudanza no han producido desviaciones en la planificación inicial. - Se reduce el potencial impacto en el proyecto, que ya ha avanzado parcialmente.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
07/03/2022	Medio	Medio	Muy bajo	Muy bajo	0,21	- Los perfiles del trabajo contraen el COVID, perdiéndose así dos jornadas de trabajo productivo. - Se reajusta el cronograma, retrasando toda la actividad del proyecto en dos jornadas laborales. Este cambio se refleja en la planificación del

						proyecto añadiendo una excepción al calendario de trabajo habitual de los perfiles.
Probabilidad revisada Fecha	**Impacto**				**Impacto Total**	**Respuestas**
	Presupuesto	**Planificación**	**Alcance**	**Calidad**		
08/04/2022	Bajo	Bajo	Muy bajo	Muy bajo	0,05	- No se registran más incidentes y el proyecto avanza según lo planeado en lo que al riesgo respecta. - El impacto potencial del riesgo es bajo a estas alturas del desarrollo.

Riesgos derivados de éste:

- Retrasos impredecibles en cualquiera de las fases del proyecto que atrasen la fecha de finalización del proyecto.

Riesgo residual:

Retrasos en la fecha de entrega del proyecto. Reducción del nivel de satisfacción del cliente por las demoras.

Plan de Contingencia:

1. Este riesgo contempla factores impredecibles y difíciles de estimar, por ello la política frente al riesgo es aceptarlo.
2. De cualquier modo, se pone el mayor número de facilidades a los trabajadores con el objetivo de minimizar el impacto.
3. En el caso de darse demoras por estos riesgos, se retrasará la planificación según corresponda.

Presupuesto para contingencias:

Total: 0,00 €

Planificación temporal de las contingencias:

Las hipotéticas correcciones de planificación se llevarán a cabo cuando se dé la incidencia.

Comentarios:

Si bien el riesgo presente no es de mayor gravedad, se considera hacerle un seguimiento exhaustivo dada la situación sanitaria global por la enfermedad COVID-19.

Monitorización (Descripción de la relación de los factores de riesgo y los indicadores y descripción de los planes de monitorización)**:**

Se han identificado los siguientes indicadores asociados a los factores de riesgo:

	Valor del riesgo				
INDICADORES	***Muy alto***	***Alto***	***Medio***	***Bajo***	***Muy Bajo***
Indicador 1: Horas productivas consumidas por situaciones de ámbito personal y familiar	> 30	> 20	> 15	> 10	> 5
Indicador 2: Horas productivas consumidas por situaciones sanitarias	> 30	> 20	> 15	> 10	> 5
Indicador 3: Horas productivas consumidas por accidente laboral	> 30	> 20	> 15	> 10	> 5

El riesgo se valorará como el más alto valor de cualquiera de los indicadores.

Indicadores (Descripción de los indicadores, del modo de evaluación y de la consolidación de indicadores si existe)**:**

Indicador 1: Horas productivas consumidas por situaciones de ámbito personal y familiar	**Evaluación:** Se evalúa cuando se pierden horas de trabajo productivas por asuntos personales y familiares
Indicador 2: Horas productivas consumidas por situaciones sanitarias	**Evaluación:** Se evalúa cuando se pierden horas de trabajo productivas por asuntos de salud
Indicador 3: Horas productivas consumidas por accidente laboral	**Evaluación:** Se evalúa cuando se pierden horas de trabajo productivas por motivo de accidente laboral

A3. INFORME "POST MORTEM" RDFSHAPE

A3.1 Descripción

El día 4 de abril de 2022 se registra una caída generalizada de los servicios desplegados en los servidores de prueba, notificándose inaccesibles por parte del cliente.

A lo largo del mismo día, se llevan a cabo las tareas correctivas correspondientes y se elabora el siguiente informe *post mortem* a fin de:

- Documentar y registrar de forma rigurosa las causas de la incidencia.
- Utilizar la situación a modo de informe de lecciones aprendidas y aplicables en el plan de implantación final del sistema.

