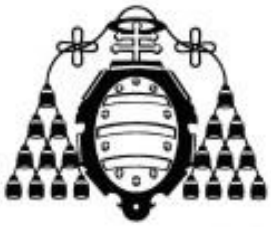
UNIVERSIDAD DE OVIEDO



ESCUELA DE INGENIERÍA INFORMÁTICA

TRABAJO FIN DE MÁSTER

Portal del empleado para Ecocomputer S.L.

DIRECTOR: María Lourdes Tajes Martínez

AUTOR: Mario Rey Regúlez

Vº Bº del Director del Proyecto

Agradecimientos

A mi familia y amigos, por animarme y apoyarme durante los momentos más difíciles del proyecto.

A Ecocomputer S.L., en especial a Myrtha y Agustín, por su colaboración e interés en el devenir del proyecto.

A mis compañeros Borja Garrido y David Cabañeros, con quien se compartieron penas y glorias durante la implantación del entorno de trabajo común.

Y a Lourdes Tajes, mi directora de Trabajo Fin de Máster, siempre aportando ideas y resolviendo dudas sobre cómo afrontar este o aquel problema.

Resumen

La aplicación desarrollada permite a los empleados de una empresa gestionar diferentes aspectos de su vida laboral proporcionándoles una interfaz en forma de portal web amigable al que pueden acceder desde cualquier navegador web moderno. La aplicación ha sido desarrollada a petición de la empresa Ecocomputer S.L., bajo su supervisión y en colaboración con ellos.

Concretamente, en la aplicación los empleados podrán gestionar sus propios días libres, según se trate un tipo de permiso u otro. Dicha gestión se realizará a través de una interfaz en la que toda la información se podrá visualizar de una manera clara.

Adicionalmente, los empleados tendrán la posibilidad de gestionar sus dietas desde dicho portal, pudiendo indicar individualmente para cada día los diferentes aspectos que han conformado esa dieta (dinero gastado en aparcamiento, kilometraje de los desplazamientos…), así como subir archivos y fotografías que demuestren lo indicado. Todo esto se podrá visualizar un informe resumen para cada empleado.

El acceso al portal estará restringido por un nombre de usuario y una contraseña, existiendo diferentes perfiles con distintos niveles de privilegio para el acceso o administración de cada una de las funcionalidades.

Cabe destacar que esta funcionalidad se construirá sobre una intranet desarrollada por otra persona, e igualmente convivirá con aplicaciones adicionales de las que no se posee control alguno.

La aplicación se desplegará en un servidor Apache Tomcat, desarrollada bajo las tecnologías Java y la base de datos estará bajo SQL Server 2005.

Palabras Clave

Gestión empresarial, Dietas, Calendario laboral, Desarrollo web, Struts2, Java.

Abstract

The developed application allows employees of a company to manage different aspectos of their laboral basis by providing a friendly website that can be accessed from any modern web browser. The application has been developed at the request of Ecocomputer S.L., under its supervision and in collaboration with them.

Specifically, on the application the employees can manage their own days off, depending on the specified type of permission. This management is done through an interface in which all the information is displayed in a clear manner.

In addition, the employees will be able to manage their own expenses while working, and may indicate individually for each day the different aspects of them (money spent on parking, mileage…), as well as upload files and photographs demonstrating it.. All this information will be printed over a summary report for each employee.

Accessing into the website will be restricted by username and password credentials, and different access levels are provided for managing each of the functionalities.

Note that this funcionality will be built over an already existent intranet, developed by others, and also coexisting with additional applications for which there is no control whatsoever.

The application will be deployed over an Apache Tomcat web server, developed under Java texhnologies and its database provided by SQL Server 2005.

Keywords

Business management, Expenses, Business calendar, Web development, Struts2, Java.

Índice General

CAPÍTULO 1. MEMORIA DEL PROYECTO 23

1.1 RESUMEN DE LA MOTIVACIÓN, OBJETIVOS Y ALCANCE DEL PROYECTO 23
1.2 RESUMEN DE TODOS LOS ASPECTOS 24
1.2.1 Capítulo 1: Memoria del Proyecto 24
1.2.2 Capítulo 2: Introducción 24
1.2.3 Capítulo 3: Aspectos teóricos 24
1.2.4 Capítulo 4: Planificación del Proyecto 24
1.2.5 Capítulo 5: Análisis 24
1.2.6 Capítulo 6: Diseño del Sistema 24
1.2.7 Capítulo 7: Implementación del Sistema 25
1.2.8 Capítulo 8: Desarrollo de Pruebas 25
1.2.9 Capítulo 9: Manuales del Sistema 25
1.2.10 Capítulo 10: Conclusiones y ampliaciones 25
1.2.11 Capítulo 11: Presupuesto 25
1.2.12 Capítulo 12: Referencias Bibliográficas 25
1.2.13 Capítulo 13: Apéndice 25

CAPÍTULO 2. INTRODUCCIÓN 27

2.1 ESTUDIO DE LA SITUACIÓN 27
2.2 JUSTIFICACIÓN DEL PROYECTO 28
2.3 OBJETIVOS DEL PROYECTO 30
2.3.1 Gestión del calendario laboral 30
2.3.2 Gestión de dietas 31

CAPÍTULO 3. ASPECTOS TEÓRICOS 33

3.1 JAVASERVER PAGES 33
3.1.1 Descripción 33
3.1.2 Aplicación en el proyecto 33
3.2 SQL SERVER 2005 34
3.2.1 Descripción 34
3.2.2 Aplicación en el proyecto 34
3.3 APACHE TOMCAT 35
3.3.1 Descripción 35
3.3.2 Aplicación en el proyecto 35
3.4 LENGUAJE UNIFICADO DE MODELADO 36
3.4.1 Descripción 36
3.4.2 Aplicación en el proyecto 36
3.5 MÉTRICA VERSIÓN 3 37
3.5.1 Descripción 37
3.5.2 Aplicación en el proyecto 37

CAPÍTULO 4. PLANIFICACIÓN DEL PROYECTO 39

4.1 PLANIFICACIÓN 39
4.1.1 Planificación inicial 40
4.1.2 Planificación real 41
4.1.3 Justificación de las diferencias 42

CAPÍTULO 5. ANÁLISIS 45

5.1 DEFINICIÓN DEL SISTEMA 45
5.1.1 Determinación del Alcance del Sistema 45
5.2 REQUISITOS DEL SISTEMA 47
5.2.1 Obtención de los Requisitos del Sistema 47
5.2.2 Identificación de Actores del Sistema 49
5.2.3 Especificación de Casos de Uso 50
5.3 IDENTIFICACIÓN DE LOS SUBSISTEMAS EN LA FASE DE ANÁLISIS 56
5.3.1 Descripción de los Subsistemas 56
5.3.2 Descripción de los Interfaces entre Subsistemas 57
5.4 DIAGRAMA DE CLASES PRELIMINAR DEL ANÁLISIS 58
5.4.1 Diagrama de Clases 58
5.4.2 Descripción de las Clases 59
5.5 ANÁLISIS DE CASOS DE USO Y ESCENARIOS 64
5.5.1 Casos de uso: gestión de calendario 64
5.5.2 Casos de uso: gestión de dietas 72
5.6 ANÁLISIS DE INTERFACES DE USUARIO 80
5.7 ESPECIFICACIÓN DEL PLAN DE PRUEBAS 81
5.7.1 Pruebas unitarias 81
5.7.2 Pruebas de integración 83
5.7.3 Pruebas de sistema 89
5.7.4 Pruebas de usabilidad 89

CAPÍTULO 6. DISEÑO DEL SISTEMA 91

6.1 ARQUITECTURA DEL SISTEMA 91
6.1.1 Diagrama de Paquetes 91
6.1.2 Diagrama de Componentes 94
6.1.3 Diagrama de Despliegue 95
6.2 DISEÑO DE CLASES 97
6.2.1 Diagrama de Clases 97
6.3 DIAGRAMAS DE INTERACCIÓN Y ESTADOS 109
6.3.1 Caso de uso insertar dieta 110
6.3.2 Caso de uso generar informe de dietas 112
6.3.3 Caso de uso solicitar permiso de vacaciones 114
6.4 DISEÑO DE LA BASE DE DATOS 116
6.4.1 Descripción del SGBD Usado 116
6.4.2 Integración del SGBD en Nuestro Sistema 116
6.4.3 Diagrama de base de datos 116
6.5 DISEÑO DE LA INTERFAZ 125
6.5.1 Módulo de gestión del calendario laboral 126
6.5.2 Módulo de gestión de dietas 132
6.6 ESPECIFICACIÓN TÉCNICA DEL PLAN DE PRUEBAS 136
6.6.1 Pruebas Unitarias 136
6.6.2 Pruebas de Integración y del Sistema 136
6.6.3 Pruebas de Accesibilidad 136

CAPÍTULO 7. IMPLEMENTACIÓN DEL SISTEMA 137

7.1 ESTÁNDARES Y NORMAS SEGUIDOS 137
7.1.1 XTML 1.1 137
7.1.2 CSS 2 137
7.1.3 Estándares de programación 138

7.1.4 Modelo Vista Controlador 138
7.2 Lenguajes de Programación 139
7.2.1 Java 139
7.2.2 JavaScript 141
7.3 Herramientas y Programas Usados para el Desarrollo 142
7.3.1 Eclipse Juno 142
7.3.2 SQL Server Management Studio Express 142
7.3.3 Enterprise Architect 142
7.3.4 Dia 143
7.3.5 Microsoft Office Project 143
7.4 Creación del Sistema 144
7.4.1 Problemas encontrados 144
7.4.2 Descripción Detallada de las Clases 146

CAPÍTULO 8. DESARROLLO DE LAS PRUEBAS 217

8.1 Pruebas Unitarias 217
8.1.1 Insertar permiso 217
8.1.2 Obtener permiso por identificador 217
8.1.3 Obtener permisos según criterios 217
8.1.4 Modificar permiso 217
8.1.5 Eliminar permiso 218
8.1.6 Autorizar permiso 218
8.1.7 Denegar permiso 218
8.1.8 Insertar tipo de permiso 218
8.1.9 Obtener tipo de permiso por identificador 218
8.1.10 Obtener todos los tipos de permiso 219
8.1.11 Modificar tipo de permiso 219
8.1.12 Eliminar tipo de permiso 219
8.1.13 Obtener bolsa de días de vacaciones de usuario 219
8.1.14 Actualizar bolsa de días de vacaciones 220
8.1.15 Sumar días en bolsa 220
8.1.16 Restar días en bolsa 220
8.1.17 Insertar detalle en dieta nueva 220
8.1.18 Insertar detalle en dieta ya existente 221
8.1.19 Obtener dieta por identificador 221
8.1.20 Obtener dietas según criterios 221
8.1.21 Modificar detalle 221
8.1.22 Eliminar último detalle 222
8.1.23 Eliminar detalle (no último) 222
8.1.24 Cambiar estado de dieta 222
8.1.25 Obtener tipo de dieta por identificador 222
8.1.26 Obtener todos los tipos de dieta 222
8.1.27 Insertar complemento no salarial 223
8.1.28 Obtener complemento no salarial por identificador 223
8.1.29 Obtener todos los complementos no salariales 223
8.1.30 Modificar complemento no salarial 223
8.1.31 Eliminar complemento no salarial 223
8.2 Pruebas de Integración y del Sistema 225
8.3 Pruebas de Rendimiento 235
8.4 Pruebas de Accesibilidad 235

CAPÍTULO 9. MANUALES DEL SISTEMA 241

9.1 MANUAL DE INSTALACIÓN 241
9.1.1 Servidor de base de datos 241
9.1.2 Sistema de gestión de base de datos 243
9.1.3 Base de datos 244
9.1.4 Servidor web 261
9.2 MANUAL DE EJECUCIÓN 262
9.2.1 Inicio del servidor web 262
9.2.2 Desplegar la aplicación 262
9.2.3 Parada del servidor web 262
9.3 MANUAL DE USUARIO 263
9.3.1 Inicio de sesión en la aplicación 263
9.3.2 Módulo de gestión de dietas 264
9.3.3 Módulo de gestión del calendario laboral 272
9.4 MANUAL DEL PROGRAMADOR 280

CAPÍTULO 10. CONCLUSIONES Y AMPLIACIONES 283

10.1 CONCLUSIONES 283
10.2 AMPLIACIONES 284
10.2.1 Usabilidad y accesibilidad 284
10.2.2 Lógica de grupos de trabajo 284
10.2.3 Lógica de jefe de departamento 284
10.2.4 Manejo de festivos 284
10.2.5 Aplicación para Android 285

CAPÍTULO 11. PRESUPUESTO 287

11.1 DESARROLLO DE PRESUPUESTO SIMPLIFICADO 287

CAPÍTULO 12. REFERENCIAS BIBLIOGRÁFICAS 289

12.1 REFERENCIAS EN INTERNET 289
12.1.1 Documentaciones oficiales 289
12.1.2 Otras documentaciones no oficiales 289
12.1.3 Tutoriales y guías varias 290
12.1.4 Foros de ayuda 290

CAPÍTULO 13. APÉNDICES 291

13.1 GLOSARIO 291
13.2 CONTENIDO ENTREGADO EN EL CD-ROM 293

Índice de Figuras

Figura 4.1 - Planificación inicial 40
Figura 4.2 - Planificación real 41
Figura 5.1- Diagrama de casos de uso: Gestión del calendario laboral....51
Figura 5.2 – Diagrama de casos de uso: Gestión de dietas....53
Figura 5.3 Diagrama de clases preliminar....58
Figura 5.4 - Diagrama de robustez: Solicitar permiso....65
Figura 5.5 - Diagrama de robustez - Eliminar permiso 66
Figura 5.6 - Diagrama de robustez - Visualizar calendario 67
Figura 5.7 - Aceptar solicitud....70
Figura 5.8 Diagrama de robustez - Actualizar bolsa de vacaciones 72
Figura 5.9 - Diagrama de robustez - Insertar dieta....73
Figura 5.10 Diagrama de robustez - Visualizar informe....75
Figura 5.11 - Diagrama de robustez - Generar informe PDF....76
Figura 6.1 - Diagrama de paquetes....91
Figura 6.2 - Diagrama de componentes 94
Figura 6.3 - Diagrama de despliegue....95
Figura 6.4 - Diagrama de clases: calendario.org.model....97
Figura 6.5 - Diagrama de clases: calendario.org.persistence....98
Figura 6.6 - Diagrama de clases: calendario.org.business 99
Figura 6.7 - Diagrama de clases: calendario.impl.persistence 100
Figura 6.8 - Diagrama de clases: calendario.impl.business 101
Figura 6.9 Diagrama de clases: calendario.impl.presentation....102
Figura 6.10 - Diagrama de clases: dietas.org.model....103
Figura 6.11 - Diagrama de clases: dietas.org.persistence....104
Figura 6.12 - Diagrama de clases: dietas.org.business 105
Figura 6.13 - Diagrama de clases: dietas.impl.persistence 106
Figura 6.14 - Diagrama de clases: dietas.impl.business 107
Figura 6.15 Diagrama de clases: dietas.impl.presentation....108
Figura 6.16 - Diagrama de interacción: insertar dieta 110
Figura 6.17 - Diagrama de estados: dieta....111
Figura 6.18 - Diagrama de interacción: generar informe de dietas 112
Figura 6.19 - Diagrama de interacción: solicitar permiso de vacaciones 114
Figura 6.20 - Diagrama de estados: permiso de vacaciones 115
Figura 6.21 Diagrama de base de datos: Usuarios 117
Figura 6.22 - Diagrama de base de datos – Calendario 120
Figura 6.23 - Diagrama de base de datos – Dietas 122
Figura 6.24 - Interfaz calendario laboral: resumen permisos 126
Figura 6.25 - Interfaz calendario laboral: buscador....127
Figura 6.26 - Interfaz calendario laboral: nueva solicitud de permiso 128
Figura 6.27 - Interfaz calendario laboral: administrador de tipos de permisos 129
Figura 6.28 - Interfaz calendario laboral: nuevo tipo de permiso....130
Figura 6.29 - Interfaz calendario laboral: bolsa de vacaciones 131
Figura 6.30 - Interfaz de gestión de dietas: gestor de dietas....132
Figura 6.31 - Interfaz de gestión de dietas: nueva dieta 133
Figura 6.32 - Interfaz de gestión de dietas: administrador de complementos no salariales 134
Figura 6.33 - Interfaz de gestión de dietas: nuevo complemento no salarial 135

Figura 9.1 - Manual de instalación: datos del usuario 241
Figura 9.2 - Manual de instalación: componentes a instalar 242
Figura 9.3 - Manual de instalación: proceso finalizado 243
Figura 9.4 - Manual de instalación: instalando el SGBD 244
Figura 9.5 - Manual de usuario: conectando a la instancia 244
Figura 9.6 - Manual de usuario: inicio de sesión 263
Figura 9.7 - Manual de usuario: sin dietas declaradas 264
Figura 9.8 - Manual de usuario: inserción de dieta 265
Figura 9.9 – Manuel de usuario: insertando dieta 265
Figura 9.10 - Manual de usuario: visualización de dietas 266
Figura 9.11 - Manual de usuario: imagen asociada a dieta 266
Figura 9.12 - Manual de usuario: editar dieta 267
Figura 9.13 - Manual de usuario: dieta validada 268
Figura 9.14 - Manual de usuario: informe PDF 268
Figura 9.15 - Manual de usuario: visualizar dietas como administrador 269
Figura 9.16 - Manual de usuario: autorizar y rechazar dieta 270
Figura 9.17 - Manual de usuario: listado de complementos no salariales 270
Figura 9.18 - Manual de usuario: insertar complemento no salarial 271
Figura 9.19 - Manual de usuario: editar complemento no salarial 271
Figura 9.20 - Manual de usuario: resumen de permisos vacío 272
Figura 9.21 - Manual de usuario: insertar permiso 273
Figura 9.22 - Resumen de permisos 273
Figura 9.23 - Manual de usuario: editar permiso 274
Figura 9.24 - Manual de usuario: calendario de permisos 275
Figura 9.25 - Manual de usuario: buscador y filtro de permisos 276
Figura 9.26 - Manual de usuario: tipos de permiso 277
Figura 9.27 - Manual de usuario: insertar nuevo tipo de permiso 278
Figura 9.28 - Manual de usuario: editar tipo de permiso 278
Figura 9.29 - Manual de usuario: bolsa de vacaciones 279

Capítulo 1. Memoria del Proyecto

1.1 Resumen de la Motivación, Objetivos y Alcance del Proyecto

La presente aplicación ha sido desarrollada debido al interés del proyectante en realizar un proyecto con y para una empresa real, y con el objetivo de facilitar en cierta medida la consecución de aspectos de gestión interna de la empresa.

De esta manera se contribuye a mejorar la productividad de una empresa, lo que acabará reportando beneficios como el ahorro de tiempo que supone realizar las tareas de una manera más rápida.

La aplicación fue propuesta por los propios empleados de Ecocomputer S.L., empresa donde el alumno realiza prácticas de empresa, para cubrir unas necesidades que ellos mismos detectaron. De esta manera, los objetivos que se pretenden alcanzar están orientados a subsanar una serie de carencias reales de una empresa real, por lo que se dará a la aplicación un uso real y continuado.

Se pretenden cubrir unas necesidades concretas en la gestión interna de Ecocomputer para con sus empleados, como son la gestión de su calendario laboral en base a los días libres de cada uno de ellos o la fácil creación y almacenamiento de las dietas de los empleados. El acceso a la funcionalidad debe ser mediante interfaces web, identificándose en el sistema de manera segura bajo un nombre de usuario y contraseña.

Adicionalmente, la seguridad se extiende al punto de poder determinar qué usuarios tienen acceso a una funcionalidad u otra, dependiendo de su nivel de privilegios dentro de la jerarquía propia de la empresa.

1.2 Resumen de Todos los Aspectos

Este documento contiene los siguientes capítulos:

1.2.1 Capítulo 1: Memoria del Proyecto

En esta sección se resume en general el contenido del proyecto, para quienes no posean un conocimiento avanzado y técnico de las diferentes tecnologías empleadas. El público objetivo es cualquier persona que quiera saber a grandes rasgos y sin profundizar demasiado lo que se pretende conseguir con la realización de este proyecto.

1.2.2 Capítulo 2: Introducción

Aquí se explica el propósito por el que se ha desarrollado el proyecto. Se detallan las necesidades que se pretenden satisfacer, así como qué se pretende conseguir con ello.

1.2.3 Capítulo 3: Aspectos teóricos

En este apartado describiremos los conceptos relevantes y necesarios para el correcto desarrollo del proyecto. Se trata de una pequeña descripción de las diferentes tecnologías o herramientas empleadas durante todo el proceso de creación.

1.2.4 Capítulo 4: Planificación del Proyecto

Este capítulo contiene la planificación temporal del proyecto: el tiempo que ha llevado desarrollarlo desde la concepción de la idea hasta el último aspecto antes de la entrega definitiva.

1.2.5 Capítulo 5: Análisis

Esta sección abarca todo lo concerniente a los límites del desarrollo de la aplicación. Todas las funcionalidades que ofrecerá son aquí desglosadas, estudiando los diferentes elementos necesarios para el correcto cumplimiento de los requisitos listados. Finalmente, el lector podrá encontrar un primer boceto de la interfaz de usuario y las diferentes pruebas que la aplicación deberá superar para cumplir con lo requerido.

1.2.6 Capítulo 6: Diseño del Sistema

Se explica la arquitectura de la aplicación desarrollada con la ayuda de diferentes diagramas que faciliten su entendimiento. También se incluye el resultado final de la interfaz de usuario y se define cómo se harán las diferentes pruebas.

1.2.7 Capítulo 7: Implementación del Sistema

En este capítulo el lector podrá conocer los distintos lenguajes de programación empleados para construir la aplicación, así como normas o patrones de desarrollo y los programas externos que han sido empleados para ello. También se desglosan los diferentes problemas surgidos durante el desarrollo del proyecto, además de una descripción detallada de las clases del código fuente del mismo.

1.2.8 Capítulo 8: Desarrollo de Pruebas

Aquí podremos encontrar los resultados de las pruebas realizadas a la aplicación y las conclusiones obtenidas tras su estudio.

1.2.9 Capítulo 9: Manuales del Sistema

Bajo este apartado se agrupan los diferentes manuales o guías que se necesitan para instalar la aplicación en el entorno de explotación. También se encuentra el manual de usuario que explica cómo se debe usar la aplicación, además de una guía para otros desarrolladores que pueda ayudarles a ampliar, modificar o entender diferentes conceptos del proyecto.

1.2.10 Capítulo 10: Conclusiones y ampliaciones

En esta sección se localiza la opinión personal sobre lo que ha supuesto la creación de este proyecto para su desarrollador, así como un desglose de las diferentes posibles mejoras y ampliaciones a las que la aplicación podría ser sometida en un futuro.

1.2.11 Capítulo 11: Presupuesto

Aquí podrá encontrarse el presupuesto detallado de lo que este proyecto podría haber costado financieramente.

1.2.12 Capítulo 12: Referencias Bibliográficas

Bajo este capítulo se muestra una lista de toda la documentación consultada para la realización del proyecto, ya sean libros divulgativos, explicativos o enlaces de Internet con dicha información.

1.2.13 Capítulo 13: Apéndice

Se detalla el contenido del CD entregado y las tareas que deben realizarse para su correcta instalación, ejecución y configuración.

Capítulo 2. Introducción

2.1 Estudio de la situación

Este proyecto surge con el fin de subsanar una serie de carencias en la gestión interna de una empresa real, Ecocomputer S.L.

Se trata de una empresa de servicios informáticos con sede en Avilés (Asturias) con un amplio abanico de soluciones a disposición de sus clientes. Cuenta con dos centros de trabajo, en Asturias (con más de 500 metros cuadrados, data center, sala de formación, laboratorio electrónico) y en León. Su zona geográfica habitual cubre Asturias, Castilla y León y Cantabria.

Ecocomputer cuenta con 16 empleados que le han permitido desarrollar proyectos muy variados, en algunos casos con una fuerte componente tecnológica y de innovación. Está fuertemente enfocada a la PYME, aunque también trabaja el sector de la administración y la gran cuenta.

Entre los servicios que ofrece destacan el desarrollo de aplicaciones y portales web a medida, hospedaje, registro y alojamiento de dominios para cliente, así como la implantación de sistemas de gestión empresarial.

La situación actual de la empresa en ciertos aspectos de su gestión interna está fundamentada en métodos y mecanismos un tanto anticuados hoy en día: uso de complicadas hojas de cálculo para llevar cuentas interna, o comunicaciones verbales o por correo electrónico para notificar ciertos aspectos de los que es difícil llevar un registro manual sencillo.

Este proyecto tiene como objetivo mejorar esas carencias, detectadas por los propios empleados y gerentes de la empresa, a través de una aplicación informática potente y que sea capaz de ayudar a mantener la gestión interna de una manera sencilla y rápida.

2.2 Justificación del Proyecto

Este proyecto se realiza para mejorar la productividad de la empresa en lo que a la gestión de diferentes aspectos laborales de sus empleados se refiere. Una importante cantidad de tareas que se realizan en la actualidad de manera independiente podrán ser ejecutadas fácil y directamente desde un navegador web moderno cualquiera.

Dichas tareas se agruparían en dos bloques o módulos independientes entre sí:

- **Gestión del calendario laboral**: mediante el uso de este módulo los empleados podrán asignarse sus propios días libres según se traten de vacaciones, asuntos propios, descansos compensatorios y bajas laborales. Los días libres marcados podrán ser visualizados según diferentes filtros y parámetros. Existirá una jerarquía de aprobación de las peticiones para que todo esté supervisado por un responsable y las solicitudes no causen problemas de organización.

- **Gestión de dietas**: con este módulo los empleados tendrán la posibilidad de cumplimentar un formulario en línea para detallar sus dietas mensuales. Para cada día del mes se podrán especificar, si corresponde, diferentes conceptos que supongan un gasto para el empleado, además de la posibilidad de adjuntar documentos (tickets, facturas…) que demuestren dichos gastos. Para cada mes podrá visualizarse un informe de las dietas de cada empleado y los tickets escaneados. Existirá un flujo de supervisión corrección y aprobación del informe presentado, a cargo de la persona responsable.

A este portal se accederá mediante un nombre de usuario y contraseña, de manera que la identificación de los empleados sea inequívoca. Existirán diferentes perfiles de usuario:

- **Administrador**: tendrá permiso para dar de alta nuevos usuarios, eliminarlos o cambiar su nivel de acceso y acceder al calendario laboral y las dietas reportadas por todos los empleados.

- **Personal**: solo tendrá acceso a gestionar sus días libres y anotar sus dietas, sin tareas administrativas.

Adicionalmente a estos perfiles predefinidos, los usuarios de Administración podrán **otorgar permisos concretos a individuos específicos**, de manera que alguien pueda realizar una función aunque su perfil no lo permita.

La realización de este proyecto se justifica debido a la actual gestión que realizan en Ecocomputer para la realización de estas tareas: documentación impresa y hojas de cálculo que se pueden extraviar y que no están centralizadas y localizadas en un entorno común. Por tanto, y como ya se ha dicho con anterioridad, la razón de ser de este proyecto es el aumento de la comodidad y la productividad de la empresa a la hora de realizar y gestionar una serie de tareas de importancia a nivel interno.

Cabe destacar que otras personas relacionadas con la empresa están realizando componentes adicionales que convivirán con lo desarrollado en este proyecto, se servirán de él o se

ofrecerán para proporcionar una base común sobre la que trabajar. A este respecto se debe señalar que la base operativa ofrecida por una "Intranet" no ha sido desarrollada en este proyecto, al igual que la gestión de los usuarios de la aplicación. Sin embargo, se ha participado en la toma de decisiones sobre la naturaleza de dichos componentes ajenos a este proyecto para llegar a un consenso común. Del mismo modo, se han aceptado condiciones impuestas por otros participantes, con la consecuente adaptación a las mismas.

Por este motivo se justifican las carencias de ciertos aspectos del proyecto, que se irán desglosando en los siguientes apartados de esta documentación. Las distintas condiciones impuestas por el cliente juegan un papel muy importante en el resultado final del producto, si no el que más. Este proyecto no es una excepción.

2.3 Objetivos del Proyecto

2.3.1 Gestión del calendario laboral

1. **Solicitar días libres:** cada empleado dispondrá de un número determinado de días libres que cubrir según su naturaleza: vacaciones, asuntos propios y descansos compensatorios, así como permisos por bajas laborales por uno u otro motivo. Los días marcados deberán estar supeditados a las condiciones del convenio laboral de la empresa. Esta funcionalidad estará disponible para todos los empleados.

2. **Eliminar días libres:** cada empleado podrá eliminar días libres reservados con anterioridad, para así poder liberar esos días ya anotados. Esta funcionalidad estará disponible para todos los empleados, con la adición de que el perfil Administrador podrá eliminar también los días del resto de trabajadores.

3. **Modificar días libres:** cada empleado podrá modificar los días libres ya reservados, siempre y cuando no hayan sido ya autorizados o disfrutados. Esta funcionalidad estará disponible para todos los empleados, con la adición de que el perfil Administrador podrá modificar también los días del resto de trabajadores.

4. **Mantener bolsa de vacaciones:** los empleados dispondrán de una "bolsa de vacaciones" donde se acumulen los días de vacaciones aún no disfrutados. Dicha bolsa será mantenida por el perfil Administrador.

5. **Aceptación de vacaciones:** existirá una o varias personas que se encarguen de la aprobación de los días de vacaciones solicitados por los empleados. Esas personas serán las que dispongan de perfil Administrador.

6. **Visualización del calendario:** los usuarios podrán visualizar su calendario de manera mensual en una interfaz que simule la disposición de un calendario estándar. Esta funcionalidad podrá ser realizada por cualquier usuario de la aplicación.

7. **Inserción de tipo de permiso:** la aplicación permitirá incluir nuevos tipos de permiso en el sistema, si así lo requieren las necesidades de la empresa. Esta funcionalidad estará disponible para los usuarios con perfil Administrador.

8. **Eliminación de tipo de permiso:** la aplicación permitirá eliminar tipos de permiso ya existentes en el sistema, si así lo requieren las necesidades de la empresa. Esta funcionalidad estará disponible para los usuarios con perfil Administrador.

9. **Modificación de tipo de permiso:** la aplicación permitirá modificar los tipos de permisos ya existentes en el sistema, si así lo requieren las necesidades de la empresa. Esta funcionalidad estará disponible para los usuarios con perfil Administrador.

2.3.2 Gestión de dietas

10. **Insertar nueva dieta:** los empleados podrán cumplimentar un formulario online detallando para cada día del mes información como la descripción de la actividad, número de albarán, gastos de aparcamiento, gastos de dietas alimenticias, kilometraje realizado, repostaje de combustible, horas extra y otros conceptos directamente relacionados con la elaboración de la nómina, además de la posibilidad de adjuntar documentos (tickets, facturas…) que ilustren esos gastos. Las dietas pueden ir asociadas a convenios que afecten de una manera u otra al importe final. Esta funcionalidad estará disponible para todos los usuarios.

11. **Modificar dieta:** las dietas ya introducidas y aún no aceptadas por los supervisores podrán ser sometidas a su edición. Esta funcionalidad estará disponible para todos los empleados, con la adición de que el perfil Administrador podrá modificar también las dietas del resto de trabajadores.

12. **Eliminar dietas:** las dietas ya introducidas y aún no aceptadas por los supervisores podrán ser borradas del sistema. Esta funcionalidad estará disponible para todos los empleados, con la adición de que el perfil Administrador podrá eliminar también los días del resto de trabajadores.

13. **Visualizar informe mensual:** todas las dietas reportadas en un mes por un empleado podrán ser visualizadas en un informe descriptivo que podrá ser impreso. Dicho informe contendrá para cada día del mes las dietas generadas, si alguna, estando indicadas por su descripción y detalles, además de la suma final de los gastos que deberán ser reembolsados al empleado en cuestión. Esta funcionalidad estará disponible para todos los empleados, con la adición de que el perfil Administrador podrá visualizar el informe de todos los empleados.

14. **Supervisar y aprobar informe:** los responsables de la supervisión de las dietas de cada empleado podrán realizar cambios, correcciones, desestimaciones o propuestas de modificación una vez sus autores confirmen su completado.

15. **Generación del informe:** el informe mensual de las dietas de cada empleado podrá ser generado en formato PDF y descargado para su posterior guardado digital o generación en papel. Esta funcionalidad estará disponible para todos los empleados, con la adición de que el perfil Administrador podrá hacer lo propio con la de los empleados a su cargo.

16. **Insertar nuevo convenio:** los administradores del sistema podrán insertar nuevos convenios asociados a tipos de dietas indicando sus características.

17. **Eliminar convenio:** los administradores del sistema podrán eliminar los convenios ya existentes en el sistema para que no puedan volver a ser utilizados.

18. **Modificar convenio:** los administradores del sistema podrán modificar los convenios existentes en el sistema.

Capítulo 3. Aspectos Teóricos

3.1 JavaServer Pages

3.1.1 Descripción

JavaServer Pages (también conocidas por su acrónimo, JSP) se trata de una tecnología Java para generar contenido dinámico para la web dando como resultado documentos HTML. En general, el servidor web interpreta el código fuente contenido en una página JSP para construir la página resultante, ejecutando internamente y de manera transparente al usuario las operaciones necesarias para obtener el resultado esperado.

Esta tecnología ha sido desarrollada por Sun Microsystems, y fue lanzada por primera vez en 1999. Actualmente se encuentra en su versión 2.2, que ha sido la empleada en el desarrollo de esta aplicación.

3.1.2 Aplicación en el proyecto

Dado que la aplicación desarrollada será desarrollada en Java y es plenamente de naturaleza web, el porqué de la utilización de JavaServer Pages salta a la vista: es la manera más cómoda a la par que eficiente de mostrar contenido dinámico en función de la acción concreta que haya solicitado el usuario.

De esta manera, diferentes usuarios ejecutando acciones sobre el mismo documento JSP generarán resultados diferentes, puesto que cada uno de ellos tendrá una identidad distinta y un estado de la sesión distinto. Esto provocará operaciones independientes diferentes sin que por ello sea necesario un documento de visualización diferente.

3.2 SQL Server 2005

3.2.1 Descripción

SQL Server 2005 es un sistema de gestión de bases de datos basado en el modelo relacional, utilizando el lenguaje T-SQL para realizar las consultas pertinentes. Entre sus características cabe destacar el soporte de transacciones, soporte de procedimientos almacenados, la posibilidad de trabajar desde un entorno gráfico, realizar las consultas en modo cliente-servidor, concurrencia de usuarios y la posibilidad de administrar información de otros servidores de base de datos. Su instalación solo es posible bajo sistemas operativos Windows.

Este sistema de gestión de bases de datos fue lanzado en 1989 por Microsoft. En este proyecto se ha utilizado la versión 2005, puesto que es la existente la empresa cliente.

3.2.2 Aplicación en el proyecto

Debido a que es el sistema de gestión de base de datos impuesto por el cliente, se utilizará Microsoft SQL Server como base de datos. En ella se almacenará toda la información existente en la aplicación, así como otros datos que no tienen relevancia en el devenir de este proyecto, pero que son empleados por el resto de aplicaciones de la empresa.

3.3 Apache Tomcat

3.3.1 Descripción

Apache Tomcat Consiste en un servidor HTTP para diferentes plataformas (como UNIX, Windows…) que implementa el protocolo HTTP/1.1 e introduce el concepto de "sitio virtual". Funciona como un contenedor de servlets e implementa las especificaciones de las JavaServer Pages en su última versión.

Es empleado principalmente para enviar páginas web estáticas o dinámicas en la web a los clientes que las soliciten, ejecutando en el servidor las operaciones necesarias para su correcto funcionamiento. Permite ser instalado de manera local, sin acceso externo a la red, con el fin de probar las aplicaciones antes de ser expuestas en un entorno de explotación real.

Tomcat es mantenido por la Apache Software Foundation y voluntarios independientes. Se trata de software libre gratuito, y los usuarios pueden acceder a su código fuente y binarios en su totalidad, ya que se encuentra bajo una licencia Apache Software License. Fue lanzado al mercado en 1999, siendo la versión 6.0 la utilizada para el despliegue de esta aplicación en la infraestructura de la empresa.

3.3.2 Aplicación en el proyecto

Será el entorno donde se despliegue la aplicación final por el motivo obvio: soporte de JavaServer Pages. También se usará como entorno de pruebas de manera local.

3.4 Lenguaje Unificado de Modelado

3.4.1 Descripción

Más conocido por sus siglas en inglés UML, es el lenguaje para modelar software más conocido y empleado en la actualidad. Se trata de un lenguaje gráfico para visualizar, definir, detallar, construir y documentar gran parte de los aspectos de un sistema informático. UML cuenta con varios tipos de diagramas, que muestran y especifican diferentes aspectos de lo que se quiere representar.

Fue desarrollado por Grady Booch, Ivar Jacobson y Jim Rumbaugh en la década de los 90, y adoptado por la Object Management Group en 1997. No fue aceptado como estándar ISO hasta el año 2000. Tras una serie de revisiones de mayor o menor envergadura, la versión actual de la especificación UML, la 2.5, fue lanzada en octubre de 2012.

3.4.2 Aplicación en el proyecto

El lenguaje UML ha sido empleado durante las fases de análisis y diseño de este proyecto, con el fin de definir y documentar todos los aspectos de la arquitectura completa de la aplicación. Para ello se ha utilizado la herramienta Enterprise Architect en su versión 6, software propietario desarrollado por Sparx Systems.

3.5 MÉTRICA Versión 3

3.5.1 Descripción

MÉTRICA se trata de una metodología de planificación, desarrollo y mantenimiento de sistemas de información, promovida por el Ministerio de Administraciones Públicas del Gobierno de España para la sistematización de actividades del ciclo de vida de los proyectos software en el ámbito de las administraciones públicas.

Está basada en el modelo de desarrollo ISO/EIC 12207 (Information Technology – Software Life Cycle Processes) y en la norma ISO/EIC 15504 SPICE (Software Process Improvement And Assurance Standards Capability Determination).

La aplicación de MÉTRICA Versión 3 permite alcanzar unos objetivos claros, como la correcta definición de sistemas de información, la dotación de un producto software satisfactorio, la mejora de la productividad de los departamentos de sistemas y tecnologías de la información y las comunicaciones, la facilitación de la comunicación entre los diferentes participantes en la producción del software y la consecución de la operación, mantenimiento y uso del producto software.

3.5.2 Aplicación en el proyecto

La versión 3 de la metodología MÉTRICA será la empleada, al menos parcialmente, en la estructura de la presente documentación, organizando por capítulos y secciones determinados según aconseja la metodología. Pese a tratarse de un proyecto no dirigido a una administración pública, que son los organismos objetivo de MÉTRICA, su aplicación se considera un manual de buenas prácticas.

Capítulo 4. Planificación del Proyecto

4.1 Planificación

Es importante destacar que, como en una gran parte de los proyectos software, la planificación inicial de este proyecto no se corresponde enteramente con la realidad. Esto es debido a una serie de factores, tanto relacionados con el desempeño del programador como de agentes externos fuera de su control.

Por este motivo se incluye en el presente apartado dos planificaciones: la inicial y la real, para que puedan ser contrastadas directamente.

Nótese que las imágenes incorporadas en el documento son demasiado grandes para el tamaño del papel, por lo que han debido de colapsarse secciones y reducido el tamaño general para que puedan visualizarse completamente. No obstante, se entregan también los documentos originales para que puedan ser consultados si el lector lo considerase necesario. Pueden encontrarse bajo los nombres “Planificación inicial.mpp” y “Planificación real.mpp", respectivamente. Pueden ser abiertos con Microsoft Project.

4.1.1 Planificación inicial

	Nombre de tarea	Duración
1	**Portal de autogestión del empleado**	**460 horas**
2	**Negociación inicial**	**18 horas**
7	**Preparación del entorno de trabajo**	**8 horas**
10	**Análisis**	**36 horas**
11	Identificación de los módulos	2 horas
12	Diagrama de clases preliminar	2 horas
13	Análisis de casos de uso y escenarios	12 horas
14	Análisis de interfaces de usuario	8 horas
15	Especificación del plan de pruebas	8 horas
16	Negociación con el cliente	4 horas
17	**Desarrollo de la aplicación**	**368 horas**
18	**Diseño del sistema**	**54 horas**
19	**Arquitectura del sistema**	**24 horas**
26	**Diseño de las interfaces**	**10 horas**
30	**Especificación técnica del plan de pruebas**	**20 horas**
36	**Implementación del sistema**	**284 horas**
37	Modelado de la base de datos	8 horas
38	Desarrollo de las interfaces	12 horas
39	**Módulo de gestión de usuarios**	**26 horas**
44	**Módulo de gestión del calendario del empleado**	**92 horas**
49	**Módulo de gestión de dietas**	**92 horas**
54	**Módulo de mensajes**	**62 horas**
59	**Pruebas**	**276 horas**
67	**Documentación**	**432 horas**
68	Documentación de análisis	12 horas
69	Documentación de diseño	20 horas
70	Documentación de implementación	30 horas
71	**Redacción de manuales**	**12 horas**
72	Manuales de descripción técnica	12 horas
73	Manuales de usuario	8 horas
74	Informe de seguimiento	2 horas
75	Preparación y descripción de entregables	4 horas
76	Impresión	12 horas
77	Entrega final	2 horas

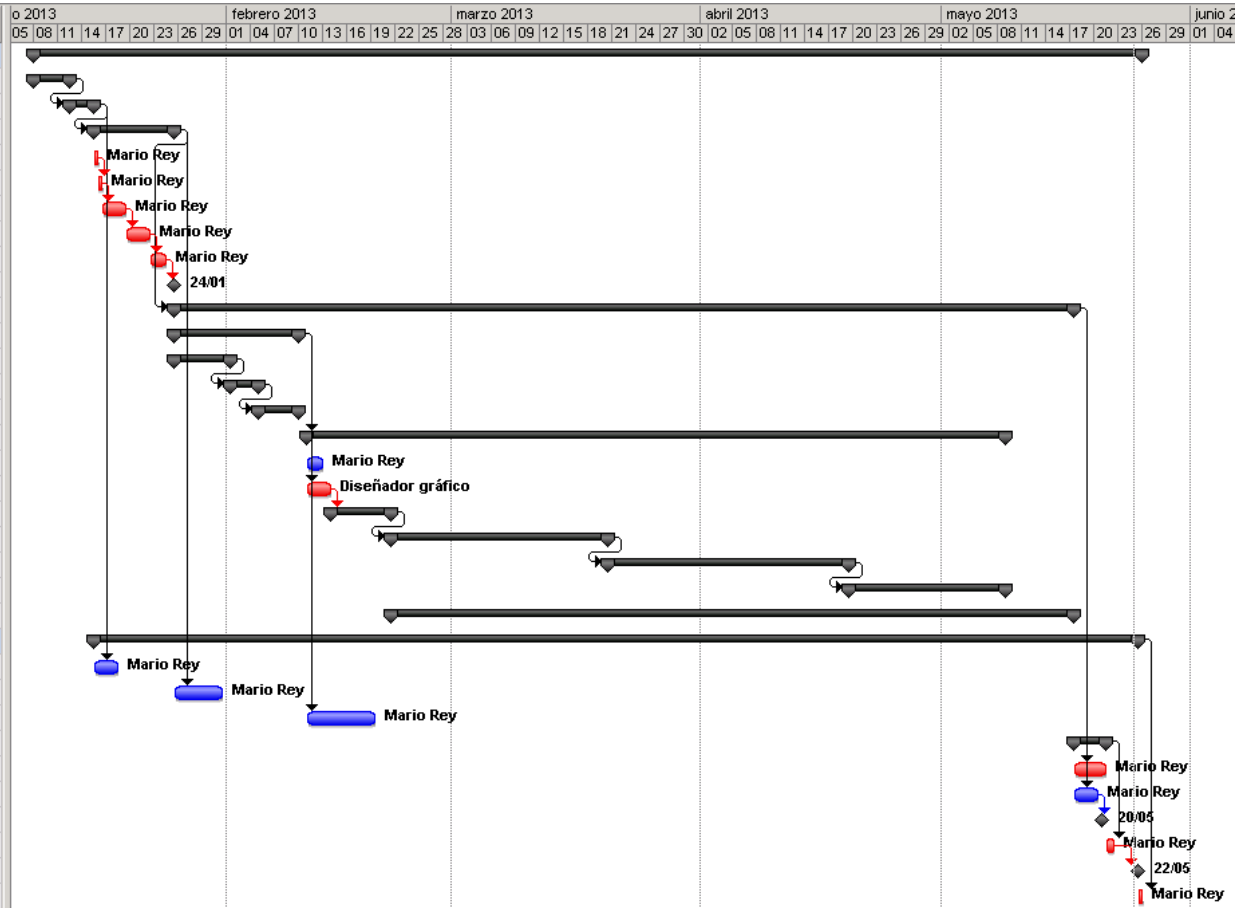


Figura 4.1 - Planificación inicial

4.1.2 Planificación real

	Nombre de tarea	Duración
1	**Portal de autogestión del empleado**	**436 horas**
2	**Negociación inicial**	**26 horas**
3	Estudio de los objetivos del proyecto	4 horas
4	Especificación técnica de requisitos	8 horas
5	Estudio de viabilidad de requisitos	4 horas
6	Negociación de modificaciones	4 horas
7	Establecer requisitos modificados	6 horas
8	**Preparación del entorno de trabajo**	**16 horas**
9	Estudio de las tecnologías necesarias	8 horas
10	Instalación y configuración del equipo	8 horas
11	**Análisis**	**54 horas**
12	Identificación de los módulos	2 horas
13	Diagrama de clases preliminar	4 horas
14	Análisis de casos de uso y escenarios	20 horas
15	Especificación del plan de pruebas	12 horas
16	Negociación con el cliente	4 horas
17	Modificación del análisis realizado	12 horas
18	**Desarrollo de la aplicación**	**290 horas**
19	**Diseño del sistema**	**86 horas**
20	**Arquitectura del sistema**	**58 horas**
28	**Especificación técnica del plan de pruebas**	**28 horas**
34	**Implementación del sistema**	**184 horas**
35	Modelado de la base de datos	12 horas
36	Desarrollo de las interfaces	12 horas
37	**Módulo de gestión de dietas**	**148 horas**
46	**Módulo de gestión del calendario**	**172 horas**
56	**Pruebas**	**44 horas**
62	Soporte	50 horas
63	**Documentación**	**326 horas**
64	Documentación de análisis	26 horas
65	Documentación de diseño	34 horas
66	Documentación de implementación	30 horas
67	**Redacción de manuales**	**14 horas**
71	Conclusiones y ampliaciones	6 horas
72	Preparación y descripción de entregables	6 horas
73	Diseño y grabación de los CD	6 horas
74	Entrega final	2 horas

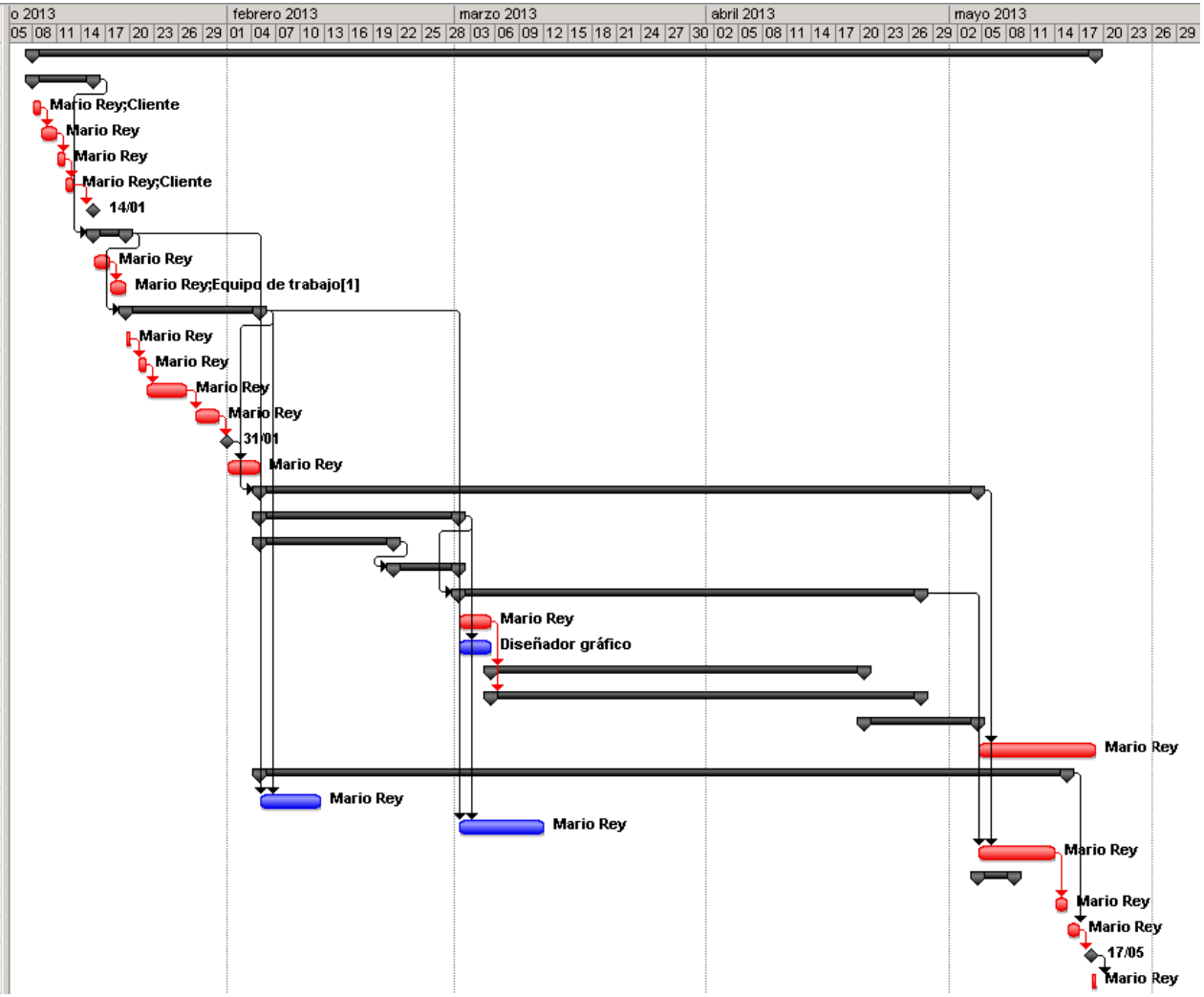


Figura 4.2 - Planificación real

4.1.3 Justificación de las diferencias

Planificar un proyecto software no es una tarea trivial. Muchos aspectos que sobre el papel se estiman de una manera resultan ser totalmente erróneos durante la ejecución de las tareas. Problemas inesperados, mala comprensión de los objetivos planteados o nuevas necesidades del cliente son algunos de los aspectos que retrasan considerablemente un proyecto y que fuerzan una toma de decisiones si se quieren respetar en cierta medida los plazos establecidos por ambas partes.