A3.2 Contenidos

A continuación, se hallan transcritos los contenidos del informe elaborado por el programador del proyecto.

A.3.2.1 Caída generalizada de RDFShape (API y clientes) por caída del servidor Nginx (+ error de certificados)

A3.2.1.1 Fecha

07/04/2022 18:15 CEST (hora aproximada)

A3.2.1.2 Autores

Eduardo Ulibarri Toledo

A3.2.1.3 Estado

- Dominios en servicios.
- Certificados SSL actualizados y funcionando.
- HSTS desactivado: los dominios son accesibles mediante HTTP como *fallback* en caso de futuros errores de certificados.

A3.2.1.4 Resumen

A última hora de la tarde se recibe notificación de que todas las aplicaciones desplegadas en la máquina RDFShape (*156.35.94.30*) son inaccesibles: no se puede establecer una conexión segura pues el certificado se detecta como auto-firmado. Se juntan varios factores que dificultan identificar el problema:

1. Acceder a las páginas mediante HTTP o aceptando la advertencia del navegador NO es posible, puesto que HSTS está activado en el servidor Nginx que distribuye cada petición a cada aplicación.
2. Acceder directamente a la IP de la máquina mediante el navegador muestra la página de error por defecto de Nginx: "*503 Service Temporarily Unavailable*". Por tanto, no sólo los certificados fallan sino el propio servidor.
3. Debido a los recientes cambios en políticas de ciberseguridad de la Universidad de Oviedo por la guerra en Ucrania, el servidor sólo está disponible para su acceso a través de la VPN de la universidad.

Tras acceder al servidor mediante SSH se toman los siguientes pasos:

1. **Desactivar HSTS para intentar conectar a las aplicaciones por HTTP**
    a. Con esto se alcanza, de nuevo, la página de error de Nginx, confirmando que el punto de entrada a las aplicaciones está caído.
    b. HSTS se desactiva cambiando las variables de entorno del contenedor nginx-proxy: *HSTS=off*
    c. Aún sin HSTS, los navegadores siguen redireccionando a HTTPS todo lo posible: *HTTPS_METHOD=redirect*
2. **Lanzar todo el stack de aplicaciones en modo NO detached (-d), para observar el output**:
    a. docker-compose up
    b. Se observa que el contenedor encargado de los certificados hace algunos reajustes, tras lo que el contenedor nginx procede a relanzar Nginx: *nginx -s reload*, **que falla con código de error 1**.
    c. Se ejecuta bash directamente dentro del contenedor Nginx y se ejecuta *nginx -s reload* para ver más información del fallo. El error es que hay rutas duplicadas definidas en */etc/nginx/vhost.d/default*.
    d. Tras eliminar los duplicados, Nginx deja de fallar al relanzarse y la aplicación funciona con normalidad
        i. Contenidos de */etc/nginx/vhost.d/default* (esta sección se encontraba duplicada 4+ veces):

```
## Start of configuration add by letsencrypt container
location ^~ /.well-known/acme-challenge/ {
    auth_basic off;
    auth_request off;
    allow all;
    root /usr/share/nginx/html;
    try_files $uri =404;
    break;
}
## End of configuration add by letsencrypt container
```

Una vez eliminados los duplicados, se relanza Nginx, el cual ya funciona con normalidad. Confirmado esto, se relanzan los contenedores con normalidad y se recupera el servicio. Se recomienda limpiar la caché del navegador si el acceso a la página sigue restringido.

A3.2.1.5 Impacto

Total, imposible acceso a todas las aplicaciones detrás del proxy Nginx.

A3.2.1.6 Antecedentes y causas

Fallo inesperado en la compatibilidad entre el contenedor Docker con el proxy nginx (jwilder/nginx-proxy) y su *companion* encargado de administrar los certificados SSL (nginxproxy/acme-companion).