En el caso del proyecto que nos atañe cabe destacar los siguientes motivos de retraso:

- **Curva de aprendizaje más pronunciada**: Pese a que el desarrollador ya había realizado otros proyectos de menor complejidad con las tecnologías empleadas, no se disponía de la suficiente experiencia para realizar una aplicación más profesional, amplia y compleja. Esta curva de aprendizaje se consideró en la planificación inicial, pero resultó estar subestimada. Las horas empleadas para realizar una tarea aparentemente trivial se vieron incrementadas conforme aumentada la complejidad.

- **Cambios de requisitos:** Durante las diferentes reuniones e informes de seguimiento realizados con el cliente se detectaron carencias de lo planteado hasta el momento, tanto en la fase inicial de análisis como en las últimas etapas del desarrollo. Ya sea por fallos de comunicación entre ambas partes, errores de definición de requisitos del programador o cambios en las necesidades del cliente, fue necesario modificar aspectos fundamentales del modelo del sistema. Estos cambios tanto a nivel de documentación como de desarrollo provocaron retrasos insalvables.

- **Modificación del alcance del proyecto:** En las primeras fases de la ejecución del proyecto estaba planteado como una aplicación independiente a cualquier otra funcionalidad de la empresa: su propia base de datos, su propia gestión de usuarios y aspectos de seguridad… Conforme avanzaba el desarrollo se notó que sería conveniente compartir funcionalidad con otros colaboradores que estaban realizado aplicaciones independientes pero con una base común. Así, con el fin de compartir base de datos de usuario, inicio de sesión conjunto y apariencia gráfica común, fue necesario redefinir la arquitectura completa del sistema compartido. Esto provocó desechar aspectos ya planteados y desarrollados para adaptarse a las nuevas necesidades.

Los retrasos sufridos se vieron reflejados de manera drástica en la planificación real: pese a que la cantidad de horas no varíe demasiado, fue necesario eliminar funcionalidad considerada secundaria para dar cabida a los cambios y retrasos surgidos.

De este modo se desechó:

- **Aplicación para dispositivos móviles Android**, puesto que los navegadores web de los dispositivos móviles modernos no difieren mucho de los de escritorio, y la aplicación podría ser utilizada con cierta normalidad.

- **Módulo de gestión de usuarios y entorno común**, ya que fue realizado por otro compañero. Se trata de una aplicación independiente que permite la inserción, modificación y borrado de usuarios y sus perfiles, así como de los permisos asociados a cada uno de ellos. También abarca el inicio y fin de sesión, control de acceso a los módulos y elaboración de los aspectos comunes.

- **Módulo de mensajería interna**, puesto que se encontraron similitudes directas con el módulo de gestión de incidencias desarrollado por otro compañero. Se consideró, por tanto, que no era necesario replicar funcionalidad cuando el entorno de trabajo era el mismo.

Cabe destacar que aunque no se eliminó totalmente la funcionalidad planteada desde un inicio, sí que fue adaptada o repartida entre los demás colaboradores para que el cliente no se viese perjudicado en la medida de lo posible. Sí se modificó, por tanto, el alcance de este presente proyecto.

Capítulo 5. Análisis

5.1 Definición del Sistema

5.1.1 Determinación del Alcance del Sistema

El proyecto consta de dos modularidades bien diferenciadas. Por un lado encontramos el módulo de gestión de dietas que controla todo lo relativo a los gastos que los empleados tienen que asumir durante el desarrollo de sus funciones, para que posteriormente les sea devuelta esa cantidad acorde a lo dictado por el convenio laboral.

Por el otro lado tenemos el módulo de gestión del calendario laboral del empleado, donde los usuarios podrán indicar sus bajas, permisos o vacaciones. En función de lo dictado por el convenio laboral, cada tipo de permiso tiene sus características y se tramitan y gestionan de una manera diferente.

Todo esto está envuelto en un entorno de trabajo común donde se integran otras modularidades independientes a las antes descritas (gestión de copias de seguridad, gestión de incidencias, gestión de dominios…) de las que no se tiene control. Entre dichos módulos encontramos el de gestión de usuarios, en la que se podrán crear nuevos usuarios, asignarles perfiles y permisos concretos, especificar qué partes de la aplicación global pueden utilizar… Como ya se ha explicado, no se tiene control sobre esta modularidad, sin embargo colabora en su elaboración mediante conversaciones periódicas con su desarrollador, en las que se expondrán las necesidades de cada uno para llegar a un consenso común.

La aplicación estará desplegada sobre un servidor Apache Tomcat proporcionado por la empresa (cliente demandante de la aplicación) pero que deberá ser configurado por los desarrolladores, en función de las necesidades que tenga que cubrir.

Del mismo modo, se utilizará el servidor de base de datos proporcionado por la empresa en que existirá una base de datos utilizada por todos los módulos de la aplicación global. Dicha base de datos será también gestionada por los desarrolladores, aunque para su diseño se contará con la ayuda de los empleados de la empresa, más experimentados, para así evitar graves problemas de inicio.

Para lograr la satisfacción de la empresa cliente se realizarán informes periódicos de seguimiento en los que se expondrá el estado actual del desarrollo del proyecto, se mostrarán avances y prototipos y se establecerán qué cambios deben realizarse sobre el mismo. Estos informes podrán consistir en reuniones presenciales con las personas designadas por la empresa a tal efecto o simplemente en intercambio de correos electrónicos en los que se establezcan los puntos antes mencionados.

Estos informes y reuniones pueden provocar retrasos en el proyecto, puesto que por falta de comunicación o entendimiento es posible que haya que retroceder en el desarrollo, modificar

funcionalidad ya implementada para replantearla de nuevo, eliminar funcionalidad completa o incluir nuevas operaciones no contempladas anteriormente.

Se tiene en cuenta que dichos retrasos son parte del proceso de desarrollo de software para un cliente real y que es algo habitual. Este es uno de los motivos por los que este proyecto resulta interesante para el desarrollador, puesto que es lo más parecido a lo que se va a encontrar una vez en el mundo laboral. Por muchos inconvenientes que puedan surgir durante el desarrollo del proyecto, serán inconvenientes con una base real, enfocados a cubrir las necesidades de un cliente real que necesita la aplicación para su entorno de trabajo real.

5.2 Requisitos del Sistema

5.2.1 Obtención de los Requisitos del Sistema

5.2.1.1 Requisitos funcionales

5.2.1.1.1 Módulo de gestión del calendario laboral

Código	Nombre requisito	Descripción del requisito
RF1.1	Solicitar permiso	Se debe poder solicitar el permiso especificando su tipo, duración y descripción.
RF1.2	Eliminar permiso	Se debe poder eliminar una solicitud de permiso ya existente.
RF1.3	Modificar permiso	Se debe poder modificar el permiso solicitado siempre y cuando las condiciones del mismo lo permitan.
RF1.4	Modificar bolsa de vacaciones	Debe ser posible incrementar o decrementar el número de días que un empleado puede utilizar como vacaciones.
RF1.5	Aceptar solicitudes	Los administradores del sistema deben poder aprobar las solicitudes de permiso.
RF1.6	Denegar solicitudes	Los administradores del sistema deben poder denegar las solicitudes de permiso.
RF1.7	Visualizar calendario por mes	Debe ser posible ver un resumen del calendario a nivel mensual.
RF1.8	Insertar nuevo tipo de permiso	Será posible que los administradores creen nuevos tipos de permiso especificando sus características y duración.
RF1.9	Eliminar tipo de permiso	Será posible que los administradores eliminen tipos de permiso ya existentes.
RF1.10	Modificar tipo de permiso	Será posible que los administradores modifiquen los tipos de permiso existentes.

5.2.1.1.2 Módulo de gestión de dietas

Código	Nombre requisito	Descripción del requisito
RF2.1	Insertar dieta	Se debe crear una nueva dieta con los parámetros y conceptos indicados por el usuario.
RF2.2	Adjuntar fichero	Debe ser posible incorporar ficheros a las dietas (papeles escaneados o fotografiados, facturas, documentos…).
RF2.3	Modificar dieta	Debe ser posible modificar los datos introducidos en la creación de la dieta antes de que haya sido aprobada.
RF2.4	Eliminar dieta	Debe ser posible eliminar una dieta antes de que haya sido aprobada.
RF2.5	Visualizar informe	Debe ser posible visualizar todas las dietas y documentos asociados reportados por cada empleado a nivel mensual.

RF2.6	Supervisar dieta	Los administradores deben poder realizar acciones correctoras sobre las mismas (modificaciones, desestimaciones…)
RF2.7	Aprobar dieta	Los administradores deben poder aceptar la dieta enviada, estableciendo así su conformidad con la misma antes de su entrega definitiva.
RF2.8	Generar informe	El informe mensual de las dietas del empleado debe ser generado y descargado en formato PDF.
RF2.9	Insertar convenio	Los administradores podrán crear nuevos convenios especificando sus características.
RF2.10	Eliminar convenio	Los administradores podrán eliminar los convenios existentes.
RF2.11	Modificar convenio	Los administradores podrán modificar los detalles de los convenios existentes.

5.2.1.2 Requisitos no funcionales

5.2.1.2.1 Requisitos de usuario

Código	Nombre requisito	Descripción del requisito
RN1.1	Conocimientos previos	El usuario debe saber desenvolverse con un dispositivo informático y el funcionamiento general de un navegador web.
RN1.2	Empleado	El usuario debe ser empleado de la empresa.

5.2.1.2.2 Requisitos de organización empresarial

Código	Nombre requisito	Descripción del requisito
RN2.1	Composición de departamentos	La empresa debe conocer qué departamentos existen y qué personas integran cada uno.
RN2.2	Jerarquía	La empresa debe conocer qué personas deben disponer de qué permisos y nivel de privilegios.
RN2.3	Composición de grupos de trabajo	La empresa debe conocer qué grupos de trabajo existen y qué personas integran cada uno.
RN2.4	Reglas de evitación de conflictos de calendario	La empresa debe conocer qué reglas se establecen para que un grupo de trabajo no quede paralizado por la ausencia de sus integrantes.
RN2.5	Flujo de aprobación de calendario	La empresa debe conocer qué personas deben encargarse de la aprobación del calendario laboral de cada empleado.
RN2.6	Flujo de aprobación de dietas	La empresa debe conocer qué personas deben encargarse del supervisado y aprobación de las dietas presentadas por cada empleado.

5.2.1.2.3 Requisitos tecnológicos

Código	Nombre requisito	Descripción del requisito
RN3.1	Conexión a Internet	El usuario necesita acceso a Internet para poder utilizar la aplicación.
RN3.2	Navegador web	El usuario debe tener instalado un navegador web compatible en su equipo.

5.2.1.2.4 Requisitos de usabilidad

Código	Nombre requisito	Descripción del requisito
RN4.1	Interfaces amigables	El manejo de la aplicación debe ser sencillo, por lo que se requieren unas interfaces amigables y limpias que faciliten su utilización, acordes a la propuesta del cliente.
RN4.2	Estilos de los informes	Los informes que el usuario puede imprimir deben corresponderse a la solución propuesta por el cliente.

5.2.1.2.5 Requisitos de seguridad

Código	Nombre requisito	Descripción del requisito
RN5.1	Inicio de sesión	Se requiere que el usuario disponga de una cuenta unipersonal en el sistema para poder iniciar sesión en el mismo.
RN5.2	Base de datos segura	Las operaciones de base de datos se harán de manera segura para garantizar su integridad.
RN5.3	Niveles de acceso	Debe garantizarse que cada usuario solo pueda realizar aquellas operaciones que tiene permitidas.
RN5.4	Comunicación segura	Las comunicaciones entre cliente y servidor tienen que estar debidamente protegidas.

5.2.1.2.6 Requisitos de tiempo de respuesta

Código	Nombre requisito	Descripción del requisito
RN5.1	Respuesta servidor web	El tiempo de respuesta de cualquier operación realizada sobre el servidor web debe ser lo más pequeña posible, no obstante también depende de la calidad de la conexión de red y del nivel de carga de trabajo del servidor.
RN5.2	Respuesta servidor base de datos	El tiempo de respuesta de cualquier operación realizada sobre el servidor de base de datos debe ser lo más pequeña posible, no obstante también depende de la calidad de la conexión de red y del nivel de carga de trabajo del servidor.

5.2.2 Identificación de Actores del Sistema

5.2.2.1 Usuario con perfil Personal

- **Descripción:** es aquella persona que trabaja con la aplicación a nivel de usuario final, con las características y funcionalidades propias del perfil Personal, ya desglosadas en apartados anteriores.

- **Interacciones:** está involucrado en las tareas de gestión de su propio calendario y dietas. También puede iniciar y finalizar sesión en el sistema.

5.2.2.2 Usuario con perfil Administrador

- **Descripción:** es aquella persona que trabaja con la aplicación a nivel de usuario final, con las características y funcionalidades propias del perfil Administración ya mencionadas, además de las propias del perfil Personal.
- **Interacciones:** está involucrado en las tareas relacionadas con la visualización de los datos de todos los usuarios sobre gestión de permisos y dietas, así como su aprobación, denegación o edición total.

5.2.2.3 Usuario anónimo

- **Descripción:** es aquella persona que no ha iniciado sesión en la aplicación y que, por tanto, no tiene acceso a la funcionalidad interna de la misma.
- **Interacciones:** este tipo de usuarios sin identificar solo pueden realizar la tarea de iniciar sesión en el sistema y solicitar el restablecimiento de su contraseña.

5.2.3 Especificación de Casos de Uso

En los siguientes diagramas se han omitido los actores “Servidor web” y “Servidor de base de datos” por intentar mantener limpio el esquema. No obstante, ambos actores participan en todas las acciones, puesto que tienen que intervenir para poder realizar con éxito las tareas solicitadas por el usuario.

5.2.3.1 Gestión del calendario laboral

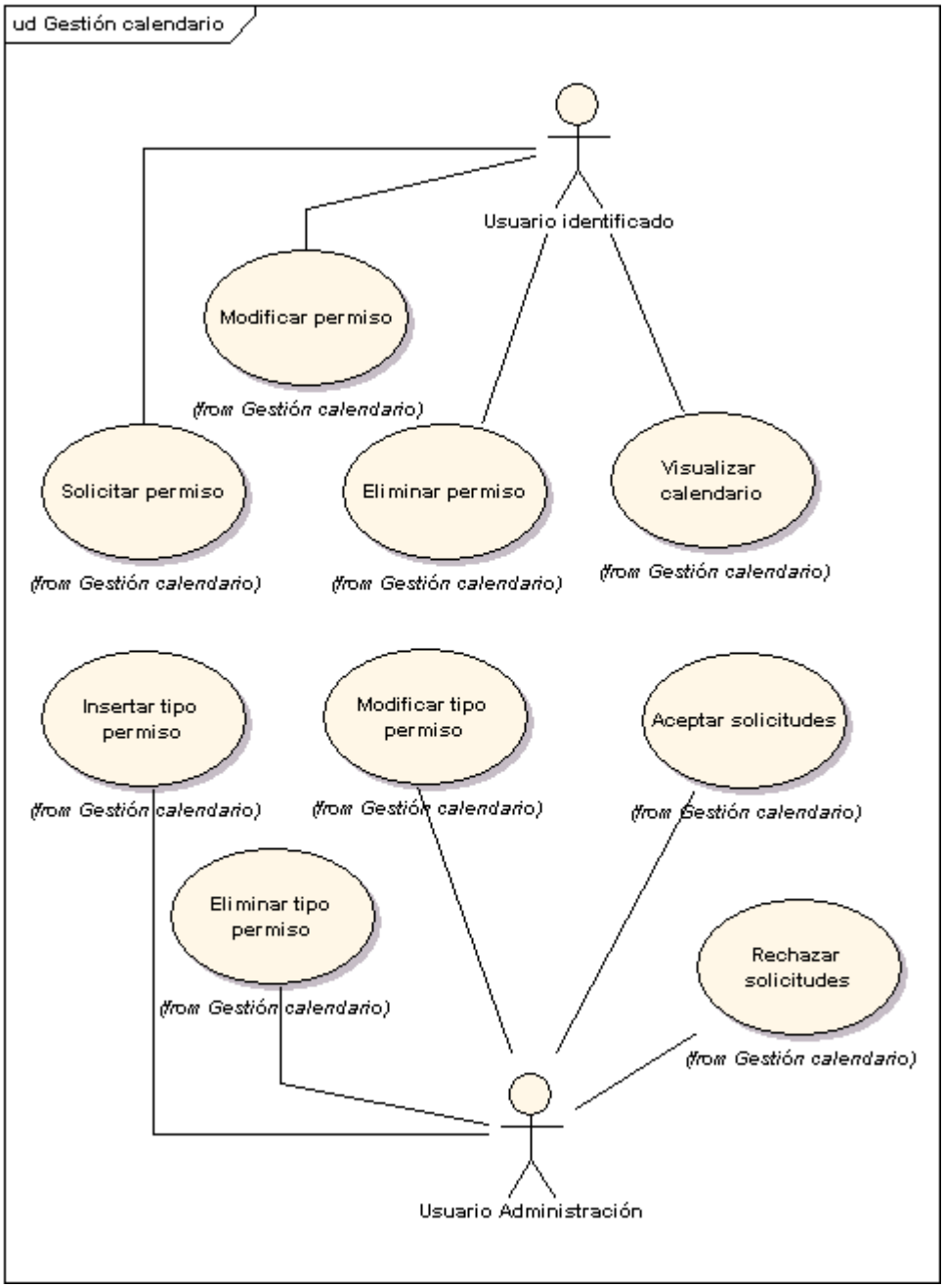


Figura 5.1- Diagrama de casos de uso: Gestión del calendario laboral

Nombre del caso de uso
Solicitar permiso
Descripción
El usuario identificado solicitará un tipo de permiso, indicando cuál es el motivo (vacaciones, asuntos propios, baja laboral…). La aplicación trabaja con el servidor web y el servidor de base de datos para llevar a cabo la operación.

Nombre del caso de uso
Eliminar permiso
Descripción

El usuario identificado seleccionará uno de los permisos solicitados anteriormente para eliminarlo.

Nombre del caso de uso
Modificar permiso
Descripción
El usuario identificado seleccionará uno de los permisos solicitados anteriormente para modificar alguno de sus valores.

Nombre del caso de uso
Visualizar calendario
Descripción
El usuario identificado seleccionará mes del que quiere visualizar sus permisos.

Nombre del caso de uso
Insertar tipo de permiso
Descripción
El usuario administrador agregará al sistema un nuevo tipo de permiso solicitable, así como sus características.

Nombre del caso de uso
Eliminar tipo de permiso
Descripción
El usuario administrador eliminará del sistema un tipo de permiso existente.

Nombre del caso de uso
Modificar tipo de permiso
Descripción
El usuario administrador agregará al sistema un nuevo tipo de permiso solicitable, así como sus características.

Nombre del caso de uso
Aceptar solicitud
Descripción
El usuario administrador aprobará la solicitud de permiso de los usuarios.

Nombre del caso de uso
Rechazar solicitud
Descripción
El usuario administrador rechazará la solicitud de permiso de los usuarios.

5.2.3.2 Gestión de dietas

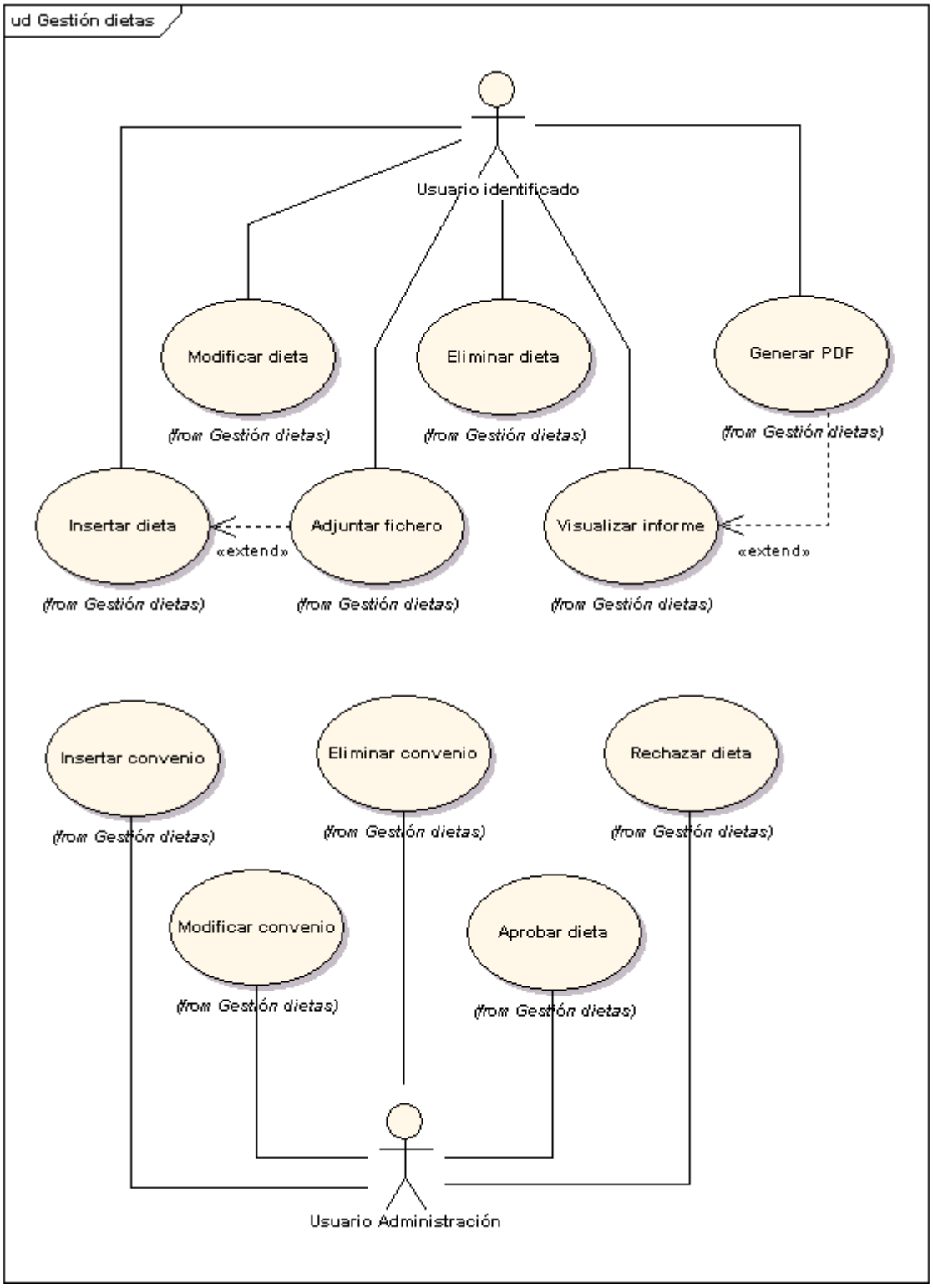


Figura 5.2 – Diagrama de casos de uso: Gestión de dietas

Nombre del caso de uso
Insertar dieta
Descripción
El usuario identificado rellenará el formulario online que le mostrará el servidor web con los datos que considere convenientes. La aplicación trabaja con el servidor web y el servidor de

base de datos para llevar a cabo la operación.

Nombre del caso de uso
Insertar dieta – Adjuntar fichero
Descripción
En adición al rellenado del formulario, el usuario identificado podrá adjuntar uno o varios ficheros que previamente se encuentren en el dispositivo desde el que está trabajando mediante una opción que le permita examinar la estructura de ficheros. El fichero será subido al servidor web.

Nombre del caso de uso
Modificar dieta
Descripción
El usuario identificado podrá modificar las dietas previamente introducidas en el sistema, para cambiar el valor de los campos o su justificación. Esto podrá hacerse antes de que sea aprobada por el supervisor.

Nombre del caso de uso
Eliminar dieta
Descripción
El usuario identificado podrá eliminar las dietas previamente introducidas en el sistema. Esto podrá hacerse antes de que sea aprobada por el supervisor.

Nombre del caso de uso
Visualizar informe
Descripción
El usuario identificado seleccionará la opción que le permita visualizar un informe resumen que englobe todas las dietas indicadas por un usuario específico en un mes concreto.

Nombre del caso de uso
Generar PDF
Descripción
El usuario identificado seleccionará la opción de generar informe de dietas del empleado para obtener una copia en formato PDF de la información

Nombre del caso de uso
Aprobar dieta
Descripción
Los administradores tendrán la posibilidad de aprobar las dietas enviadas por los usuarios.

Nombre del caso de uso
Rechazar dieta
Descripción
Los administradores tendrán la posibilidad de rechazar las dietas enviadas por los usuarios.

Nombre del caso de uso
Insertar convenio
Descripción
Los administradores podrán insertar nuevos convenios asociados a los tipos de dietas existentes, especificando sus características.

Nombre del caso de uso
Eliminar convenio
Descripción
Los administradores podrán eliminar los convenios existentes en el sistema.

Nombre del caso de uso
Modificar convenio
Descripción
Los administradores podrán modificar los convenios existentes en el sistema.

5.3 Identificación de los Subsistemas en la Fase de Análisis

5.3.1 Descripción de los Subsistemas

En la aplicación se distinguen varios subsistemas, cada una de ellas dedicada a una parte específica de la funcionalidad implementada, ya sea a nivel de funcionamiento lógico interno, de interconexión con los diferentes actores o de presentación:

5.3.1.1 Subsistema de presentación

La función de este subsistema es la de establecer entre la aplicación y el usuario final un punto de intercomunicación, es decir, cómo se transmiten información el uno al otro. De esta manera, la presentación proporciona al usuario el acceso a las diferentes funciones que la aplicación pone a su disposición, mostrándole los controles que puede utilizar para cada una de ellas. Del mismo modo se encarga de capturar las órdenes que el usuario ejecuta y la información que introduce para desencadenar la acción a realizar.

5.3.1.2 Subsistema de persistencia

El subsistema de persistencia se encarga de comunicar los diferentes subsistemas de lógica interna con el servidor de base de datos, para leer y escribir la información pertinente que cada funcionalidad desencadena a petición del usuario. Se tratarían de los mecanismos de interconexión entre el servidor web y la base de datos.

5.3.1.3 Subsistema de intranet

Su cometido es el de proporcionar al resto de la funcionalidad una base sobre la que construirse. Se encarga de la identificación del usuario durante todo su paso por la aplicación, así como de proporcionar las restricciones de seguridad a las que el usuario se ve sometido en función de su nivel de acceso.

5.3.1.4 Subsistema de lógica de gestión del calendario

Este subsistema se encarga de realizar las tareas referentes a la gestión del calendario laboral de los empleados tras recibir las órdenes desde el subsistema de presentación. Gracias a la identificación del usuario recibido desde el subsistema de intranet, puede proporcionar unos servicios u otros en función de su nivel de privilegios. Se comunica con el subsistema de persistencia para escribir o leer la información que necesite en la base de datos.

5.3.1.5 Subsistema de lógica de gestión de dietas

Funciona de manera análoga al subsistema de gestión de calendario, recibiendo órdenes e información desde la interfaz y comunicándose con la persistencia, para lograr realizar las tareas relacionadas con la gestión de las dietas de los usuarios.

5.3.2 Descripción de los Interfaces entre Subsistemas

La comunicación entre el subsistema de presentación y los diferentes subsistemas de lógica interna se rige mediante los mecanismos proporcionados por las librerías Java de Struts2, así como de una arquitectura dividida en capas siguiendo el patrón Modelo Vista Controlador, haciendo que las peticiones recibidas en el navegador sean atendidas por controladores, llamados Servlets, que desencadenarán la ejecución de las clases Action encargadas de su procesamiento.

La comunicación con el subsistema de persistencia se realiza dentro de la misma aplicación mediante el habitual paso de parámetros de Java entre clases e interfaces. Éste, a su vez, se comunica con el servidor de base de datos mediante el controlador de comunicación proporcionada por JDBC.

Todo ello se realiza gracias a la comunicación existente con el subsistema de intranet, que se encarga de identificar y colocar al usuario y sus atributos en sesión, introduciendo sus datos en una *cookie* para su posterior identificación.

5.4 Diagrama de Clases Preliminar del Análisis

5.4.1 Diagrama de Clases

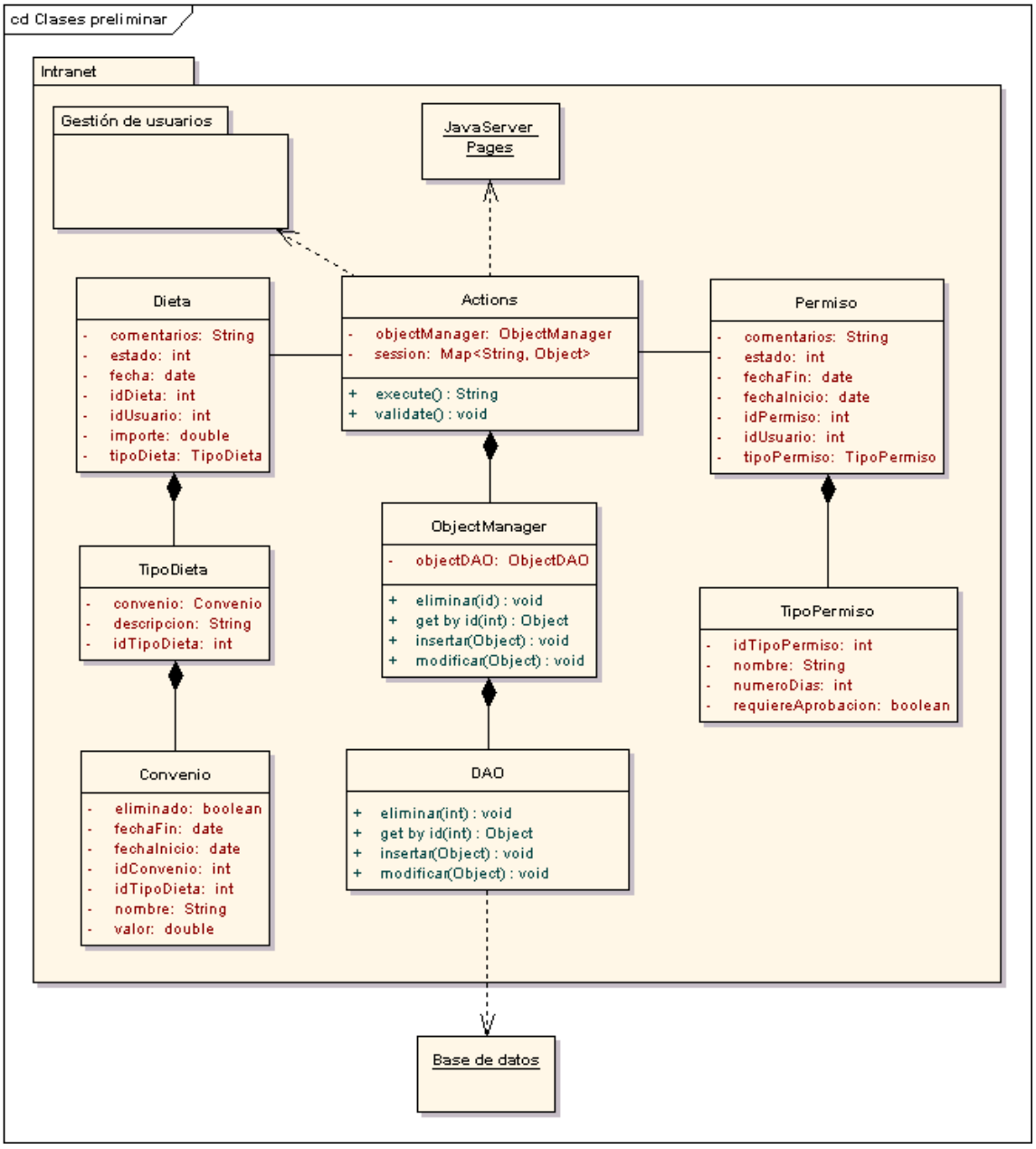


Figura 5.3 Diagrama de clases preliminar

En este diagrama preliminar de clases podemos ver que los Actions, las clases que se encargan de tramitar en primer lugar la información que el usuario visualiza o escribe en las JavaServer Pages, hacen uso de la gestión de usuarios incluida en la intranet para obtener los datos del usuario realizando la operación: identificación y si tiene acceso a lo que está intentando realizar. Esa información es utilizada luego para poder trabajar a lo largo del resto de las capas y clases de la arquitectura planteada.

5.4.2 Descripción de las Clases

Será necesario tener en cuenta que esto es un modelo simplificado para entender de un simple vistazo la arquitectura de clases de la aplicación. Por ejemplo, se han englobado en una clase general "Actions" todas las clases que realizan una función similar. Del mismo modo, se utiliza "Object" de manera genérica para referirse a las diferentes entidades del modelo y evitar así complicar el diagrama con información redundante.

5.4.2.1 Subsistema de intranet

Este subsistema, fuera del control y la gestión de este proyecto, es el que engloba toda la lógica y arquitectura general bajo la que se sustenta el resto de subsistemas.

5.4.2.2 Subsistema de persistencia

Nombre de la Clase
DAO
Descripción
Clase que representa de manera genérica a los diferentes objetos de acceso a datos que se comunican directamente con la base de datos.
Responsabilidades
Se encarga de implementar los métodos CRUD (crear, obtener, actualizar y borrar, en sus siglas en inglés) o aquellos que se consideren necesarios de acceso a la base de datos. Recibe la información con la que debe trabajar y la prepara para lectura o escritura en base de datos en función a los mismos.
Atributos Propuestos
Ninguno.
Métodos Propuestos
• **eliminar (int)**: Recibe el identificador del objeto y lo elimina de la base de datos. • **get by id (int):** Recibe el identificador del objeto y busca en la base de datos a qué hace referencia. Una vez encontrado lo retorna. • **insertar (Object):** Recibe el objeto, con toda su información, y lo inserta en la base de datos. • **modificar (Object):** Recibe el objeto, con toda su información, busca en la base de datos a qué hace referencia y lo actualiza.

5.4.2.3 Subsistema de presentación

En este subsistema encontramos lo que supone las interacciones del usuario con la aplicación a través de las JavaServer Pages. En ellas se muestra la información solicitada de la base de datos, obtenida a través de las capas de la arquitectura, así como los controles para realizar peticiones y escrituras a la misma. Dichas interacciones son manejadas a través de los Actions.

Nombre de la Clase
Actions
Descripción
Clase que representa de manera genérica los diferentes Actions que se encargan de recoger

las solicitudes del usuario y tramitarlas para pasarle las órdenes a la siguiente capa de la arquitectura, así como preparar la información para mostrarla en el navegador del usuario.
Responsabilidades
Recibir las peticiones del usuario y procesarlas para solicitar la información a la lógica de la aplicación. También valida que los datos introducida por el usuario sean correctos.
Atributos Propuestos
• **objectManager**: Objeto genérico que representa a todos los Manager de los diferentes objetos que necesite cada Action, y se encarga de realizar las diferentes llamadas a la capa de persistencia para llevar a cabo las operaciones solicitadas por el usuario. • **session:** Contiene los datos de la sesión del usuario (incluida la identificación del mismo), y es utilizado también para transportar cuando sea necesario la información a las JavaServer Pages.
Métodos Propuestos
• **execute():** Este método es el ejecutado por defecto cuando Struts llama al Action encargado de gestionar una acción del usuario concreta. Recoge la información y la prepara para transmitírsela a la siguiente capa de la arquitectura a través del objectManager. • **validate():** Este método, si declarado, es llamado antes del execute(), y su función es la de verificar si los datos introducidos por el usuario son adecuados y pueden ser utilizados.

5.4.2.4 Subsistema de lógica de gestión de calendario

Nombre de la Clase
ObjectManager
Descripción
Clase que representa a todos los manejadores de los diferentes objetos del modelo de la aplicación que necesiten acceso a datos.
Responsabilidades
Representar un puente lógico entre la capa de presentación en la que el usuario solicita información y la capa de persistencia donde se obtiene. Hace las llamadas oportunas a los objetos DAO para conseguirlo.
Atributos Propuestos
• **objectDAO**: Objeto genérico que representa todos los DAO de los diferentes objetos que necesite cada ObjectManager, y se encarga de realizar las diferentes llamadas a los métodos que trabajan directamente con la base de datos.
Métodos Propuestos
• **eliminar (int)**: Recibe el identificador del objeto y hace la llamada correspondiente al método de eliminación del DAO a través del objectDAO. • **get by id (int):** Recibe el identificador del objeto y hace la llamada correspondiente al método de lectura del DAO a través del objectDAO. • **insertar (Object):** Recibe el identificador del objeto y hace la llamada correspondiente al método de inserción del DAO a través del objectDAO. • **modificar (Object):** Recibe el objeto, con toda su información, y hace la llamada correspondiente al método de modificación del DAO a través del objectDAO.

Nombre de la Clase
Permiso
Descripción

Clase que representa a los permisos que el usuario solicita a la aplicación.
Responsabilidades
Contener la información relativa al permiso y ofrecer métodos de lectura y escritura de sus atributos.
Atributos Propuestos
• **comentarios**: Cadena de texto que recoge la información adicional que el usuario quiera aportar de manera libre al permiso solicitado. • **estado:** Número entero que representa al estado en el que se encuentra el permiso solicitado (solicitado, autorizado, denegado). • **fechaInicio:** Momento puntual en el tiempo que indica qué día comienza el permiso solicitado. • **fechaFin:** Momento puntual en el tiempo que indica qué día termina el permiso solicitado. • **idPermiso:** Número entero utilizado como identificador único del permiso. • **idUsuario:** Número entero utilizado como identificador único del usuario solicitante del permiso. • **tipoPermiso:** Objeto que representa de qué tipo es el permiso solicitado.
Métodos Propuestos
Ninguno a destacar, además de los "getters" y "setters" de los atributos.

Nombre de la Clase
TipoPermiso
Descripción
Clase que representa a los tipos de permisos que el sistema tiene registrados.
Responsabilidades
Contener la información relativa al tipo de permiso y ofrecer métodos de lectura y escritura de sus atributos.
Atributos Propuestos
• **idTipoPermiso:** Número entero utilizado como identificador único del tipo de permiso. • **nombre:** Cadena de texto que representa el nombre del tipo de permiso. • **numeroDias:** Número entero que indica la duración máxima en días del tipo de permiso. • **requiereAprobacion:** Booleano que especifica si el tipo de permiso requiere aprobación directa por parte del administrador del sistema.
Métodos Propuestos
Ninguno a destacar, además de los "getters" y "setters" de los atributos.

5.4.2.5 Subsistema de lógica de gestión de dietas

Nombre de la Clase
ObjectManager
Descripción
Clase que representa a todos los manejadores de los diferentes objetos del modelo de la aplicación que necesiten acceso a datos.
Responsabilidades
Representar un puente lógico entre la capa de presentación en la que el usuario solicita información y la capa de persistencia donde se obtiene. Hace las llamadas oportunas a los objetos DAO para conseguirlo.
Atributos Propuestos

• **objectDAO**: Objeto genérico que representa todos los DAO de los diferentes objetos que necesite cada ObjectManager, y se encarga de realizar las diferentes llamadas a los métodos que trabajan directamente con la base de datos.
Métodos Propuestos
• **eliminar (int)**: Recibe el identificador del objeto y hace la llamada correspondiente al método de eliminación del DAO a través del objectDAO. • **get by id (int):** Recibe el identificador del objeto y hace la llamada correspondiente al método de lectura del DAO a través del objectDAO. • **insertar (Object):** Recibe el identificador del objeto y hace la llamada correspondiente al método de inserción del DAO a través del objectDAO. • **modificar (Object):** Recibe el objeto, con toda su información, y hace la llamada correspondiente al método de modificación del DAO a través del objectDAO.

Nombre de la Clase
Dieta
Descripción
Clase que representa a las dietas del usuario en la aplicación.
Responsabilidades
Contener la información relativa a la dieta y ofrecer métodos de lectura y escritura de sus atributos.
Atributos Propuestos
• **comentarios**: Cadena de texto que recoge la información adicional que el usuario quiera aportar de manera libre a la dieta. • **estado:** Número entero que representa al estado en el que se encuentra la dieta (sin validar, validada, autorizada, rechazada). • **fecha:** Momento puntual en el tiempo que indica a qué día hace referencia la dieta. • **idDieta:** Número entero utilizado como identificador único de la dieta. • **idUsuario:** Número entero utilizado como identificador único del usuario autor de la dieta. • **importe:** Número racional que indica la cantidad de dinero a retribuir asociada a la dieta. • **tipoDieta:** Objeto que representa de qué tipo es la dieta.
Métodos Propuestos
Ninguno a destacar, además de los “getters” y “setters” de los atributos.

Nombre de la Clase
TipoDieta
Descripción
Clase que representa a los tipos de permisos que el sistema tiene registrados.
Responsabilidades
Contener la información relativa al tipo de permiso y ofrecer métodos de lectura y escritura de sus atributos.
Atributos Propuestos
• **convenio:** Objeto que representa al convenio asociado al tipo de dieta. • **descripcion:** Cadena de texto utilizada para describir al tipo de dieta. • **idTipoDieta:** Número entero utilizado como identificador único del tipo de dieta.
Métodos Propuestos
Ninguno a destacar, además de los “getters” y “setters” de los atributos.

Nombre de la Clase
Convenio
Descripción
Clase que representa al convenio al que se ve sometida una dieta en función del tipo de dieta asociado.
Responsabilidades
Contener la información relativa al convenio y ofrecer métodos de lectura y escritura de sus atributos.
Atributos Propuestos
• **eliminado:** Booleano que representa si el convenio ha sido eliminado del sistema. • **fechaInicio:** Momento puntual en el tiempo que indica qué día comienza el permiso solicitado. • **fechaFin:** Momento puntual en el tiempo que indica qué día termina el permiso solicitado. • **idConvenio:** Número entero utilizado como identificador único del convenio. • **idTipoDieta:** Número entero utilizado como identificador único del tipo de dieta asociado al convenio. • **nombre:** Cadena de texto utilizada para otorgar un nombre descriptivo al convenio. • **valor:** Número racional que especifica a qué cantidad de dinero se ve transformada la dieta en función de su tipo.
Métodos Propuestos
Ninguno a destacar, además de los “getters” y “setters” de los atributos.

5.5 Análisis de Casos de Uso y Escenarios

En esta sección se describirán los casos de uso identificados anteriormente de forma detallada, a través de sus escenarios. Los escenarios describen las interacciones entre los usuarios y el sistema e incluyen información acerca de los objetivos, expectativas, motivaciones, acciones y reacciones que se llevan a cabo.

Nótese que no todos los casos de uso llevan asociados un diagrama de robustez, sino que solo se muestran los más ilustrativos, puesto que el proceso de muchos de ellos es similar entre sí o excesivamente trivial como para requerir una explicación adicional.

5.5.1 Casos de uso: gestión de calendario

5.5.1.1 Solicitar permiso

Solicitar permiso	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo.
Poscondiciones	• El permiso debe aparecer en la lista de solicitudes para su posterior gestión por parte del administrador.
Actores	• Usuario.
Descripción	El usuario: 1. Rellenará el formulario de solicitud de permiso con los campos requeridos. El sistema: 2. Comprobará si es posible realizar la solicitud en base a las condiciones de cada permiso. 3. Reflejará en base de datos la solicitud de permiso. 4. Actualizará la interfaz para notificar al usuario.
Variaciones	• **Escenario alternativo 1:** el usuario no puede realizar la solicitud de ese permiso por no cumplir las condiciones del mismo. ○ Notificar el hecho al usuario. ○ Volver al paso 1 del escenario principal.
Excepciones	• **La base de datos no está disponible:** no se puede escribir la solicitud del empleado. ○ Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. ○ Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	

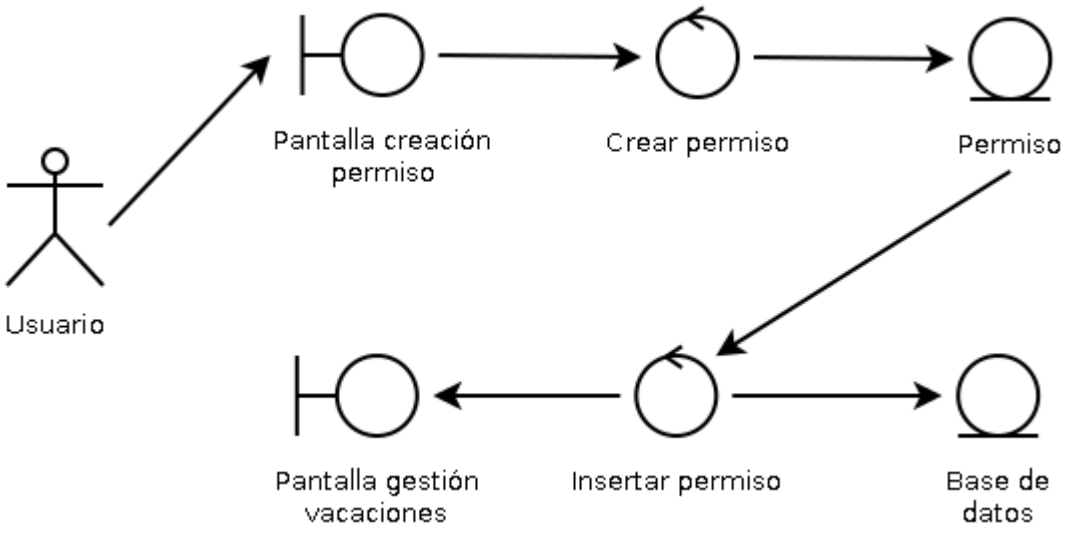


Figura 5.4 - Diagrama de robustez: Solicitar permiso

5.5.1.2 Eliminar permiso

Eliminar permiso	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • La solicitud debe existir en el sistema y no ser anterior al día actual.
Poscondiciones	• La solicitud de permiso debe dejar de aparecer en interfaz y base de datos. • En caso de ser una solicitud de vacaciones, la bolsa de días disponibles debe incrementarse en la cantidad equivalente.
Actores	• Usuario.
Descripción	El usuario: 1. Seleccionará el permiso que desea eliminar. El sistema: 2. Reflejará en base de datos la eliminación de la solicitud. 3. Actualizará la interfaz para notificar al usuario.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede escribir la eliminación de ese día al empleado. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos.
Notas	

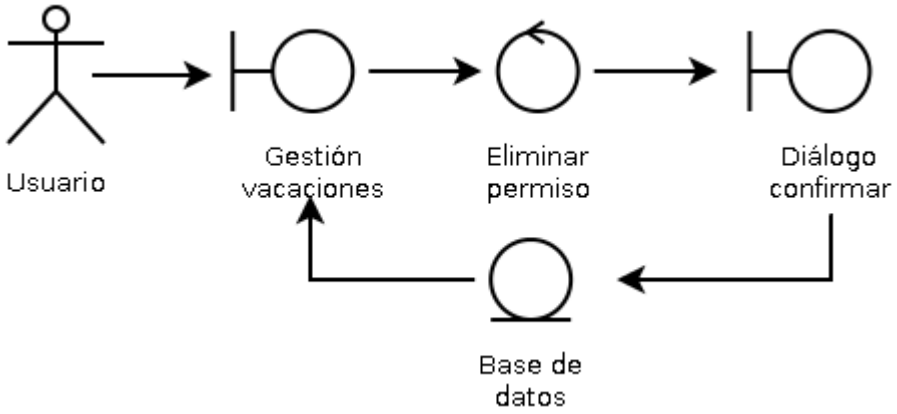


Figura 5.5 - Diagrama de robustez - Eliminar permiso

5.5.1.3 Modificar permiso

Modificar permiso	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • La solicitud debe existir en el sistema y no ser anterior al día actual.
Poscondiciones	• La solicitud de permiso reflejar los cambios realizados en interfaz y base de datos. • En caso de ser una solicitud de vacaciones, deberá pasar a estado no autorizado y actualizar en consecuencia la bolsa de días de vacaciones disponible.
Actores	• Usuario.
Descripción	El usuario: 1. Seleccionará el permiso que desea modificar. 2. Realizará los cambios que considere oportunos. El sistema: 3. Reflejará en base de datos los cambios de la solicitud. 4. En caso de ser una solicitud de vacaciones, modificará su estado y actualizará la bolsa. 5. Actualizará la interfaz para notificar al usuario.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede escribir la eliminación de ese día al empleado. ○ Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. ○ Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	

5.5.1.4 Visualizar calendario

Seleccionar visualización	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo
Poscondiciones	• Se muestra al usuario el calendario solicitado.
Actores	• Usuario.
Descripción	El usuario: 1. Seleccionará el mes que quiere ver. El sistema: 2. Leerá de la base de datos la información relacionada con la fecha escogida. 3. Actualizará la interfaz con la nueva visualización.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información relacionada. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos.
Notas	

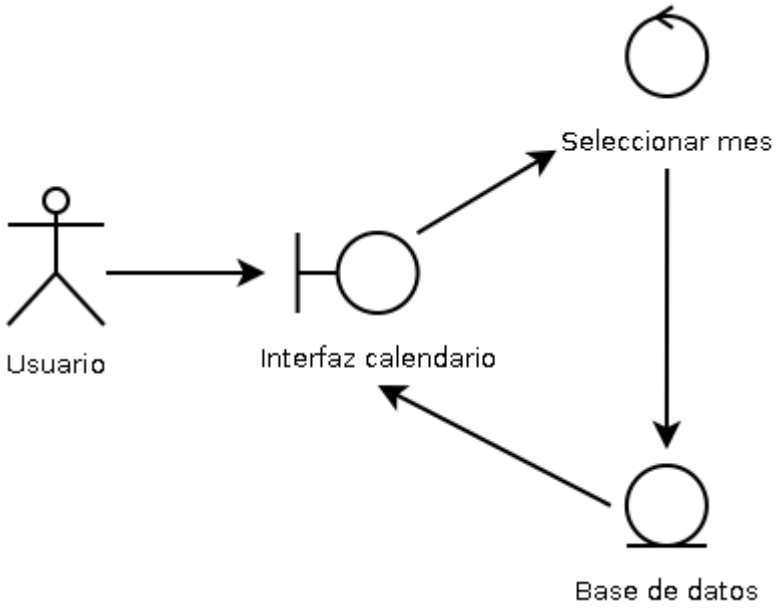


Figura 5.6 - Diagrama de robustez - Visualizar calendario

5.5.1.5 Insertar tipo de permiso

Insertar tipo de permiso	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del grupo Administradores.
Poscondiciones	• La base de datos refleja el nuevo tipo de permiso insertado, que

	pasará a ser seleccionable a la hora de realizar una solicitud.
Actores	• Administrador.
Descripción	El administrador: 1. Rellenará el formulario de inserción de tipo de permiso con los campos requeridos. El sistema: 2. Insertará en la base de datos el tipo de permiso.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información relacionada. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. o Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	

5.5.1.6 Eliminar tipo de permiso

Eliminar tipo de permiso	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del grupo Administradores. • El permiso a eliminar debe estar presente en el sistema.
Poscondiciones	• El tipo de permiso eliminado no puede volver a ser seleccionado durante una solicitud. • Los permisos afectados reflejan la invalidez de su tipo de permiso para que el usuario los actualice en consecuencia.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará el tipo de permiso que desea eliminar y pulsará el botón a tal efecto. El sistema: 2. Actualizará la base de datos para reflejar que ese tipo de permiso pasa a estar eliminado.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información relacionada. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos.
Notas	

5.5.1.7 Modificar tipo de permiso

Modificar tipo de permiso	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del grupo Administradores. • El permiso a modificar debe estar presente en el sistema.
Poscondiciones	• Los nuevos datos del permiso aparecen en interfaz y base de datos.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará el tipo de permiso que desea modificar y pulsará el botón a tal efecto. 2. Editará los campos que considere oportunos. El sistema: 3. Actualizará la base de datos para reflejar los nuevos datos del tipo de permiso.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información relacionada. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. o Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	

5.5.1.8 Aceptar solicitud

Aceptar solicitud	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del grupo Administradores. • El permiso debe requerir autorización. • El permiso no debe ser pasado. • Si se trata de un permiso de vacaciones, el usuario solicitante deberá poseer en su bolsa la cantidad de días requerida.
Poscondiciones	• La solicitud de permiso aparece aprobada. • Si se trata de un permiso de vacaciones, la bolsa de días de vacaciones del usuario solicitante se ha decrementado en consecuencia
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará el tipo de permiso que desea aprobar y pulsará el

	botón a tal efecto. El sistema: 2. Actualizará la base de datos para reflejar el estado del permiso. 3. Si se trata de un permiso de vacaciones, descontará de la bolsa de días disponibles la cantidad solicitada.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información relacionada. ○ Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **No se dan las condiciones necesarias para aceptar la solicitud:** por cuestiones de lógica interna la solicitud no puede ser aceptada pese a que el administrador lo ordene. ○ Notificar al usuario el motivo del problema.
Notas	

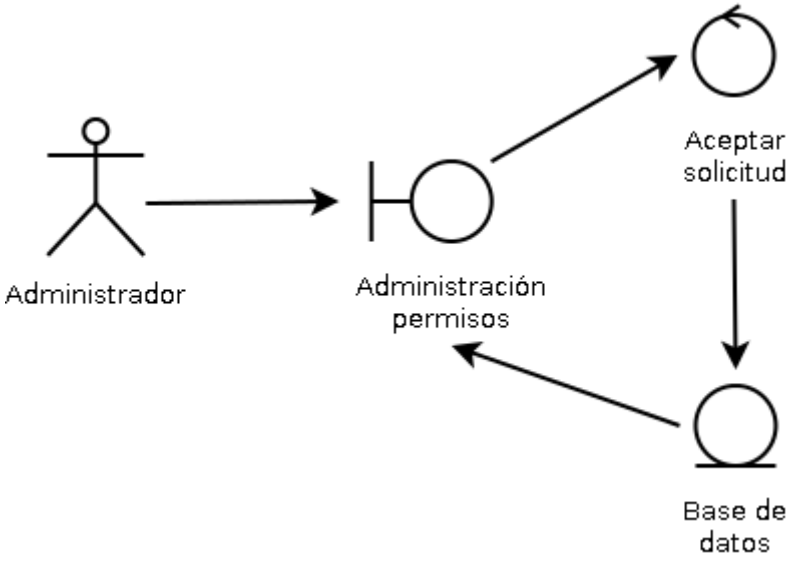


Figura 5.7 - Aceptar solicitud

5.5.1.9 Rechazar solicitud

Rechazar solicitud	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del grupo Administradores. • El permiso debe requerir autorización. • El permiso no debe estar ya autorizado. • El permiso no debe ser pasado.
Poscondiciones	• La solicitud de permiso aparece rechazada.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará el tipo de permiso que desea rechazar y pulsará el

	botón a tal efecto. El sistema: 2. Actualizará la base de datos para reflejar el estado del permiso.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información relacionada. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos.
Notas	

5.5.1.10 Actualizar bolsa de días de vacaciones

Actualizar bolsa de días de vacaciones	
Precondiciones	• El usuario debe estar validado en el sistema y tener permiso sobre el módulo. • El usuario debe formar parte del grupo Administradores.
Poscondiciones	• La bolsa de días de vacaciones reflejará el nuevo valor indicado.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará el usuario al que desea editar la bolsa de vacaciones. 2. Introducirá el nuevo valor. El sistema: 3. Actualizará la base de datos en función del valor introducido.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información relacionada. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. o Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	

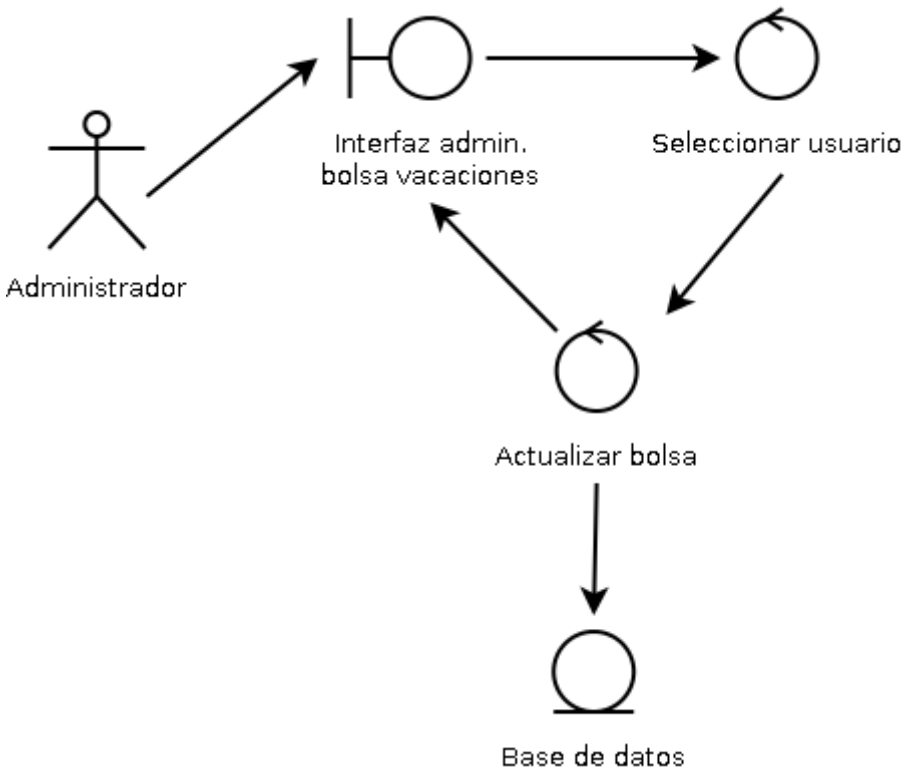


Figura 5.8 Diagrama de robustez - Actualizar bolsa de vacaciones

5.5.2 Casos de uso: gestión de dietas

5.5.2.1 Insertar dieta

Insertar dieta	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe disponer en su equipo de la imagen que justifique la dieta.
Poscondiciones	• La dieta debe estar insertada en la base de datos. • Los importes totales correspondientes deben estar actualizados. • La imagen debe estar alojada en el servidor web.
Actores	• Usuario.
Descripción	El usuario: 1. Rellenará los campos que necesite del formulario que tendrá en pantalla. 2. Navegará por su estructura local de ficheros para encontrar la fotografía asociada a la dieta. 3. Pulsará sobre el botón “Aceptar”. El sistema: 4. Recogerá la información introducida por el usuario. 5. Comprobará la validez de los datos introducidos. 6. Calculará el importe de la dieta en base a los valores

	introducidos. 7. Escribirá en la base de datos la dieta introducida. 8. Guardará en el servidor la fotografía adjunta.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. o Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	-

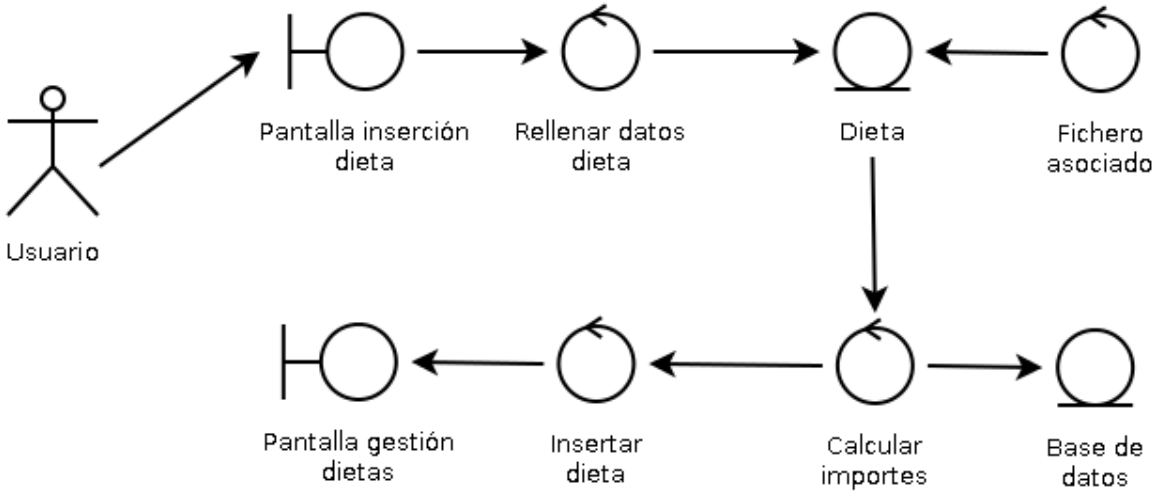


Figura 5.9 - Diagrama de robustez - Insertar dieta

5.5.2.2 Eliminar dieta

Eliminar dieta	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • La dieta no debe haber sido ya autorizada por el administrador
Poscondiciones	• La dieta debe dejar de existir en la base de datos. • Los importes totales deben estar actualizados. • El fichero adjunto a la dieta debe dejar de existir en el servidor.
Actores	• Usuario.
Descripción	El usuario: 1. Seleccionará la dieta a eliminar y pulsará el botón a tal efecto. El sistema: 2. Eliminará la dieta de la base de datos. 3. Recalculará los importes totales en base a la desaparición de la dieta. 4. Eliminará el fichero asociado a la dieta recién eliminada.

Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta. ○ Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos.
Notas	El administrador puede eliminar dietas de los demás usuarios. El proceso es el mismo.

5.5.2.3 Modificar dieta

Modificar dieta	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • La dieta no debe haber sido ya autorizada por el administrador
Poscondiciones	• La dieta debe reflejar los nuevos valores introducidos.
Actores	• Usuario.
Descripción	El usuario: 1. Seleccionará la dieta a editar y pulsará el botón a tal efecto. 2. Modificará los campos que considere convenientes. El sistema: 3. Actualizará los datos de la dieta en la base de datos. 4. Recalculará los importes totales en base a los nuevos valores de la dieta. 5. Sustituirá la imagen anterior por la nueva en el servidor.
Variaciones	**Si el campo imagen se deja vacío:** 1. La imagen ya existente seguirá siendo la asociada a la dieta en cuestión. **Si la dieta se encuentra rechazada:** 1. Pasará a estado “nueva” para su posterior revisión.

Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta.
	  ○ Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos.
	• **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia.
	  ○ Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.

Notas	El administrador puede modificar dietas de los demás usuarios. El proceso es el mismo.

5.5.2.4 Visualizar informe

Visualizar informe	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo.
Poscondiciones	• La interfaz ha sido actualizada con la visualización seleccionada

	por el usuario.
Actores	• Usuario.
Descripción	El usuario: 1. Seleccionará el mes que desea visualizar. El sistema: 2. Leerá de la base de datos la información solicitada en el mes seleccionado por el usuario. 3. Procesará y preparará la información para mostrarla. 4. Actualizará la interfaz para mostrar el informe mensual.
Variaciones	**Si el usuario dispone de perfil Administrador:** 1. Podrá seleccionar, además del mes, el usuario del que quiere consultar la información.
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información solicitada. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **No hay información:** no existen dietas para los valores solicitados. o Informar al usuario del hecho.
Notas	-

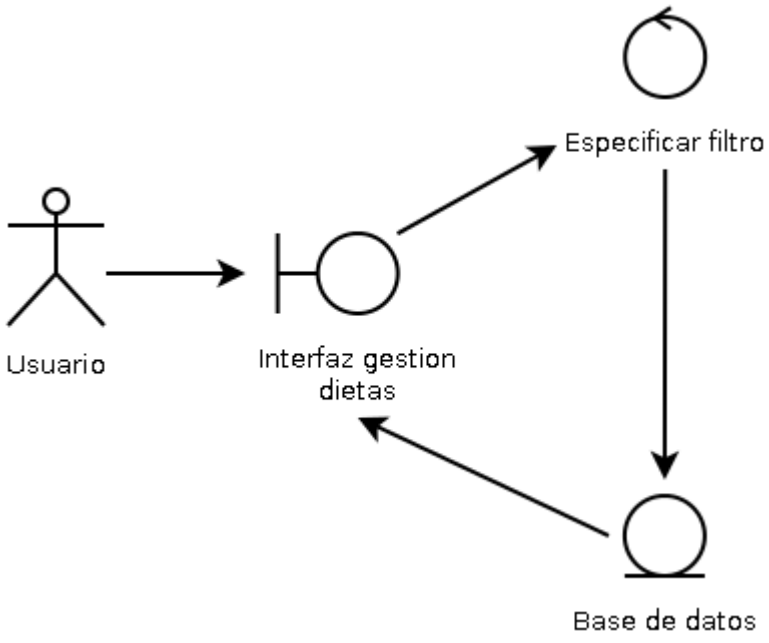


Figura 5.10 Diagrama de robustez - Visualizar informe

5.5.2.5 Generar informe en PDF

Generar informe en PDF	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo.
Poscondiciones	• El informe en formato PDF se crea en el servidor.

	• El informe en formato PDF se descarga en el equipo del usuario.
Actores	• Usuario.
Descripción	El usuario: 1. Pulsará sobre el botón de generar informe del contenido que está visualizando. El sistema: 2. Leerá de la base de datos la información solicitada en el mes seleccionado por el usuario. 3. Procesará y preparará la información para su formateo al documento 4. Almacenará el documento en su estructura de ficheros. 5. Ofrecerá el documento al navegador web del usuario solicitante del informe.
Variaciones	**Si el usuario dispone de perfil Administrador:** 1. Del mismo modo que en la visualización de informe, el administrador puede seleccionar también de qué usuario quiere generar el PDF.
Excepciones	• **La base de datos no está disponible:** no se puede obtener la información solicitada. ○ Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **No hay información:** no existen dietas para los valores solicitados. Informar al usuario del hecho.
Notas	-

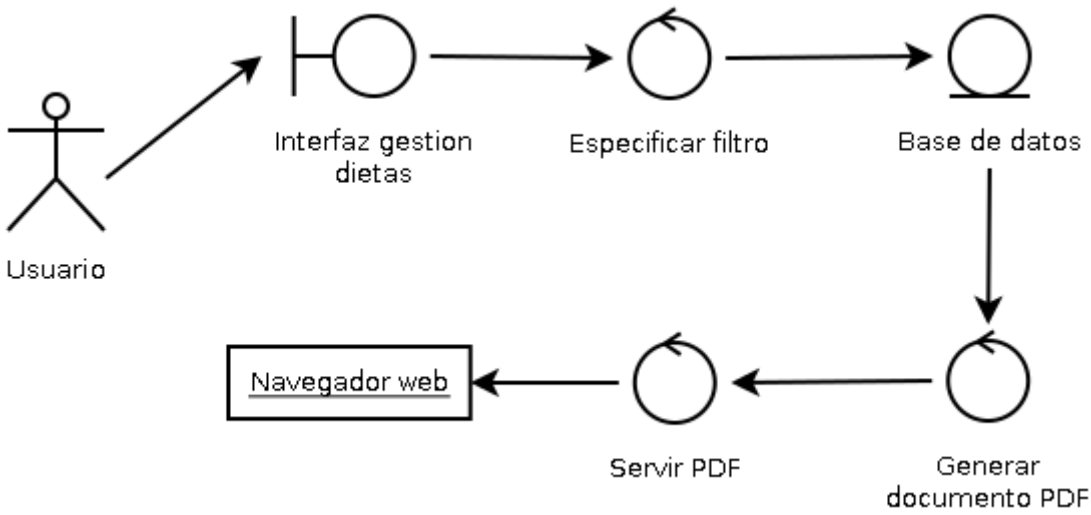


Figura 5.11 - Diagrama de robustez - Generar informe PDF

5.5.2.6 Insertar nuevo convenio

Insertar nuevo convenio	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos

	sobre el módulo. • El usuario debe formar parte del perfil Administradores.
Poscondiciones	• El nuevo convenio debe estar en la base de datos y ser seleccionable en las posteriores inserciones de dieta.
Actores	• Administrador.
Descripción	El administrador: 1. Rellenará los campos del formulario que tendrá en pantalla. 2. Pulsará sobre el botón "Aceptar". El sistema: 3. Recogerá la información introducida por el usuario. 4. Comprobará la validez de los datos introducidos. 5. Escribirá en la base de datos el convenio introducido.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. o Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	-

5.5.2.7 Eliminar convenio

Eliminar convenio	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del perfil Administradores.
Poscondiciones	• El convenio debe aparecer en la base de datos como "eliminado". • Las dietas afectadas por la eliminación del convenio avisarán de ello para que sean modificadas.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará el convenio a eliminar y pulsará el botón a tal efecto. El sistema: 2. Actualizará el convenio en base de datos para reflejar su estado "eliminado". 3. Notificará a las dietas asociadas a ese convenio la invalidez del mismo para su edición.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta. o Notificar al usuario un error asociado al problema

	encontrado y pedirle que lo intente de nuevo pasados unos minutos.
Notas	-

5.5.2.8 Modificar convenio

Modificar convenio	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del perfil Administradores.
Poscondiciones	• El convenio debe reflejar en la base de datos los cambios introducidos. • Las dietas afectadas por la modificación del convenio avisarán de ello para que sean modificadas.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará el convenio a editar y pulsará el botón a tal efecto. 2. Modificará los campos que considere convenientes. El sistema: 3. Recogerá la información introducida por el usuario. 4. Comprobará la validez de los datos introducidos. 5. Actualizará los datos del convenio en la base de datos. 6. Notificará a las dietas que se vean afectadas a los cambios la invalidez del convenio para su edición.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta. ○ Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. ○ Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	-

5.5.2.9 Rechazar dieta

Modificar convenio	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del perfil Administradores. • La dieta no debe estar ya rechazada o aprobada.
Poscondiciones	• La dieta debe aparecer como “rechazada”.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará la dieta a rechazar y presionará el botón a tal

	efecto. El sistema: 2. Actualizará la base de datos para reflejar el nuevo estado de la dieta. 3. Notificará al usuario del rechazo de su dieta para su eliminado o modificación.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. o Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	-

5.5.2.10 Aprobar dieta

Aprobar dieta	
Precondiciones	• El usuario debe estar validado en el sistema y tener permisos sobre el módulo. • El usuario debe formar parte del perfil Administradores. • La dieta no debe estar ya rechazada o aprobada.
Poscondiciones	• La dieta debe aparecer como “aprobada”.
Actores	• Administrador.
Descripción	El administrador: 1. Seleccionará la dieta a aprobar y presionará el botón a tal efecto. El sistema: 2. Actualizará la base de datos para reflejar el nuevo estado de la dieta. 3. Notificará al usuario de la aprobación de su aprobación.
Variaciones	-
Excepciones	• **La base de datos no está disponible:** no se puede insertar la dieta. o Notificar al usuario un error asociado al problema encontrado y pedirle que lo intente de nuevo pasados unos minutos. • **Los datos introducidos no son válidos:** los valores de los campos escritos contienen algún error o incongruencia. o Notificar al usuario la naturaleza del error y pedirle que lo intente nuevamente.
Notas	-

5.6 Análisis de Interfaces de Usuario

Dado que este proyecto se realiza en colaboración con una empresa que actúa como cliente de la aplicación a desarrollar, y dicha empresa ya dispone de un diseñador gráfico, no ha sido necesario realizar un análisis de las interfaces.

La empresa, siguiendo los procedimientos y estilos habituales, y en consonancia con la apariencia general de su web corporativa, diseñó las interfaces de usuario de este proyecto. La tarea del programador fue simplemente adaptar lo máximo posible la interfaz final a las peticiones de la empresa.

Como la responsabilidad del análisis de interfaces recayó en el cliente, no se desglosará en este apartado. No obstante, podrá observarse el resultado final de las interfaces en el Capítulo 6 que desglosa el diseño del sistema.

5.7 Especificación del Plan de Pruebas

En esta sección se desglosa el plan de pruebas a tener en cuenta durante el desarrollo del proyecto. Para cada tarea será necesario especificar los diferentes escenarios que pueden darse y el resultado esperado para cada uno de ellos. También será necesario tener en cuenta los aspectos de integración de la modularidad entre sí y para con el resto de componentes del sistema.

5.7.1 Pruebas unitarias

Debido a la naturaleza de las aplicaciones struts, solo tiene sentido realizar pruebas a las clases del modelo, ya que es quien en última instancia recae todo el peso de la lógica. Se realizarán por tanto pruebas sobre las clases que se encargan del acceso a base de datos, puesto que el resto de la arquitectura solo son llamadas a uno u otro método de dicho nivel.

5.7.1.1 Gestión de calendario de vacaciones

Prueba	Resultado esperado
Insertar permiso	El permiso aparece en base de datos con los datos tal y como deben.
Obtener permiso por identificador	Se recupera de base de datos un objeto permiso con los datos tal y como aparecen en ella.
Obtener permisos según criterios	Se recupera de base de datos una colección de objetos permisos con sus datos correctamente cargados y que responen a los criterios dictados (usuario, tipo, fecha….).
Modificar permiso	Los nuevos datos indicados para el permiso aparecen en base de datos, sobreescribiendo los antiguos.
Eliminar permiso	El permiso ya no existe en la base de datos.
Autorizar permiso	El estado del permiso pasa a ser “autorizado”.
Denegar permiso	El estado del permiso pasa a ser “denegado”.
Insertar tipo de permiso	El tipo de permiso aparece en base de datos con los datos tal y como deben.
Obtener tipo de permiso por identificador	Se recupera de base de datos un objeto tipo permiso con los datos tal y como aparecen en ella.
Obtener todos los tipos de permiso	Se recupera de base de datos una colección de objetos tipo de permiso con sus datos correctamente cargados.
Modificar tipo de	Los nuevos datos indicados para el tipo de permiso aparecen en

permiso	base de datos, sobreescribiendo los antiguos.
Eliminar tipo de permiso	El bit "eliminado" del tipo de permiso pasa a ser "1".
Obtener bolsa de días de vacaciones de usuario	Se recupera de base de datos el objeto bolsa correspondiente al usuario del que se solicita.
Actualizar bolsa de días de vacaciones	El número de días de la bolsa de vacaciones del usuario pasa a ser el indicado, sobreescribiendo el anterior.
Sumar días en bolsa	El número de días de la bolsa de vacaciones del usuario se incrementa en la cantidad dada.
Restar días en bolsa	El número de días de la bolsa de vacaciones del usuario se decrementa en la cantidad dada.

5.7.1.2 Gestión de dietas

Prueba	Resultado esperado
Insertar detalle en dieta nueva	Una nueva dieta aparece en base de datos con los datos indicados. Un nuevo detalle de dieta, asociado a dicha dieta, se introduce en base de datos con la información adecuada. Se calcula el importe según el complemento no salarial aplicado y se actualizan los totales de su dieta asociada.
Insertar detalle en dieta ya existente	Un nuevo detalle de dieta, asociado a la dieta a la que pertenece, se introduce en base de datos con la información especificada. Se calcula el importe según el complemento no salarial aplicado y se actualizan los totales de su dieta asociada.
Obtener dieta por identificador	Se recupera un objeto dieta, así como sus detalles asociados, correspondiente al identificador especificado.
Obtener dietas según criterios	Se recupera una colección de dietas, así como sus respectivos detalles, respondiendo a los criterios indicados (usuario, tipo, fecha…).
Modificar detalle	Los nuevos datos indicados para el detalle aparecen en base de datos, sobreescribiendo los antiguos. Se calcula el importe según el complemento no salarial aplicado y se actualizan los totales de su dieta asociada.
Eliminar último detalle	Desaparece de la base de datos el detalle indicado, así como su dieta asociada.
Eliminar detalle (no último)	Desaparece de la base de datos el detalle indicado. Se recalculan los totales de su dieta asociada.

Cambiar estado de dieta	La dieta refleja en base de datos el nuevo estado indicado.
Obtener tipo de dieta por identificador	Se recupera un objeto tipo de dieta de la base de datos correspondiente al identificador especificado.
Obtener todos los tipos de dieta	Se recupera una colección de objetos tipo de dieta de la base de datos con sus respectivos valores.
Insertar complemento no salarial	La base de datos contiene un nuevo objeto complemento no salarial con los datos indicados.
Obtener complemento no salarial por identificador	Se recupera un objeto complemento no salarial con sus valores de la base de datos correspondiente al indicador especificado.
Obtener todos los complementos no salariales	Se recupera una colección de objetos complemento no salarial de la base de datos, con sus respectivos valores.
Modificar complemento no salarial	Los nuevos datos indicados para el complemento no salarial aparecen reflejados en la base de datos, sobreescribiendo los anteriores.
Eliminar complemento no salarial	El bit “eliminado” del complemento no salarial pasa a ser “1”.

5.7.2 Pruebas de integración

5.7.2.1 Gestión del calendario de vacaciones

5.7.2.1.1 Solicitar permiso

Prueba	Resultado esperado
Crear permiso que no requiere aprobación	El sistema registra la solicitud de permiso y lo establece como ya aprobado.
Crear permiso que requiere aprobación	El sistema registra la solicitud de permiso y lo establece como aún no aprobado.
Crear permiso tipo laborable en días laborables	El sistema registra la solicitud de permiso correctamente.
Crear permiso tipo laborable en fin de semana	El sistema no registra la solicitud de permiso y notifica al usuario del error cometido.
Solicitar vacaciones con suficiente	El sistema registra la solicitud de permiso y la marca

bolsa	como pendiente de aprobación.
Solicitar vacaciones sin suficiente bolsa	El sistema no registra la solicitud de permiso, no resta de la bolsa y notifica al usuario del error.
Solicitar permiso con límite anual ya superado	El sistema no registra la solicitud de permiso y notifica al usuario del error.
Solicitar permiso sin límite anual ya superado	El sistema registra la solicitud de permiso correctamente.
Solicitar permiso por más días del máximo permitido	El sistema no registra la solicitud de permiso y notifica al usuario del motivo.
Solicitar permiso con alguno de los campos vacíos	El sistema no registra la solicitud de permiso y notifica al usuario del error cometido.
Solicitar permiso con fecha de inicio posterior a fecha final	El sistema no registra la solicitud de permiso y notifica al usuario del error cometido.
Solicitar permiso para una fecha ya asignada a otro permiso	El sistema no registra la solicitud de permiso y notifica al usuario del motivo.

5.7.2.1.2 Eliminar permiso

Prueba	Resultado esperado
Eliminar permiso aún no concluido	El permiso se elimina del sistema correctamente.
Eliminar permiso o vacación ya concluido	La operación no está disponible para su ejecución.
Eliminar vacación no aprobada	El permiso se elimina del sistema correctamente.
Eliminar vacación ya aprobada	El permiso se elimina del sistema y la bolsa de vacaciones se incrementa adecuadamente.

5.7.2.1.3 Modificar permiso

Considérese que modificar permiso está supeditado a las mismas restricciones que "insertar permiso" (no puede sobrepasar el límite de días, deben estar todos los campos rellenos…).

Prueba	Resultado esperado
Modificar permiso o vacación ya concluido	La operación no está disponible para su ejecución.
Modificar permiso	El permiso se modifica correctamente.

Modificar vacación (indep. de su estado)	El permiso se modifica, pasa a estado en espera de aprobación y la bolsa se incrementa adecuadamente.

5.7.2.1.4 Visualizar calendario

Prueba	Resultado esperado
Seleccionar mes y año con datos	El calendario se recarga para mostrar todos los permisos del usuario solicitante para la fecha indicada.
Seleccionar mes y año sin datos	El calendario se recarga pero se muestra vacío. Se indica al usuario por qué.

5.7.2.1.5 Insertar tipo de permiso

Prueba	Resultado esperado
Insertar tipo de permiso	El sistema inserta el tipo de permiso correctamente.
Insertar tipo de permiso con algún campo incorrecto	El sistema no registra el tipo de permiso y avisa al usuario del error cometido.

5.7.2.1.6 Eliminar tipo de permiso

Prueba	Resultado esperado
Eliminar tipo de permiso	El sistema actualiza el permiso y lo establece como "eliminado" para evitar su utilización en el futuro, pero no lo borra.

5.7.2.1.7 Modificar tipo de permiso

Prueba	Resultado esperado
Modificar tipo de permiso	El sistema modifica el tipo de permiso correctamente.
Modificar tipo de permiso con algún campo incorrecto	El sistema no registra los cambios sobre el tipo de permiso y avisa al usuario del error cometido.

5.7.2.1.8 Aceptar solicitud

Prueba	Resultado esperado
Aceptar solicitud con suficiente bolsa de vacaciones	El sistema cambia el estado de la solicitud a "aceptado" y resta de la bolsa de vacaciones los días correspondientes.

Aceptar solicitud sin suficiente bolsa de vacaciones	El sistema no cambia el estado de la solicitud, no modifica la bolsa de vacaciones y notifica al usuario del motivo.

5.7.2.1.9 Rechazar solicitud

Prueba	Resultado esperado
Rechazar solicitud	El sistema cambia el estado de la solicitud a "rechazado".

5.7.2.2 Gestión de dietas

5.7.2.2.1 Insertar dieta

Prueba	Resultado esperado
Insertar dieta sin complemento no salarial	El sistema registra los datos de la dieta y almacena el fichero asociado en el servidor. Recalcula valores totales de dieta.
Insertar dieta con complemento salarial	El sistema registra los datos de la dieta y almacena el fichero asociado en el servidor. Calcula valor de dieta según el complemento indicado y recalcula valores totales.
Insertar dieta con alguno de los campos obligatorios vacío o incorrecto	El sistema no registra los datos de la dieta y avisa al usuario del error cometido.

5.7.2.2.2 Modificar dieta

Prueba	Resultado esperado
Modificar dieta especificando nuevo fichero	El sistema registra los cambios de la dieta y almacena el nuevo fichero asociado en el servidor, borrando el anterior. Recalcula valores de dieta.
Modificar dieta sin especificar nuevo fichero	El sistema registra los cambios de la dieta sin modificar el fichero asociado. Recalcula valores de dieta.
Modificar dieta con alguno de los campos obligatorios vacío o incorrecto	El sistema no registra los datos de la dieta y avisa al usuario del error cometido.
Modificar dieta si ya ha sido	La operación no está disponible

autorizada

5.7.2.2.3 Eliminar dieta

Prueba	Resultado esperado
Eliminar dieta	El sistema borra la dieta, recalcula los importes totales y elimina el fichero asociado a la misma.
Eliminar dieta si ya ha sido autorizada	La operación no está disponible.

5.7.2.2.4 Visualizar informe

Prueba	Resultado esperado
Visualizar por defecto	El sistema busca y muestra las dietas del usuario del mes en curso.
Visualizar especificando mes y año	El sistema busca y muestra las dietas del usuario para el mes y año indicados.
Visualizar especificando mes, año y usuario (Administrador)	El sistema busca y muestra las dietas del usuario especificado para el mes y año indicado.

5.7.2.2.5 Generar informe PDF

Prueba	Resultado esperado
Generar informe	El sistema utiliza los criterios especificados en la visualización del informe para generar el documento PDF. Lo almacena en el servidor y se lo sirve al navegador del usuario.

5.7.2.2.6 Aprobar dieta

Prueba	Resultado esperado
Aprobar dieta	El sistema modifica el estado de la dieta y lo establece como “aprobada”.

5.7.2.2.7 Rechazar dieta

Prueba	Resultado esperado
Rechazar dieta	El sistema modifica el estado de la dieta y lo establece

	como “rechazada”.

5.7.2.2.8 Insertar complemento no salarial

Prueba	Resultado esperado
Insertar complemento	El sistema inserta el complemento no salarial y sus datos para su futura utilización.
Insertar complemento con algún campo vacío o incorrecto	El sistema no inserta el complemento no salarial y notifica al usuario del error cometido.
Insertar complemento con fecha de inicio posterior a finalización	El sistema no inserta el complemento no salarial y notifica al usuario del error cometido.

5.7.2.2.9 Modificar complemento no salarial

Téngase en cuenta que la validación de datos y campos debería arrojar los mismos resultados que la inserción de complemento no salarial.

Prueba	Resultado esperado
Modificar complemento	El sistema modifica el complemento no salarial y actualiza las dietas asociadas para recalcular sus importes.

5.7.2.2.10 Eliminar complemento no salarial

Prueba	Resultado esperado
Eliminar complemento	El sistema actualiza el complemento no salarial para marcarlo como “Eliminado” y evitar su uso posterior. Se actualizan las dietas asociadas para recalcular sus importes.

5.7.2.3 Integración en intranet

La integración de las dos modularidades independientes entre sí (gestor de dietas y gestor de calendario laboral) no supone ninguna complejidad, puesto que no comparten nada más que el entorno común o intranet.

Es por eso que en este apartado analizaremos cuál debe ser el resultado de la integración con dicha intranet, que envuelve a ambos módulos y les ofrecen servicios con los que trabajar, aunque no se tenga control directo sobre ellas.

Prueba	Resultado esperado

Otorgar a un perfil permisos de escritura sobre el módulo	Los usuarios con dicho perfil podrán realizar tareas de usuario normal sobre el módulo en cuestión.
Revocar el permiso de escritura a un perfil	Los usuarios con dicho perfil ya no podrán realizar tareas en el módulo.
Otorgar a un usuario permiso de escritura sobre el módulo	El usuario especificado podrá realizar tareas de usuario normal sobre el módulo en cuestión.
Revocar el permiso de escritura a un usuario	El usuario especificado ya no podrá realizar tareas en el módulo, a no ser que su perfil se lo permita.
Insertar a un usuario en el perfil "Administradores"	El usuario en cuestión podrá realizar tareas de usuario administrador en los módulos.

5.7.3 Pruebas de sistema

Este tipo de pruebas comprueban la buena construcción de los mecanismos que comunican el conjunto de todo el sistema con otros sistemas externos.

Prueba	Resultado esperado
Comunicar con el servidor de base de datos	La base de datos responde correctamente a la orden SQL ejecutada.

5.7.4 Pruebas de usabilidad

Como ya se ha comentado en apartados anteriores, el cliente fue el responsable de diseñar las interfaces de usuario del proyecto. Por tanto, y puesto que se considera que el cliente ha estudiado los aspectos relativos a ello, no se realizarán estudios sobre la satisfacción del cliente con la usabilidad de la aplicación a desarrollar.

Capítulo 6. Diseño del Sistema

6.1 Arquitectura del Sistema

6.1.1 Diagrama de Paquetes

Debido a la gran cantidad de clases que tiene una aplicación construida con Struts y bajo el Modelo Vista Controlador, es necesario un diagrama de paquetes que agrupe de manera lógica bloques de funcionalidad similar.

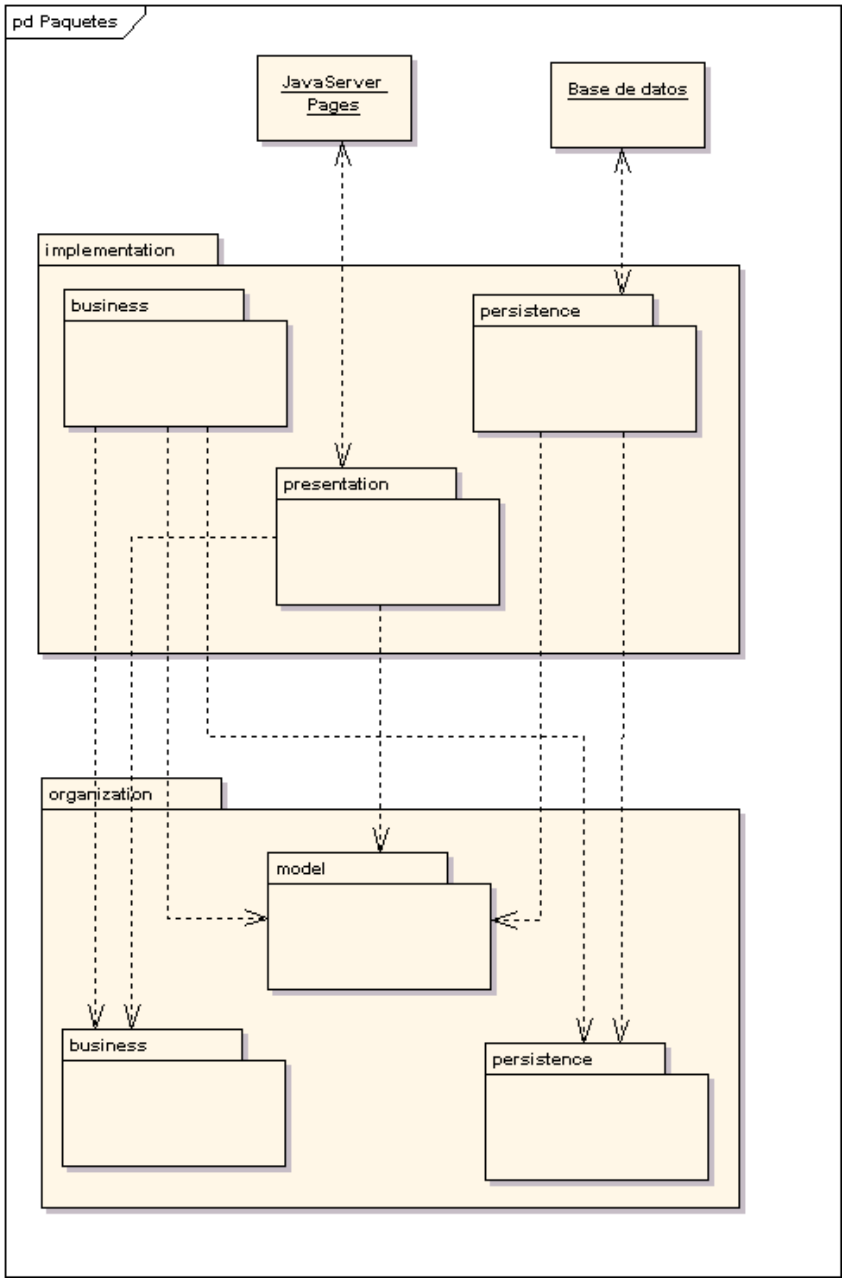


Figura 6.1 - Diagrama de paquetes

Este diagrama, en el que aparentemente hay mucha dependencia entre paquetes, responde directamente a las necesidades del Modelo Vista Controlador.

6.1.1.1 Paquete implementation

Este paquete agrupa los subpaquetes correspondientes a la implementación de las clases, donde se realiza toda la lógica. En su interior encontramos:

6.1.1.1.1 Paquete presentation

Contiene las clases "Action" que colaboran directamente con las JavaServer Pages para recoger datos o mostrarlos. Mediante la implementación de la interfaz "ModelDriven" de struts, los datos se transportan en objetos del modelo, de ahí que necesite colaborar con dicho paquete "organization.model".

Para llamar a la siguiente capa de la arquitectura necesita utilizar las interfaces de objetos contenidos en el paquete "organization.business". Dichas interfaces declaran las signaturas de los métodos que necesitan las Action para llevar a cabo las tareas ordenadas por el usuario.

6.1.1.1.2 Paquete persistence

Agrupa a los Data Access Object (DAO), que son las clases que suministran el acceso y comunicación al sistema de persistencia, en este caso una base de datos. Implementan los métodos de creación, obtención, actualización y borrado de datos necesarios para la ejecución de las tareas solicitadas en la capa de lógica de la aplicación.

Dichas operaciones las realiza utilizando los objetos del paquete "organization.model", de ahí que necesite interactuar con él. Los DAO implementan interfaces del paquete "organization.persistence" que dictan los métodos que deben incorporar, y es por eso que existe una relación en el diagrama de paquetes.

6.1.1.1.3 Paquete business

Este paquete contiene los "Manager" o gestores, que se encargan de llamar a los diferentes métodos de la capa de persistencia según reciba las órdenes de la capa de presentación. Es por eso que necesita colaborar con todos los subpaquetes de "organization", ya que son las clases que ponen a todas las capas en comunicación.

6.1.1.2 Paquete organization

Este paquete agrupa los subpaquetes que contienen los elementos del modelo de la aplicación, así como las interfaces que se implementan desde las correspondientes clases del paquete "implementation". En su interior encontramos:

6.1.1.2.1 Paquete model

Contiene los modelos del sistema, es decir, las clases que en última instancia representan los datos que el usuario maneja en la aplicación. Dichas clases (Dieta, Permiso, TipoPermiso…) contienen únicamente atributos, constructores y "getters" y "setters" de los atributos.

6.1.1.2.2 Paquete persistence

Este paquete agrupa a las interfaces que los objetos de acceso a datos (DAO) tienen que implementar. Solo disponen de las signaturas de los métodos para especificar su nombre, parámetros y tipo de retorno.

6.1.1.2.3 Paquete business

Contiene las interfaces que los "Manager" deben implementar para llevar a cabo sus funciones. Solo disponen de las signaturas de los métodos.