El *companion*, posiblemente en un intento de renovar los certificados introdujo un duplicado en las rutas de Nginx, causando que este errase al lanzarse.

Para más inri, los dos contenedores utilizan un volumen persistente *vhost:/etc/nginx/vhost.d* para compartir información, por tanto, sólo con lanzar los contenedores el fallo no se arreglaba pues el fichero corrupto no se sobrescribía.

A3.2.1.7 Causante

No identificado con total certeza, diversas hipótesis:

- Cambios recientes en las infraestructuras de la universidad de Oviedo:
  - Los registros muestran que la máquina sufrió un reinicio horas antes de descubrir la incidencia:
    - `reboot system boot 5.4.0-107-generi Wed Apr 6 17:22 - 17:22 (00:00)`
  - La máquina ha pasado a ser sólo accesible mediante VPN y el firewall `ufw` se hallaba desactivado

- Posible fecha cercana a la renovación de certificados.

A3.2.1.8 Solución

- (ver resumen)
- Actualizar las imágenes Docker de Nginx y su companion a las últimas versiones compatibles.
- Relanzar el firewall y activar que se lance por defecto cuando se inicia el sistema.

A3.2.1.9 Detección

Los usuarios avisaron de que la web no era accesible.

A3.2.1.10 Lecciones aprendidas

- Revisar más exhaustivamente las máquinas cuando se conoce que se están haciendo cambios en infraestructuras.
- Mantener los clientes y credenciales VPN de todos actualizados para poder conectarse a la máquina cuanto antes.
- Dificultar el acceso HTTP a los servicios mediante redirecciones a HTTPS, pero no impedirlo directamente mediante HSTS, puesto que los servicios quedan del todo inaccesibles ante cualquier fallo de certificados.

A4. CONTENIDO ENTREGADO

A4.1 Contenidos

Junto al documento principal del trabajo de finde máster, se ha entregado un archivo comprimido de nombre **anexos.zip** que, en su interior, alberga la siguiente estructura de contenidos:

Directorio	Contenido
./(directorio raíz)	Contiene un fichero `LEEME.TXT` explicando esta estructura de carpetas.
./validacion_flujos_rdf	Contiene el código fuente de cada uno de los módulos.
./validacion_flujos_rdf/comet	Código y documentación asociada del módulo de validación de flujos.
./validacion_flujos_rdf/api	Código y documentación asociada del módulo API.
./validacion_flujos_rdf/cliente	Código y documentación asociada del módulo Cliente.
./instalacion	Contiene el archivo e instrucciones necesarias para arrancar el sistema completo localmente mediante *docker-compose* sin necesidad de instalar o compilar los módulos del proyecto.
./documentacion	Contiene toda la documentación asociada al desarrollo del proyecto. Ver *Estructura de Directorios de "documentacion"* para más información.

Tabla 105. Estructura general de los anexos entregados

Estructura de Directorios de "documentacion"

La carpeta ***documentacion*** contiene los documentos complementarios elaborados durante el desarrollo del proyecto:

Directorio	Contenido
./gestion_proyectos	Documentos creados y mencionados para el capítulo de gestión de proyectos (ver *4.3-Planificación y Gestión del TFM*).
./gestion_proyectos/planificacion	Contiene los cronogramas del proyecto para cada una de las etapas definidas (*Microsoft Project*).
./gestion_proyectos/presupuesto	Contiene los presupuestos del proyecto para cada una de las etapas definidas (*Microsoft Excel*).

./gestion_proyectos/riesgos	Contiene el registro de riesgos del proyecto (*Microsoft Excel*).
./media	Archivos multimedia variados con los diagramas y bocetos proyecto.
./media/diagramas	Contiene los diagramas elaborados para el proyecto en formato imagen (PNG) y UMLet (UXF), subdivididos por tipo de diagrama y etapa del desarrollo.
./media/interfaces	Contiene los bocetos de las interfaces de usuario en formato imagen (PNG).
./pruebas	Ficheros con los resultados de las pruebas del sistema.
./pruebas/rendimiento	Resultados de las pruebas de rendimiento del sistema (*Microsoft Excel*).