6.1.2 Diagrama de Componentes

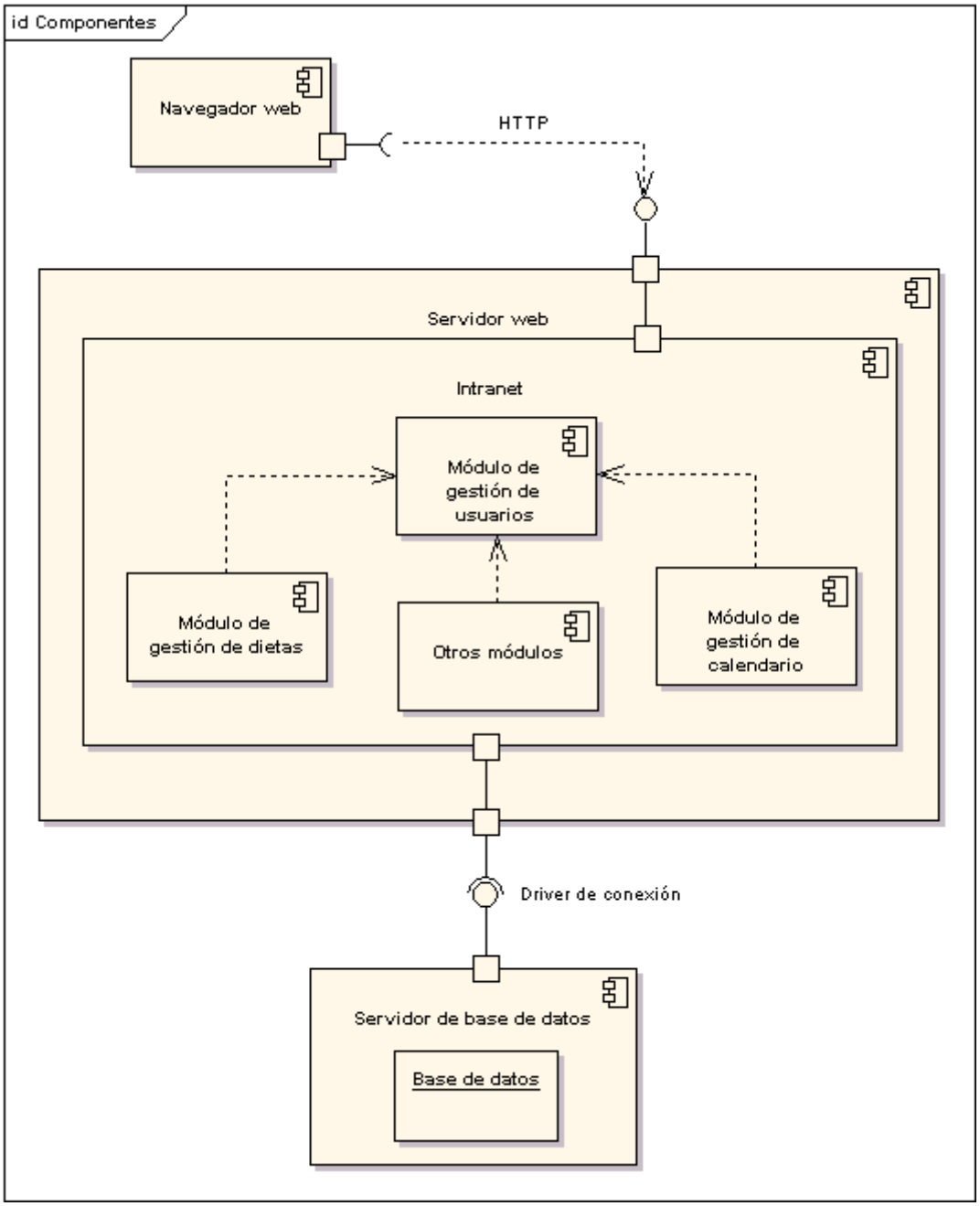


Figura 6.2 - Diagrama de componentes

Este diagrama de componentes, que tiene como objetivo mostrar las dependencias e interconexiones entre componentes lógicos de un sistema, refleja la arquitectura general planteada.

Podemos ver cómo los diferentes módulos necesitan la autenticación otorgada por el módulo de gestión de usuarios, estando todos ellos contenidos en la base de la intranet. Todo ello se encuentra desplegado dentro de un servidor web, que mediante el puerto correspondiente se comunica con el servidor de base de datos (a través del driver de conexión). A su vez dispone de una interfaz abierta, representada por un puerto, para que el navegador del cliente pueda conectarse a través del protocolo HTTP.

6.1.3 Diagrama de Despliegue

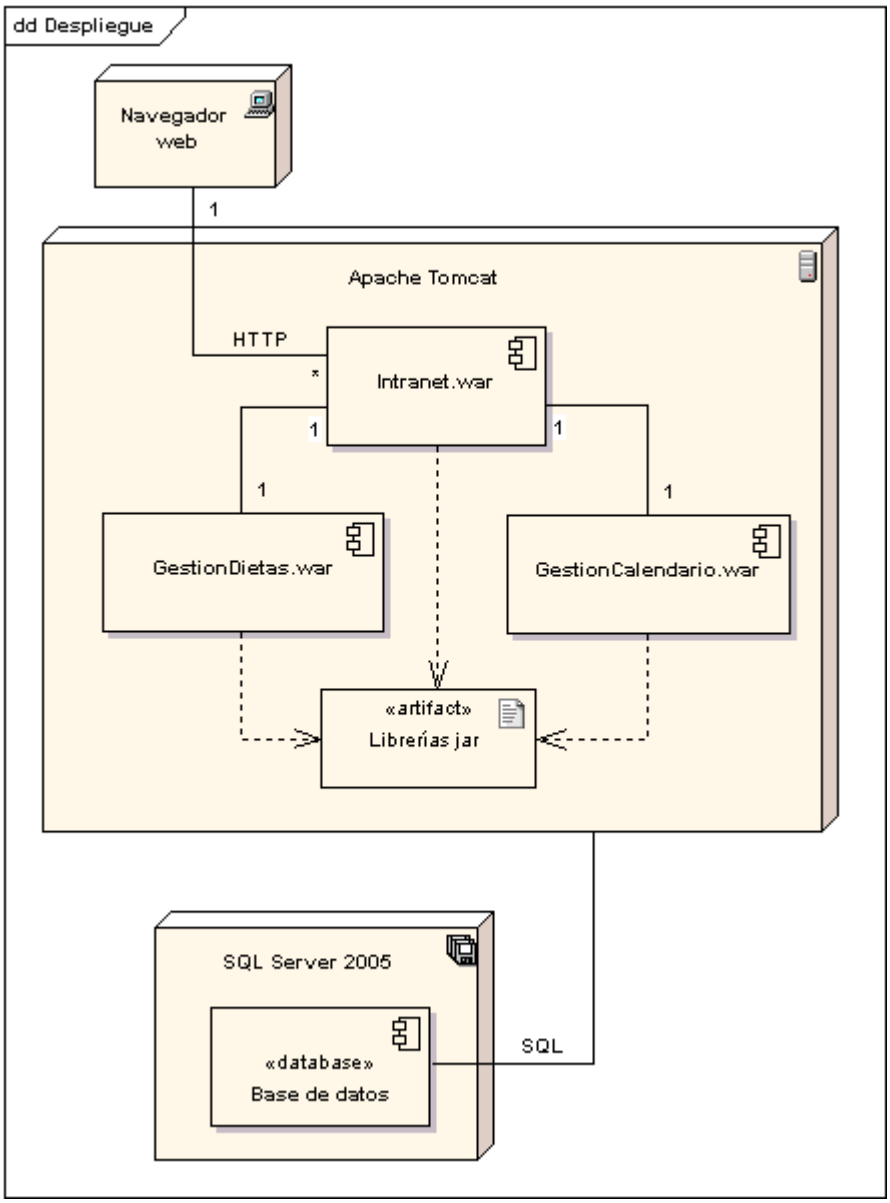


Figura 6.3 - Diagrama de despliegue

Una vez el sistema esté terminado y trabajando, tendrá una estructura como la que esboza el anterior diagrama de despliegue. A continuación se detallan sus componentes:

6.1.3.1 Navegador web

Se trata del elemento que sirve de interfaz de comunicación con el usuario. Desencadena las acciones y realiza las peticiones al servidor web para poder trabajar con la aplicación. La comunicación se realiza mediante el protocolo HTTP.

6.1.3.2 Apache Tomcat

Este servidor web se encarga de alojar las aplicaciones desplegadas y todo lo necesario para el correcto funcionamiento del sistema. En su interior podemos encontrar:

- **Intranet.war**: Aplicación independiente, desarrollada por otros colaboradores, que proporciona el soporte base de inicio se sesión de los usuarios. Se trata de un archivo .war que contiene las clases Java, páginas JavaServer Pages, librerías propias y todos los demás elementos internos que necesita.

- **GestionDietas.war**: Uno de los dos módulos desarrollados en este proyecto, que proporciona la funcionalidad correspondiente a la gestión de dietas. Este archivo desplegable contiene todos los componentes internos para su funcionamiento.

- **GestionCalendario.war**: El segundo de los módulos, que proporciona la funcionalidad relativa a la gestión del calendario laboral. Del mismo modo, contiene todo lo necesario para su ejecución y despliegue.

- **Librerías jar**: Este componente representa el grueso de las librerías que necesita el sistema en general para funcionar. Estas librerías son comunes a todos los módulos, de ahí que para conservar la modularidad y ahorrar costes computacionales se haya decidido ponerlas en el servidor en lugar de en cada módulo por separado.

Recordemos que el servidor web contiene otros módulos desarrollados por otros colaboradores. Como no tienen relación directa con este proyecto se han omitido del diagrama de despliegue.

6.1.3.3 SQL Server 2005

Se trata del servidor de base de datos utilizado por la aplicación para almacenar la información. Se encuentra en una máquina independiente a la del servidor web, aunque en la misma red interna.

El driver de conexión con la base de datos se encarga de la comunicación entre ambas máquinas, siendo necesario hacer las peticiones en lenguaje SQL.

6.2 Diseño de Clases

6.2.1 Diagrama de Clases

Como ya hemos visto en secciones anteriores, el proyecto se divide en dos módulos diferentes con distinta funcionalidad. Al ser independientes entre sí, se detallarán sus diagramas de cases de manera separada.

Es necesario también recordar el esquema general de las clases presentado tanto en el diagrama preliminar del análisis (capítulo 5.4) como el diagrama de paquetes de la fase de diseño (capítulo 6.6.1). Para segmentar el diagrama de clases y facilitar su comprensión se utilizará la división en paquetes mencionada.

6.2.1.1 Módulo de gestión del calendario laboral

6.2.1.1.1 Paquete org.model

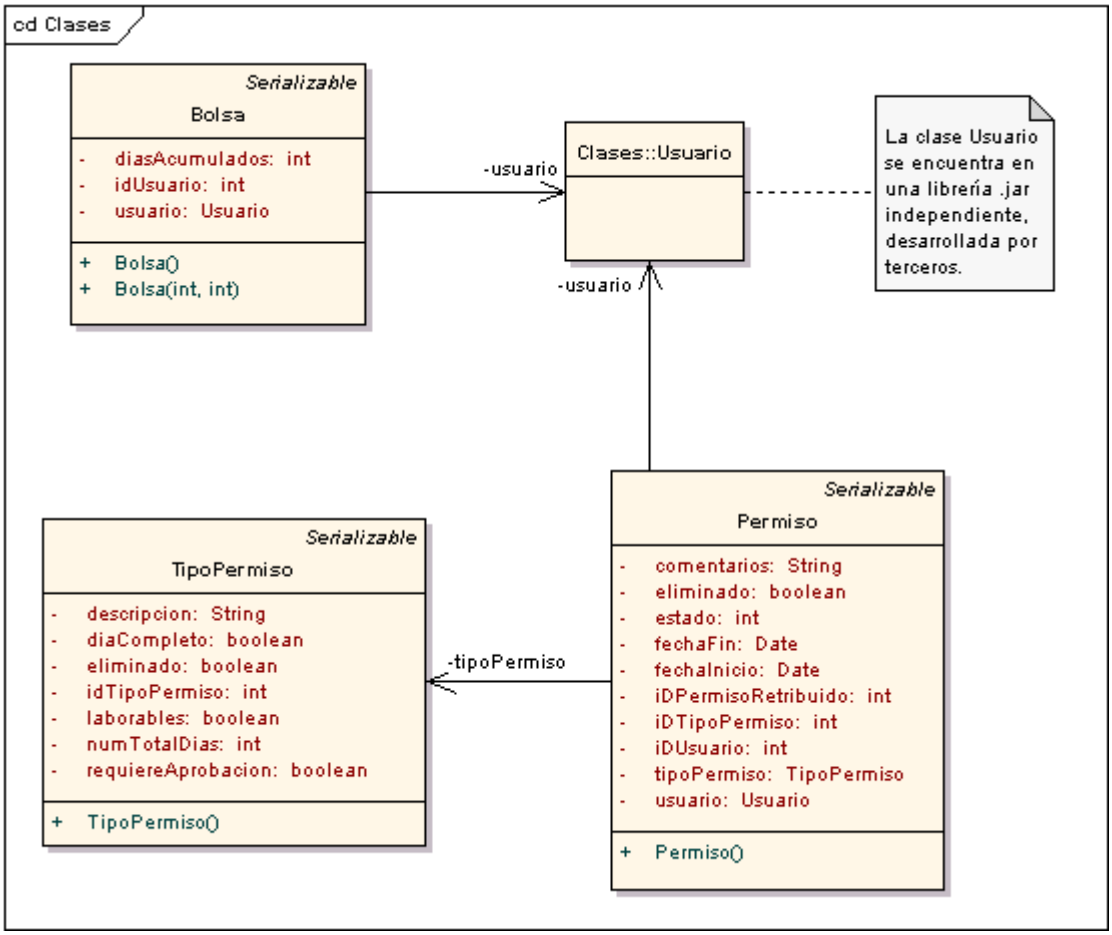


Figura 6.4 - Diagrama de clases: calendario.org.model

Clases que representan al modelo del dominio del módulo. Están formadas por sus atributos, constructor y métodos “getters” y “setters” de los atributos (omitidos en el diagrama para evitar sobrecargarlo con información redundante).

Dado que son objetos que tienen relación directa con las tablas de la base de datos, implementan la interfaz “Serializable”.

6.2.1.1.2 Paquete org.persistence

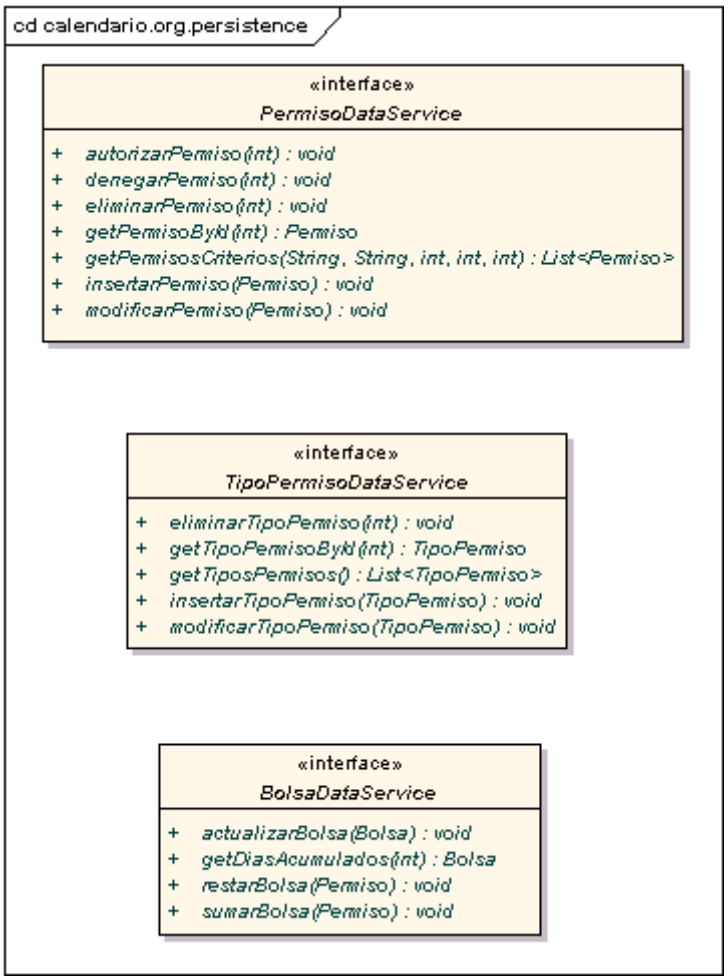


Figura 6.5 - Diagrama de clases: calendario.org.persistence

En este paquete encontramos las interfaces de acceso a base de datos, que contienen las signaturas de los métodos necesarios. Nótese que se corresponden directamente con las clases del modelo.

6.2.1.1.3 Paquete org.persistence.business

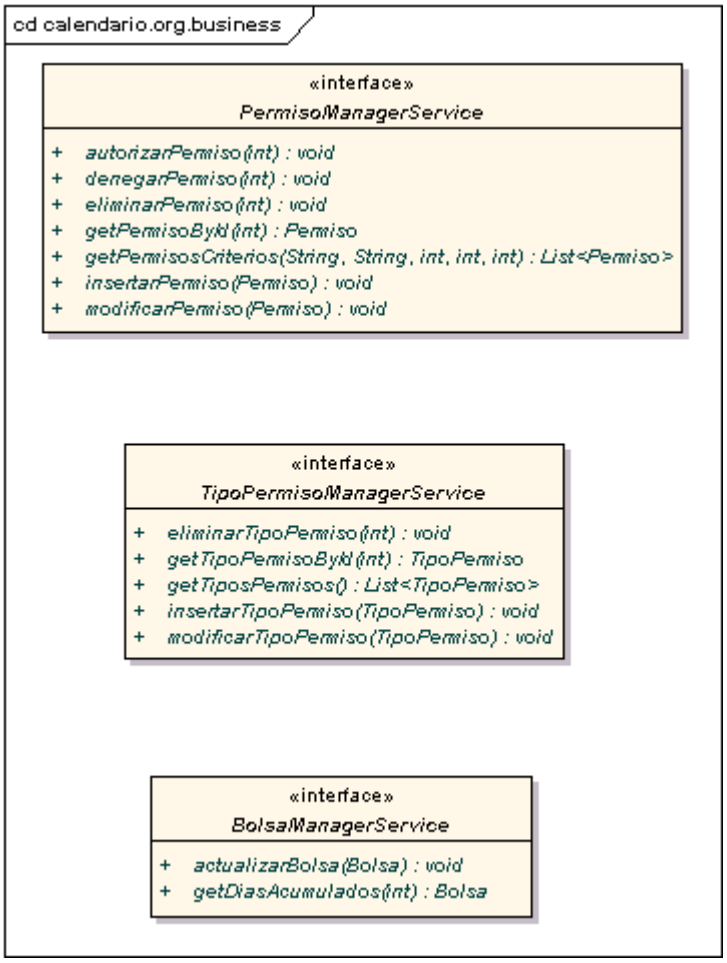


Figura 6.6 - Diagrama de clases: calendario.org.business

Estas interfaces ofrecen las signaturas de los métodos que desencadenan toda la lógica de la aplicación, llamados directa o indirectamente desde las capas superiores por acción del usuario.

Nótese que se corresponden directamente con las clases del modelo, y que las signaturas tienen relación casi directa con las de las clases DataService. Esto se debe a que casi la totalidad de las acciones del usuario ocasionan una llamada diferente a base de datos.

6.2.1.1.4 Paquete calendario.impl.persistence

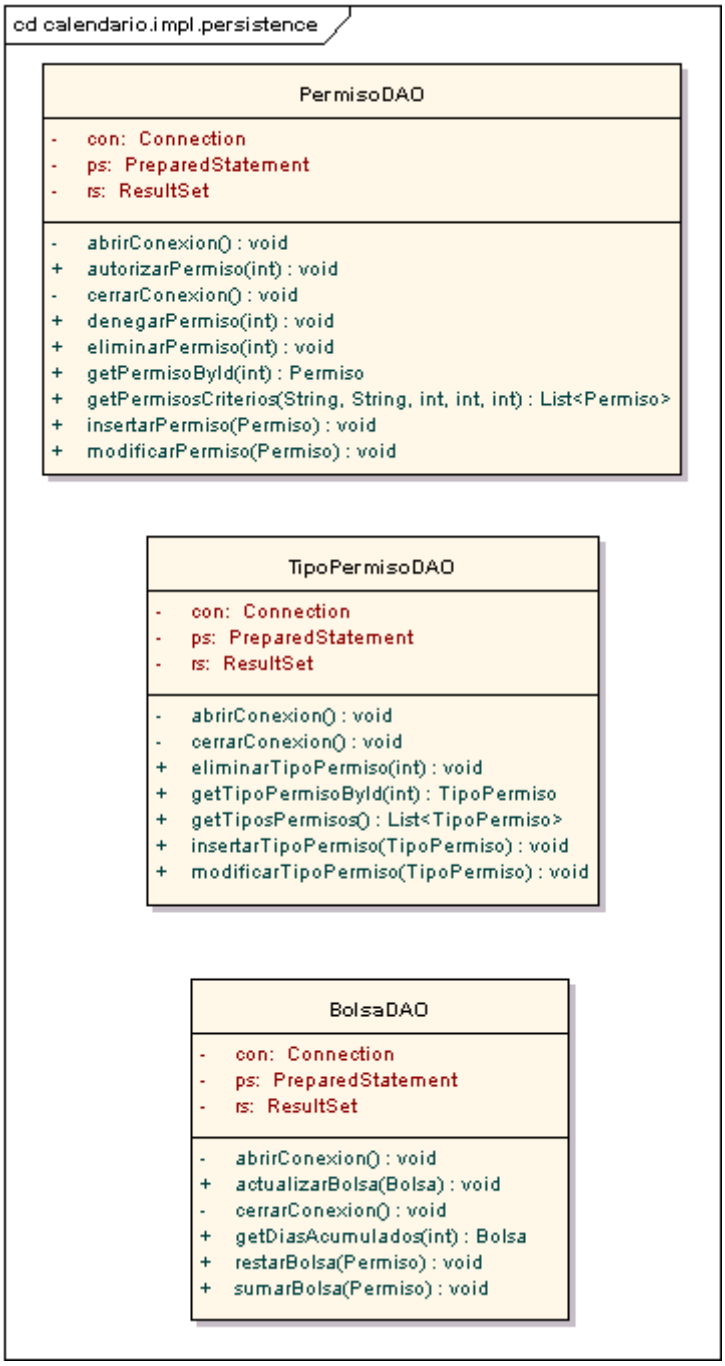


Figura 6.7 - Diagrama de clases: calendario.impl.persistence

En este paquete podemos encontrar los DAO, que además de disponer de los atributos y métodos necesarios para controlar el acceso a base de datos, implementan los métodos marcados por la signatura de la interfaz “DataService” que implementan. En dichos métodos se realizan las operaciones oportunas de escritura y lectura de base de datos.

6.2.1.1.5 Paquete calendario.impl.business

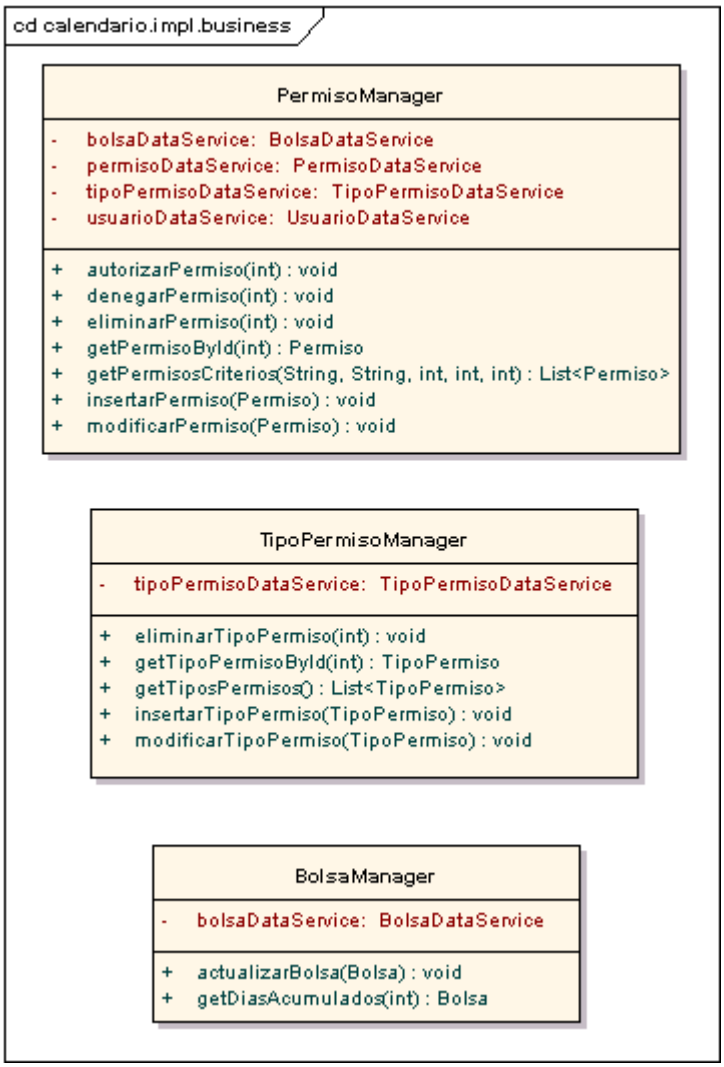


Figura 6.8 - Diagrama de clases: calendario.impl.business

Los "Manager" o manejadores implementan las interfaces de sus correspondientes "ManagerService", y por tanto definen sus métodos. Para ello realizan llamadas a la capa de base de datos a través de los "DataService" que cada uno necesite, de ahí la existencia de dichos atributos.

Cabe destacar que mientras que unas clases son sencillas y únicamente tienen que realizar operaciones básicas sobre sus homólogos en base de datos, otras (como "PermisoManager"), requieren trabajar con más elementos del modelo y por tanto declaran los atributos correspondientes para tener acceso a los mismos.

6.2.1.1.6 Paquete calendario.impl.presentation

En este paquete se encuentran todos los "Actions" del módulo, es decir, el código que se ejecuta en primera instancia cuando el usuario solicita una operación. Dado que cada acción independiente posee una clase diferente y estas son muy similares entre sí, en el siguiente diagrama solo se incluyen las que se han considerado más relevantes e ilustrativas a modo de ejemplo.

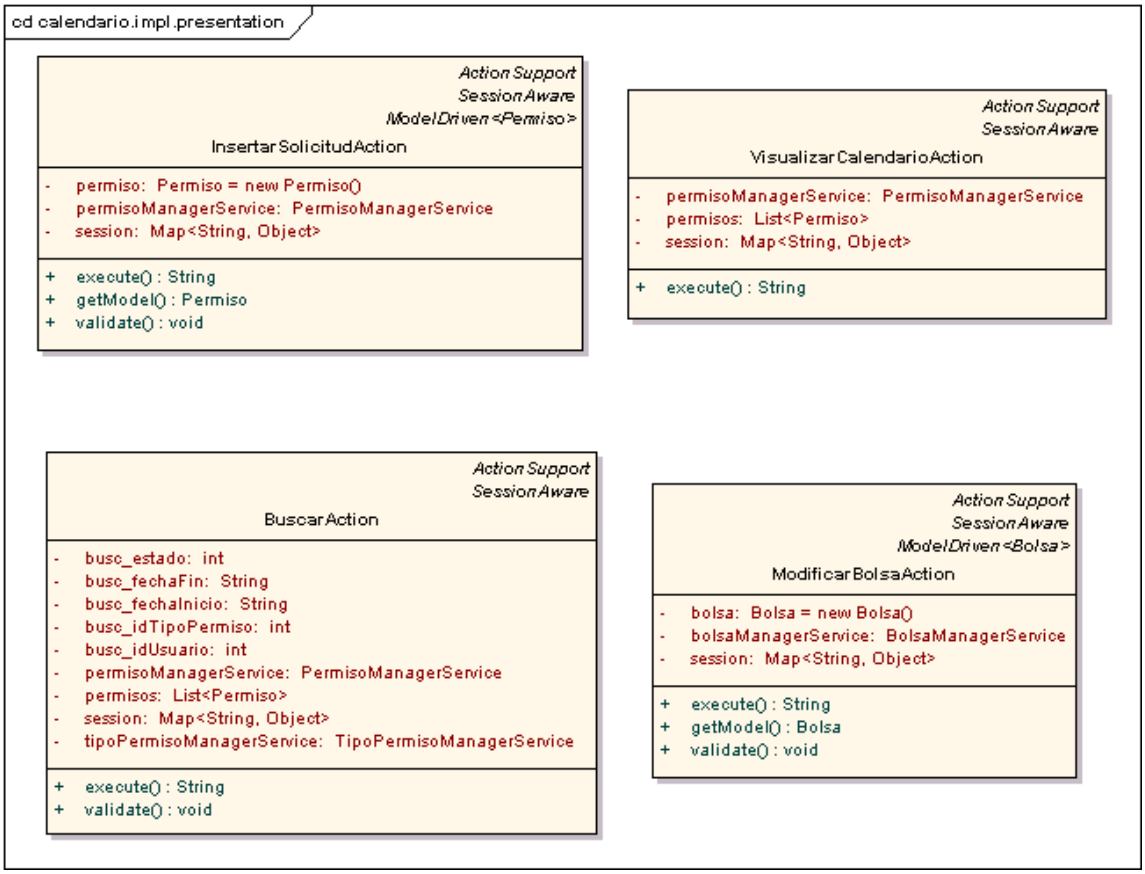


Figura 6.9 Diagrama de clases: calendario.impl.presentation

Como ya hemos adelantado, las clases Action son muy similares entre sí. Todas ellas implementan las interfaces "ActionSupport" de struts, necesario para su funcionamiento, y "SessionAware", para tener acceso al objeto "session" con los datos de la sesión del usuario.

Algunas de ellas también implementan la interfaz "ModelDriven<T>" de struts, que obliga a declarar el método getModel(). Este método devuelve un objeto del modelo de datos cuyos atributos han sido poblados automáticamente según los campos del formulario introducido por el usuario en la página jsp.

Cuando el formulario no responde a un objeto del modelo de datos es necesario declarar los atributos independientemente (véase la clase BuscarAction) y acceder a ellos uno a uno.

Una vez con los datos obtenidos se ejecuta el método execute(), en el que se preparan los datos y se utilizan los ManagerService para llamar a la siguiente capa de la arquitectura y, a la larga, acceder a base de datos. Si algún dato debe ser mostrado en pantalla por la petición del usuario se declara como atributo para que pueda ser recuperado (véase la lista de permisos de BuscarAction).

Nótese también la existencia del método validate() en alguna de las clases. Este método, si declarado, se ejecuta antes del execute(), y su función es la de validar si los datos introducidos por el usuario son válidos (sin campos nulos, letras en lugar de números…). Ante cualquier incongruencia detectada, interrumpe la ejecución del Action y avisa al usuario mediante un mensaje de error personalizado en pantalla.

6.2.1.2 Módulo de gestión de dietas

Este módulo se ha planteado con la misma arquitectura de clases que el de gestión de calendario laboral ya visto. Las justificaciones de sus atributos, métodos y relaciones son las mismas, por lo que solo se añadirán las explicaciones adicionales que no hayan sido contempladas hasta ahora.

6.2.1.2.1 Paquete dietas.org.model

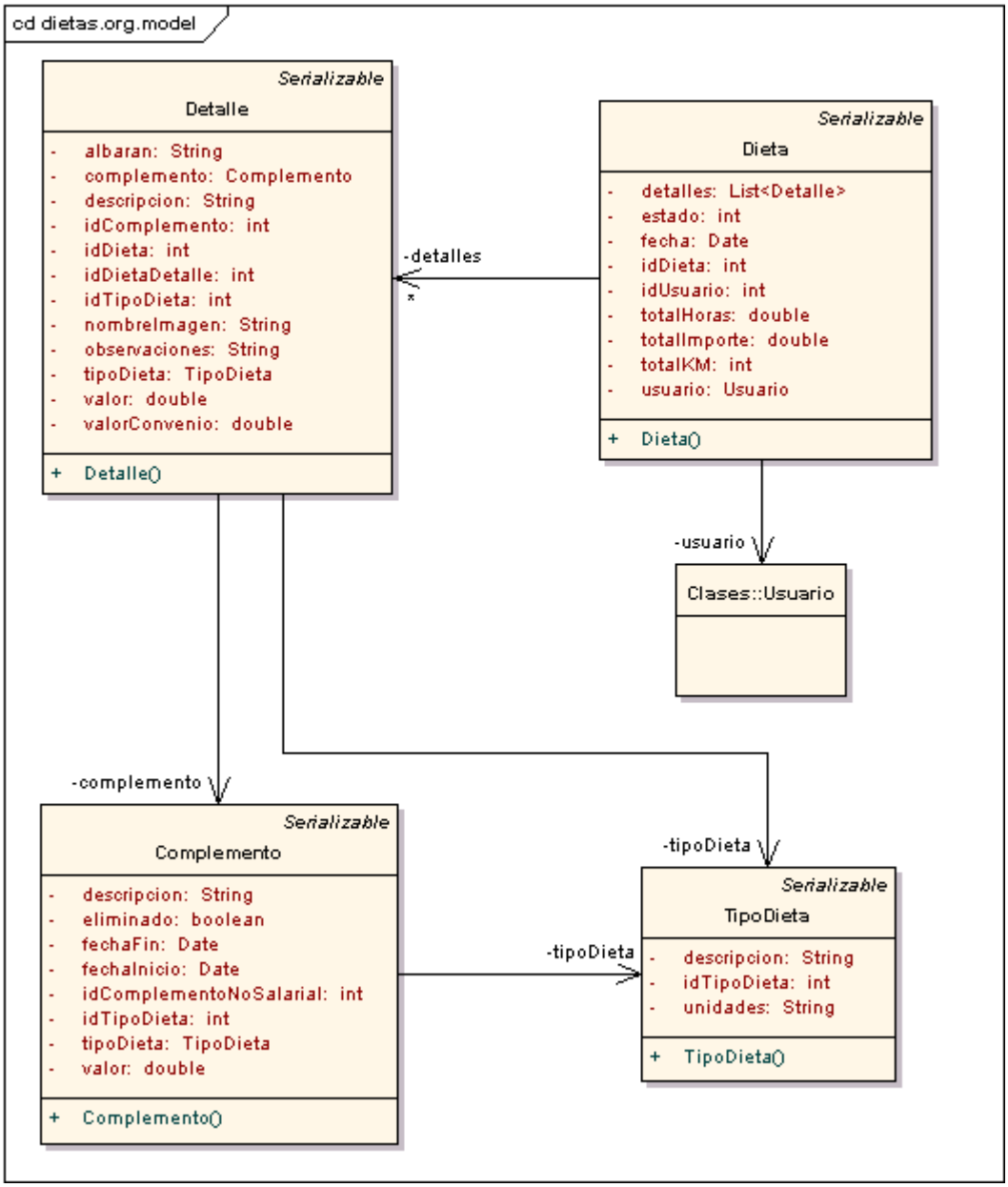


Figura 6.10 - Diagrama de clases: dietas.org.model

A lo ya explicado anteriormente solo cabe destacar que los objetos "Dieta" disponen de una lista de "Detalles" individuales. Como se puede observar, una dieta se considera todos aquellos gastos surgidos en un día concreto, y se aprueban o rechazan en bloque.

Cada detalle dispone de dos valores, el real y el total una vez calculado en base a lo dictado por el complemento no salarial del convenio laboral. Estos valores pueden ser iguales si el complemento no modifica el valor.

6.2.1.2.2 Paquete dietas.org.persistence

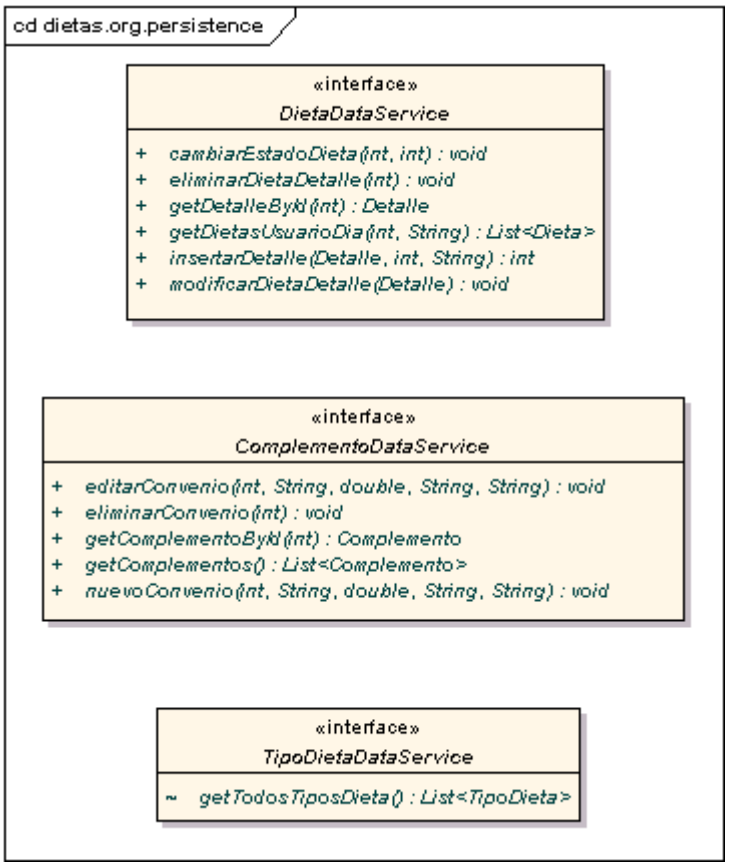


Figura 6.11 - Diagrama de clases: dietas.org.persistence

Como explicación adicional cabe destacar que, dado que las dietas y los detalles son objetos con una estrecha relación entre sí y que uno no puede existir sin la presencia del otro, no se plantea un DataService para cada uno de ellos, ya que los accesos a la base de datos serán conjuntos. Esta decisión se trasladará del mismo modo al resto de paquetes.

6.2.1.2.3 Paquete dietas.org.business

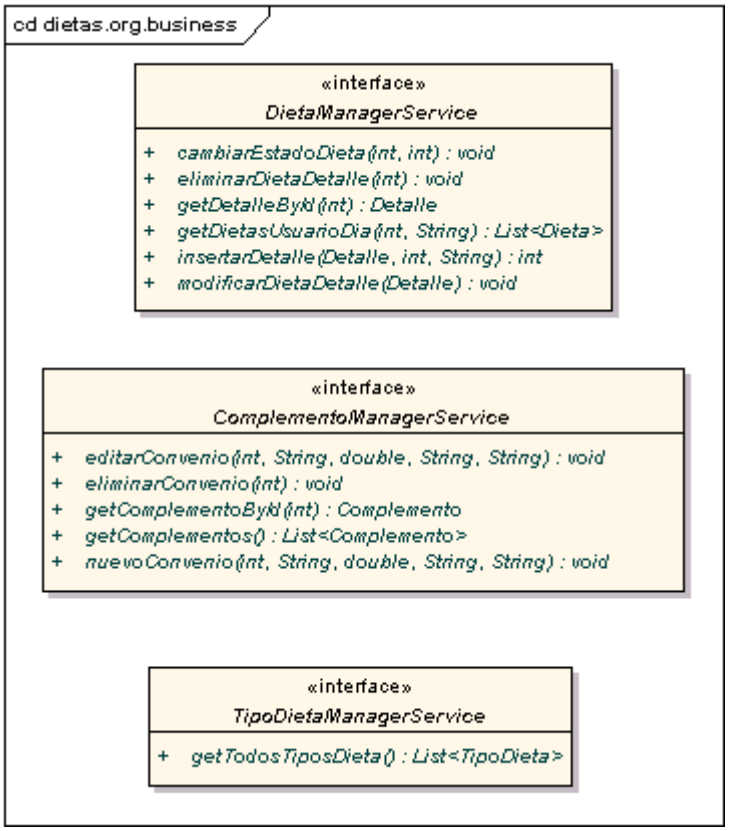


Figura 6.12 - Diagrama de clases: dietas.org.business

Del mismo modo que antes, en este paquete tenemos las interfaces que los respectivos "Manager" deberán implementar.

6.2.1.2.4 Paquete dietas.impl.persistence

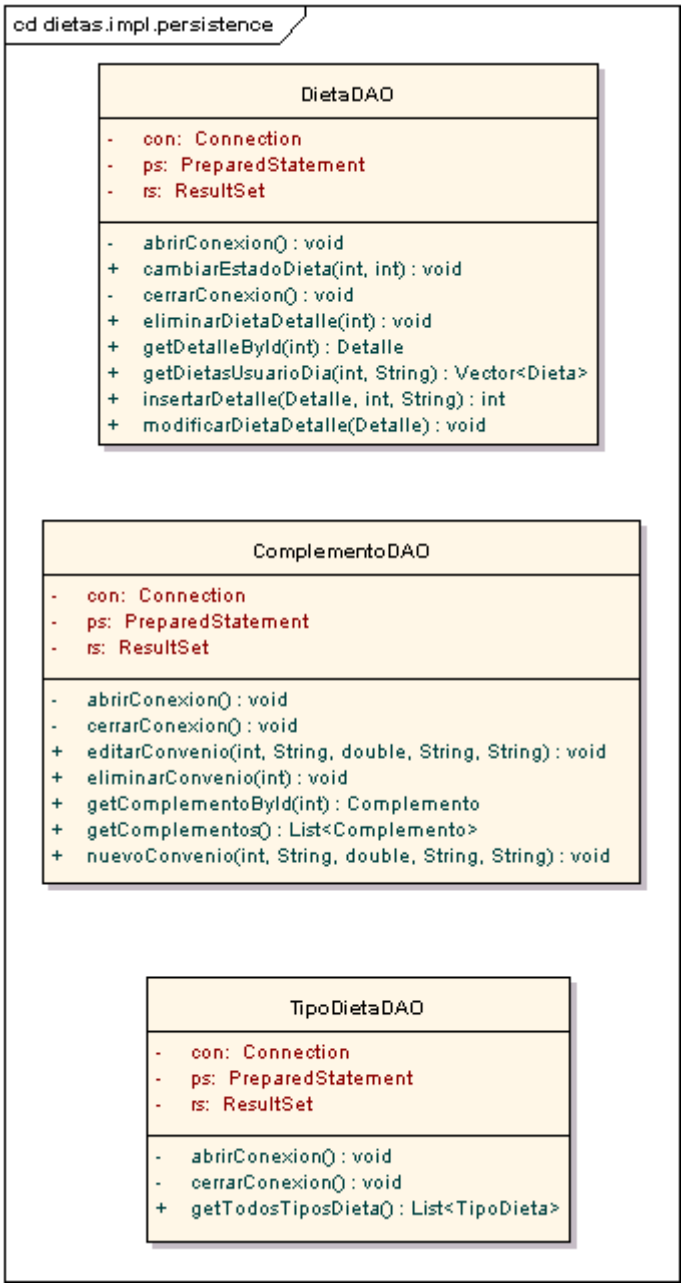


Figura 6.13 - Diagrama de clases: dietas.impl.persistence

Como ya se ha explicado, estas clases implementan los métodos de acceso a base de datos dictados por su interfaz DataService.

6.2.1.2.5 Paquete dietas.impl.business

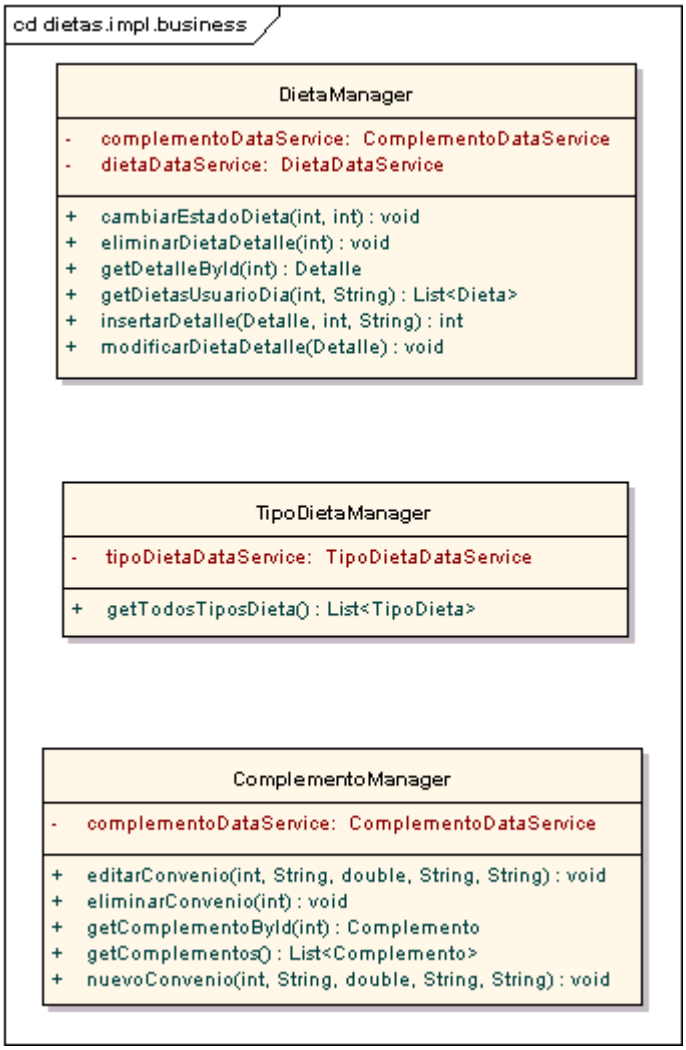


Figura 6.14 - Diagrama de clases: dietas.impl.business

Como ya hemos visto, en este paquete se implementan los métodos indicados por sus respectivas interfaces ManagerService, además de declarar objetos DataService para poder hacer uso de sus métodos de acceso a base de datos.

6.2.1.2.6 Paquete dietas.impl.presentation

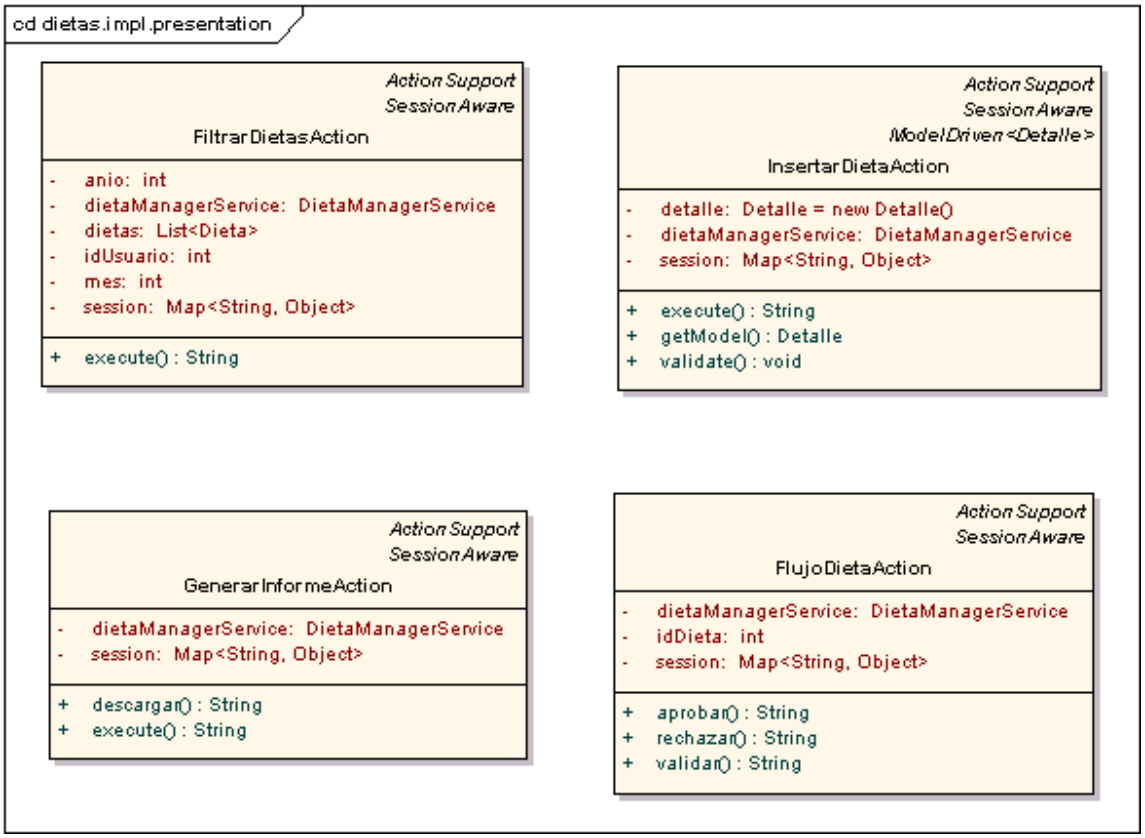


Figura 6.15 Diagrama de clases: dietas.impl.presentation

Nuevamente se muestran solo algunos de los Action más representativos puesto que todos son semejantes en estructura.

Como explicación adicional cabría destacar la existencia del método descargar() a la hora de generar informe, puesto que además de su creación es necesario que el usuario pueda descargárselo y almacenarlo en su equipo.