Tabla 106. Estructura de la carpeta "documentacion" de los anexos

A5. REFERENCIAS BIBLIOGRÁFICAS

[1] J. M. Redondo (17 de 6 de 2019). *Documentos-modelo para Trabajos de Fin de Grado/Máster de la Escuela de Informática de Oviedo*. Disponible: https://www.researchgate.net/publication/327882831_Plantilla_de_Proyectos_de_Fin_de_Carrera_de_la_Escuela_de_Informatica_de_Oviedo.

[2] Gobierno de España (5 de 11 de 2021). *PAe – Métrica v.3*. Disponible: https://administracionelectronica.gob.es/pae_Home/pae_Documentacion/pae_Metodolog/pae_Metrica_v3.html.

[3] W3C (5 de 11 de 2021). *RDF – Semantic Web Standards*. Disponible: https://www.w3.org/RDF/.

[4] ShEx Community Group (5 de 11 de 2021). *ShEx – Shape Expressions*. Disponible: https://shex.io/.

[5] Wikipedia (5 de 11 de 2021). *Stream (computing)*. Disponible: https://en.wikipedia.org/wiki/Stream_(computing)

[6] Mozilla Developer Network, Web Docs (18 de 3 de 2022). *The WebSocket API (WebSockets)*. Disponible: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.

[7] Apache Software Foundation (5 de 11 de 2021). *Apache Kafka*. Disponible: https://kafka.apache.org/.

[8] AMQP – Advanced Message Queueing Protocol. *The internet protocol for business messaging*. Disponible: https://www.amqp.org/about/what.

[9] WESO Research Group (5 de 11 de 2021). *WESO webpage*. Disponible: https://www.weso.es/.

[10] WESO Research Group (5 de 11 de 2021). *RDFShape API*. Disponible: https://www.weso.es/rdfshape-api/.

[11] WESO Research Group (5 de 11 de 2021). *RDFShape – Playground for RDF, ShEx, SHACL and more*. Disponible: https://rdfshape.weso.es/.

[12] W3C (5 de 11 de 2021). *Shapes Constraint Language (SHACL)*. Disponible: https://www.w3.org/TR/shacl/.

[13] Wikidata (5 de 11 de 2021). *Wikidata (home)*. Disponible: https://www.wikidata.org/wiki/Wikidata:Main_Page.

[14] Wikipedia (5 de 11 de 2021). *Recent Changes*. Disponible: https://en.wikipedia.org/wiki/Special:RecentChanges.

[15] Mozilla Developer Network, Web Docs (6 de 11 de 2021). *Server-sent events – Web APIs*. Disponible: https://developer.mozilla.org/en-US/docs/Web/API/Server-sent_events.

[16] Swagger (6 de 11 de 2021). *OpenAPI Specification*. Disponible: https://swagger.io/specification/.

[17] SwaggerHub (6 de 11 de 2021). *SwaggerHub – API design and documentation with OpenAPI*. Disponible: https://swagger.io/tools/swaggerhub/.

[18] SwaggerHub (6 de 11 de 2021). *RDFShape | weso*. Disponible: https://app.swaggerhub.com/apis/weso/RDFShape.

[19] IBM (6 de 11 de 2021). *What are containers?* Disponible: https://www.ibm.com/cloud/learn/containers.

[20] W3C (6 de 11 de 2021). *Web Content Accessibility Guidelines (WCAG 2) – Level AA Conformance*. Disponible: https://www.w3.org/WAI/WCAG2AA-Conformance.

[21] Mozilla Developer Network, Web Docs (6 de 11 de 2021). *HTTPS*. Disponible: https://developer.mozilla.org/en-US/docs/Glossary/https.

[22] Wikipedia (4 de 11 de 2021). *History of the World Wide Web*. Disponible: https://en.wikipedia.org/wiki/History_of_the_World_Wide_Web.

[23] W3C (4 de 11 de 2021). *Tim Berners-Lee*. Disponible: https://www.w3.org/People/Berners-Lee/.

[24] World Wide Web Foundation (4 de 11 de 2021). *World Wide Web Foundation*. Disponible: https://webfoundation.org/.

[25] W3C (4 de 11 de 2021). *OWL – Web Ontology Language*. Disponible: https://www.w3.org/OWL/.

[26] Wikipedia (4 de 11 de 2021). *Resource Description Framework – Serialization Formats*. Disponible: https://en.wikipedia.org/wiki/Resource_Description_Framework#Serialization_formats.

[27] RDFa (4 de 11 de 2021). *RDFa*. Disponible: http://rdfa.info/.

[28] W3C (4 de 11 de 2021). *Shape Expressions Community Group*. Disponible: https://www.w3.org/community/shex/.

[29] W3C (4 de 11 de 2021). *SHACL Community Group*. Disponible: https://www.w3.org/community/shacl/.

[30] GitHub (4 de 11 de 2021). *ShEx Specification*. Disponible: https://github.com/shexSpec.

[31] ShEx Community Group (4 de 11 de 2021). *ShEx Tools and Validators*. Disponible: https://shex.io/#platforms.

[32] GitHub (4 de 11 de 2021). *Shex-s*. Disponible: https://www.weso.es/shex-s/.

[33] WESO (4 de 11 de 2021). *SHaclEX library*. Disponible: https://www.weso.es/shaclex/.

[34] GitHub (4 de 11 de 2021). *Shex.js*. Disponible: https://github.com/shexjs/shex.js.

[35] GitHub (4 de 11 de 2021). *Shex-Java*. Disponible: https://github.com/iovka/shex-java.

[36] GitHub (4 de 11 de 2021). *PyShex*. Disponible: https://github.com/hsolbrig/PyShEx.

[37] Apache Jena (4 de 11 de 2021). *Apache Jena*. Disponible: https://jena.apache.org/documentation/shex/.

[38] Apache Software Foundation (4 de 11 de 2021). *Apache Software Foundation*. Disponible: https://apache.org/.

[39] RabbitMQ (4 de 11 de 2021). *RabbitMQ - Messaging that just works*. Disponible: https://www.rabbitmq.com/.

[40] ActiveMQ (4 de 11 de 2021). *Apache ActiveMQ*. Disponible: https://activemq.apache.org/.

[41] Jean-Paul Calbimonte (4 de 11 de 2021). *Jean-Paul Calbimonte*. Disponible: https://jeanpi.org/wp/.

[42] Stream Reasoning (4 de 11 de 2021). *Triplewave – A tool to bring RDF streams on the web*. Disponible: https://streamreasoning.org/TripleWave/.