Además, se han agrupado acciones de similar función en FlujoDietaAction, prescindiendo del método execute() y añadiendo aprobar(), rechazar() y validar(), que pueden convivir en la misma clase siempre y cuando estén bien definidos en el fichero de mapeo de los Actions de struts.

6.3 Diagramas de Interacción y Estados

A continuación podremos encontrar una serie de diagramas en los que se pretende explicar el funcionamiento interno de alguna de las acciones más importantes del proyecto a lo largo de su ejecución por los diferentes pasos que la comprenden.

Mediante los diagramas de interacciones veremos el conjunto de pasos y métodos llamados desde que se inicia una operación hasta que finaliza, además de los elementos implicados en ello. A través de los diagramas de estados se pretende ilustrar qué fases atraviesan los objetos durante una u otra operación.

Se muestran los diagramas más representativos de la aplicación, ya que como hemos visto las operaciones son relativamente similares entre sí.

6.3.1 Caso de uso insertar dieta

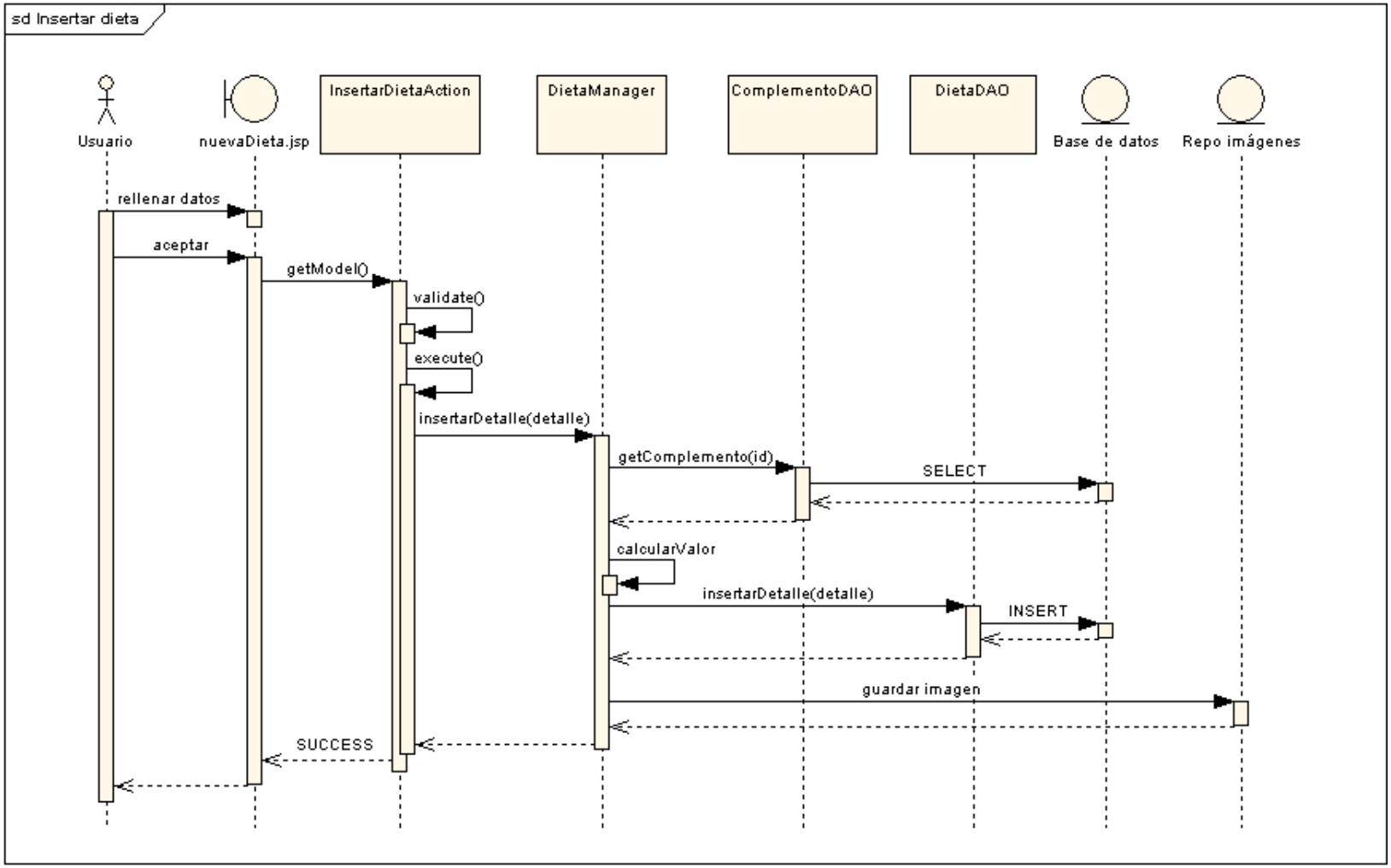


Figura 6.16 - Diagrama de interacción: insertar dieta

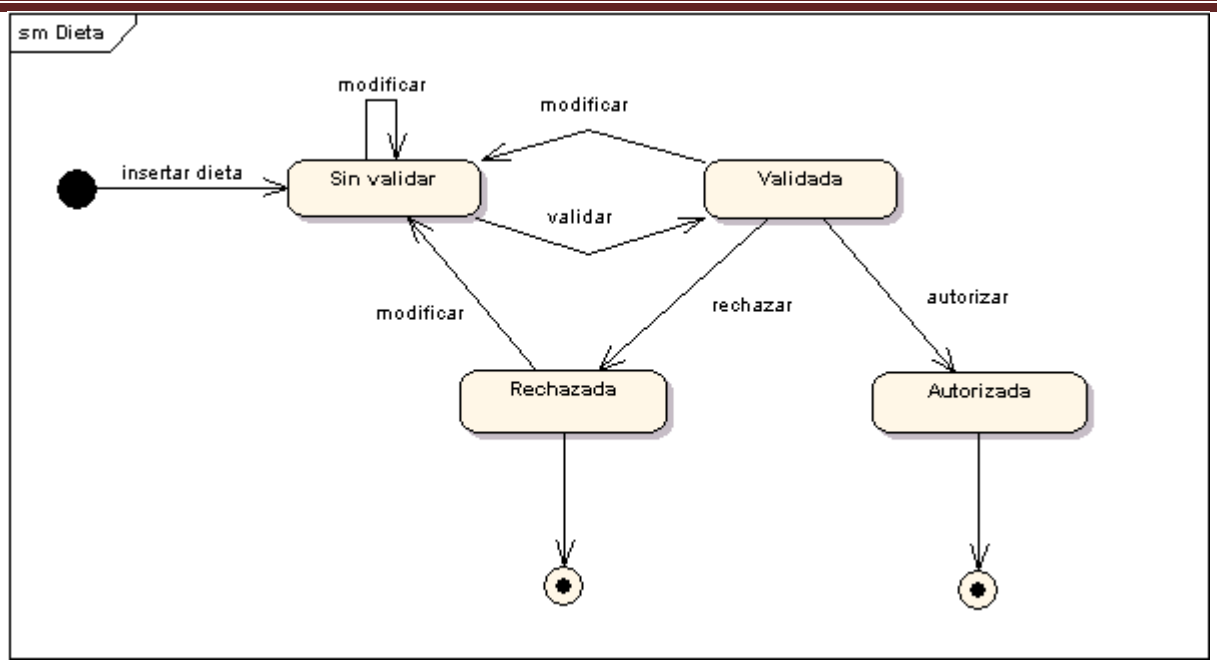


Figura 6.17 - Diagrama de estados: dieta

Mediante los dos diagramas anteriores se pretende explicar el ciclo de vida de una dieta desde que el usuario la crea hasta que termina su flujo de supervisión.

Por un lado, el diagrama de interacciones muestra los pasos que tiene que ejecutar el sistema para crearla, así como los agentes involucrados. Vemos que el usuario interactúa con la página jsp de su navegador para rellenar los datos y aceptarlos. El Action valida que los datos sean correctos y posteriormente ejecuta. En dicha ejecución se llama en primer lugar al Manager, que obtiene de la base de datos a través del DAO correspondiente los datos del complemento no salarial asociado al detalle. El Manager seguidamente calcula los valores de la dieta e inserta todo en la base de datos, nuevamente a través de su DAO. Tras esto, almacena la imagen asociada al detalle en el directorio de imágenes del servidor.

Por otro lado, el diagrama de estados muestra el ciclo de vida de la dieta. Una vez insertada se encuentra en estado “sin validar”, y así seguirá se modifique o no hasta que el usuario decida validarla. Una vez hecho, cualquier modificación la devolverá al estado anterior. El administrador podrá entonces bien rechazarla para que el usuario la modifique de nuevo y reiniciar el proceso si así lo desea, o bien autorizarla para especificar su fin.

6.3.2 Caso de uso generar informe de dietas

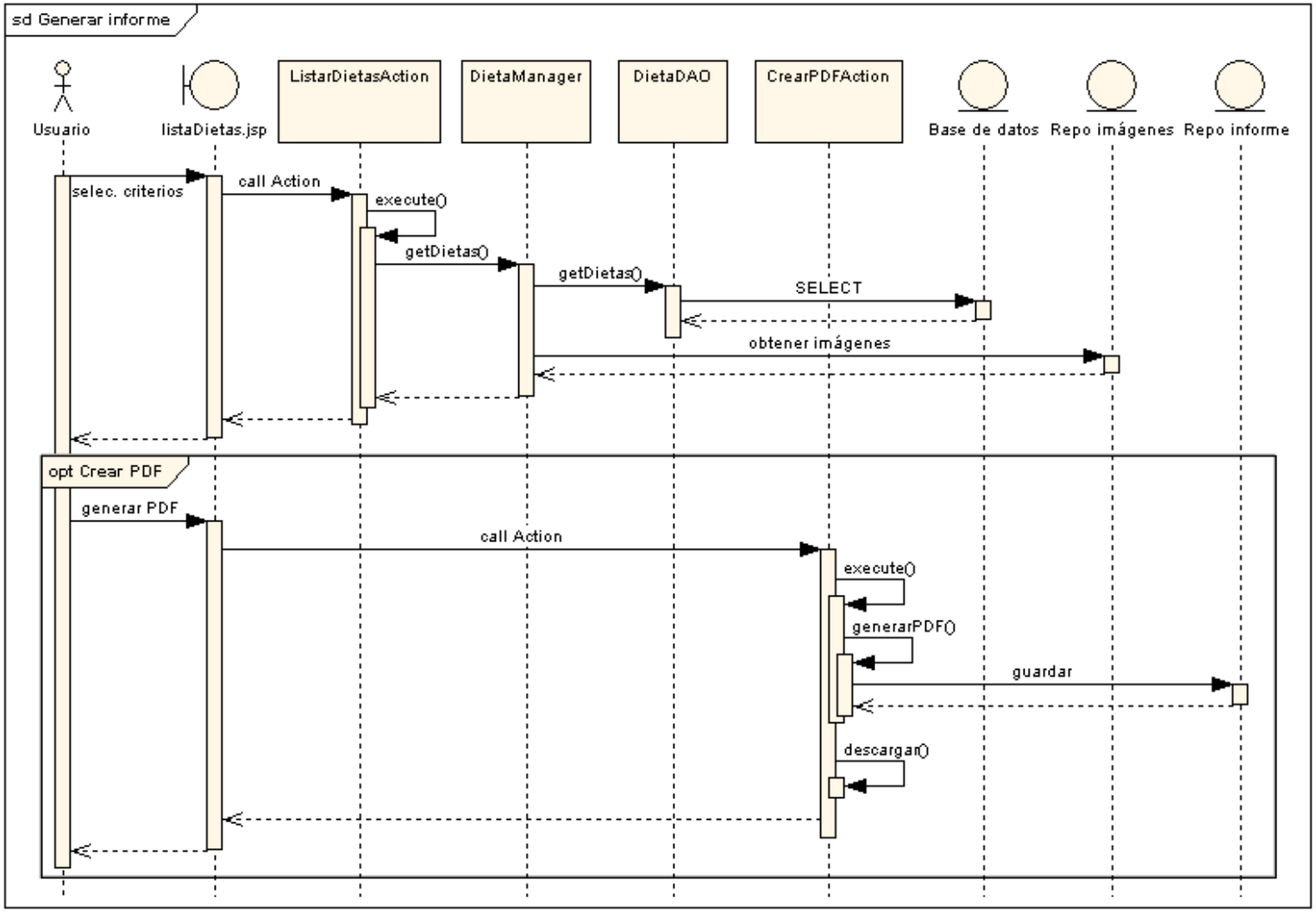


Figura 6.18 - Diagrama de interacción: generar informe de dietas

Para este caso hemos omitido el diagrama de estados puesto que las dietas no modifican su estado durante el proceso de generación de informe.

Podemos observar que la estructura es paralela a la ya vista anteriormente (y por extensión a todas las demás): el usuario interacciona con la interfaz y especifica los criterios del informe que quiere crear, llama al Action y llama al Manager. Este obtiene las dietas de la base de datos a través del DAO de dietas y recupera las imágenes asociadas del servidor. La información viaja de vuelta a la página jsp para que el usuario la visualice.

Opcionalmente el usuario puede generar el informe PDF de las dietas que acaba de obtener. Para ello interacciona con la interfaz mediante el botón correspondiente para llamar a su Action. Durante su ejecución genera el informe con los datos anteriores, lo almacena en el servidor y posteriormente se lo sirve al usuario a través del navegador web.

6.3.3 Caso de uso solicitar permiso de vacaciones

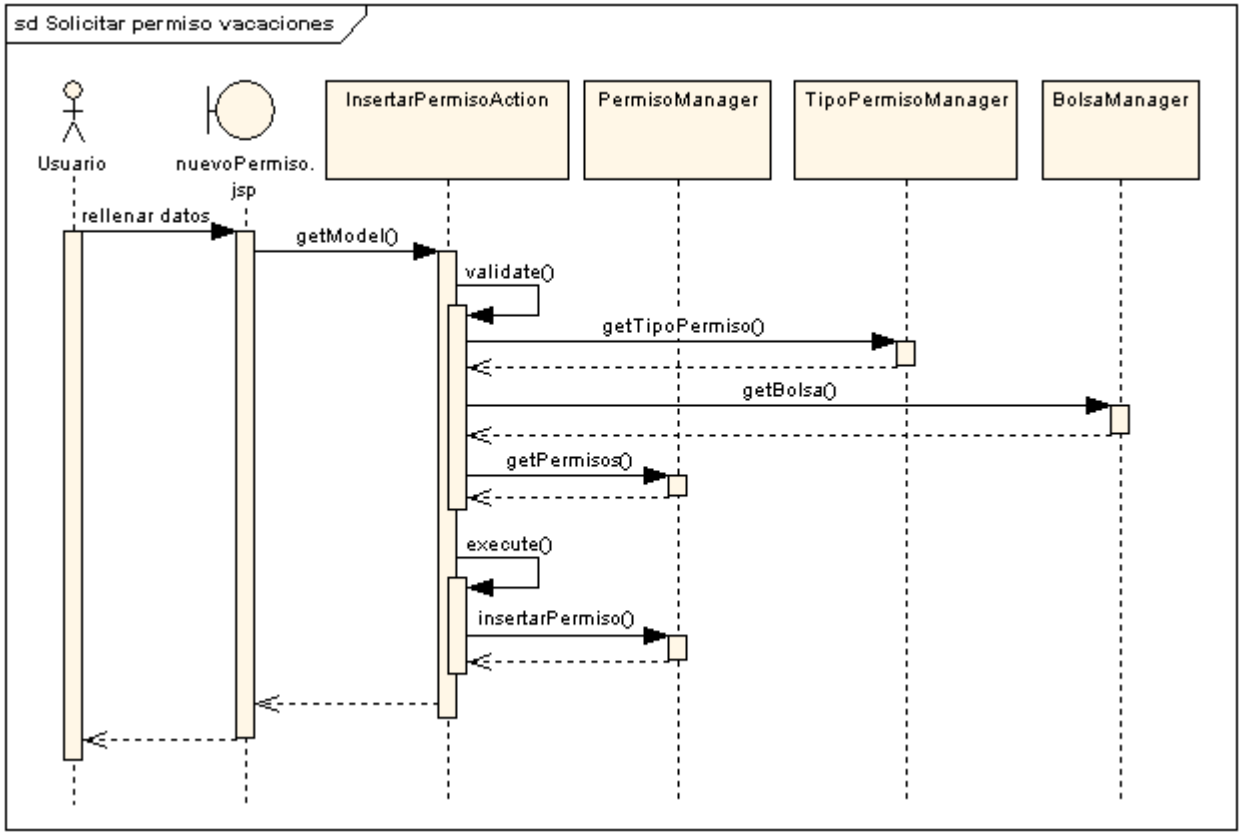


Figura 6.19 - Diagrama de interacción: solicitar permiso de vacaciones

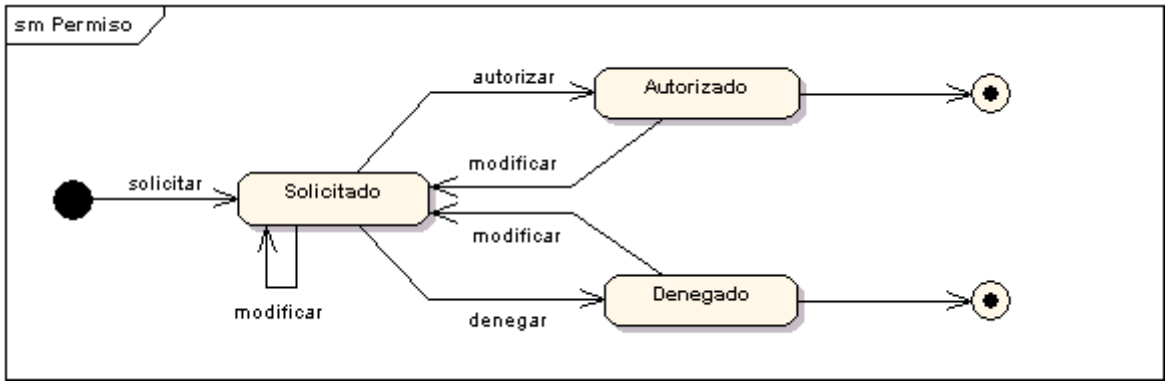


Figura 6.20 - Diagrama de estados: permiso de vacaciones

Estos diagramas ilustran el proceso del sistema para crear una solicitud de permiso de tipo vacaciones, así como la máquina de estados que indica cómo se encuentra el permiso a lo largo de su ciclo de vida.

En primer lugar, en el diagrama de secuencia nos encontramos con una secuencia de pasos ya familiar: el usuario interacciona con la interfaz y llama al Action que se encarga de todo el proceso. Vemos cómo el método "validate" necesita hacer llamadas a la base de datos (recorriendo la arquitectura de capas habitual) para obtener la información del tipo de permiso solicitado, el número de días disponible en la bolsa de vacaciones del usuario y los permisos ya existentes que coincidan en el tiempo con el permiso a solicitar. Con todos esos datos en la mano, comprueba que las condiciones necesarias se cumplan para poder insertar el permiso, es decir, que disponga de suficientes días de vacaciones y que no coincida con otros permisos. Nótese que no se accede a la bolsa de vacaciones para restar el número de días correspondiente: ese proceso se realiza cuando el administrador autoriza el permiso.

Cabe destacar que se han obviado los elementos "PermisoDAO", "TipoPermisoDAO", "BolsaDAO" y "Base de datos" puesto que no aportaban nada nuevo a lo ya visto y recargarían el diagrama de información redundante.

En cuanto al diagrama de estados se destaca que un permiso de tipo vacaciones pasa a estado "solicitado" en el momento de su creación. En este momento el administrador puede autorizarlo o denegarlo para finalizar el proceso. En cualquier momento de la máquina de estados el permiso puede ser modificado, haciendo que este retorne al estado inicial. Como nota adicional, los permisos que no requieren aprobación se crean directamente con estado "autorizado".

6.4 Diseño de la Base de Datos

En la aplicación podemos distinguir una única base de datos en la que se almacenará toda la información necesaria para el correcto funcionamiento del proyecto, además de otras tablas y datos existentes, utilizadas por otras aplicaciones y secciones de la intranet general pero que no tienen relación con lo aquí expuesto.

Es por esto que se hablará únicamente de las tablas empleadas por este proyecto y no de la totalidad de la base de datos, que no ha sido diseñada ni gestionada por el alumno.

6.4.1 Descripción del SGBD Usado

El Sistema de Gestión de Base de Datos utilizado es un Microsoft SQL Server 2005, impuesto por la empresa cliente como condición necesaria, ya que toda su infraestructura está construida sobre este sistema.

Este SGBD contempla soporte para transacciones, procedimientos almacenados, dispone de un entorno gráfico para su administración y la posibilidad de trabajar en modo cliente-servidor. Gracias al cliente nativo de SQL que tiene incorporado se pueden ejecutar las sentencias sobre dicho lenguaje de consulta.

6.4.2 Integración del SGBD en Nuestro Sistema

La Java Database Connectivity empleada en el proyecto dispone de los mecanismos y controladores necesarios para conectarse y trabajar con un SQL Server 2005, proporcionando su localización, nombre de la base de datos y un usuario y contraseña con acceso a la misma.

Las conexiones se realizan a través de un *connection pool*, que mantiene una colección de conexiones abiertas que pueden ser reutilizadas, agilizando así los mecanismos de conexión para con ella y evitando la sobrecarga de recursos, puesto que en una aplicación multitarea con varios usuarios simultáneos crear nuevas conexiones continuamente para cada petición ralentiza mucho la ejecución general.

6.4.3 Diagrama de base de datos

El diagrama de base de datos se ha dividido en tres secciones para ilustrar de manera separada y diferenciada las diferentes partes que conforman el total.

6.4.3.1 Tablas concernientes al usuario

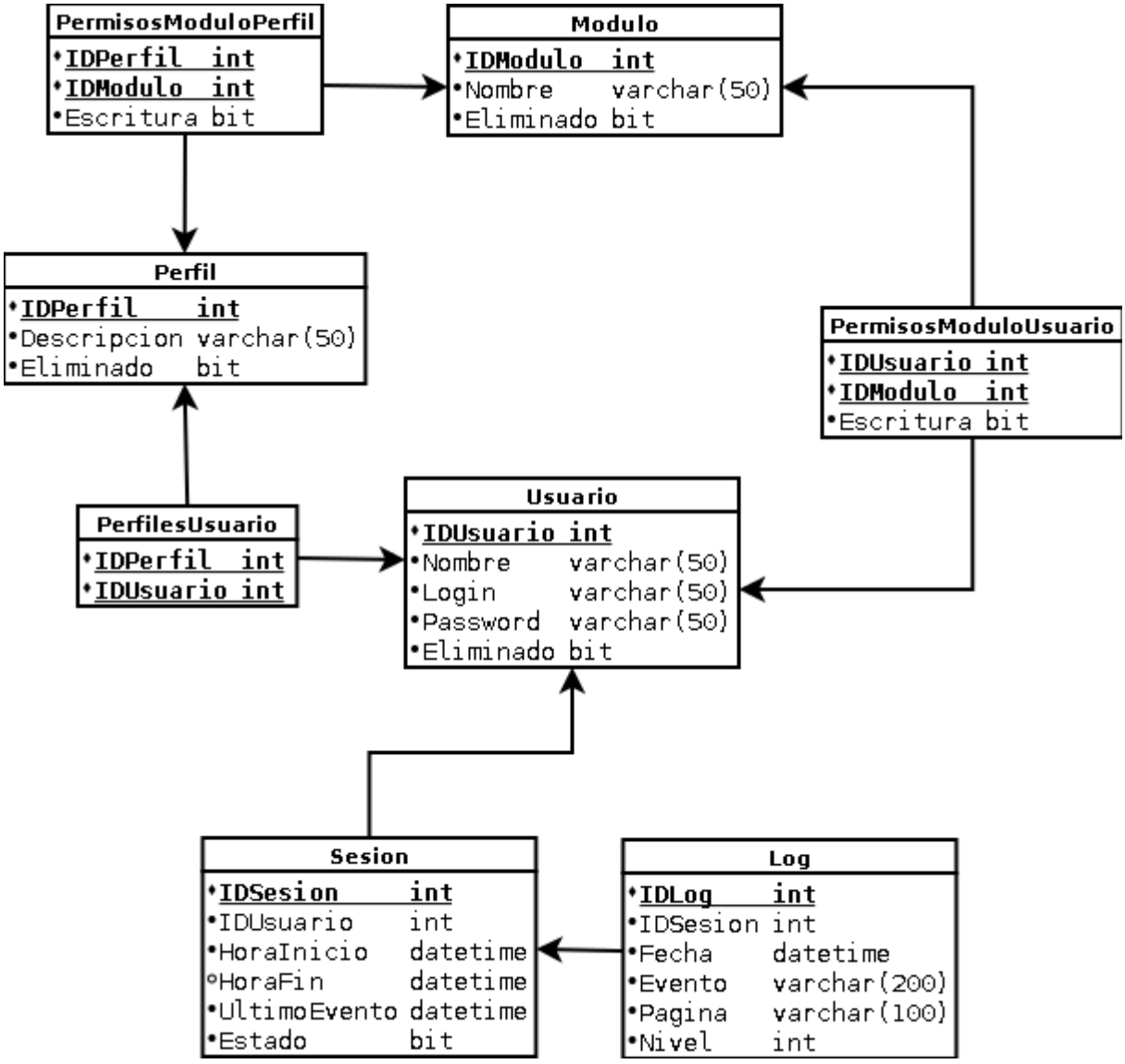


Figura 6.21 Diagrama de base de datos: Usuarios

En este primer diagrama podemos observar cómo se organiza la información en torno a los usuarios:

Nombre de la tabla
Usuario
Descripción
Tabla que almacena los datos relevantes a los usuarios del sistema.
Atributos
• **IDUsuario (PK, int, no NULL):** Identificador único del usuario. • **Nombre (varchar(50), no NULL):** Nombre del usuario. • **Login (varchar(50), no NULL):** Nombre con el que el usuario inicia sesión. • **Password (varchar(50), no NULL):** Contraseña con la que el usuario inicia sesión. • **Eliminado (bit, no NULL):** Indica si el usuario ha sido eliminado o no.

Nombre de la tabla
Perfil
Descripción
Tabla que almacena los datos relevantes a los diferentes perfiles del sistema.
Atributos
• **IDUsuario (PK, int, no NULL):** Identificador único del perfil. • **Descripcion (varchar(50), no NULL):** Descripción y nombre del perfil. • **Eliminado (bit, no NULL):** Indica si el usuario ha sido eliminado o no.

Nombre de la tabla
PerfilesUsuario
Descripción
Tabla que relaciona a un usuario con su perfil.
Atributos
• **IDPerfil (PK, FK, int, no NULL):** Identificador único del perfil. • **IDUsuario (PK, FK, int, no NULL):** Identificador único del usuario.

Nombre de la tabla
Modulo
Descripción
Tabla que almacena los datos relevantes a los módulos o bloques de funcionalidad diferente del sistema.
Atributos
• **IDModulo (PK, int, no NULL):** Identificador único del módulo. • **Nombre (varchar(50), no NULL):** Nombre del módulo. • **Eliminado (bit, no NULL):** Indica si el módulo ha sido eliminado o no.

Nombre de la tabla
PermisosModuloPerfil
Descripción
Tabla que especifica qué permisos tiene un perfil determinado sobre un módulo concreto.
Atributos
• **IDPerfil (PK, FK, int, no NULL):** Identificador único del perfil. • **IDModulo (PK, FK, int, no NULL):** Identificador único del módulo. • **Escritura (bit, no NULL):** Indica si el perfil posee permisos de escritura (bit a 1) sobre el módulo o solo de lectura (bit a 0).

Nombre de la tabla
PermisosModuloUsuario
Descripción
Tabla que especifica qué permisos tiene un usuario determinado sobre un módulo concreto.
Atributos
• **IDPerfil (PK, FK, int, no NULL):** Identificador único del perfil. • **IDUsuario (PK, FK, int, no NULL):** Identificador único del usuario. • **Escritura (bit, no NULL):** Indica si el usuario posee permisos de escritura (bit a 1) sobre el módulo o solo de lectura (bit a 0).

Nombre de la tabla
Sesion
Descripción
Tabla que almacena los datos de la sesión del usuario.
Atributos
• **IDSesion (PK, int, no NULL):** Identificador único de la sesión. • **IDUsuario (FK, int, no NULL):** Identificador único del usuario. • **HoraInicio (datetime, no NULL):** Fecha y hora a la que el usuario inició la sesión. • **HoraFin (datetime, NULL):** Fecha y hora a la que el usuario finalizó la sesión. • **UltimoEvento (datetime, no NULL):** Fecha y hora en la que el usuario realizó la última operación. • **Estado (bit, no NULL):** Indica el estado de la sesión.

Nombre de la tabla
Log
Descripción
Tabla que funciona como registro o log de todas las acciones que el usuario realiza durante una sesión concreta.
Atributos
• **IDLog (PK, int, no NULL):** Identificador único de registro. • **IDSesion (FK, int, no NULL):** Identificador único de la sesión. • **Fecha (datetime, no NULL):** Fecha y hora a la que el usuario inició la sesión. • **Evento (varchar(200), no NULL):** Descripción del evento que causó la escritura en el registro. • **Pagina (varchar(100), no NULL):** Nombre de la página que causó la escritura en el registro. • **Nivel (bit, no NULL):** Nivel de abstracción del tipo de evento que causó la escritura en registro.

6.4.3.2 Tablas concernientes a la gestión de calendario

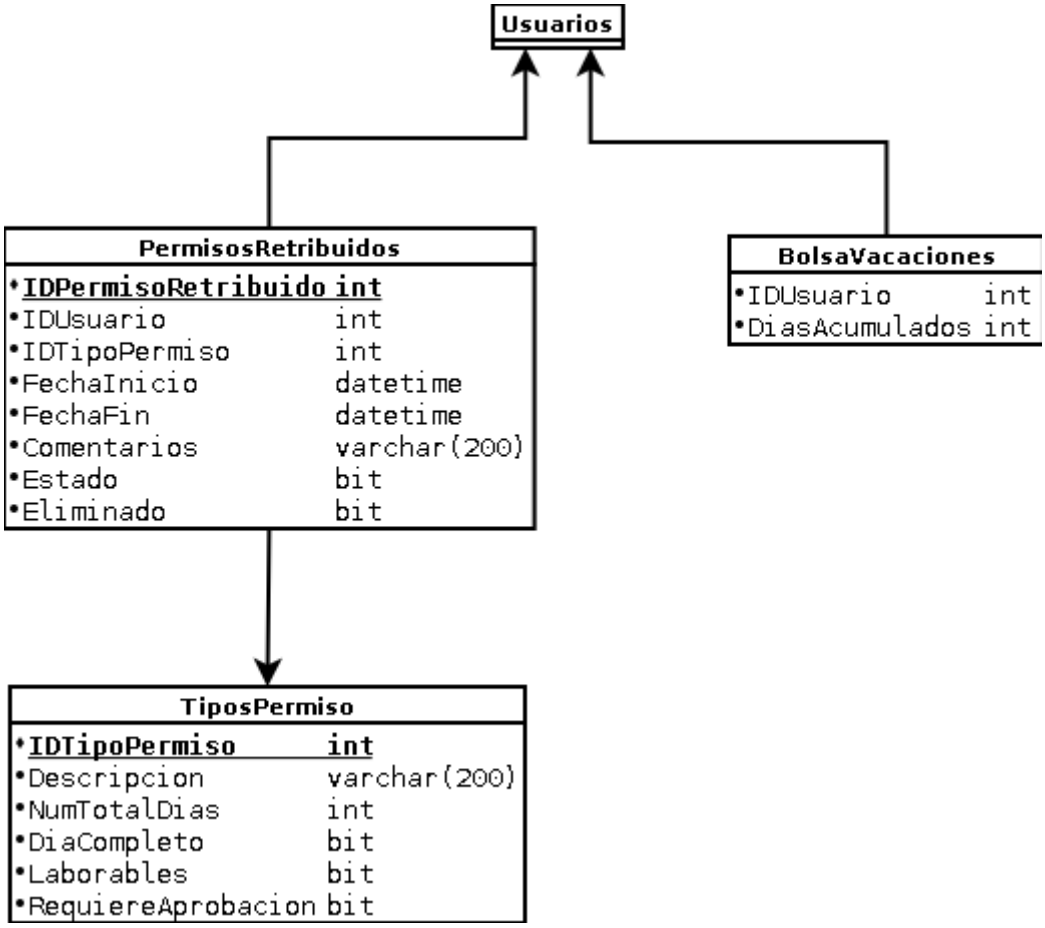


Figura 6.22 - Diagrama de base de datos – Calendario

Esta sección del diagrama de la base de datos muestra lo relativo a la gestión del calendario. Nótese que la entidad "Usuarios" hace referencia a la tabla con el mismo nombre del anterior segmento del diagrama.

Nombre de la tabla
PermisosRetribuidos
Descripción
Tabla que almacena los datos relativos a los permisos solicitados por los usuarios de la aplicación.
Atributos
• **IDPermisoRetribuido (PK, int, no NULL):** Identificador único del permiso. • **IDUsuario (FK, int, no NULL):** Identificador único del usuario solicitante del permiso. • **IDTipoPermiso (FK, int, no NULL):** Identificador único del tipo de permiso. • **FechaInicio (datetime, no NULL):** Fecha de inicio del permiso solicitado. • **FechaFin (datetime, no NULL):** Fecha de finalización del permiso solicitado. • **Comentarios (varchar(200), no NULL):** Cualquier cadena de texto libre en la que el usuario puede justificar la razón de la solicitud. • **Estado (bit, no NULL):** Indica el estado del permiso (1 solicitado, 2 autorizado, 3 denegado). • **Eliminado (bit, no NULL):** Indica si el permiso ha sido eliminado o no.

Nombre de la tabla
TiposPermiso
Descripción
Tabla que almacena los datos relativos a los tipos de permiso que maneja la aplicación y que pueden ser seleccionados por los usuarios.
Atributos
• **IDTipoPermiso (PK, int, no NULL):** Identificador único del tipo de permiso. • **Descripcion (varchar(200), no NULL):** Cadena de texto que define al tipo de permiso. • **NumTotalDias (int, no NULL):** Cantidad de días que como máximo pueden escogerse para ese tipo de permiso. • **DiaCompleto (bit, no NULL):** Indica si el tipo de permiso comprende un día completo o únicamente horas de un solo día. • **Laborables (bit, no NULL):** Indica si el tipo de permiso comprende días laborables o días naturales. • **RequiereAprobación (bit, no NULL):** Especifica si el tipo de permiso requiere aprobación por parte del administrador o su autorización es directa.

Nombre de la tabla
BolsaVacaciones
Descripción
Tabla que recoge cuántos días acumulados de vacaciones tiene disponible cada usuario del sistema.
Atributos
• **IDUsuario (FK, único, int, no NULL):** Identificador único del usuario. • **DiasAcumulados (int, no NULL):** Cantidad de días que el usuario tiene disponible para sus vacaciones.

6.4.3.3 Tablas concernientes a la gestión de dietas

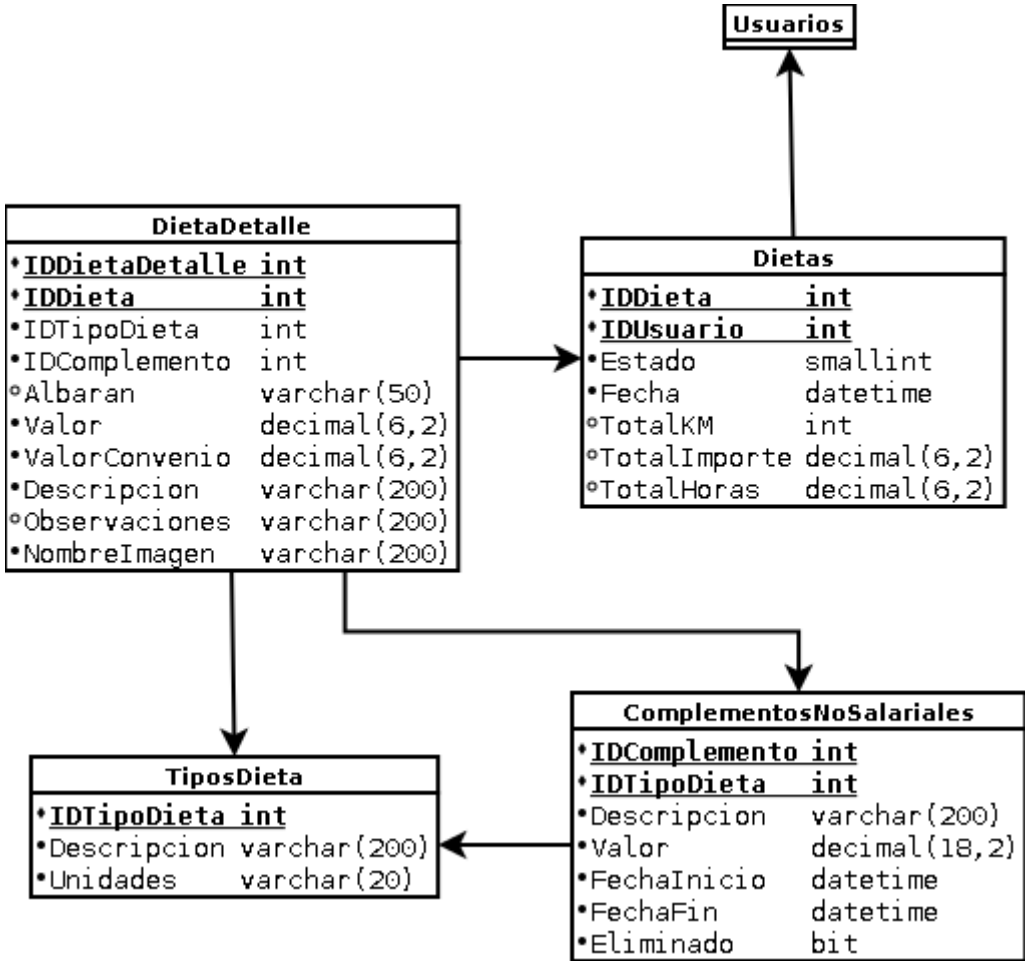


Figura 6.23 - Diagrama de base de datos – Dietas

Esta sección del diagrama de la base de datos muestra lo relativo a la gestión de las dietas. Nótese que la entidad "Usuarios" hace referencia a la tabla con el mismo nombre del anterior segmento del diagrama.

Nombre de la tabla
Dietas
Descripción
Tabla que almacena los datos relativos a las dietas especificadas por el usuario. Cada dieta hace referencia a un conjunto de detalles de un solo día.
Atributos
• **IDDieta (PK, int, no NULL):** Identificador único de la dieta. • **IDUsuario (PK, FK, int, no NULL):** Identificador único del usuario. • **Estado (smallint, no NULL):** Indica el estado de la dieta (1 sin validar, 2 validada, 3 autorizada, 4 denegada).. • **Fecha (datetime, no NULL):** Especifica el día al que hace referencia la dieta. • **TotalKM (int, no NULL):** Indica la cantidad total de kilómetros realizados en la dieta de ese día. • **TotalImporte (decimal(6,2), no NULL):** Indica la cantidad total de dinero para la dieta de ese día. • **TotalHoras (decimal(6,2), no NULL):** Indica la cantidad total de horas extra realizadas

en la dieta de ese día.

Nombre de la tabla
DietaDetalle
Descripción
Tabla que almacena los datos relativos a los detalles de la dieta de un día concreto.
Atributos
• **IDDietaDetalle (PK, int, no NULL):** Identificador único del detalle de la dieta. • **IDDieta (PK, FK, int, no NULL):** Identificador único de la dieta. • **IDTipoDieta (FK, int, no NULL):** Identificador único del tipo del detalle de dieta. • **IDComplemento (FK, int, no NULL):** Identificador único del complemento asociado al detalle de la dieta. • **Albaran (varchar(50), NULL):** Número de albarán del detalle de la dieta, si existe. • **Valor (decimal(6,2), no NULL):** Valor monetario real del detalle de la dieta. • **ValorConvenio (decimal(6,2), no NULL):** Valor monetario del detalle de la dieta una vez aplicado la transformación impuesta por el complemento del convenio. • **Descripcion (varchar(200), no NULL):** Texto libre que define la naturaleza del permiso, escrito por el usuario. • **Observaciones (varchar(200), NULL):** Texto libre en donde el usuario puede especificar cualquier comentario adicional que considere oportuno. • **NombreImagen (varchar(200), no NULL):** Nombre de la imagen asociada al detalle de la dieta del usuario.

Nombre de la tabla
TiposDieta
Descripción
Tabla que almacena los datos relativos a los tipos de dieta que maneja la aplicación y que pueden ser seleccionados por los usuarios.
Atributos
• **IDTipoDieta (PK, int, no NULL):** Identificador único del tipo de dieta. • **Descripcion (varchar(200), no NULL):** Cadena de texto que define al tipo de dieta. • **Unidades (varchar(20), no NULL):** Texto que indica cuál es la unidad en la que se mide la dieta (euros, kilómetros u horas).

Nombre de la tabla
ComplementosNoSalariales
Descripción
Tabla que almacena los datos relativos a los complementos no salariales definidos por el convenio laboral, que afecta al precio final de la dieta en función de su tipo.
Atributos
• **IDComplemento (PK, int, no NULL):** Identificador único del tipo del complemento no salarial. • **IDTipoDieta (FK, int, no NULL):** Identificador único del tipo de dieta al que está asociado. • **Descripcion (varchar(200), no NULL):** Cadena de texto que define el complemento no salarial. • **Valor (decimal(18,2), no NULL):** Especifica el valor monetario al que se ve sometido la dieta asociada al complemento. • **FechaInicio (datetime, no NULL):** Fecha de inicio de validez del complemento.

- **FechaFin (datetime, no NULL):** Fecha de fin de validez del complemento.
- **Laborables (bit, no NULL):** Indica si el tipo de permiso comprende días laborables o días naturales.
- **Eliminado (bit, no NULL):** Indica si el complemento ha sido eliminado o no, y si por tanto deja de tener validez.

6.5 Diseño de la interfaz

Como ya se comentó en la fase de análisis, fue responsabilidad de la empresa el diseñar las interfaces de la aplicación, distribuir sus contenidos, establecer su gama de colores y todo lo relacionado con la visualización de los contenidos.

El desarrollador solo tiene la responsabilidad de adaptar la información de la manera más semejante posible a los deseos de la empresa. Como apenas se ha participado en el diseño de la interfaz, se muestran a continuación las capturas enviadas, sin justificar las mismas debido a que se desconocen las razones.

Cabe señalar que por motivos técnicos o logísticos quizá no sea posible adaptar las interfaces en su totalidad a lo especificado por el cliente. Estas posibles discrepancias, no obstante, solo afectarán a la distribución de los contenidos y no a su funcionalidad.