[43] GitHub (4 de 11 de 2021). *WESO – Turbine*. Disponible: https://github.com/weso/turbine.
[44] Typelevel (6 de 11 de 2021). *Cats-effect – The pure asynchronous runtime for Scala*. Disponible: https://typelevel.org/cats-effect/.
[45] ZIO (6 de 11 de 2021). *ZIO*. Disponible: https://zio.dev/.
[46] Monix (6 de 11 de 2021). *Monix Documentation*. Disponible: https://typelevel.org/cats-effect/.
[47] Fs2 (6 de 11 de 2021). *Fs2 - Functional, effectful, concurrent streams for Scala*. Disponible: https://fs2.io/#/.
[48] Lightbend – Akka (7 de 11 de 2021). *Introduction to Akka*. Disponible: https://doc.akka.io/docs/akka/current/typed/guide/introduction.html.
[49] http4s (7 de 11 de 2021). *http4s*. Disponible: https://http4s.org/.
[50] Circe (7 de 11 de 2021). *circe: Home*. Disponible: https://circe.github.io/circe/.
[51] GitHub (7 de 11 de 2021). *http4s/rho*. Disponible: https://github.com/http4s/rho.
[52] Logback (7 de 11 de 2021). *Logback – Home*. Disponible: https://logback.qos.ch/.
[53] Log4J (7 de 11 de 2021). *Apache Log4J*. Disponible: https://logging.apache.org/log4j/2.x/.
[54] Lightbend - Scala Logging (7 de 11 de 2021). Disponible: https://github.com/lightbend/scala-logging.
[55] React (7 de 11 de 2021). *React- a JS library for building user interfaces*. Disponible: https://reactjs.org/.
[56] React Bootstrap (7 de 11 de 2021). *React-Bootstrap*. Disponible: https://react-bootstrap.github.io/.
[57] Sonatype (7 de 11 de 2021). *Open Source Software Repository Hosting*. Disponible: https://central.sonatype.org/publish/publish-guide/.
[58] MVN Repository (7 de 11 de 2021). *Maven Repository: Central*. Disponible: https://mvnrepository.com/repos/central.
[59] GitHub (7 de 11 de 2021). *Packages*. Disponible: https://github.com/features/packages.
[60] Docker (7 de 11 de 2021). *Docker*. Disponible: https://www.docker.com/.
[61] Open Containers (7 de 11 de 2021). *Open Container Iniciative*. Disponible: https://opencontainers.org/.
[62] Podman (7 de 11 de 2021). *Podman*. Disponible: https://podman.io/.
[63] Docker (7 de 11 de 2021). *Docker Hub*. Disponible: https://hub.docker.com/.
[64] GitHub (7 de 11 de 2021). *Working with the container registry*. Disponible: https://docs.github.com/en/packages/working-with-a-github-packages-registry/working-with-the-container-registry.
[65] GitHub (7 de 11 de 2021). *About Wikis*. Disponible: https://docs.github.com/en/communities/documenting-your-project-with-wikis/about-wikis.
[66] Scalameta (8 de 11 de 2021). *mdoc*. Disponible: https://scalameta.org/mdoc/.
[67] Docusaurus (8 de 11 de 2021). *Docusaurus*. Disponible: https://docusaurus.io/.
[68] Scala-SBT (8 de 11 de 2021). *sbt – The interactive build tool*. Disponible: https://www.scala-sbt.org/.

[69] Jekyll LRB (8 de 11 de 2021). *Jekyll | Simple, blog-aware static sites*. Disponible: https://jekyllrb.com/.
[70] GitHub (8 de 11 de 2021). *About GitHub Pages and Jekyll*. Disponible: https://docs.github.com/en/pages/setting-up-a-github-pages-site-with-jekyll/about-github-pages-and-jekyll.
[71] Travis CI (8 de 11 de 2021). *Travis CI – Start building today!* Disponible: https://www.travis-ci.com/.
[72] GitHub (8 de 11 de 2021). *GitHub Actions*. Disponible: https://github.com/features/actions.
[73] MongoDB (8 de 11 de 2021). *MongoDB*. Disponible: https://www.mongodb.com/.
[74] MongoDB Atlas (8 de 11 2021). *MongoDB Atlas*. Disponible: https://www.mongodb.com/atlas.
[75] MySQL (8 de 11 de 2021). *MySQL*. Disponible: https://www.mysql.com/.
[76] Fs2 Kafka (8 de 11 de 2021). *Functional Kafka Streams for Scala*. Disponible: https://fd4s.github.io/fs2-kafka/.
[77] Axios (11 de 11 de 2021). *Axios*. Disponible: https://axios-http.com/.
[78] NPM JS (11 de 11 de 2021). *Prop-types*. Disponible: https://www.npmjs.com/package/prop-types.
[79] vis.js (11 de 11 de 2021). *vis.js*. Disponible: https://visjs.org/.
[80] Cytoscape.js (11 de 11 de 2021). *Cytoscape.js*. Disponible: https://js.cytoscape.org/.
[81] CodeMirror (11 de 11 de 2021). *CodeMirror*. Disponible: https://codemirror.net/.
[82] Perfectkb (11 de 11 de 2021). *YASHE*. Disponible: https://perfectkb.github.io/yate/.
[83] WESO (11 de 11 de 2021). *YASHE*. Disponible: https://www.weso.es/YASHE/.
[84] NPM JS (11 de 11 de 2021). *react-use-websocket*. Disponible: https://www.npmjs.com/package/react-use-websocket.
[85] NPM JS (11 de 11 de 2021). *react-bootstrap-table-next*. Disponible: https://www.npmjs.com/package/react-bootstrap-table-next.
[86] Docker Docs (11 de 11 de 2021). *Overview of Docker Compose*. Disponible: https://docs.docker.com/compose/.
[87] NGINX Docs (11 de 11 de 2021). *NGINX Reverse Proxy*. Disponible: https://docs.nginx.com/nginx/admin-guide/web-server/reverse-proxy/.
[88] Docker Hub (11 de 11 de 2021). *jwilder/nginx-proxy*. Disponible: https://github.com/nginx-proxy/nginx-proxy.
[89] Docker Hub (11 de 11 de 2021). *willfarrel/autoheal*. Disponible: https://hub.docker.com/r/willfarrell/autoheal/.
[90] WESO (12 de 11 de 2021). Labra's Home Page. Disponible: https://labra.weso.es/.
[91] Internet Engineering Task Force (10 de 12 de 2021). *RFC 1738 – Uniform Resource Locators (URL)*. Disponible: https://www.ietf.org/rfc/rfc1738.txt.
[92] AChecker (13 de 12 de 2021). *Web Accessibility Checker for WCAG 2.0*. Disponible: https://achecker.achecks.ca/checker/index.php.
[93] PlusQA (13 de 12 de 2021). *Top Desktop Browsers 2021*. Disponible: https://plusqa.com/2021/06/17/top-desktop-browsers-2021.

[94] GitHub (14 de 12 de 2021). *About GitHub Pages*. Disponible: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages.

[95] MongoDB Manual (10 de 1 de 2022). *MongoDB Wire Protocol*. Disponible: https://www.mongodb.com/docs/manual/reference/mongodb-wire-protocol/.

[96] MongoDB Manual (13 de 1 de 2022). *ObjectId*. Disponible: https://www.mongodb.com/docs/manual/reference/method/ObjectId/.

[97] Postman (19 de 1 de 2022). *Postman API Platform*. Disponible: https://www.postman.com/.

[98] Wikipedia (25 de 1 de 2022). *Monad (functional programming)*. Disponible: https://en.wikipedia.org/wiki/Monad_(functional_programming).

[99] MongoDB Atlas (27 de 1 de 2022). *Additional Cluster Settings*. Disponible: https://www.mongodb.com/docs/atlas/cluster-additional-settings/#select-the-mongodb-version-of-the-cluster.

[100] GitHub (27 de 1 de 2022). *Kirill5k/mongo4cats*. Disponible: https://github.com/kirill5k/mongo4cats.

[101] GitHub (28 de 1 de 2022). *scalatest/scalatest: A testing tool for Scala and Java developers*. Disponible: https://www.scalatest.org/.

[102] GitHub (28 de 1 de 2022). *typelevel/cats-effect-testing: Integration between cats-effect and test frameworks*. Disponible: https://github.com/typelevel/cats-effect-testing.

[103] W3C (2 de 2 de 2022). *How to meet WCAG (Quick Reference)*. Disponible: https://www.w3.org/WAI/WCAG21/quickref/?versions=2.0.

[104] Manpages (24 de 4 de 2022). *man inxi: Command line system information script for console and IRC*. Disponible: https://manpages.org/inxi.

[105] Docker Docs (6 de 5 de 2022). *Use multi-stage builds*. Disponible: https://docs.docker.com/develop/develop-images/multistage-build/.