6.5.1 Módulo de gestión del calendario laboral

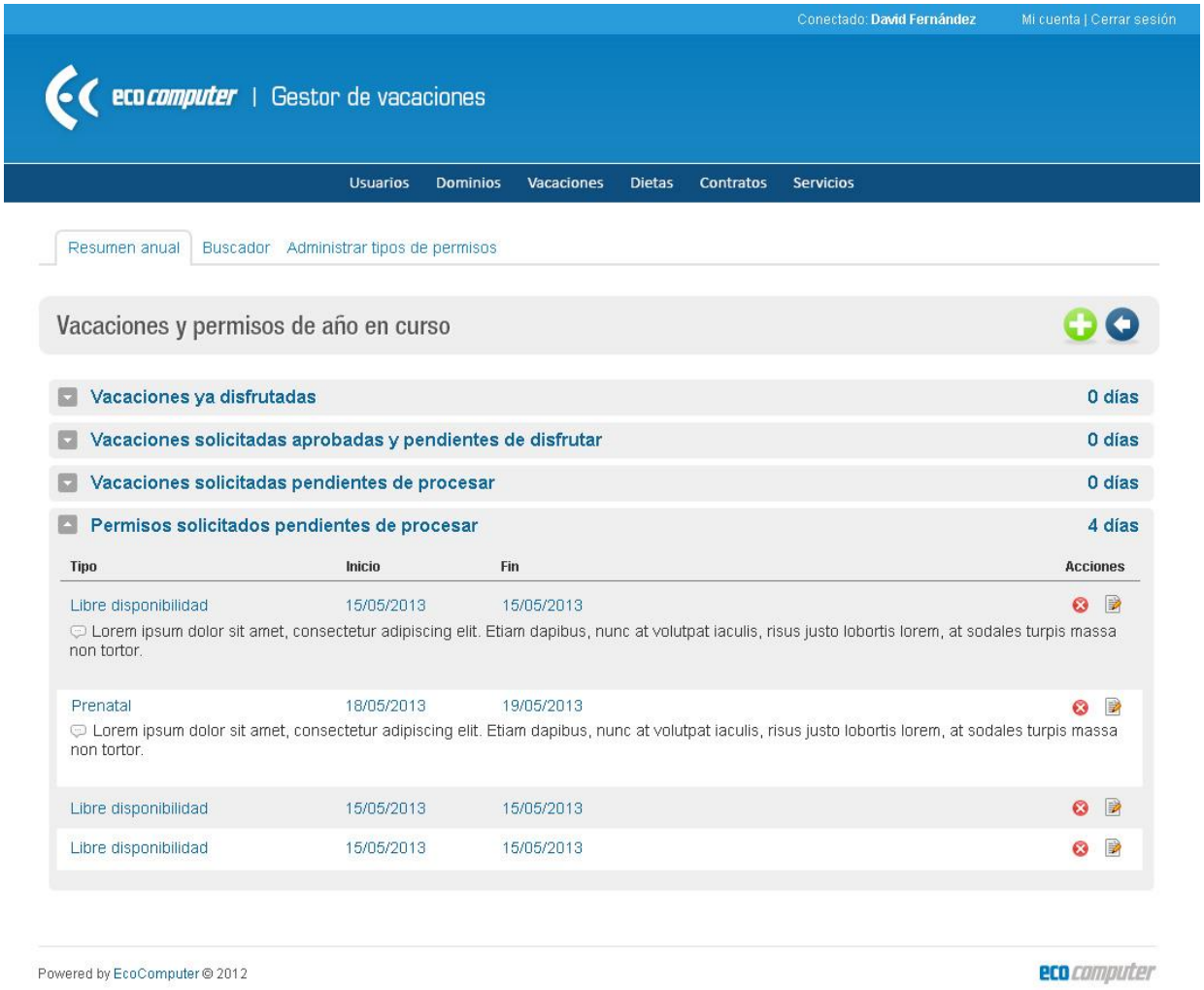


Figura 6.24 - Interfaz calendario laboral: resumen permisos

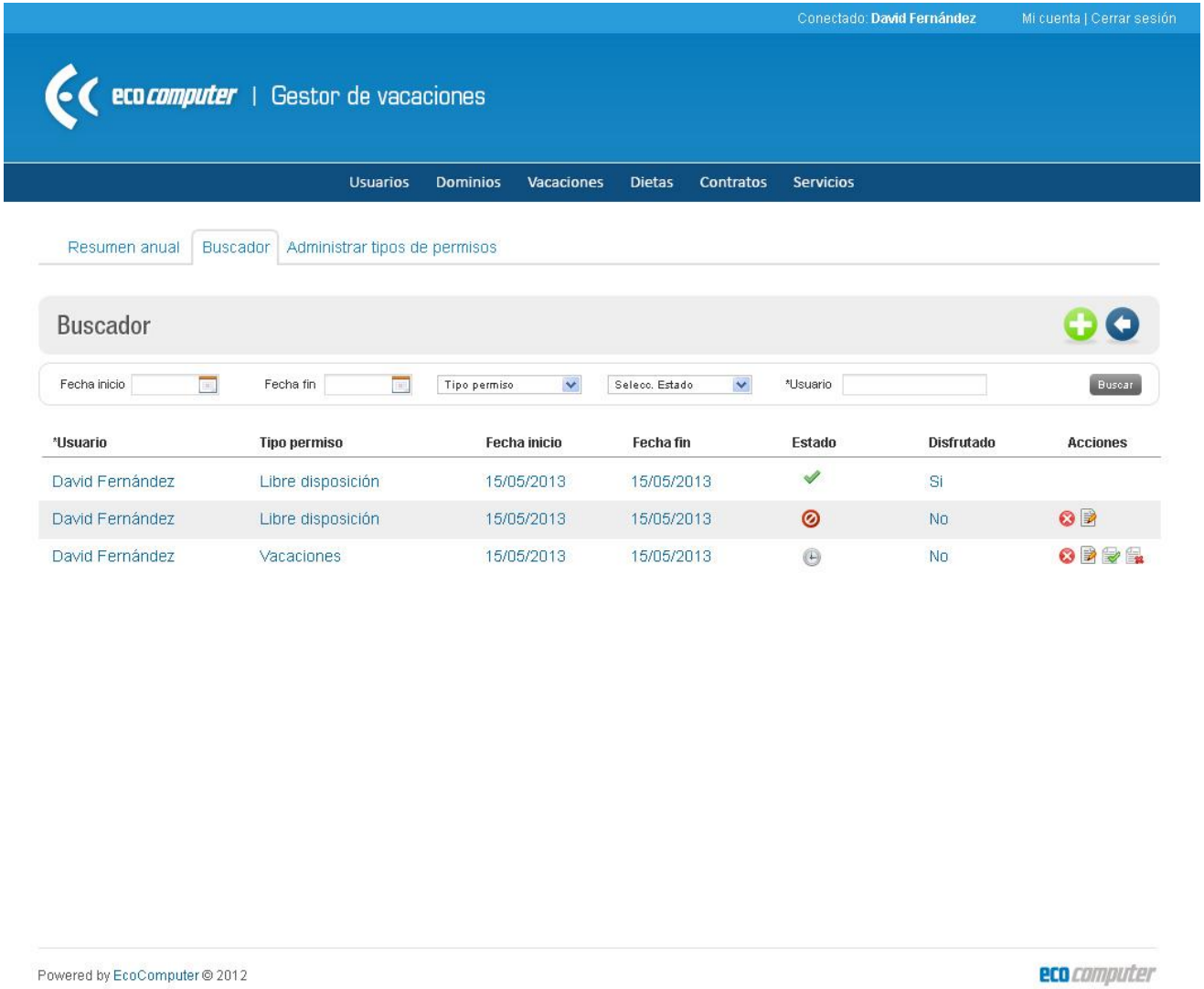


Figura 6.25 - Interfaz calendario laboral: buscador

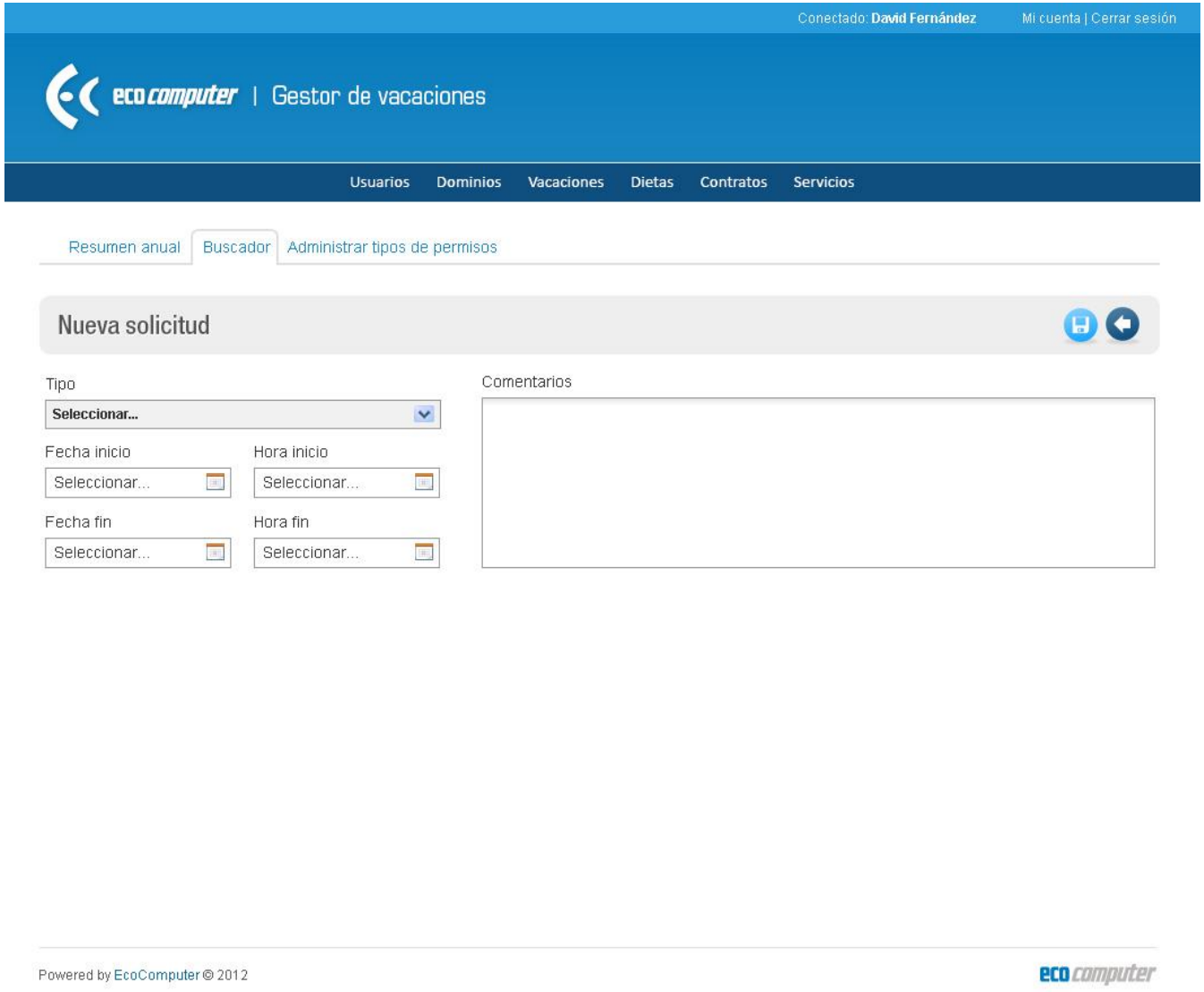


Figura 6.26 - Interfaz calendario laboral: nueva solicitud de permiso

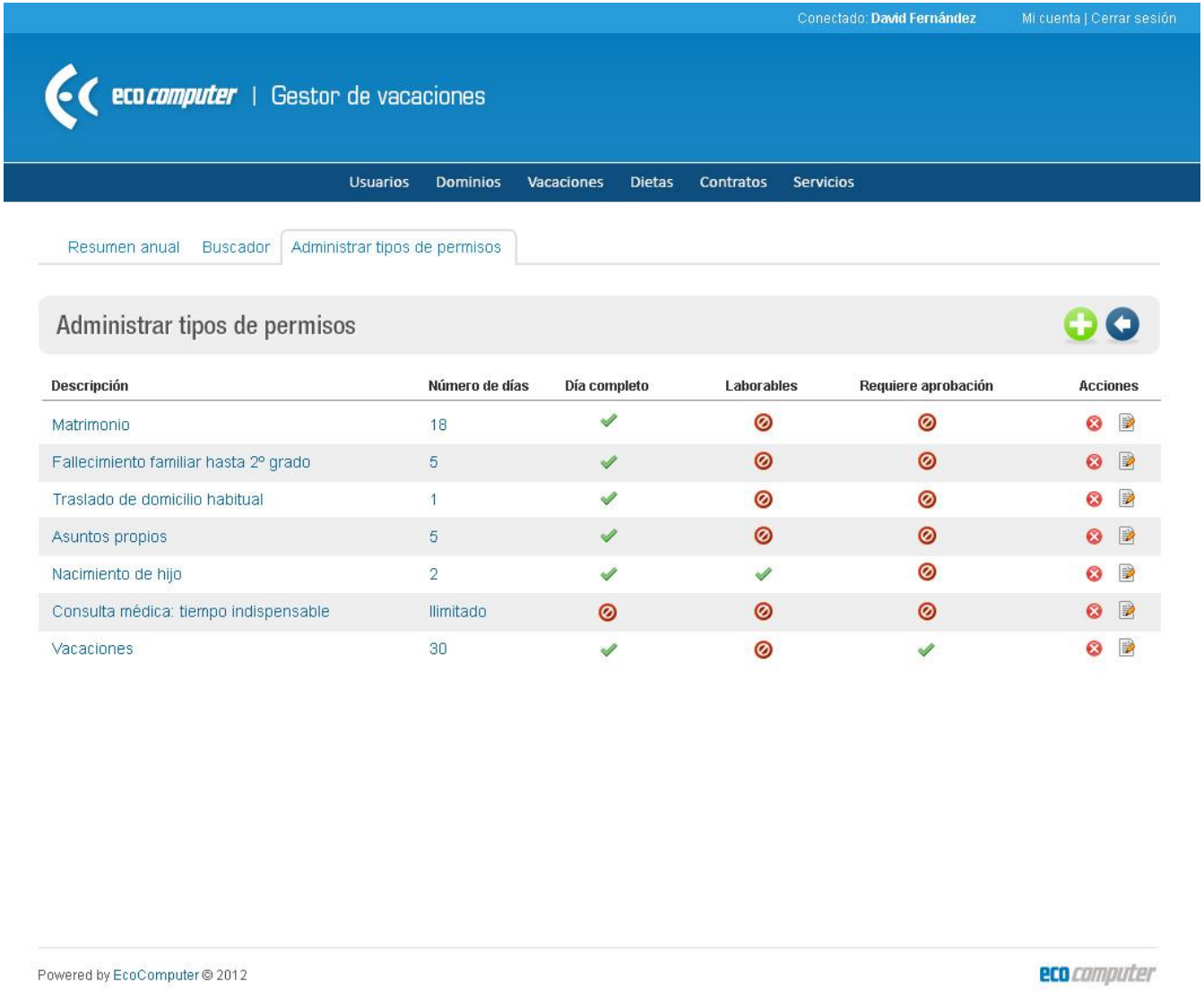


Figura 6.27 - Interfaz calendario laboral: administrador de tipos de permisos

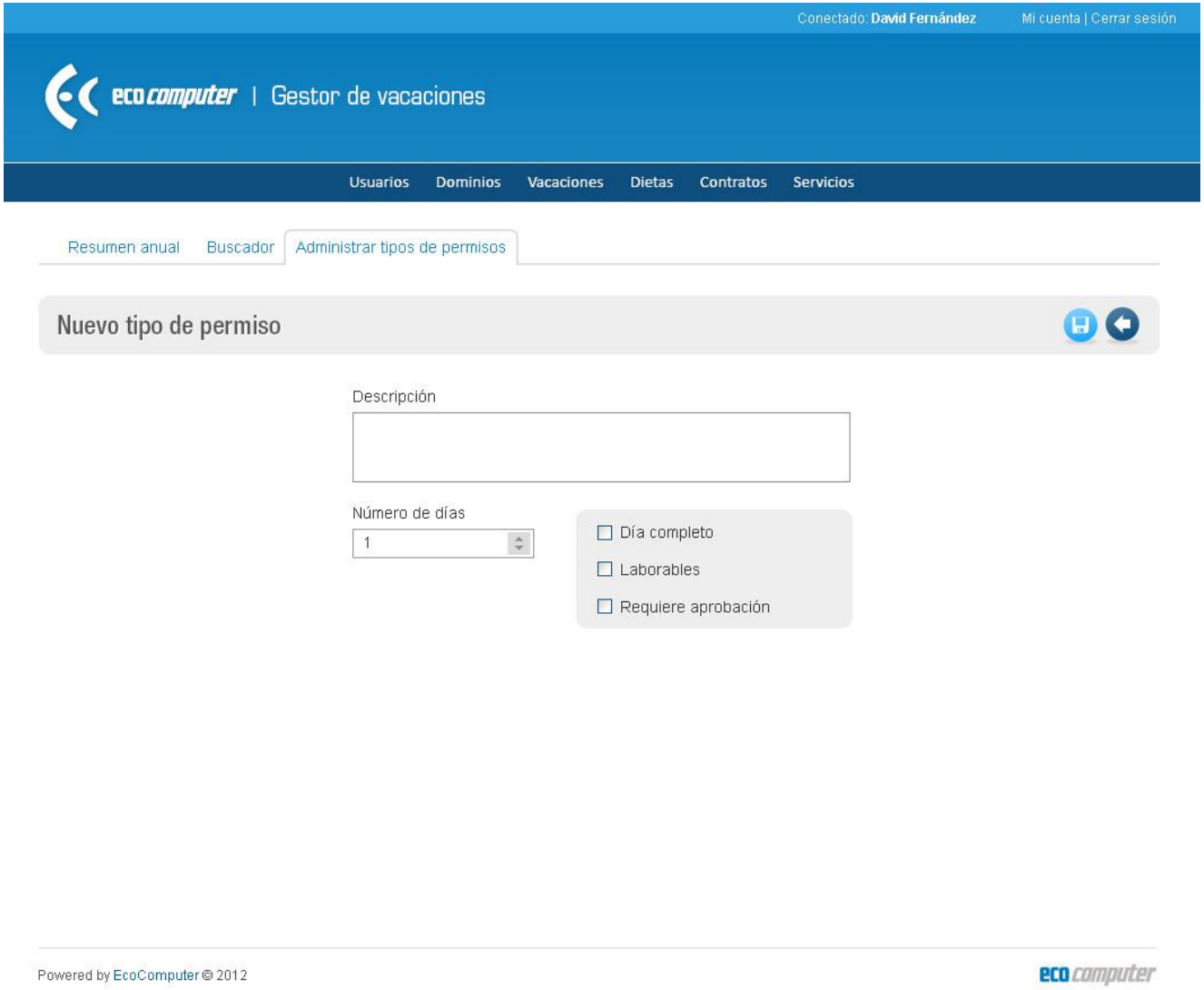


Figura 6.28 - Interfaz calendario laboral: nuevo tipo de permiso

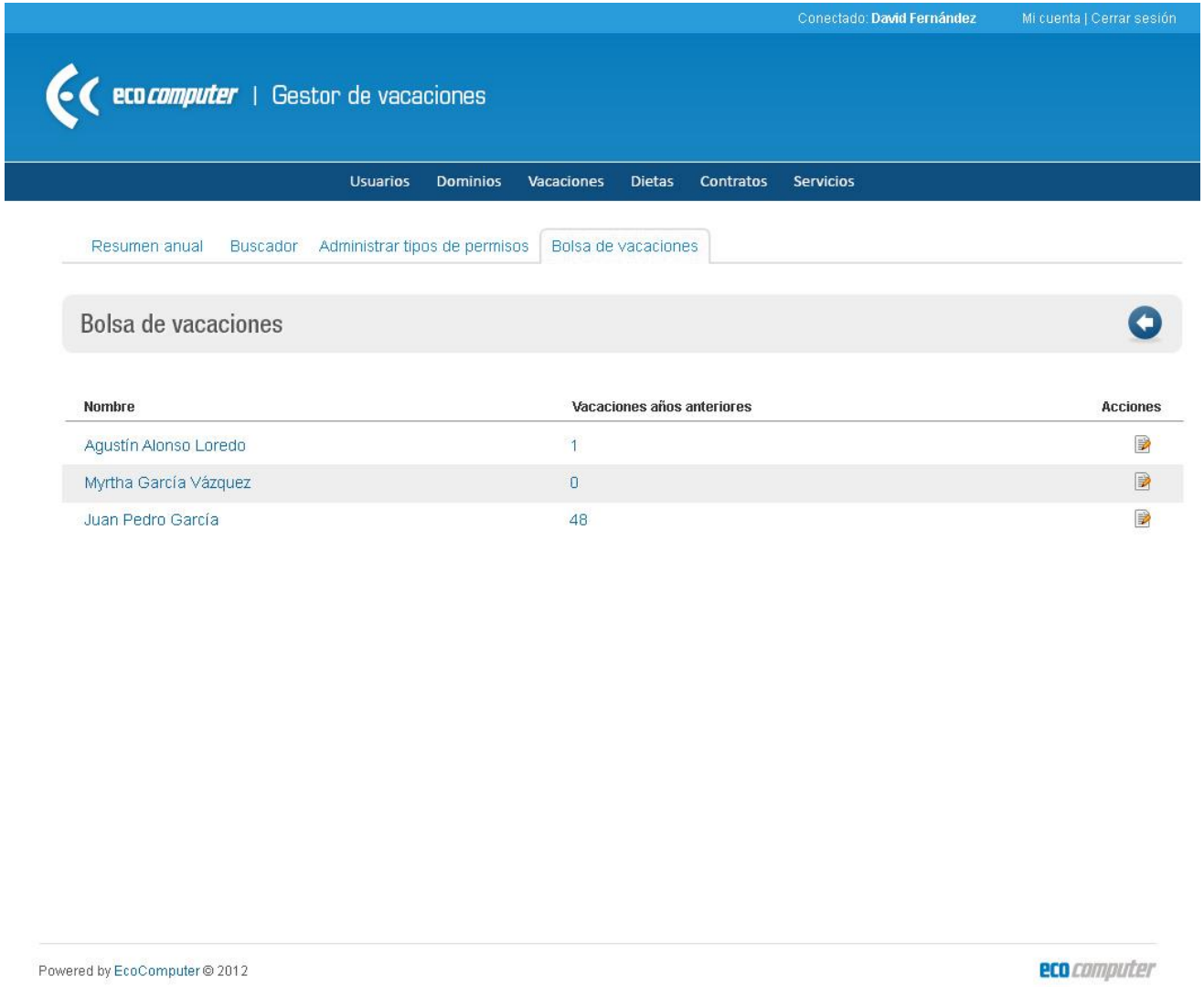


Figura 6.29 - Interfaz calendario laboral: bolsa de vacaciones

6.5.2 Módulo de gestión de dietas

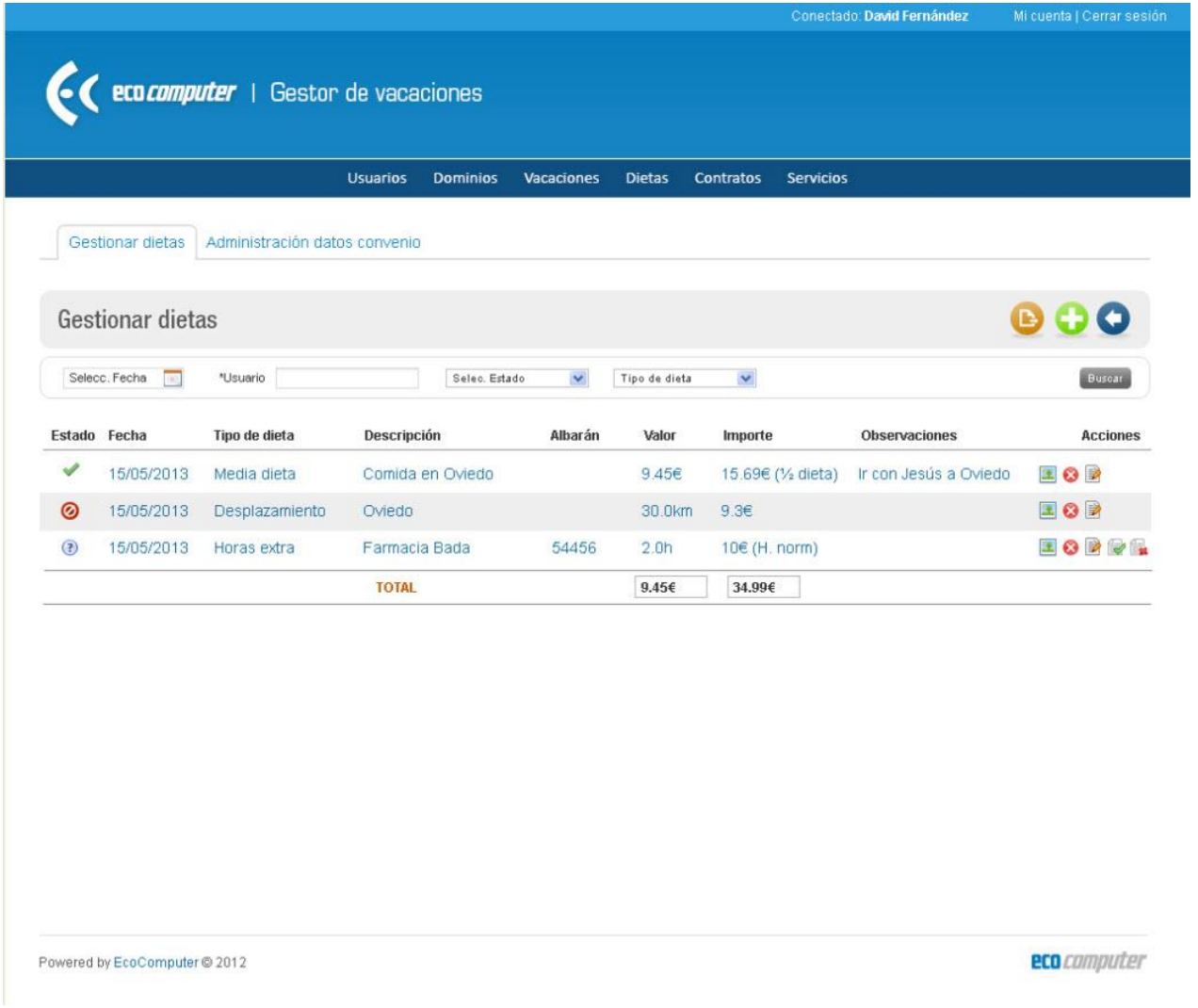


Figura 6.30 - Interfaz de gestión de dietas: gestor de dietas

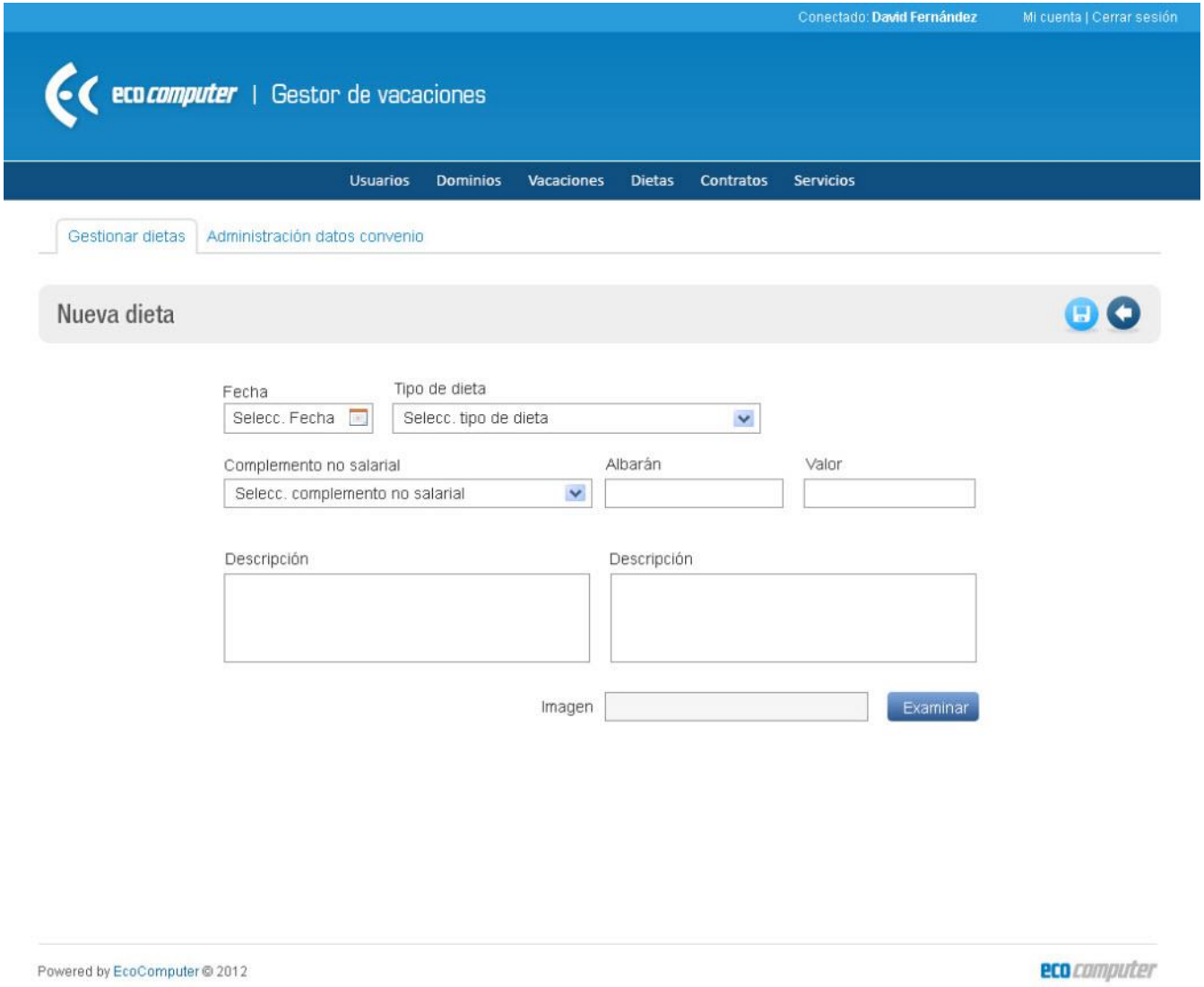


Figura 6.31 - Interfaz de gestión de dietas: nueva dieta

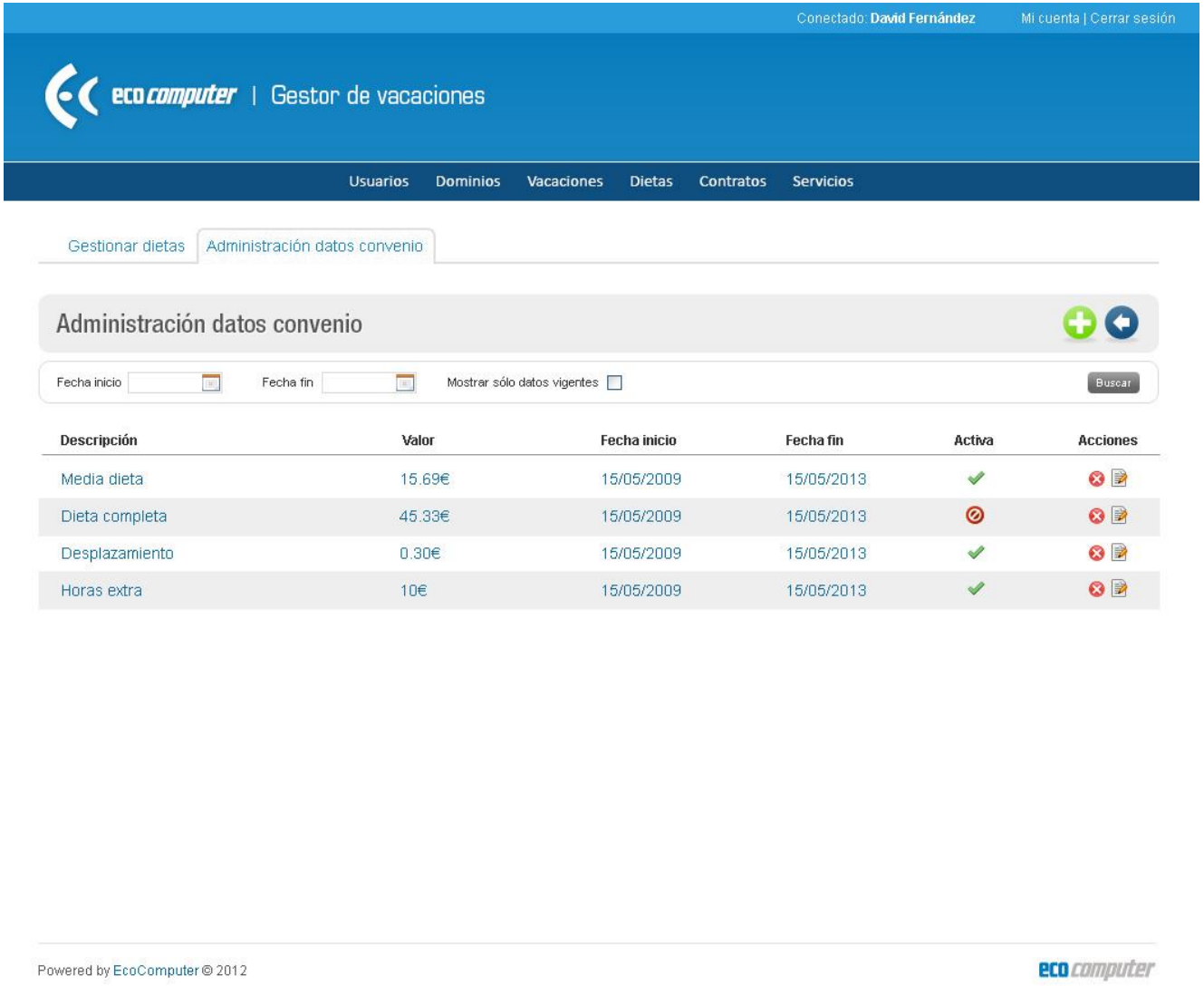


Figura 6.32 - Interfaz de gestión de dietas: administrador de complementos no salariales

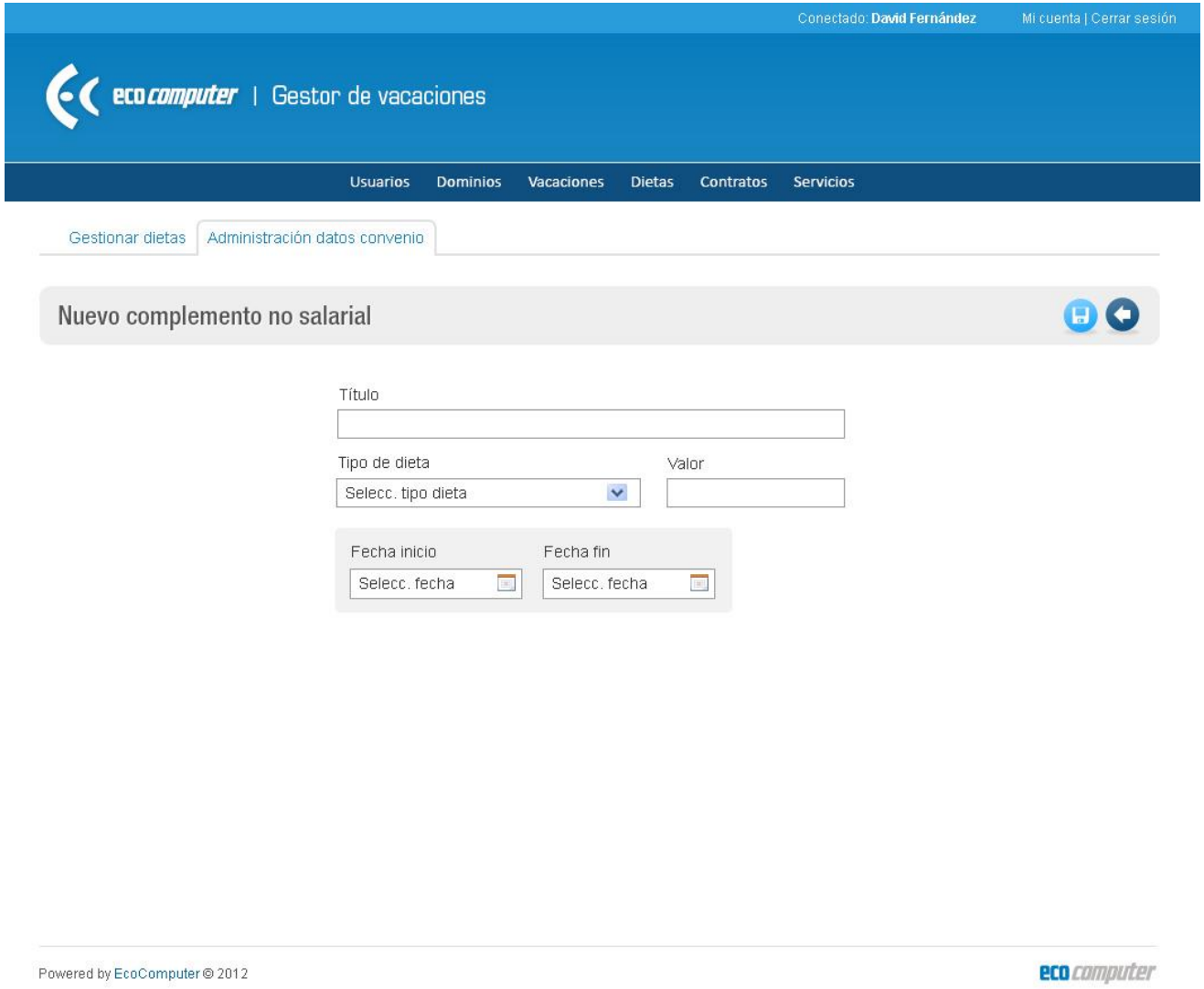


Figura 6.33 - Interfaz de gestión de dietas: nuevo complemento no salarial

6.6 Especificación Técnica del Plan de Pruebas

6.6.1 Pruebas Unitarias

Se realizarán pruebas unitarias sobre las clases del dominio que realiza las operaciones importantes de agregado de datos, modificación y demás casos de uso relevantes. Con ello pretendemos corroborar que los procesos serealizan correctamente, sin errores bajo los diferentes escenarios y que el resultado final es el esperado.

6.6.2 Pruebas de Integración y del Sistema

Mediante estas pruebas de integración se pretende comprobar que la combinación de cada módulo individual en el total funciona como es esperado. Se realizarán las definidas previamente en la fase de análisis. Habrá que teneren cuenta que las pruebas de visualización de interfaz son más directas y basadas en la experiencia del usuario: se comprobará su buen funcionamiento y eficacia de manera directa, observando la interfaz y los posibles cambios que en ella provoquen las operaciones de los demás módulos.

6.6.3 Pruebas de Accesibilidad

Considerando la naturaleza puramente web de esta aplicación y que está supeditada a los requisitos, se prioriza la adaptación a las preferencias del cliente por encima de la accesibilidad. No obstante, se realizarán las comprobaciones propuestas por el WCAG para intentar obtener la máxima accesibilidad posible sin sacrificar los requisitos del cliente en la medida de lo posible.

Capítulo 7. Implementación del Sistema

7.1 Estándares y Normas Seguidos

7.1.1 XTML 1.1

Acrónimo en inglés de ***eXtensible Hypertext Markup Language***, se trata del lenguaje de marcado pensado para ser estándar en sustitución de HTML. Es solamente la versión XML de HTML, que tiene la misma funcionalidad pero cumple con las especificaciones de XML, más estrictas. Tiene como objeto la obtención de una web semántica, en la que estén claramente separadas la información a mostrar de la forma de mostrarla.

Durante el desarrollo de la aplicación se intenta seguir lo máximo posible las normas establecidas por este estándar. Sin embargo, por la naturaleza de Struts2 y los deseos del cliente es muy posible que no se consiga un marcado que cumpla totalmente con la norma. Al no ser un requisito del cliente no se perseguirá la total perfección del estándar.

7.1.2 CSS 2

Acrónimo en inglés de ***Cascading Style Sheets***, u "hojas de estilo en cascada", es un lenguaje formal cuya función es la definir la presentación de un documento escrito en HTML o XML. Dichas hojas se pueden aplicar mediante un documento CSS externo llamado desde el archivo asociado, desde una hoja de estilo interna en el mismo documento, o insertando las propiedades CSS directamente dentro de las etiquetas HTML, si bien la opción recomendada por ser la más potente es la primera de ellas. Aplicando hojas de estilo conseguimos las siguientes ventajas:

- Control centralizado de la presentación de un sitio web, agilizando considerablemente su actualización.
- Posibilidad de que el usuario pueda especificar desde el navegador su propia hoja de estilos para cubrir algunas posibles necesidades como, por ejemplo, aumentar el tamaño del texto o cambiarlo de color.
- Las páginas pueden disponer de diferentes hojas de estilo en función del dispositivo que las lea. Así, la visualización de la web no será la misma desde un ordenador de sobremesa que en un dispositivo móvil, por ejemplo.
- Aumenta la legibilidad del código HTML por contener menos información, además de reducir considerablemente el tamaño del documento.

A diferencia de XHTML, seguir las normas establecidas por CSS es más sencillo puesto que no están tan supeditas a los deseos del cliente. Por tanto, se perseguirá que las hojas de estilo CSS validen completamente con el estándar.

7.1.3 Estándares de programación

Pese a que no es necesario para la correcta ejecución del código fuente, sí que es muy recomendable aplicar los estándares de cada lenguaje de programación utilizado. Seguir las normas dictadas aporta principalmente Mantenibilidad, entendimiento y legibilidad del código. Esto permite que la aplicación sea más fácil de entender y mantener por el propio autor original del código u otros programadores que deban hacerlo posteriormente.

Durante el desarrollo del proyecto se seguirán los estándares propios de cada lenguaje (organización física y lógica de ficheros, correcta declaración de objetos, adecuada indentación y separación…).

7.1.4 Modelo Vista Controlador

Este patrón de arquitectura de software se encarga de separar la los datos de la aplicación, la lógica de negocio y la interfaz de usuario que representa la información y recoge las interacciones del usuario. Aplicar este patrón supone aplicar las ideas de reutilización de código y separación de conceptos, lo que facilita el desarrollo y posterior mantenimiento de una aplicación software.

La realización del proyecto se ha planteado bajo la propuesta de este patrón, como ya se ha visto en los capítulos anteriores. Por un lado las JavaServer Pages muestran la información (vista), las Action recogen las interacciones del usuario (controlador) y los Manager, Service, DAO y demás clases se encargan de la lógica de negocio (modelo).

7.2 Lenguajes de Programación

7.2.1 Java

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directo de punteros o memoria.

Las aplicaciones Java están típicamente compiladas en un código intermedio, aunque la compilación en código máquina nativo es posible. En el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución.

La implementación original y de referencia del compilador, la máquina virtual (JVM) y las librerías de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process, si bien otros han desarrollado también implementaciones alternativas de estas tecnologías de Sun, algunas incluso bajo licencias de software libre.

Java es el lenguaje de programación utilizado en toda la lógica del proyecto que nos atañe. Además de ello, se utilizan componentes adicionales para lograr los objetivos deseados.

7.2.1.1 Struts

Struts es una herramienta de soporte para el desarrollo de aplicaciones web bajo el patrón modelo-vista-controlador para Java. Su utilización permite reducir considerablemente el tiempo de desarrollo, separando automáticamente las capas de interfaz de usuario, lógica de negocio y el acceso a datos. Esto lo realiza mediante Servlets, que se encargan de realizar las operaciones desencadenadas por los Actions del usuario en la interfaz HTML.

Este software libre fue creado originalmente por Craig McClanahan y donado a la Apache Foundation en el año 2000, pero no fue hasta 2005 cuando se convirtió en un proyecto de alta importancia para Apache. La versión empleada en este proyecto es la 2.2.1 por cuestiones de compatibilidad con el resto de software del proyecto.

Con el fin de agilizar, facilitar y asegurar el correcto desarrollo del proyecto y su separación en capas, se emplearán las librerías de Struts en el entorno de desarrollo eclipse.

7.2.1.2 Spring Framework

Spring es un conjunto de herramientas para el desarrollo de aplicaciones en Java. Aunque Spring no impone ningún modelo de programación específico, se ha popularizado entre los programadores al ser considerado una alternativa al modelo de Enterprise JavaBeans.

Sus módulos proveen una gran variedad de servicios, técnicas y paradigmas, como un contenedor de Inversión de Control, programación orientada a aspectos, acceso a datos,

gestor transaccional, modelo-vista-controlador, framework de acceso remoto, convención sobre configuración, procesamiento por lotes, autenticación y autorización, administración remota, mensajería y herramientas de testado.

Se trata de un software libre desarrollado por Rod Johnson cuya primera versión fue lanzada en 2004. En el desarrollo del proyecto se utilizará la versión 2.2.1 distribuida junto con las librerías de Struts.

Las librerías de Spring se incorporan en el entorno de desarrollo de eclipse para proporcionar de manera más directa las herramientas que ofrece. Concretamente, se utilizará sobre todo la funcionalidad concerniente a la inyección directa de dependencias.

7.2.1.3 Java Database Connectivity

Más conocido por sus siglas JDBC, se trata de una API que permite realizar de manera sencilla operaciones sobre una base de datos empleando el lenguaje Java, independientemente del sistema operativo y base de datos a la que se quiera acceder. Una vez establecida la conexión mediante los manejadores pertinentes, se puede realizar cualquier tipo de tarea con la base de datos, siempre y cuando se tengan en la misma los permisos adecuados, utilizando el conjunto de objetos e interfaces que pone a disposición del programador

Su desarrollo depende de Sun Microsystems puesto que se integra en la Java Development Kit desde su versión 1.1 de febrero de 1997. Su actual versión, JDBC 4.1, está incluida en la Java SE 7 de julio de 2011.

La utilización de las librerías de JDBC en el desarrollo del proyecto se limita a la conectividad de la aplicación con la base de datos SQL Server 2005 para solicitar o escribir información.

7.2.1.4 iText

iText se trata de una librería open source que permite la creación y manipulación de documentos PDF, RTF y HTML de una manera sencilla. Dispone de una gran cantidad de funcionalidades y características que la convierten en una potente herramienta para gestionar adecuadamente este tipo de información.

Programada para Java y C#, fue desarrollada por Bruno Lowagie y Paulo Soares, se encuentra en su versión 5.3.5, que ha sido la empleada en este proyecto. Las herramientas proporcionadas por la librería iText tienen como objetivo en este proyecto la creación de los ficheros PDF que se generan durante el uso de la gestión de dietas de la aplicación.

7.2.1.5 Joda-Time

La librería Java Joda-Time ofrece un reemplazo para los tipos estándar de Java relacionados con la fecha y hora, facilitando su manipulación e incorporando nuevas funcionalidades y operaciones con las que trabajar con dicho tipo de datos a través de su API. Se trata de una librería open source.

La versión 1.0 de Joda-Time vio la luz en 2005, y se encuentra actualmente en la 2.2, siendo la utilizada en el desarrollo de este proyecto. Su cometido es el de hacer más fácil para el programador el manejo de fechas en Java, haciendo uso de sus funcionalidades con el fin de ahorrar coste computacional y tiempo de programación.

7.2.2 JavaScript

JavaScript es un lenguaje de programación interpretado, dialecto del estándar ECMAScript. Se trata de un lenguaje orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Aunque existe una forma de JavaScript ejecutada en el servidor, normalmente se ejecuta en la máquina del cliente a través del navegador web, otorgando funcionalidad extra a aquellas páginas web que tengan componentes desarrollados con este lenguaje. También es significativo su uso en aplicaciones externas a la web, como widgets de escritorio o documentos PDF.

Aunque JavaScript pueda ser relacionado con el lenguaje Java, lo cierto es que no es así: aunque es de Java de quien adopta nombres y convenciones, tienen semánticas y propósitos diferentes, siendo el lenguaje C con quien comparte una sintaxis similar. También tiene influencias de otros lenguajes como Perl, Self o Python. Fue desarrollado originalmente por Brendan Eich de Netscape Communications Corp. en 1995 bajo el nombre de "Mocha", siendo el término "JavaScript" una marca registrada de Oracle Corporation. Su última versión estable es la 1.8.5 de marzo de 2011.

JavaScript se utiliza para otorgar dinamismo y funcionalidad extra a las JavaServer Pages que comprenden el proyecto. Como módulos adicionales, directamente vinculados a JavaScript, se emplea también:

7.2.2.1 jQuery

jQuery es una biblioteca, creada inicialmente por John Resig y presentada en enero de 2006, que simplifica notablemente la manera de interactuar con documentos HTML, su árbol DOM, manejar eventos, desarrollar animaciones y agregar interacción AJAX, entre otras.

En este proyecto se utiliza la versión 1.9.1 para facilitar y mejorar la ejecución del JavaScript empleado en las JavaServer Pages.

7.2.2.2 jQueryUI

jQueryUI se trata de una biblioteca creada como adición a jQuery, anunciada en septiembre de 2007, que le otorgan un conjunto de plugins, widgets y efectos visuales para mejorar la interacción del usuario con la página web.

La aplicación de su versión 1.10.2 en este proyecto pasa por añadir calendarios desplegables, tooltips más ricos, efectos visuales varios…

7.3 Herramientas y Programas Usados para el Desarrollo

7.3.1 Eclipse Juno

Eclipse es un entorno de desarrollo multiplataforma utilizado para la realización de proyectos de cierta envergadura, ya que da soporte a una gran cantidad de lenguajes de programación, como son C, C++, Fortran, Java, Perl, PHP, Python o Ruby, entre otros, gracias a las herramientas de desarrollo que incorpora y la posibilidad de instalar plugins adicionales.

El entorno de desarrollo integrado emplea módulos que pueden ser activados y desactivados en función de las necesidades de cada usuario, lo que le confiere una ligereza de ejecución considerable. Eclipse provee al programador de una serie de frameworks para realizar una gran cantidad de tareas relacionadas con el modelado y desarrollo de aplicaciones, además de un compilador interno de Java muy eficaz.

Se trata de una aplicación de software libre mantenida por la Eclipse Foundation, fundada en 2003, y su comunidad de usuarios, pese a que inicialmente fue producto de un proyecto de IBM. Su última versión estable, 4.2 y llamada Juno, es la empleada en el desarrollo de este proyecto.

Eclipse es la herramienta elegida para el desarrollo del proyecto, debido a que soporta todas las tecnologías y librerías necesarias, su ligereza, eficacia y la familiaridad del programador con el entorno.

7.3.2 SQL Server Management Studio Express

La aplicación SQL Server Management Studio Express es una herramienta empleada para configurar, manejar y administrar todos los componentes de una base de datos Microsoft SQL Server. Soporta tanto la entrada de datos por comandos como mediante las interfaces gráficas proporcionadas.

Fue desarrollada por el propio equipo de Microsoft y lanzada conjuntamente con SQL Server 2005. En este proyecto se utiliza la versión 9 “Express”, gratuita, puesto que es la última compatible con la versión del servidor de base de datos empleado. El motivo de su empleo aquí es obvio: la fácil administración de la base de datos del sistema.

7.3.3 Enterprise Architect

La aplicación Enterprise Architect, desarrollada por Sparx Systems, es una completa herramienta con la que crear todo tipo de diagramas UML. Utiliza la última especificación de UML y dispone de gran cantidad de funcionalidades extra además de las posibilidades de diseño que ofrece. Se trata de una herramienta comercial, si bien existe una versión de prueba de 30 días de uso.

Incluye soporte para los diagramas de comportamiento, compuestos por casos de uso, actividades, estado, interacción, secuencia y comunicación; además de diagramas estructurales como lo son los de paquetes, clases, objetos, composición, componentes y despliegue. Entre otras características, ofrece la creación automática de código fuente en más de diez lenguajes diferentes partiendo de los diagramas, y viceversa; reportes en formato HTML y RTF; manejo de requisitos; integración con entornos de desarrollo como Microsoft Visual Studio o Eclipse…

Enterprise Architect ha sido usado en el desarrollo de este proyecto como herramienta de modelaje de algunos diagramas, los cuales se pueden ver en las diferentes secciones de esta documentación. Concretamente, la versión utilizada de la aplicación ha sido la 6.0.

7.3.4 Dia

Dia es una aplicación con la que crear diagramas, desarrollada como parte del proyecto GNOME de sistemas Unix, si bien también existe una versión para Windows. Es un programa gratuito de código abierto distribuido bajo la Licencia Pública General de GNU.

Con Dia se pueden modelar diferentes tipos de diagramas, entre los que se encuentran los de UML, de flujo o incluso de circuitos eléctricos. Gracias al paquete dia2code, Dia también permite generar el esqueleto del código fuente a partir del diagrama UML diseñado. La aplicación fue originalmente creada por Alexander Larsson hasta que James Henstridge tomó su relevo como desarrollador jefe. Actualmente es mantenido por una serie de desarrolladores independientes. Su última versión estable, la 0.97.1, data de enero de 2010.

Al igual que con el Enterprise Architect, Dia ha sido empleado para el diseño de algunos diagramas, puesto que en algunos casos las herramientas de modelaje de una supera en prestaciones y comodidad a las del otro.

7.3.5 Microsoft Office Project

Project se trata de una aplicación desarrollada por Microsoft cuya función es la de administrar proyectos para poder mantener un control sobre el trabajo, la evolución del desarrollo, los plazos temporales y las finanzas, entre otros. Se trata de un software comercial que fue creado para DOS en 1984 a partir de las especificaciones de Alan M. Boyd de Microsoft, aunque fue una empresa de Seattle quien lo desarrolló para que Microsoft comprase los derechos en 1985.

Entre sus tareas se pueden destacar la posibilidad de organizar diferentes tareas, personal dedicado a cada una y el presupuesto previsto; la visualización y gestión del progreso, solucionando problemas del proceso y prediciendo posibles escenarios; y el control de las finanzas de los proyectos pudiendo elaborar presupuestos

Se ha utilizado la versión 2007 (12.0) para realizar la planificación temporal del proyecto, así como la definición y duración de sus diferentes fases. El resultado de ello puede verse en el capítulo 4 de esta documentación.

7.4 Creación del Sistema

7.4.1 Problemas encontrados

A continuación se enumeran y explican brevemente algunos de los problemas más importantes encontrados durante el desarrollo del proyecto, y qué se hizo para solucionarlos.

7.4.1.1 Retrasos por cambios de requisitos

Como ya se ha comentado en capítulos anteriores, se buscaba realizar un proyecto que tuviese un uso real en una empresa. Esto significa que la descripción de la aplicación, y por tanto los requisitos derivados de la misma, provienen directamente de los deseos del cliente transmitidos al desarrollador a través de reuniones, documentos o correos electrónicos.

Las necesidades del cliente pueden cambiar, nuevas ideas pueden surgir, la comunicación entre ambas partes puede no ser óptima… Son muchos los factores que pueden repercutir en el desarrollo de un proyecto. Y este proyecto no ha sido una excepción.

Por unas u otras razones, los requisitos de la aplicación cambiaron según se iba avanzando en el desarrollo, y aunque ya se contaba con la posibilidad de que sucediese, provocó retrasos y cancelación o modificación sustancial de parte de la funcionalidad inicial. Estos aspectos se encuentran más detallados en el Capítulo 4 de esta documentación, concerniente a la planificación del proyecto.

7.4.1.2 No disponibilidad de los recursos proporcionados por el cliente

Al tratarse de una aplicación que va a ser entregada al cliente, el servidor de producción y la base de datos se encuentran bajo el control del cliente. Pese a que el desarrollador tiene acceso a ciertos aspectos administrativos de ellos, no dejan de ser equipos localizados en las instalaciones de Ecocomputer, por lo que su mantenimiento y monitorización corre a cargo de la empresa. Esto en alguna ocasión ha provocado problemas por caídas del servidor, necesidad de administración de más alto nivel del disponible…

La solución a estos problemas pasaba por ponerse en contacto con la empresa y solicitar la asistencia que cada caso requiriese.

Por otro lado, y relacionado con esto, no podemos olvidar que Ecocomputer se trata de una empresa con sus proyectos propios, ajenos a esta aplicación, y clientes a los que hay que ofrecer soporte y asistencia. Comprensiblemente, unas tareas tienen mayor prioridad que otras, por lo que los empleados no siempre podían atender peticiones o dedicar el tiempo necesario en el momento deseado.

Este inconveniente se abordó simplemente dejando las tareas y funcionalidades afectadas en pausa mientras se dedicaba tiempo a otras que no requerían la atención directa de los empleados de la empresa.

7.4.1.3 Aparición de la lógica de complementos no salariales

Uno de los requisitos del cliente que no se contemplaron al inicio del desarrollo fue la posibilidad de que las dietas tuviesen asociados complementos no salariales y que estos pudiesen ser editados o eliminados. Esto se veía totalmente inabarcable en el modelo planteado, la lógica de la aplicación, las operaciones de acceso a base de datos…

Para poder incorporar dicha funcionalidad fue necesario rehacer una gran parte de la lógica de gestión de dietas, lo que provocó un considerable retraso en las fases finales de la implementación del proyecto. Por suerte, estar aplicando el patrón Modelo Vista Controlador evitó que este retraso fuese mucho mayor de lo que podría haber sido.

Finalmente, ya que la edición o eliminación de complementos no salariales repercutía directamente en los cálculos totales del importe a percibir, se decidió que aquellas dietas que se viesen afectadas por estos cambios no se modificasen automáticamente, si no que se notificaría al usuario de la discrepancia detectada para que él decidiese manualmente cómo actuar.

7.4.1.4 Creación de informe PDF

Para generar el informe PDF que el módulo de gestión de dietas requería fue necesario investigar diferentes librerías y su documentación. Ya que el cliente solo quería utilizar software libre gratuito, la elección se vio resuelta en favor de la librería iText.

Esta librería es sumamente completa, con una gran cantidad de clases Java propias y funcionalidades que incrementaban mucho la complejidad del desarrollo del código responsable de generar el informe. Muchas horas fueron empleadas en intentar lograr un resultado adecuado, sin éxito.

Finalmente, la solución adoptada pasó por utilizar la API que proporcionaba la librería para crear los documentos mediante notación HTML, no exenta de bugs, pero mucho más sencillo que utilizando las clases Java de iText. El resultado, quizá algo menos vistoso a nivel de código fuente, logró el objetivo deseado por el cliente.

7.4.1.5 Subida de imágenes al servidor

Este proceso, aparentemente trivial, generó una serie de problemas con los que no se contaba desde un principio. Por un lado, los mecanismos de struts2 para este tipo de campos de información no son tan intuitivos como cabría desear. Fue necesaria la implementación de una serie de validadores e interceptores que gestionasen que el fichero a subir fuese del formato correcto. También se realizaron una serie de modificaciones a nivel de configuración del servidor para permitir la subida de ficheros a una ruta relativa determinada de la forma y características requeridas.

La documentación de struts2 carecía de explicaciones adecuadas para lograr satisfactoriamente la meta deseada, por lo que hubo que recurrir a una serie de tutoriales, guías, explicaciones y foros no oficiales para llevar a cabo la tarea paso a paso.

7.4.1.6 Visualización de imágenes privadas

Parcialmente relacionado con el punto anterior, la visualización de las imágenes de las dietas del usuario a través de la web generó más dificultades de las esperadas. Para evitar problemas de seguridad y violación de privacidad las imágenes no podían estar localizadas en un directorio público al que cualquiera pudiese acceder a través de HTTP, tuviese o no la sesión iniciada en la aplicación. Por tanto, las imágenes debían ser retornadas a través de un Action al que solo tuviesen acceso los usuarios adecuados.

Nuevamente a través de información no oficial de struts2 se consiguió que un Action retornase una imagen que recibiese como parámetro. Varias clases adicionales y configuraciones internas tuvieron que ser añadidas para lograrlo.

En cuanto a mostrar las imágenes en los informes PDF no fue mayor problema, puesto que en la base de datos se almacena el nombre de la imagen. El proceso era tan sencillo como acceder a la ruta relativa y crear una etiqueta *img* en el código que genera el informe, ya que como hemos comentado anteriormente la generación se hace a través de HTML.

7.4.1.7 Posibilidad de eliminar tipos de permisos

Otro de los requisitos del cliente con los que no se contaba en un principio fue la posibilidad de eliminar tipos de permiso para el módulo de gestión de calendario laboral de los empleados. Este requisito no supuso tanta dificultad como el visto anteriormente sobre los complementos no salariales, pero aun así hubo que replantear el modelo inicial.

Obviamente, eliminar un tipo de permiso que ya esté siendo utilizado por una dieta causa problemas de integridad en la base de datos y la aplicación en general. La solución adoptada pasa por actualizar el tipo de permiso para activar un bit “Eliminado” en lugar de eliminarlo. De este modo, el tipo de permiso no puede volver a ser seleccionado, y las dietas que lo tengan ya asignado pueden fácilmente avisar al usuario que el tipo de permiso ya no es válido.

7.4.2 Descripción Detallada de las Clases

A continuación se muestra una descripción de todas las clases, con todos sus atributos y métodos. Para ello se usa una tabla prototipo donde aparece ordenada toda la información requerida.

Cabe señalar que se han omitido los numerosos métodos “getters” y “setters” de atributos de la aplicación, puesto que no aportan explicación alguna al funcionamiento del sistema y solo alargarían innecesariamente la longitud de las tablas.

7.4.2.1 Módulo de gestión del calendario laboral

7.4.2.1.1 BolsaManager

Nombre	Tipo	Implementa
BolsaManager		BolsaManagerService
Descripción		
Se encarga de gestionar las acciones relacionadas con la bolsa de vacaciones del usuario, llamando a unos métodos u otros para cada operación.		
Métodos		
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<Bolsa>	getDiasAcumulados	
public	int	getDiasAcumulados	Int idUsuario
public	void	actualizarBolsa	Bolsa bolsa
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		BolsaDataService	bolsaDataService
Observaciones			

7.4.2.1.2 PermisoManager

Nombre	Tipo	Implementa
PermisoManager		PermisoManagerService
Descripción		
Se encarga de gestionar las acciones relacionadas con los permisos de vacaciones del usuario, llamando a unos métodos u otros para cada operación.		
Métodos		
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	insertarPermiso	Permiso permiso
public	List<Permiso>	getPermisosUsuarioMes	int idusuario, int mes, int anio
public	List<Permiso>	getPermisosAnio	
public	List<Permiso>	getPermisosCriterios	String fechaInicio, String fechaFin, int idTipoPermiso, int estado, int idUsuario
public	void	eliminarPermiso	int idPermiso
public	void	autorizarPermiso	int idPermiso
public	void	denegarPermiso	int idPermiso
public	Permiso	getPermisoById	int idPermiso
public	void	modificarPermiso	Permiso permiso
public	int	getDiasAcumPermisoAnio	int idUsuario, int idTipoPermiso
Atributos		
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		BolsaDataService	bolsaDataService
private		PermisoDataService	permisoDataService
private		UsuarioDataService	usuarioDataService
private		TipoPermisoDataService	tipoPermisoDataService
Observaciones		

7.4.2.1.3 TipoPermisoManager

Nombre	Tipo	Implementa
TipoPermisoManager		

Descripción
Se encarga de gestionar las acciones relacionadas con los tipos de permisos, llamando a unos métodos u otros para cada operación.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	insertarTipoPermiso	TipoPermiso tipo
public	List<TipoPermiso>	getTiposPermisos	
public	void	eliminarTipoPermiso	int idTipoPermiso
public	TipoPermiso	getTipoPermisoById	int idTipoPermiso
public	void	modificarTipoPermiso	TipoPermiso tipo

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		TipoPermisoDataService	tipoPermisoDataService

Observaciones

7.4.2.1.4 EstadosPermisosUtil

Nombre	Tipo	Implementa
EstadosPermisosUtil		

Descripción
Clase de utilidad de la que se sirven otras clases y se encarga de rellenar una colección con el nombre de los estados de los permisos.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public static	Map<Integer, String>	rellenarEstados	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.1.5 FechaUtil

Nombre	Tipo	Implementa
FechaUtil		

Descripción
Clase de utilidad de la que se sirven otras clases para resolver operaciones comunes relacionadas con la gestión de fechas.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public static	List<DateTime>	daysBetween	Date inicio, Date fin
public static	boolean	isFinDeSemana	DateTime dia
public static	String	getFirstDayYear	
public static	String	getLastDayYear	
public static	String	conversorFecha	Date date
public static	boolean	isInicioAnteriorFin	String inicio, String fin

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.1.6 BolsaDAO

Nombre	Tipo	Implementa
BolsaDAO		BolsaDataService

Descripción			
Implementa los métodos dictados por su interfaz que se encargan del acceso a base de datos de todo lo relacionado con la bolsa de vacaciones del usuario.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
private	void	abrirConexion	
private	void	cerrarConexion	
public	List<Bolsa>	getDiasAcumulados	
public	int	getDiasAcumulados	int idUsuario
public	void	actualizarBolsa	Bolsa bolsa
private	void	crearBolsa	int idUsuario
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Connection	con
private		PreparedStatement	ps
private		ResultSet	rs
Observaciones			

7.4.2.1.7 PermisoDAO

Nombre	Tipo	Implementa
PermisoDAO		PermisoDataService

Descripción
Implementa los métodos dictados por su interfaz que se encargan del acceso a base de datos de todo lo relacionado con los permisos del usuario.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
private	void	abrirConexion	
private	void	cerrarConexion	
public	void	insertarPermiso	Permiso permiso
public	List<Permiso>	getPermisosUsuarioMes	int idusuario, int mes, int anio
public	List<Permiso>	getPermisosAnio	
public	List<Permiso>	getPermisosCriterios	String fechaInicio, String fechaFin, int idTipoPermiso, int estado, int idUsuario
public	void	eliminarPermiso	int idPermiso
public	void	autorizarPermiso	int idPermiso
public	void	denegarPermiso	int idPermiso
public	Permiso	getPermisoById	int idPermiso
public	void	modificarPermiso	Permiso permiso
public	int	getDiasAcumPermisoAnio	int idUsuario, int idTipoPermiso

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Connection	con
private		PreparedStatement	ps
private		ResultSet	rs

Observaciones

7.4.2.1.8 TipoPermisoDAO

Nombre	Tipo	Implementa
TipoPermisoDAO		TipoPermisoDataService

Descripción
Implementa los métodos dictados por su interfaz que se encargan del acceso a base de datos de todo lo relacionado con los tipos de permiso.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
private	void	abrirConexion	
private	void	cerrarConexion	
public	void	insertarTipoPermiso	TipoPermiso tipo
public	List<TipoPermiso>	getTiposPermisos	
public	void	eliminarTipoPermiso	int idTipoPermiso
public	TipoPermiso	getTipoPermisoById	int idTipoPermiso
public	void	modificarTipoPermiso	TipoPermiso tipo

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Connection	con
private		PreparedStatement	ps
private		ResultSet	rs

Observaciones

7.4.2.1.9 InterfazBolsaAction

Nombre	**Tipo**	**Implementa**	**Hereda de**
InterfazBolsaAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de preparar la interfaz de gestión de la bolsa de vacaciones de los usuarios.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		List<Bolsa>	bolsas
private		UsuarioManagerService	usuarioManagerService
private		BolsaManagerService	bolsaManagerService
Observaciones			

7.4.2.1.10 ModificarBolsaAction

Nombre	Tipo	Implementa	Hereda de
ModificarBolsaAction		SessionAware, ModelDriven<Bolsa>	ActionSupport

Descripción
Action que se encarga de recoger el nuevo valor de bolsa y llamar a la siguiente capa de la arquitectura para actualizarlo.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	void	validate	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		Bolsa	bolsa
private		BolsaManagerService	bolsaManagerService

Observaciones

7.4.2.1.11 BuscarAction

Nombre	Tipo	Implementa	Hereda de
BuscarAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger los datos introducidos por el usuario y realiza una búsqueda de permisos en base a ellos.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	void	validate	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		List<Usuario>	usuarios
private		Map<Integer, String>	estados
private		List<TipoPermiso>	tiposPermiso
private		List<Permiso>	permisos
private		String	busc_fechaInicio
private		String	busc_fechaFin
private		int	busc_idTipoPermiso
private		int	busc_estado
private		int	busc_idUsuario
private		UsuarioManagerService	usuarioManagerService
private		TipoPermisoManagerService	tpManagerService
private		PermisoManagerService	permisoManagerService
Observaciones			

7.4.2.1.12 InterfazBuscadorAction

Nombre	Tipo	Implementa	Hereda de
InterfazBuscadorAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de preparar la interfaz de búsqueda de permisos de los usuarios.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		List<Usuario>	usuarios
private		Map<Integer, String>	estados
private		List<TipoPermiso>	tiposPermiso
private		List<Permiso>	permisos
private		String	busc_fechaInicio
private		String	busc_fechaFin
private		int	busc_idTipoPermiso
private		int	busc_estado
private		int	busc_idUsuario
private		UsuarioManagerService	usuarioManagerService
private		TipoPermisoManagerService	tpManagerService
private		PermisoManagerService	permisoManagerService
Observaciones			

7.4.2.1.13 VisualizarCalendarioAction

Nombre	Tipo	Implementa	Hereda de
VisualizarCalendarioAction		SessionAware	ActionSupport

Descripción
Action que se encarga de mostrar el calendario mensual con los permisos del usuario y sus estados.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		Int	mes
private		Int	anio
private		List<Permiso>	permisos
private		PermisoManagerService	permisoManagerService

Observaciones

7.4.2.1.14 MostrarResumenAction

Nombre	Tipo	Implementa	Hereda de
MostrarResumenAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de preparar la interfaz de gestión de la bolsa de vacaciones de los usuarios.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		List<Permiso>	permisosNoConcluidos
private		List<Permiso>	permisosConcluidos
private		List<Permiso>	vacacionesPendientes
private		int	diasPermisosConcl
private		int	diasPermisosNoConcl
private		int	diasVacacionesPend
private		PermisoManagerService	permManagerService
Observaciones			

7.4.2.1.15 EliminarSolicitudAction

Nombre	**Tipo**	**Implementa**	**Hereda de**
EliminarSolicitudAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger la acción del usuario para eliminar una solicitud de permiso y llama a la siguiente capa de la arquitectura para realizarlo.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		String	busc_fechaInicio
private		String	busc_fechaFin
private		int	busc_idTipoPermiso
private		int	busc_estado
private		int	busc_idUsuario
private		int	idPermiso
Observaciones			

7.4.2.1.16 EstadosSolicitudAction

Nombre	Tipo	Implementa	Hereda de
EstadosSolicitudAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger la acción del usuario para cambiar el estado de un permiso y llama a la siguiente capa de la arquitectura para realizarlo.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	autorizar	
public	String	denegar	
public	void	validate	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		String	busc_fechaInicio
private		String	busc_fechaFin
private		int	busc_idTipoPermiso
private		int	busc_estado
private		int	busc_idUsuario
private		int	idPermiso
private		PermisoManagerService	permisoManagerService
private		BolsaManagerService	bolsaManagerService
Observaciones			

7.4.2.1.17 InsertarSolicitudAction

Nombre	Tipo	Implementa	Hereda de
InsertarSolicitudAction		SessionAware, ModelDriven<Permiso>	ActionSupport

Descripción

Action que se encarga de recoger los datos para insertar un permiso y llama a la siguiente capa de la arquitectura para realizarlo.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	
public	void	validate	
public	String	esPosible	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		Permiso	permiso
private		int	id
private		PermisoManagerService	permisoManagerService
private		BolsaManagerService	bolsaManagerService
private		TipoPermisoManagerService	tpManagerService

Observaciones

7.4.2.1.18 ModificarSolicitudAction

Nombre	Tipo	Implementa	Hereda de
ModificarSolicitudAction		SessionAware, ModelDriven<Permiso>	ActionSupport

Descripción
Action que se encarga de recoger los datos para modificar un permiso y llama a la siguiente capa de la arquitectura para realizarlo.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	void	validate	
public	String	esPosible	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		Permiso	permiso
private		int	idPermiso
private		int	idTipoPermiso
private		PermisoManagerService	permisoManagerService
private		BolsaManagerService	bolsaManagerService
private		TipoPermisoManagerService	tpManagerService

Observaciones

7.4.2.1.19 NuevaSolicitudAction

Nombre	Tipo	Implementa	Hereda de
NuevaSolicitudAction		SessionAware, ApplicationAware	ActionSupport

Descripción
Action que se encarga de preparar la interfaz para insertar o editar un permiso.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	
public	String	modificando	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		Map<String, Object>	application
private		Permiso	permiso
private		int	idPermiso
private		PermisoManagerService	permisoManagerService
private		TipoPermisoManagerService	tpManagerService

Observaciones

7.4.2.1.20 ObtenerDatosTipoAction

Nombre	Tipo	Implementa	Hereda de
ObtenerDatosTipoAction		SessionAware	ActionSupport

Descripción

Action que se encarga de obtener los datos asociados a un tipo de permiso para mostrárselos al usuario cuando esté insertando uno.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		Map<String, Object>	application
private		TipoPermiso	tipoPermiso
private		Bolsa	bolsa
private		int	idTipo
private		int	diasGastados
private		PermisoManagerService	permisoManagerService
private		BolsaManagerService	bolsaManagerService
private		TipoPermisoManagerService	tpManagerService

Observaciones

7.4.2.1.21 EliminarTipoAction

Nombre	**Tipo**	**Implementa**	**Hereda de**
EliminarTipoAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger la orden de usuario de eliminar un tipo de permiso y llamar a la siguiente capa de la arquitectura para realizarlo.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		int	idTipoPermiso
private		TipoPermisoManagerService	tpManagerService
Observaciones			

7.4.2.1.22 InsertarTipoAction

Nombre	Tipo	Implementa	Hereda de
InsertarTipoAction		SessionAware, ModelDriven<TipoPermiso>	ActionSupport
Descripción			
Action que se encarga de recoger la orden de usuario de insertar un tipo de permiso, sus datos y llamar a la siguiente capa de la arquitectura para realizarlo.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	void	validate	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		TipoPermiso	tipoPermiso
private		TipoPermisoManagerService	tpManagerService
Observaciones			

7.4.2.1.23 ModificarTipoAction

Nombre	Tipo	Implementa	Hereda de
ModificarTipoAction		SessionAware, ModelDriven<TipoPermiso>	ActionSupport

Descripción

Action que se encarga de recoger la orden de usuario de modificar un tipo de permiso, sus datos y llamar a la siguiente capa de la arquitectura para realizarlo.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	
public	void	validate	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		TipoPermiso	tipoPermiso
private		TipoPermisoManagerService	tpManagerService

Observaciones

7.4.2.1.24 MostrarTiposAction

Nombre	Tipo	Implementa	Hereda de
MostrarTiposAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recargar la interfaz con los tipos de permiso disponibles.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		List<TipoPermiso>	tiposPermiso
private		TipoPermisoManagerService	tpManagerService
Observaciones			

7.4.2.1.25 NuevoTipoAction

Nombre	**Tipo**	**Implementa**	**Hereda de**
NuevoTipoAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de preparar la interfaz de inserción o edición de tipos de permiso.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	String	modificar	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		TipoPermiso	tipoPermiso
private		int	idTipoPermiso
private		TipoPermisoManagerService	tpManagerService
Observaciones			

7.4.2.1.26 BolsaManagerService

Nombre	Tipo	Implementa
BolsaManagerService	interface	

Descripción
Interfaz que dicta las signaturas de los métodos que deberá implementar BolsaManager.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<Bolsa>	getDiasAcumulados	
public	Bolsa	getDiasAcumulados	int idUsuario
public	void	actualizarBolsa	Bolsa bolsa

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.1.27 PermisoManagerService

Nombre	Tipo	Implementa
PermisoManagerService		
Descripción		
Interfaz que dicta las signaturas de los métodos que deberá implementar PermisoManager.		
Métodos		
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	insertarPermiso	Permiso permiso
public	List<Permiso>	getPermisosUsuarioMes	int idusuario, int mes, int anio
public	List<Permiso>	getPermisosAnio	
public	List<Permiso>	getPermisosCriterios	String fechaInicio, String fechaFin, int idTipoPermiso, int estado, int idUsuario
public	void	eliminarPermiso	int idPermiso
public	void	autorizarPermiso	int idPermiso
public	void	denegarPermiso	int idPermiso
public	Permiso	getPermisoById	int idPermiso
public	void	modificarPermiso	Permiso permiso
public	int	getDiasAcumPermisoAnio	int idUsuario, int idTipoPermiso
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Observaciones			

7.4.2.1.28 TipoPermisoManagerService

Nombre	Tipo	Implementa
TipoPermisoManager Service		

Descripción
Se encarga de gestionar las acciones relacionadas con los tipos de permisos, llamando a unos métodos u otros para cada operación.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	insertarTipoPermiso	TipoPermiso tipo
public	List<TipoPermiso>	getTiposPermisos	
public	void	eliminarTipoPermiso	int idTipoPermiso
public	TipoPermiso	getTipoPermisoById	int idTipoPermiso
public	void	modificarTipoPermiso	TipoPermiso tipo

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.1.29 Bolsa

Nombre	Tipo	Implementa
Bolsa		Serializable

Descripción
Clase del modelo que representa al objeto bolsa de vacaciones del usuario.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public		Bolsa	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		int	idUsuario
private		int	diasAcumulados
private		Usuario	usuario

Observaciones

7.4.2.1.30 Permiso

Nombre	Tipo	Implementa
Permiso		Serializable

Descripción			
Clase del modelo que representa al objeto permiso.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public		Permiso	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		int	idPermiso
private		int	idUsuario
private		int	idTipoPermiso
private		Date	fechaInicio
private		Date	fechaFin
private		String	comentarios
private		int	estado
private		boolean	eliminado
private		TipoPermiso	tipoPermiso
private		Usuario	usuario
Observaciones			

7.4.2.1.31 BolsaDataService

Nombre	Tipo	Implementa
BolsaDataService	interface	

Descripción
Interfaz que dicta las signaturas de los métodos que tiene que implementar BolsaDAO.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	insertarTipoPermiso	TipoPermiso tipo
public	List<TipoPermiso>	getTiposPermisos	
public	void	eliminarTipoPermiso	int idTipoPermiso
public	TipoPermiso	getTipoPermisoById	int idTipoPermiso
public	void	modificarTipoPermiso	TipoPermiso tipo

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.1.32 PermisoDataService

Nombre	Tipo	Implementa
PermisoDataService	interface	

Descripción
Interfaz que dicta las signaturas de los métodos que tiene que implementar PermisoDAO.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	insertarPermiso	Permiso permiso
public	List<Permiso>	getPermisosUsuarioMes	int idusuario, int mes, int anio
public	List<Permiso>	getPermisosAnio	
public	List<Permiso>	getPermisosCriterios	String fechaInicio, String fechaFin, int idTipoPermiso, int estado, int idUsuario
public	void	eliminarPermiso	int idPermiso
public	void	autorizarPermiso	int idPermiso
public	void	denegarPermiso	int idPermiso
public	Permiso	getPermisoById	int idPermiso
public	void	modificarPermiso	Permiso permiso
public	int	getDiasAcumPermisoAnio	int idUsuario, int idTipoPermiso

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.1.33 TipoPermisoDataService

Nombre	Tipo	Implementa
TipoPermisoDataService	interface	

Descripción
Interfaz que dicta las signaturas de los métodos que tiene que implementar TipoPermisoDAO.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	insertarTipoPermiso	TipoPermiso tipo
public	List<TipoPermiso>	getTiposPermisos	
public	void	eliminarTipoPermiso	int idTipoPermiso
public	TipoPermiso	getTipoPermisoById	int idTipoPermiso
public	void	modificarTipoPermiso	TipoPermiso tipo

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.2 Módulo de gestión de dietas

7.4.2.2.1 ComplementoManager

Nombre	Tipo	Implementa
ComplementoManager		

Descripción			
Se encarga de gestionar las acciones relacionadas con los complementos no salariales de las dietas, llamando a unos métodos u otros para cada operación.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	nuevoConvenio	Complemento compl
public	List<Complemento>	getComplementos	
public	void	eliminarConvenio	int idComplemento
public	Complemento	getComplementoById	int idComplemento
public	void	editarConvenio	Complemento compl
public	List<Complemento>	getComplByTipoDieta	Int idTipoDieta, String fecha
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		ComplementoDataService	complementoDataService
Observaciones			

7.4.2.2.2 DietaManager

Nombre	Tipo	Implementa
DietaManager		

Descripción
Se encarga de gestionar las acciones relacionadas con las dietas y los detalles de las dietas, llamando a unos métodos u otros para cada operación.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<Dieta>	getTodasDietas	int idUsuario, String fecha
public	int	insertarDetalle	Detalle detalle, int idUsuario, String fecha
public	List<Dieta>	getDietasUsuarioDia	int idUsuario, String fecha
public	void	eliminarDietaDetalle	int idDietaDetalle
public	void	modificarDietaDetalle	Detalle detalle
public	void	cambiarEstadoDieta	int idDieta, int estado
public	Detalle	getDetalleById	Int idDietaDetalle

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		DietaDataService	dietaDataService
private		ComplementoDataService	complementoDataService

Observaciones

7.4.2.2.3 TipoDietaManager

Nombre	Tipo	Implementa
TipoDietaManager		

Descripción			
Se encarga de gestionar las acciones relacionadas con los tipos de dieta, llamando a unos métodos u otros para cada operación.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<TipoDieta>	getTodosTiposDieta	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		TipoDietaDataService	tipoDietaDataService
Observaciones			

7.4.2.2.4 EstadosDietaUtil

Nombre	Tipo	Implementa
EstadosDietaUtil		

Descripción
Clase de utilidad de la que se sirven otras clases y se encarga de rellenar una colección con el nombre de los estados de las dietas.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public static	List<String>	rellenarEstados	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.2.5 FechaUtil

Nombre	Tipo	Implementa
FechaUtil		

Descripción			
Clase de utilidad de la que se sirven otras clases para resolver operaciones comunes relacionadas con la gestión de fechas.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public static	List<DateTime>	daysBetween	Date inicio, Date fin
public static	boolean	isFinDeSemana	DateTime dia
public static	String	getFirstDayYear	
public static	String	getLastDayYear	
public static	String	conversorFecha	Date date
public static	boolean	isInicioAnteriorFin	String inicio, String fin
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Observaciones			

7.4.2.2.6 FileUtil

Nombre	Tipo	Implementa
FileUtil		

Descripción
Clase de utilidad de la que se sirven otras clases para gestionar el acceso a los ficheros que maneja la aplicación: imágenes y documentos PDF.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public static	void	saveFile	File imagen, String nombre
public static	void	deleteFile	String nombre
public static	void	deleteInforme	String nombre
public static	byte[]	dameImagen	String nombre

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static	String	absoluteFilePath
private	static	String	pathInformes

Observaciones

7.4.2.2.7 InformeDietaUtil

Nombre	Tipo	Implementa
InformeDietaUtil		

Descripción
Clase de utilidad que se encarga por completo de la generación del documento PDF de dietas.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	generarInforme	List<Dieta> dietas, int idUsuario, String fecha

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private	static final	String	absoluteFilePath
private		TipoDietaDataService	tipoDietaDataService
private		UsuarioDataService	usuarioDataService

Observaciones

En su interior alberga la clase estática HeaderFooter:

Nombre	Tipo	Hereda de
HeaderFooter	static	PdfPageEventHelper

Descripción
Clase interna de InformeDietaUtil que se encarga de generar los encabezados y pies de página de los documentos.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public		HeaderFooter	String nombreUsuario, String fecha
public	void	onEndPage	PdfWriter writer, Document document

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		String	nombreUsuario
private		String	fecha

Observaciones

7.4.2.2.8 ComplementoDAO

Nombre	Tipo	Implementa
ComplementoDAO		ComplementoDataService

Descripción

Implementa los métodos dictados por su interfaz que se encargan del acceso a base de datos de todo lo relacionado con los complementos no salariales de dietas.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
private	void	abrirConexion	
private	void	cerrarConexion	
public	void	nuevoConvenio	Complemento compl
public	List<Complemento>	getComplementos	
public	void	eliminarConvenio	int idComplemento
public	Complemento	getComplementoById	int idComplemento
public	void	editarConvenio	Complemento compl
public	List<Complemento>	getComplByTipoDieta	int idTipoDieta, String fecha
private	TipoDieta	getTipoDieta	int idTipoDieta

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Connection	con
private		Prepared Statement	ps
private		ResultSet	rs

Observaciones

7.4.2.2.9 DietaDAO

Nombre	Tipo	Implementa
DietaDAO		DietaDataService

Descripción			
Implementa los métodos dictados por su interfaz que se encargan del acceso a base de datos de todo lo relacionado con las dietas y los detalles de dietas.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
private	void	abrirConexion	
private	void	cerrarConexion	
public	List<Dieta>	getTodasDietas	int idUsuario, String fecha
public	int	insertarDetalle	Detalle detalle, int idUsuario, String fecha
public	List<Dieta>	getDietasUsuarioDia	int idUsuario, String fecha
public	void	eliminarDietaDetalle	int idDietaDetalle
public	void	modificarDietaDetalle	Detalle detalle
public	void	cambiarEstadoDieta	int idDieta, int estado
public	Detalle	getDetalleById	int idDietaDetalle
private	List<Detalle>	getDetallesDieta	int idDieta
private	TipoDieta	getTipoDieta	int idTipoDieta
private	double	calcularConvenio	Detalle detalle
private	int	getIDDietaDiaUsuario	String dia, int idUsuario
private	int	nuevoDiaDieta	String dia, int idUsuario
private	void	actualizarValoresDieta	int idDieta
private	boolean	dietaVacia	int idDieta
private	void	eliminarDieta	int idDieta

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Connection	con
private		Prepared Statement	ps
private		ResultSet	rs
Observaciones			

7.4.2.2.10 TipoDietaDAO

Nombre	Tipo	Implementa
TipoDietaDAO		TipoDietaDataService

Descripción
Implementa los métodos dictados por su interfaz que se encargan del acceso a base de datos de todo lo relacionado con los tipos de dietas.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
private	void	abrirConexion	
private	void	cerrarConexion	
public	List<TipoDieta>	getTodosTiposDieta	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Connection	con
private		Prepared Statement	ps
private		ResultSet	rs

Observaciones

7.4.2.2.11 CrearConvenioAction

Nombre	Tipo	Implementa	Hereda de
CrearConvenioAction		SessionAware, ModelDriven<Complemento>	ActionSupport

Descripción
Action que se encarga de recoger los datos introducidos por el usuario para crear un nuevo complemento no salarial, y llama a la siguiente capa de la arquitectura para realizarlo.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	
public	void	validate	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		Complemento	complemento
private		ComplementoManagerService	complManagerService

Observaciones

7.4.2.2.12 EditarConvenioAction

Nombre	Tipo	Implementa	Hereda de
EditarConvenioAction		SessionAware, ModelDriven<Complemento>	ActionSupport

Descripción

Action que se encarga de recoger los datos introducidos por el usuario para editar un complemento no salarial, y llama a la siguiente capa de la arquitectura para realizarlo.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	
public	void	validate	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		Complemento	complemento
private		ComplementoManagerService	complManagerService

Observaciones

7.4.2.2.13 EliminarConvenioAction

Nombre	Tipo	Implementa	Hereda de
EliminarConvenioAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger la petición del usuario para eliminar un complemento no salarial, y llama a la siguiente capa de la arquitectura para realizarlo.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		int	idConvenio
private		ComplementoManagerService	complManagerService
Observaciones			

7.4.2.2.14 InterfazAdminAction

Nombre	Tipo	Implementa	Hereda de
InterfazAdminAction		SessionAware, ApplicationAware	ActionSupport

Descripción
Action que se encarga de preparar la interfaz en la que se mostrarán los complementos no salariales del sistema.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		Map<String, Object>	application
private		List<Complemento>	complementos
private		ComplementoManagerService	complManagerService
private		TipoDietaManagerService	tipoDietaManagerService

Observaciones

7.4.2.2.15 NuevoConvenioAction

Nombre	Tipo	Implementa	Hereda de
NuevoConvenioAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de preparar la interfaz de inserción y edición de los complementos no salariales.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		String	descripcion
private		int	idConvenio
private		double	valor
private		int	idTipoDieta
private		String	fechaInicio
private		String	fechaFin
private		List<TipoDieta>	tiposDieta
private		ComplementoManagerService	complManagerService
private		TipoDietaManagerService	tipoDietaManagerService
Observaciones			

7.4.2.2.16 EliminarDietaDetalleAction

Nombre	Tipo	Implementa	Hereda de
EliminarDietaDetalleAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger la orden de usuario de eliminar un detalle de una dieta y llama a la siguiente capa de la arquitectura para realizarlo.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		int	idDietaDetalle
private		DietaManagerService	dietaManagerService
Observaciones			

7.4.2.2.17 FlujoDietaAction

Nombre	Tipo	Implementa	Hereda de
FlujoDietaAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger la orden de usuario de validar, aprobar o rechazar una dieta y llama a la siguiente capa de la arquitectura para realizarlo.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	validar	
public	String	rechazar	
public	String	aprobar	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		int	idDieta
private		DietaManagerService	dietaManagerService
Observaciones			

7.4.2.2.18 GenerarInformeAction

Nombre	Tipo	Implementa	Hereda de
GenerarInformeAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de recoger la orden de usuario de crear el informe PDF de dietas y de descargarlo al equipo del usuario.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	String	descargar	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		int	selectUsuario
private		String	fecha_filtro
private		String	filename
private		FileInputStream	fileInputStream
private		DietaManagerService	dietaManagerService
Observaciones			

7.4.2.2.19 InsertarDietaAction

Nombre	Tipo	Implementa	Hereda de
InsertarDietaAction		SessionAware, ModelDriven<Detalle>	ActionSupport

Descripción
Action que se encarga de recoger los datos introducidos por el usuario para crear un detalle de dieta, y llama a la siguiente capa de la arquitectura para realizarlo.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	
public	void	validate	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		Detalle	detalle
private		String	fecha
private		String	contentType
private		String	filename
private		List<Complemento>	complementos
private		List<Dietas>	dietas
private		DietaManagerService	dietaManagerService
private		ComplementoManagerService	complManagerService

Observaciones

7.4.2.2.20 PrepararInterfazInsertarAction

Nombre	Tipo	Implementa	Hereda de
PrepararInterfazInsertarAction		SessionAware, ApplicationAware	ActionSupport

Descripción
Action que se encarga de preparar la interfaz de inserción de detalles de dieta.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		Map<String, Object>	application
private		int	idDietaDetalle
private		List<Complemento>	complementos
private		List<Integer>	anios
private		Map<Integer, String>	meses
private		TipoDietaManagerService	tipoDietaManagerService
private		ComplementoManagerService	complManagerService

Observaciones

7.4.2.2.21 RecargarConveniosAction

Nombre	Tipo	Implementa	Hereda de
RecargarConveniosAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de actualizar los complementos no salariales disponibles para la fecha y el tipo de dieta indicado por el usuario.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		int	idTipo
private		String	fecha
private		ComplementoManagerService	complManagerService
Observaciones			

7.4.2.2.22 CustomImageBytesResult

Nombre	Tipo	Implementa	Hereda de
CustomImageBytesResult		Result	
Descripción			
Clase que especifica el tipo de retorno específico para imágenes tras la ejecución del Action que las solicita.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	execute	ActionInvocation invocation
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Observaciones			

7.4.2.2.23 ImagenAction

Nombre	Tipo	Implementa	Hereda de
ImagenAction		SessionAware, ParameterAware, ServletRequestAware	ActionSupport

Descripción
Action que se encarga de devolver el archivo de imagen asociado al detalle de la dieta.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	byte[]	getCustomImageInByte	
public	String	cargarImagen	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		Map<String, String[]>	parameters
private		HttpServletRequest	request
private		Image	image
private		String	filename

Observaciones

7.4.2.2.24 FiltrarDietasAction

Nombre	Tipo	Implementa	Hereda de
FiltrarDietasAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de realizar una búsqueda y filtrado de dietas en función de los datos introducidos por el usuario.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
public	String	mesActual	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		String	fecha_filtro
private		int	mes
private		int	anio
private		int	selectUsuario
private		List<Dietas>	dietas
private		List<Usuarios>	usuarios
private		double	totalImporte
private		double	totalHoras
private		UsuarioManagerService	usuarioManagerService
private		DietaManagerService	dietaManagerService
Observaciones			

7.4.2.2.25 ModificarDietaDetalleAction

Nombre	Tipo	Implementa	Hereda de
ModificarDietaDetalleAction		SessionAware, ModelDricen<Detalle>	ActionSupport

Descripción
Action que se encarga de recoger los datos introducidos por el usuario para editar un detalle de dieta, y llama a la siguiente capa de la arquitectura para realizarlo.

Métodos

Acceso | Modo	Tipo de Retorno	Nombre	Parámetros y tipos
public	String	execute	
public	String	validate	

Atributos

Acceso	Modo	Tipo o Clase	Nombre
private		Map<String, Object>	session
private		Detalle	detalle
private		File	imagen
private		String	contentType
private		String	filename
private		DietaManagerService	dietaManagerService

Observaciones

7.4.2.2.26 PrepararModificarAction

Nombre	**Tipo**	**Implementa**	**Hereda de**
PrepararModificarAction		SessionAware	ActionSupport
Descripción			
Action que se encarga de preparar la interfaz de edición de detalles de dieta.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	String	execute	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		Map<String, Object>	session
private		int	idDietaDetalle
private		String	fecha
private		Detalle	detalle
private		DietaManagerService	dietaManagerService
private		ComplementoManagerService	complManagerService
Observaciones			

7.4.2.2.27 ComplementoManagerService

Nombre	Tipo	Implementa
ComplementoManager	interface	

Descripción
Interfaz que dicta las signaturas de los métodos que ComplementoManager debe implementar.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	nuevoConvenio	Complemento compl
public	List<Complemento>	getComplementos	
public	void	eliminarConvenio	int idComplemento
public	Complemento	getComplementoById	int idComplemento
public	void	editarConvenio	Complemento compl
public	List<Complemento>	getComplByTipoDieta	Int idTipoDieta, String fecha

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.2.28 DietaManagerService

Nombre	Tipo	Implementa
DietaManagerService		

Descripción
Interfaz que dicta las signaturas de los métodos que DietaManager debe implementar.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<Dieta>	getTodasDietas	int idUsuario, String fecha
public	int	insertarDetalle	Detalle detalle, int idUsuario, String fecha
public	List<Dieta>	getDietasUsuarioDia	int idUsuario, String fecha
public	void	eliminarDietaDetalle	int idDietaDetalle
public	void	modificarDietaDetalle	Detalle detalle
public	void	cambiarEstadoDieta	int idDieta, int estado
public	Detalle	getDetalleById	Int idDietaDetalle

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.2.29 TipoDietaManager

Nombre	Tipo	Implementa
TipoDietaManagerService	interface	

Descripción			
Interfaz que dicta las signaturas de los métodos que TipoDietaManager debe implementar.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<TipoDieta>	getTodosTiposDieta	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Observaciones			

7.4.2.2.30 Complemento

Nombre	**Tipo**	**Implementa**
Complemento		Serializable

Descripción			
Clase del modelo que representa al objeto complemento no salarial.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public		Complemento	
public	boolean	isVigente	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		int	idComplemento
private		int	idTipoDieta
private		String	descripcion
private		double	valor
private		Date	fechaInicio
private		Date	fechaFin
private		boolean	eliminado
private		TipoDieta	tipoDieta

Observaciones
El método isVigente no retorna ningún atributo booleano “vigente”, si no que se encarga de verificar si la fecha actual está comprendida entre la fecha de inicio y la fecha de finalización de vigencia del complemento.

7.4.2.2.31 Detalle

Nombre	Tipo	Implementa
Detalle		Serializable

Descripción
Clase del modelo que representa al objeto detalle de dieta.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public		Detalle	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		int	idDietaDetalle
private		int	idDieta
private		int	idTipoDieta
private		int	idComplemento
private		String	albarán
private		double	valor
private		double	valorConvenio
private		String	descripción
private		String	observaciones
private		String	nombreImagen
private		TipoDieta	tipoDieta
private		Complemento	complemento

Observaciones

7.4.2.2.32 Dieta

Nombre	**Tipo**	**Implementa**
Dieta		Serializable

Descripción			
Clase del modelo que representa al objeto dieta.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public		Dieta	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		int	idDieta
private		int	idUsuario
private		int	estado
private		Date	fecha
private		int	totalKM
private		double	totalImporte
private		double	totalHoras
private		List<Detalle>	detalles
Observaciones			

7.4.2.2.33 TipoDieta

Nombre	Tipo	Implementa
TipoDieta		Serializable

Descripción			
Clase del modelo que representa al objeto tipo de dieta.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public		TipoDieta	
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
private		int	idTipoDieta
private		String	descripcion
private		String	unidades
Observaciones			

7.4.2.2.34 ComplementoDataService

Nombre	Tipo	Implementa
ComplementoDataService		

Descripción
Interfaz que dicta las signaturas de los métodos a implementar por ComplementoDAO.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	void	nuevoConvenio	Complemento compl
public	List<Complemento>	getComplementos	
public	void	eliminarConvenio	int idComplemento
public	Complemento	getComplementoById	int idComplemento
public	void	editarConvenio	Complemento compl
public	List<Complemento>	getComplByTipoDieta	int idTipoDieta, String fecha

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

7.4.2.2.35 DietaDataService

Nombre	Tipo	Implementa
DietaDataService		

Descripción			
Interfaz que dicta las signaturas de los métodos a implementar por DietaDAO.			
Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<Dieta>	getTodasDietas	int idUsuario, String fecha
public	int	insertarDetalle	Detalle detalle, int idUsuario, String fecha
public	List<Dieta>	getDietasUsuarioDia	int idUsuario, String fecha
public	void	eliminarDietaDetalle	int idDietaDetalle
public	void	modificarDietaDetalle	Detalle detalle
public	void	cambiarEstadoDieta	int idDieta, int estado
public	Detalle	getDetalleById	int idDietaDetalle
Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**
Observaciones			

7.4.2.2.36 TipoDietaDataService

Nombre	Tipo	Implementa
TipoDietaDataService		

Descripción
Interfaz que dicta las signaturas de los métodos a implementar por TipoDietaDAO.

Métodos			
Acceso | Modo	**Tipo de Retorno**	**Nombre**	**Parámetros y tipos**
public	List<TipoDieta>	getTodosTiposDieta	

Atributos			
Acceso	**Modo**	**Tipo o Clase**	**Nombre**

Observaciones

Capítulo 8. Desarrollo de las Pruebas

8.1 Pruebas Unitarias

Una serie de pruebas se han realizado sobre un conjunto de clases para comprobar su correcto funcionamiento. Se han priorizado las pruebas que trabajan sobre las bases de datos al realizar inserciones, consultas, eliminaciones o actualizaciones de las tablas y los datos, así como los cálculos internos para conservar la integridad del sistema.

8.1.1 Insertar permiso

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Permiso con todos sus datos especificados.
- **Salida:** El permiso aparece en base de datos con los datos tal y como deben.
- **Resultado:** Correcto.

8.1.2 Obtener permiso por identificador

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del permiso a recuperar.
- **Salida:** Se recupera de base de datos un objeto permiso con los datos tal y como aparecen en ella.
- **Resultado:** Correcto.

8.1.3 Obtener permisos según criterios

- **Entrada:** Se invoca al método pasándole como parámetro los criterios deseados (identificador de usuario, fecha…).
- **Salida:** Se recupera de base de datos una colección de objetos permisos con sus datos correctamente cargados y que responen a los criterios dictados.
- **Resultado:** Correcto.

8.1.4 Modificar permiso

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Permiso con todos sus datos especificados y cuyo identificador ya hace referencia a un permiso de la base de datos.

- **Salida:** Los nuevos datos indicados para el permiso aparecen en base de datos, sobreescribiendo los antiguos.
- **Resultado:** Correcto.

8.1.5 Eliminar permiso

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del permiso a eliminar.
- **Salida:** El permiso ya no existe en la base de datos.
- **Resultado:** Correcto.

8.1.6 Autorizar permiso

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del permiso a autorizar.
- **Salida:** El estado del permiso pasa a ser “autorizado”.
- **Resultado:** Correcto.

8.1.7 Denegar permiso

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del permiso a denegar.
- **Salida:** El estado del permiso pasa a ser “denegado”.
- **Resultado:** Correcto.

8.1.8 Insertar tipo de permiso

- **Entrada:** Se invoca al método pasándole como parámetro un objeto TipoPermiso con todos sus datos especificados.
- **Salida:** El tipo de permiso aparece en base de datos con los datos tal y como deben.
- **Resultado:** Correcto.

8.1.9 Obtener tipo de permiso por identificador

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del tipo de permiso a recuperar.

- **Salida:** Se recupera de base de datos un objeto tipo permiso con los datos tal y como aparecen en ella.
- **Resultado:** Correcto.

8.1.10 Obtener todos los tipos de permiso

- **Entrada:** Se invoca al método, no es necesario especificar parámetros.
- **Salida:** Se recupera de base de datos una colección de objetos tipo de permiso con sus datos correctamente cargados.
- **Resultado:** Correcto.

8.1.11 Modificar tipo de permiso

- **Entrada:** Se invoca al método pasándole como parámetro un objeto TipoPermiso con todos sus datos especificados y cuyo identificador ya hace referencia a un tipo de permiso de la base de datos.
- **Salida:** Los nuevos datos indicados para el tipo de permiso aparecen en base de datos, sobreescribiendo los antiguos.
- **Resultado:** Correcto.

8.1.12 Eliminar tipo de permiso

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del tipo de permiso a eliminar.
- **Salida:** El bit “eliminado” del tipo de permiso pasa a ser “1”.
- **Resultado:** Correcto.

8.1.13 Obtener bolsa de días de vacaciones de usuario

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador de usuario del que se quiere recuperar la bolsa.
- **Salida:** Se recupera de base de datos el objeto Bolsa correspondiente al usuario del que se solicita.
- **Resultado:** Correcto.

8.1.14 Actualizar bolsa de días de vacaciones

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador de usuario al que se le pretende actualizar la bolsa, así como el nuevo valor de la misma.
- **Salida:** El número de días de la bolsa de vacaciones del usuario pasa a ser el indicado, sobreescribiendo el anterior.
- **Resultado:** Correcto.

8.1.15 Sumar días en bolsa

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador de usuario al que se le pretende sumar días en la bolsa, así como el número de días a sumar.
- **Salida:** El número de días de la bolsa de vacaciones del usuario se incrementa en la cantidad dada.
- **Resultado:** Correcto.

8.1.16 Restar días en bolsa

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador de usuario al que se le pretende restar días en la bolsa, así como el número de días a restar.
- **Salida:** El número de días de la bolsa de vacaciones del usuario se decrementa en la cantidad dada.
- **Resultado:** Correcto.

8.1.17 Insertar detalle en dieta nueva

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Detalle con todos sus datos especificados, así como el identificador del usuario al que hace referencia y la fecha de la dieta.
- **Salida:** Una nueva dieta aparece en base de datos con los datos indicados. Un nuevo detalle de dieta, asociado a dicha dieta, se introduce en base de datos con la información adecuada. Se calcula el importe según el complemento no salarial aplicado y se actualizan los totales de su dieta asociada.
- **Resultado:** Correcto.

8.1.18 Insertar detalle en dieta ya existente

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Detalle con todos sus datos especificados, así como el identificador del usuario al que hace referencia y la fecha de la dieta.
- **Salida:** n nuevo detalle de dieta, asociado a la dieta a la que pertenece, se introduce en base de datos con la información especificada. Se calcula el importe según el complemento no salarial aplicado y se actualizan los totales de su dieta asociada.
- **Resultado:** Correcto.

8.1.19 Obtener dieta por identificador

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador de la dieta a recuperar.
- **Salida:** Se recupera un objeto Dieta, así como sus objetos Detalle asociados, correspondiente al identificador especificado.
- **Resultado:** Correcto.

8.1.20 Obtener dietas según criterios

- **Entrada:** Se invoca al método pasándole como parámetro los criterios deseados (identificador de usuario, fecha, estado…).
- **Salida:** Se recupera una colección de objetos Dieta, así como sus respectivos objectos Detalle, respondiendo a los criterios indicados.
- **Resultado:** Correcto.

8.1.21 Modificar detalle

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Detalle con todos sus datos especificados y cuyo identificador ya hace referencia a un detalle de la base de datos.
- **Salida:** Los nuevos datos indicados para el detalle aparecen en base de datos, sobreescribiendo los antiguos. Se calcula el importe según el complemento no salarial aplicado y se actualizan los totales de su dieta asociada.
- **Resultado:** Correcto.

8.1.22 Eliminar último detalle

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del detalle a eliminar.
- **Salida:** Desaparece de la base de datos el detalle indicado, así como su dieta asociada.
- **Resultado:** Correcto.

8.1.23 Eliminar detalle (no último)

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del detalle a eliminar.
- **Salida:** Desaparece de la base de datos el detalle indicado. Se recalculan los totales de su dieta asociada.
- **Resultado:** Correcto.

8.1.24 Cambiar estado de dieta

- **Entrada:** Se invoca al método pasándole como parámetros el identificador de la dieta a la que se desee cambiar el estado, así como el nuevo estado que se le quiere otorgar.
- **Salida:** La dieta refleja en base de datos el nuevo estado indicado.
- **Resultado:** Correcto.

8.1.25 Obtener tipo de dieta por identificador

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del tipo de dieta a recuperar.
- **Salida:** Se recupera un objeto tipo de dieta de la base de datos correspondiente al identificador especificado.
- **Resultado:** Correcto.

8.1.26 Obtener todos los tipos de dieta

- **Entrada:** Se invoca al método, no es necesario ningún atributo.
- **Salida:** Se recupera una colección de objetos tipo de dieta de la base de datos con sus respectivos valores.
- **Resultado:** Correcto.

8.1.27 Insertar complemento no salarial

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Complemento con todos sus datos especificados.
- **Salida:** La base de datos contiene un nuevo objeto complemento no salarial con los datos indicados.
- **Resultado:** Correcto.

8.1.28 Obtener complemento no salarial por identificador

- **Entrada:** Se invoca al método pasándole como parámetro un número entero que represente al identificador del complemento no salarial a recuperar.
- **Salida:** Se recupera un objeto complemento no salarial con sus valores de la base de datos correspondiente al identificador especificado.
- **Resultado:** Correcto.

8.1.29 Obtener todos los complementos no salariales

- **Entrada:** Se invoca al método, no es necesario ningún atributo.
- **Salida:** Se recupera una colección de objetos complemento no salarial de la base de datos, con sus respectivos valores.
- **Resultado:** Correcto.

8.1.30 Modificar complemento no salarial

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Complemento con todos sus datos especificados y cuyo identificador ya hace referencia a un complemento no salarial de la base de datos.
- **Salida:** Los nuevos datos indicados para el complemento no salarial aparecen reflejados en la base de datos, sobreescribiendo los anteriores.
- **Resultado:** Correcto.

8.1.31 Eliminar complemento no salarial

- **Entrada:** Se invoca al método pasándole como parámetro un objeto Complemento con todos sus datos especificados y cuyo identificador ya hace referencia a un complemento no salarial de la base de datos.

- **Salida:** El bit “eliminado” del complemento no salarial pasa a ser “1”.
- **Resultado:** Correcto.

8.2 Pruebas de Integración y del Sistema

Una vez integrado el conjunto de clases y ficheros necesarios para ejecutar la aplicación al completo, nos disponemos a ejecutar las pruebas especificadas en el apartado 5.7 de esta documentación.

En esta ocasión, además de los resultados de la base de datos, que ya se especificó en las pruebas unitarias anteriores, también observaremos y vigilaremos los resultados tras todas las operaciones en la interfaz de usuario que es, en definitiva, el componente que debe reflejar los cambios de cara a que el usuario pueda trabajar adecuadamente.

8.2.1.1 Gestión del calendario de vacaciones

8.2.1.1.1 Solicitar permiso

Prueba	Resultado esperado
Crear permiso que no requiere aprobación	El sistema registra la solicitud de permiso y lo establece como ya aprobado.
	Resultado obtenido
	La base de datos contiene el nuevo permiso adecuadamente y su estado es “aprobado”. La interfaz muestra el nuevo permiso para su interacción.
Prueba	**Resultado esperado**
Crear permiso que requiere aprobación	El sistema registra la solicitud de permiso y lo establece como aún no aprobado.
	Resultado obtenido
	La base de datos contiene el nuevo permiso adecuadamente y su estado es “solicitado”. La interfaz muestra el nuevo permiso para su interacción.
Prueba	**Resultado esperado**
Crear permiso tipo laborable en días laborables	El sistema registra la solicitud de permiso correctamente.
	Resultado obtenido
	La base de datos contiene el nuevo permiso adecuadamente. La interfaz muestra el nuevo permiso para su interacción.
Prueba	**Resultado esperado**
Crear permiso tipo laborable en fin de semana	El sistema no registra la solicitud de permiso y notifica al usuario del error cometido.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.
Prueba	**Resultado esperado**
Solicitar vacaciones con suficiente	El sistema registra la solicitud de permiso y la marca

bolsa	como pendiente de aprobación.
	Resultado obtenido
	El permiso aparece en la base de datos con los valores correspondientes.
Prueba	**Resultado esperado**
Solicitar vacaciones sin suficiente bolsa	El sistema no registra la solicitud de permiso, no resta de la bolsa y notifica al usuario del error.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.
Prueba	**Resultado esperado**
Solicitar permiso con límite anual ya superado	El sistema no registra la solicitud de permiso y notifica al usuario del error.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.
Prueba	**Resultado esperado**
Solicitar permiso sin límite anual ya superado	El sistema registra la solicitud de permiso correctamente.
	Resultado obtenido
	El permiso aparece en la base de datos con los valores correspondientes.
Prueba	**Resultado esperado**
Solicitar permiso por más días del máximo permitido	El sistema no registra la solicitud de permiso y notifica al usuario del motivo.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.
Prueba	**Resultado esperado**
Solicitar permiso con alguno de los campos vacíos	El sistema no registra la solicitud de permiso y notifica al usuario del error cometido.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.
Prueba	**Resultado esperado**
Solicitar permiso con fecha de inicio posterior a fecha final	El sistema no registra la solicitud de permiso y notifica al usuario del error cometido.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.
Prueba	**Resultado esperado**
Solicitar permiso para una fecha ya asignada a otro permiso	El sistema no registra la solicitud de permiso y notifica al usuario del motivo.
	Resultado obtenido

	La base de datos no registra cambios. La interfaz muestra el motivo.

8.2.1.1.2 Eliminar permiso

Prueba	Resultado esperado
Eliminar permiso aún no concluido	El permiso se elimina del sistema correctamente.
	Resultado obtenido
	Aparece un diálogo pidiendo confirmación. Tras aceptar, el permiso desaparece de la base de datos e interfaz.
Prueba	**Resultado esperado**
Eliminar permiso o vacación ya concluido	La operación no está disponible para su ejecución.
	Resultado obtenido
	No existe el botón correspondiente para ejecutar la operación.
Prueba	**Resultado esperado**
Eliminar vacación no aprobada	El permiso se elimina del sistema correctamente.
	Resultado obtenido
	Aparece un diálogo pidiendo confirmación. Tras aceptar, el permiso de vacaciones desaparece de la base de datos e interfaz.
Prueba	**Resultado esperado**
Eliminar vacación ya aprobada	El permiso se elimina del sistema y la bolsa de vacaciones se incrementa adecuadamente.
	Resultado obtenido
	Aparece un diálogo pidiendo confirmación. Tras aceptar, el permiso desaparece de la base de datos e interfaz, y la bolsa de días del usuario se actualiza.

8.2.1.1.3 Modificar permiso

Prueba	Resultado esperado
Modificar permiso o vacación ya concluido	La operación no está disponible para su ejecución.
	Resultado obtenido
	No existe el botón correspondiente para ejecutar la operación.
Prueba	**Resultado esperado**
Modificar permiso	El permiso se modifica correctamente.
	Resultado obtenido
	La base de datos e interfaz reflejan los nuevos valores indicados para el permiso.
Prueba	**Resultado esperado**

Modificar vacación (indep. de su estado)	El permiso se modifica, pasa a estado en espera de aprobación y la bolsa se incrementa adecuadamente.
	Resultado obtenido
	La base de datos e interfaz reflejan los nuevos valores indicados para el permiso de vacaciones. Además, su estado es "solicitado" y la bolsa de días del usuario dispone de más cantidad.

8.2.1.1.4 Visualizar calendario

Prueba	**Resultado esperado**
Seleccionar mes y año con datos	El calendario se recarga para mostrar todos los permisos del usuario solicitante para la fecha indicada.
	Resultado obtenido
	La base de datos permanece sin cambios. La interfaz se actualiza para mostrar el calendario correspondiente.
Prueba	**Resultado esperado**
Seleccionar mes y año sin datos	El calendario se recarga pero se muestra vacío. Se indica al usuario por qué.
	Resultado obtenido
	La base de datos permanece sin cambios. La interfaz se actualiza para mostrar el calendario vacío, e indica por qué.

8.2.1.1.5 Insertar tipo de permiso

Prueba	**Resultado esperado**
Insertar tipo de permiso	El sistema inserta el tipo de permiso correctamente.
	Resultado obtenido
	La base de datos contiene el nuevo tipo de permiso. La interfaz se recarga para mostrarlo y pasa a ser seleccionable durante la inserción de permiso.
Prueba	**Resultado esperado**
Insertar tipo de permiso con algún campo incorrecto	El sistema no registra el tipo de permiso y avisa al usuario del error cometido.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.

8.2.1.1.6 Eliminar tipo de permiso

Prueba	**Resultado esperado**
Eliminar tipo de permiso	El sistema actualiza el permiso y lo establece como "eliminado" para evitar su utilización en el futuro, pero no lo borra.
	Resultado obtenido

	La base de datos refleja el tipo de permiso con su bit "eliminado" a "1". La interfaz deja de mostrarlo y a la hora de insertar permiso ya no es seleccionable.

8.2.1.1.7 Modificar tipo de permiso

Prueba	Resultado esperado
Modificar tipo de permiso	El sistema modifica el tipo de permiso correctamente.
	Resultado obtenido
Prueba	**Resultado esperado**
Modificar tipo de permiso con algún campo incorrecto	El sistema no registra los cambios sobre el tipo de permiso y avisa al usuario del error cometido.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.

8.2.1.1.8 Aceptar solicitud

Prueba	Resultado esperado
Aceptar solicitud con suficiente bolsa de vacaciones	El sistema cambia el estado de la solicitud a "aceptado" y resta de la bolsa de vacaciones los días correspondientes.
	Resultado obtenido
	La base de datos modifica el estado de la solicitud y actualiza la bolsa de vacaciones en consecuencia. La interfaz muestra los cambios.
Prueba	**Resultado esperado**
Aceptar solicitud sin suficiente bolsa de vacaciones	El sistema no cambia el estado de la solicitud, no modifica la bolsa de vacaciones y notifica al usuario del motivo.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.

8.2.1.1.9 Rechazar solicitud

Prueba	Resultado esperado
Rechazar solicitud	El sistema cambia el estado de la solicitud a "rechazado".
	Resultado obtenido
	La base de datos contiene el permiso con estado "rechazado". La interfaz lo muestra.

8.2.1.2 Gestión de dietas

8.2.1.2.1 Insertar dieta

Prueba	Resultado esperado
Insertar dieta sin complemento no salarial	El sistema registra los datos de la dieta y almacena el fichero asociado en el servidor. Recalcula valores totales de dieta.
	Resultado obtenido
	La base de datos contiene el detalle de dieta y sus datos, así como los nuevos valores de totales de dieta. El servidor tiene la imagen asociada al detalle. La interfaz muestra el nuevo detalle.
Prueba	**Resultado esperado**
Insertar dieta con complemento no salarial	El sistema registra los datos de la dieta y almacena el fichero asociado en el servidor. Calcula valor de dieta según el complemento indicado y recalcula valores totales.
	Resultado obtenido
	La base de datos contiene el detalle de dieta, sus datos y su valor tras calcular el complemento, así como los nuevos valores de totales de dieta. El servidor tiene la imagen asociada al detalle. La interfaz muestra el nuevo detalle.
Prueba	**Resultado esperado**
Insertar dieta con alguno de los campos obligatorios vacío o incorrecto	El sistema no registra los datos de la dieta y avisa al usuario del error cometido.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.

8.2.1.2.2 Modificar dieta

Prueba	Resultado esperado
Modificar dieta especificando nuevo fichero	El sistema registra los cambios de la dieta y almacena el nuevo fichero asociado en el servidor, borrando el anterior. Recalcula valores de dieta.
	Resultado obtenido
	La base de datos contiene los datos nuevos del detalle de dieta y su importe tras calcular el complemento, así como los nuevos valores de totales de dieta. El servidor tiene la nueva imagen asociada al detalle y desaparece la anterior.
Prueba	**Resultado esperado**
Modificar dieta sin especificar	El sistema registra los cambios de la dieta sin modificar

nuevo fichero	el fichero asociado. Recalcula valores de dieta.
	Resultado obtenido
	La base de datos contiene los datos nuevos del detalle de dieta y su importe tras calcular el complemento, así como los nuevos valores de totales de dieta. La imagen asociada sigue existiendo.
Prueba	**Resultado esperado**
Modificar dieta con alguno de los campos obligatorios vacío o incorrecto	El sistema no registra los datos de la dieta y avisa al usuario del error cometido.
	Resultado obtenido
	La base de datos no registra cambios. La interfaz muestra el motivo.
Prueba	**Resultado esperado**
Modificar dieta si ya ha sido autorizada	La operación no está disponible
	Resultado obtenido
	No existe el botón correspondiente para ejecutar la operación.

8.2.1.2.3 Eliminar dieta

Prueba	**Resultado esperado**
Eliminar dieta	El sistema borra la dieta, recalcula los importes totales y elimina el fichero asociado a la misma.
	Resultado obtenido
	La base de datos no contiene el detalle y la dieta asociada se actualiza con los nuevos valores. El fichero asociado desaparece. La interfaz ya no muestra el detalle de dieta.
Prueba	**Resultado esperado**
Eliminar dieta si ya ha sido autorizada	La operación no está disponible.
	Resultado obtenido
	No existe el botón correspondiente para ejecutar la operación.

8.2.1.2.4 Visualizar informe

Prueba	**Resultado esperado**
Visualizar por defecto	El sistema busca y muestra las dietas del usuario del mes en curso.
	Resultado obtenido
	La base de datos permanece sin cambios. La interfaz de usuario se recarga y muestra la información.

Prueba	Resultado esperado
Visualizar especificando mes y año	El sistema busca y muestra las dietas del usuario para el mes y año indicados.
	Resultado obtenido
	La base de datos permanece sin cambios. La interfaz de usuario se recarga y muestra la información.
Prueba	**Resultado esperado**
Visualizar especificando mes, año y usuario (Administrador)	El sistema busca y muestra las dietas del usuario especificado para el mes y año indicado.
	Resultado obtenido
	La base de datos permanece sin cambios. La interfaz de usuario se recarga y muestra la información.

8.2.1.2.5 Generar informe PDF

Prueba	Resultado esperado
Generar informe	El sistema utiliza los criterios especificados en la visualización del informe para generar el documento PDF. Lo almacena en el servidor y se lo sirve al navegador del usuario.
	Resultado obtenido
	La base de datos permanece sin cambios. El sistema contiene el informe en formato PDF, y una copia del mismo se descarga a través del navegador del usuario.

8.2.1.2.6 Aprobar dieta

Prueba	Resultado esperado
Aprobar dieta	El sistema modifica el estado de la dieta y lo establece como “aprobada”.
	Resultado obtenido
	La base de datos registra el cambio de estado de la dieta. La interfaz lo muestra en consecuencia.

8.2.1.2.7 Rechazar dieta

Prueba	Resultado esperado
Rechazar dieta	El sistema modifica el estado de la dieta y lo establece como “rechazada”.
	Resultado obtenido
	La base de datos registra el cambio de estado de la dieta. La interfaz lo muestra en consecuencia.

8.2.1.2.8 Insertar complemento no salarial

Prueba	Resultado esperado

Insertar complemento	El sistema inserta el complemento no salarial y sus datos para su futura utilización.
	Resultado obtenido
	La base de datos contiene el nuevo complemento no salarial. La interfaz se recarga para mostrarlo y pasa a ser seleccionable durante la inserción de dieta.
Prueba	**Resultado esperado**
Insertar complemento con algún campo vacío o incorrecto	El sistema no inserta el complemento no salarial y notifica al usuario del error cometido.
	Resultado obtenido
	El sistema no registra los datos del permiso y avisa al usuario del error cometido.
Prueba	**Resultado esperado**
Insertar complemento con fecha de inicio posterior a finalización	El sistema no inserta el complemento no salarial y notifica al usuario del error cometido.
	Resultado obtenido
	El sistema no registra los datos del permiso y avisa al usuario del error cometido.

8.2.1.2.9 Modificar complemento no salarial

Prueba	**Resultado esperado**
Modificar complemento	El sistema modifica el complemento no salarial y actualiza las dietas asociadas para recalcular sus importes.
	Resultado obtenido
	La base de datos e interfaz reflejan los nuevos valores indicados para el complemento no salarial.

8.2.1.2.10 Eliminar complemento no salarial

Prueba	**Resultado esperado**
Eliminar complemento	El sistema actualiza el complemento no salarial para marcarlo como “Eliminado” y evitar su uso posterior. Se actualizan las dietas asociadas para recalcular sus importes.
	Resultado obtenido
	La base de datos refleja el complemento no salarial con su bit “eliminado” a “1”. La interfaz deja de mostrarlo y a la hora de insertar dieta ya no es seleccionable.

8.2.1.3 Integración en intranet

Prueba	**Resultado esperado**
Otorgar a un perfil permisos de	Los usuarios con dicho perfil podrán realizar tareas de

escritura sobre el módulo	usuario normal sobre el módulo en cuestión.
	Resultado obtenido
	Los usuarios visualizan el módulo y, por tanto, tienen acceso a las tareas que en él pueden realizar.
Prueba	**Resultado esperado**
Revocar el permiso de escritura a un perfil	Los usuarios con dicho perfil ya no podrán realizar tareas en el módulo.
	Resultado obtenido
	Los usuarios ya no visualizan el módulo y, por tanto, no tienen acceso a las tareas que en él pueden realizar.
Prueba	**Resultado esperado**
Otorgar a un usuario permiso de escritura sobre el módulo	El usuario especificado podrá realizar tareas de usuario normal sobre el módulo en cuestión.
	Resultado obtenido
	El usuario visualiza el módulo y, por tanto, tiene acceso a las tareas que en él puede realizar.
Prueba	**Resultado esperado**
Revocar el permiso de escritura a un usuario	El usuario especificado ya no podrá realizar tareas en el módulo, a no ser que su perfil se lo permita.
	Resultado obtenido
	El usuario ya no visualiza el módulo y, por tanto, no tiene acceso a las tareas que en él puede realizar.
Prueba	**Resultado esperado**
Insertar a un usuario en el perfil “Administradores”	El usuario en cuestión podrá realizar tareas de usuario administrador en los módulos.
	Resultado obtenido
	El usuario visualiza nuevas opciones y submenús y, por tanto, tiene acceso a sus operaciones.

8.3 Pruebas de Rendimiento

Dado que la aplicación desarrollada no está pensada para un entorno de alta explotación, no se han previsto problemas derivados del rendimiento. Unos pocos usuarios que generarán escasa concurrencia y datos no supondrán un problema necesario de analizar y controlar. Además, la administración de los servidores y sistemas implicados en el proyecto no está dentro del control del desarrollador, si no que es el cliente quien elige el entorno.

En cuando al rendimiento del código, el desarrollo se ha realizado pensando en mantener un alto nivel de calidad y limpieza a la vez que se intentaba alcanzar la menor complejidad posible. Las conexiones con la base de datos, el aspecto más susceptible de presentar cadencias en las operaciones, son parte de la llamada Intranet de la aplicación y por tanto no es parte de este proyecto. No obstante, las conexiones se realizan a través de un pool de conexiones, por lo que el rendimiento es óptimo.

El único aspecto que quizá merezca una mención es el tamaño de las tablas de la base de datos relacionados con el registro del sistema, las sesiones de los usuarios y el log de sus operaciones, puesto que todo lo que hagan los usuarios queda registrado. Dichas tablas pueden llegar a contener una gran cantidad de información conforme la aplicación siga funcionando a lo largo de los meses. Es responsabilidad del cliente, administrador de dicha base de datos, el control del mencionado aspecto.

8.4 Pruebas de Accesibilidad

Puntos de verificación Prioridad 1:

En general (Prioridad 1)	Sí	No	N/A
1.1 Proporcione un texto equivalente para todo elemento no textual (Por ejemplo, a través de "alt", "longdesc" o en el contenido del elemento). *Esto incluye:* imágenes, representaciones gráficas del texto, mapas de imagen, animaciones (Por ejemplo, *GIFs* animados), "applets" y objetos programados, "ascii art", marcos, scripts, imágenes usadas como viñetas en las listas, espaciadores, botones gráficos, sonidos (ejecutados con o sin interacción del usuario), archivos exclusivamente auditivos, banda sonora del vídeo y vídeos.	**X**		
2.1 Asegúrese de que toda la información transmitida a través de los colores también esté disponible sin color, por ejemplo mediante el contexto o por marcadores.	**X**		
4.1 Identifique claramente los cambios en el idioma del texto del documento y en cualquier texto equivalente (por ejemplo, leyendas).			**X**
6.1 Organice el documento de forma que pueda ser leído sin hoja de estilo. Por ejemplo, cuando un documento HTML es interpretado sin asociarlo a una hoja de estilo, tiene que ser posible leerlo.	**X**		
6.2 Asegúrese de que los equivalentes de un contenido dinámico son actualizados cuando cambia el contenido dinámico.	**X**		
7.1 Hasta que las aplicaciones de usuario permitan controlarlo, evite provocar destellos en la pantalla.	**X**		
14.1 Utilice el lenguaje apropiado más claro y simple para el contenido	**X**		

de un sitio.			
Y si utiliza imágenes y mapas de imagen (Prioridad 1)	**Sí**	**No**	**N/A**
1.2 Proporcione vínculos redundantes en formato texto para cada zona activa de un mapa de imagen del servidor.			**X**
9.1 Proporcione mapas de imagen controlados por el cliente en lugar de por el servidor, excepto donde las zonas sensibles no puedan ser definidas con una forma geométrica.			**X**
Y si utiliza tablas (Prioridad 1)	**Sí**	**No**	**N/A**
5.1 En las tablas de datos, identifique los encabezamientos de fila y columna.	**X**		
5.2 Para las tablas de datos que tienen dos o más niveles lógicos de encabezamientos de fila o columna, utilice marcadores para asociar las celdas de encabezamiento y las celdas de datos.		**X**	
Y si utiliza marcos ("frames") (Prioridad 1)	**Sí**	**No**	**N/A**
12.1 Titule cada marco para facilitar su identificación y navegación.			**X**
Y si utiliza "applets" y "scripts" (Prioridad 1)	**Sí**	**No**	**N/A**
6.3 Asegure que las páginas sigan siendo utilizables cuando se desconecten o no se soporten los scripts, *applets* u otros objetos programados. Si esto no es posible, proporcione información equivalente en una página alternativa accesible.		**X**	
Y si utiliza multimedia (Prioridad 1)	**Sí**	**No**	**N/A**
1.3 Hasta que las aplicaciones de usuario puedan leer en voz alta automáticamente el texto equivalente de la banda visual, proporcione una descripción auditiva de la información importante de la banda visual de una presentación multimedia.			**X**
1.4 Para toda presentación multimedia dependiente del tiempo (por ejemplo, una película o animación) sincronice alternativas equivalentes (por ejemplo, subtítulos o descripciones de la banda visual) con la presentación.			**X**
Y si todo lo demás falla (Prioridad 1)	**Sí**	**No**	**N/A**
11.4 Si, después de los mayores esfuerzos, no puede crear una página accesible, proporcione un vínculo a una página alternativa que use tecnologías *W3C*, sea accesible, tenga información (o funcionalidad) equivalente y sea actualizada tan a menudo como la página (original) inaccesible.		**X**	

Puntos de verificación Prioridad 2:

En general (Prioridad 2)	**Sí**	**No**	**N/A**
2.2 Asegúrese de que las combinaciones de los colores de fondo y primer plano tengan el suficiente contraste para que sean percibidas por personas con deficiencias de percepción de color o en pantallas en blanco y negro [Prioridad 2 para las imágenes. Prioridad 3 para los textos].		**X**	
3.1 Cuando exista un marcador apropiado, use marcadores en vez de imágenes para transmitir la información.		**X**	
3.2 Cree documentos que estén validados por las gramáticas formales publicadas.		**X**	
3.3 Utilice hojas de estilo para controlar la maquetación y la presentación.	**X**		
3.4 Utilice unidades relativas en lugar de absolutas al especificar los valores en los atributos de los marcadores de lenguaje y en los valores		**X**	

de las propiedades de las hojas de estilo.			
3.5 Utilice elementos de encabezado para transmitir la estructura lógica y utilícelos de acuerdo con la especificación.	X		
3.6 Marque correctamente las listas y los ítems de las listas.	X		
3.7 Marque las citas. No utilice el marcador de citas para efectos de formato tales como sangrías.			X
6.5 Asegúrese de que los contenidos dinámicos son accesibles o proporcione una página o presentación alternativa.		X	
7.2 Hasta que las aplicaciones de usuario permitan controlarlo, evite el parpadeo del contenido (por ejemplo, cambio de presentación en periodos regulares, así como el encendido y apagado).	X		
7.4 Hasta que las aplicaciones de usuario proporcionen la posibilidad de detener las actualizaciones, no cree páginas que se actualicen automáticamente de forma periódica.	X		
7.5 Hasta que las aplicaciones de usuario proporcionen la posibilidad de detener el redireccionamiento automático, no utilice marcadores para redirigir las páginas automáticamente. En su lugar, configure el servidor para que ejecute esta posibilidad.	X		
10.1 Hasta que las aplicaciones de usuario permitan desconectar la apertura de nuevas ventanas, no provoque apariciones repentinas de nuevas ventanas y no cambie la ventana actual sin informar al usuario.	X		
11.1 Utilice tecnologías *W3C* cuando estén disponibles y sean apropiadas para la tarea y use las últimas versiones que sean soportadas.	X		
11.2 Evite características desaconsejadas por las tecnologías *W3C*.	X		
12.3 Divida los bloques largos de información en grupos más manejables cuando sea natural y apropiado.	X		
13.1 Identifique claramente el objetivo de cada vínculo.	X		
13.2 Proporcione metadatos para añadir información semántica a las páginas y sitios.		X	
13.3 Proporcione información sobre la maquetación general de un sitio (por ejemplo, mapa del sitio o tabla de contenidos).		X	
13.4 Utilice los mecanismos de navegación de forma coherente.	X		
Y si utiliza tablas (Prioridad 2)	**Sí**	**No**	**N/A**
5.3 No utilice tablas para maquetar, a menos que la tabla tenga sentido cuando se alinee. Por otro lado, si la tabla no tiene sentido, proporcione una alternativa equivalente (la cual debe ser una versión alineada).	X		
5.4 Si se utiliza una tabla para maquetar, no utilice marcadores estructurales para realizar un efecto visual de formato.	X		
Y si utiliza marcos ("frames") (Prioridad 2)	**Sí**	**No**	**N/A**
12.2 Describa el propósito de los marcos y cómo éstos se relacionan entre sí, si no resulta obvio solamente con el título del marco.			X
Y si utiliza formularios (Prioridad 2)	**Sí**	**No**	**N/A**
10.2 Hasta que las aplicaciones de usuario soporten explícitamente la asociación entre control de formulario y etiqueta, para todos los controles de formularios con etiquetas asociadas implícitamente, asegúrese de que la etiqueta está colocada adecuadamente.	X		
12.4 Asocie explícitamente las etiquetas con sus controles.	X		
Y si utiliza "applets" y "scripts" (Prioridad 2)	**Sí**	**No**	**N/A**
6.4 Para los *scripts* y *applets*, asegúrese de que los manejadores de eventos sean independientes del dispositivo de entrada.		X	

7.3 Hasta que las aplicaciones de usuario permitan congelar el movimiento de los contenidos, evite los movimientos en las páginas.	X		
8.1 Haga los elementos de programación, tales como *scripts* y *applets*, directamente accesibles o compatibles con las ayudas técnicas [Prioridad 1 si la funcionalidad es importante y no se presenta en otro lugar; de otra manera, Prioridad 2].		X	
9.2 Asegúrese de que cualquier elemento que tiene su propia interfaz pueda manejarse de forma independiente del dispositivo.	X		
9.3 Para los "scripts", especifique manejadores de evento lógicos mejor que manejadores de eventos dependientes de dispositivos.	X		

Puntos de verificación Prioridad 3:

En general (Prioridad 3)	Sí	No	N/A
4.2 Especifique la expansión de cada abreviatura o acrónimo cuando aparezcan por primera vez en el documento.			X
4.3 Identifique el idioma principal de un documento.	X		
9.4 Cree un orden lógico para navegar con el tabulador a través de vínculos, controles de formulario y objetos.	X		
9.5 Proporcione atajos de teclado para los vínculos más importantes (incluidos los de los mapas de imagen de cliente), los controles de formulario y los grupos de controles de formulario.		X	
10.5 Hasta que las aplicaciones de usuario (incluidas las ayudas técnicas) interpreten claramente los vínculos contiguos, incluya caracteres imprimibles (rodeados de espacios), que no sirvan como vínculo, entre los vínculos contiguos.			X
11.3 Proporcione la información de modo que los usuarios puedan recibir los documentos según sus preferencias (por ejemplo, idioma, tipo de contenido, etc.).			X
13.5 Proporcione barras de navegación para destacar y dar acceso al mecanismo de navegación.	X		
13.6 Agrupe los vínculos relacionados, identifique el grupo (para las aplicaciones de usuario) y, hasta que las aplicaciones de usuario lo hagan, proporcione una manera de evitar el grupo.	X		
13.7 Si proporciona funciones de búsqueda, permita diferentes tipos de búsquedas para diversos niveles de habilidad y preferencias.			X
13.8 Localice la información destacada al principio de los encabezamientos, párrafos, listas, etc.			X
13.9 Proporcione información sobre las colecciones de documentos (por ejemplo, los documentos que comprendan múltiples páginas).			X
13.10 Proporcione un medio para saltar sobre un *ASCII art* de varias líneas.			X
14.2 Complemente el texto con presentaciones gráficas o auditivas cuando ello facilite la comprensión de la página.			X
14.3 Cree un estilo de presentación que sea coherente para todas las páginas.	X		
Y si utiliza imágenes o mapas de imagen (Prioridad 3)	**Sí**	**No**	**N/A**
1.5 Hasta que las aplicaciones de usuario interpreten el texto equivalente para los vínculos de los mapas de imagen de cliente, proporcione vínculos de texto redundantes para cada zona activa del mapa de imagen de cliente.			X
Y si utiliza tablas (Prioridad 3)	**Sí**	**No**	**N/A**

5.5 Proporcione resúmenes de las tablas.		**X**	
5.6 Proporcione abreviaturas para las etiquetas de encabezamiento.		**X**	
10.3 Hasta que las aplicaciones de usuario (incluidas las ayudas técnicas) interpreten correctamente los textos contiguos, proporcione un texto lineal alternativo (en la página actual o en alguna otra) para *todas* las tablas que maquetan texto en paralelo, en columnas de palabras.		**X**	
Y si utiliza formularios (Prioridad 3)	**Sí**	**No**	**N/A**
10.4 Hasta que las aplicaciones de usuario manejen correctamente los controles vacíos, incluya caracteres por defecto en los cuadros de edición y áreas de texto.		**X**	

Como puede verse en las anteriores tablas, se ha intentado lograr en la medida de lo posible que las normas dictadas por WCAG se cumplan para poder tener una aplicación lo más accesible posible.

No obstante, debido a las herramientas utilizadas para el desarrollo o a los deseos del cliente sobre el diseño de las interfaces, no todas estas normas han podido ser cumplidas en su totalidad

Capítulo 9. Manuales del Sistema

9.1 Manual de Instalación

9.1.1 Servidor de base de datos

En primer lugar será necesario instalar el servidor de base de datos Microsoft SQL Server 2005. Para ello se necesita disponer de un equipo con Windows Server 2003 o superior y cumplir con los siguientes requisitos mínimos:

- Procesador Pentium III o superior a 600 MHz (recomendado 1 GHz o más);
- 512 MB de memoria RAM (recomendado 1 GB o más).
- Al menos 2 GB de disco duro.

Antes de comenzar con el proceso de instalación se recomienda tener el sistema operativo totalmente actualizado, mediante el uso de Windows Update. Seguidamente ejecutamos el instalador que comenzará el proceso.

Tras aceptar los habituales términos y condiciones de la licencia y pasar por las primeras pantallas de bienvenida, veremos cómo se ejecuta una comprobación de los requisitos hardware y software del sistema. Si todo está en orden podremos pasar al siguiente paso en el que se nos pedirán nuestros datos:

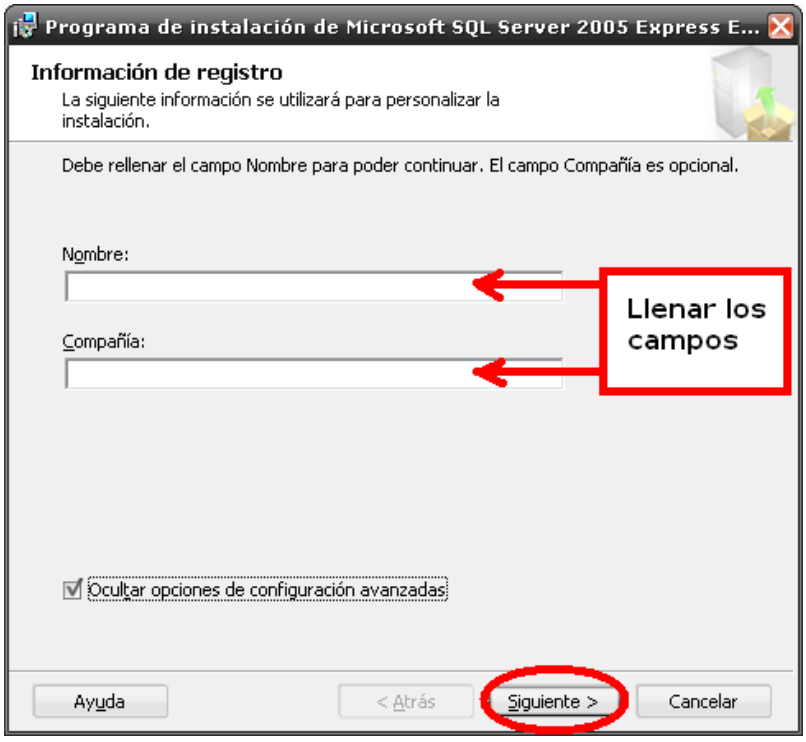


Figura 9.1 - Manual de instalación: datos del usuario

Seguidamente deberemos seleccionar qué componentes queremos instalar. Para evitar posibles problemas se aconseja marcar todo:

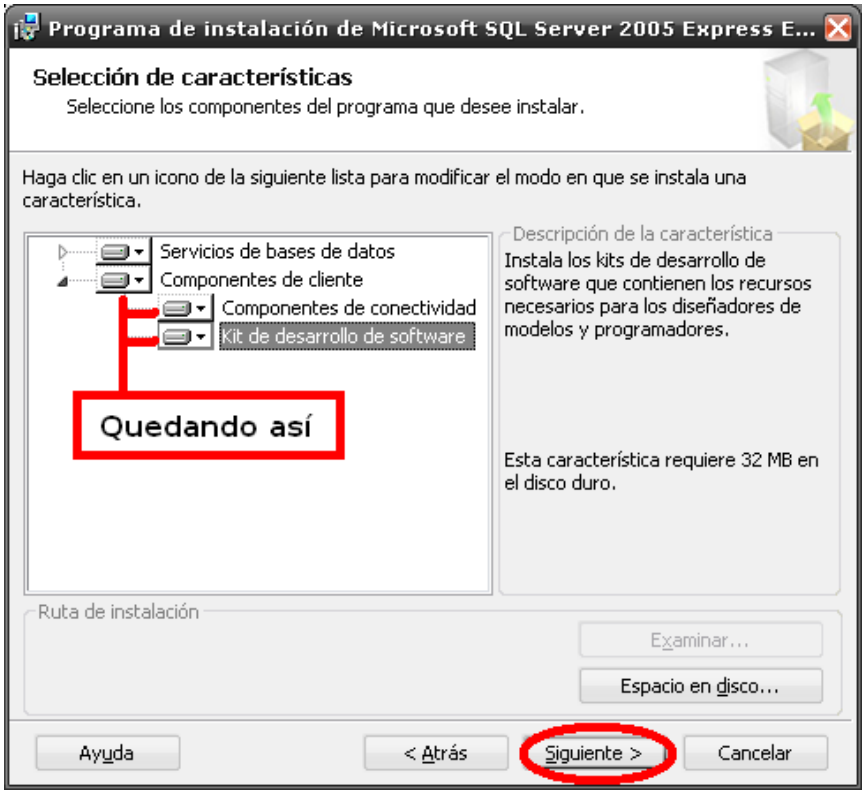


Figura 9.2 - Manual de instalación: componentes a instalar

En el siguiente paso deberemos indicar el nombre de la instancia, que podrá ser el que el usuario desee (en nuestro caso ha sido nombrado PortalEmpleado). Al dar a siguiente deberemos elegir modo de autenticación mixto para que la identificación del usuario no se limite únicamente al entorno Windows y sus usuarios. También deberemos escribir la contraseña para el usuario por defecto “sa”, a nuestra elección (en este caso la contraseña es Portal.Empleado).

Tras comprobar que estamos satisfechos con los parámetros de configuración establecidos, el servidor de base de datos se instalará y avisará cuando el proceso concluya:

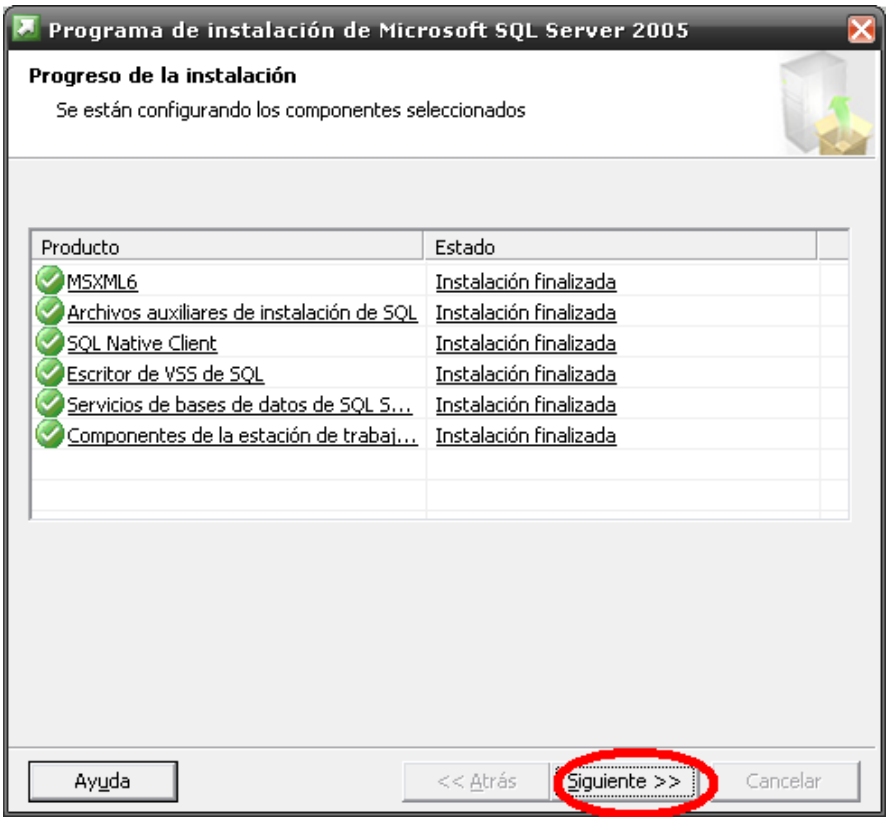


Figura 9.3 - Manual de instalación: proceso finalizado

Para finalizar quizá sea necesario indicar que el cortafuegos del equipo debe estar configurado correctamente para recibir peticiones en el puerto en el que el servidor de base de datos esté escuchando (por defecto, 1433).

9.1.2 Sistema de gestión de base de datos

Una vez con el servidor de base de datos instalado necesitamos el programa que lo gestione de una manera sencilla para el usuario: el SGBD. Por comodidad y gratuidad será Microsoft SQL Server Management Studio Express.

Tras ejecutar el instalador y aceptar las ya comunes primeras pantallas de aceptación de condiciones y bienvenida, se nos pedirá nuevamente los mismos datos de usuario que ya pusimos en la anterior instalación.

En el siguiente paso debemos elegir los componentes a instalar, que por defecto ya está seleccionado el único disponible y necesario. Seguimos aceptando los pasos del instalador y comenzará el proceso:

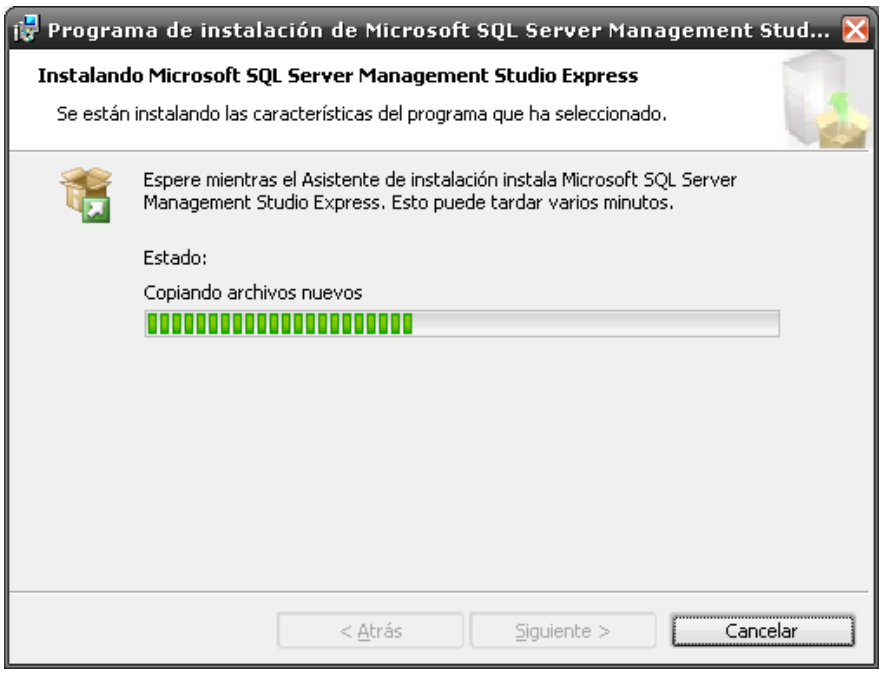


Figura 9.4 - Manual de instalación: instalando el SGBD

Una vez instalado tanto el servidor de base de datos como su sistema de gestión se recomienda de nuevo actualizar todo mediante Windows Update, puesto que es probable que exista algún parche de seguridad o rendimiento para ambos paquetes.

9.1.3 Base de datos

Ya con el servidor de base de datos y el SGBD en el equipo procedemos a instalar la base de datos, sus tablas y datos mínimos para poder trabajar en la aplicación. Para ello accedemos a Microsoft SQL Server Management Studio Express y nos pedirá los datos de conexión a la instancia del servidor, que deberemos rellenar como sigue (o con tus propios valores si decidiste usar otros parámetros):

Figura 9.5 - Manual de usuario: conectando a la instancia

Considera que el nombre del servidor hace referencia al nombre del equipo seguido de \ y el nombre de la instancia. Si la conexión no es en el equipo local deberás introducir la IP de la máquina.

Una vez conectados, en el explorador de objetos deberás crear una nueva base de datos dejando los campos por defecto excepto, claro está, el nombre. En nuestro caso hemos usado "Ecocomputer". Una vez creada, damos clic derecho sobre ella y seleccionamos la opción "Nueva consulta". En el editor de texto que aparecerá pegamos y ejecutamos el siguiente código, que genera las tablas sus columnas de la base de datos (el script puede encontrarse también en el directorio de software a instalar bajo el nombre "script.sql"). Cabe señalar que el script crea todas las tablas de la base de datos, es decir, también las relativas a los demás módulos del sistema:

```
CREATE TABLE [dbo].[Modulos](
	[IDModulo] [int] IDENTITY(1,1) NOT NULL,
	[Nombre] [varchar](50) NULL,
 CONSTRAINT [PK_Modulos] PRIMARY KEY CLUSTERED
(
	[IDModulo] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[TiposRelaciones]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[TiposRelaciones](
	[IDTipoRelacion] [int] NOT NULL,
	[Descripcion] [varchar](150) NOT NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_TiposRelaciones] PRIMARY KEY CLUSTERED
(
	[IDTipoRelacion] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[TiposDieta]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[TiposDieta](
	[IDTipoDieta] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](200) NULL,
	[Unidades] [varchar](20) NULL,
 CONSTRAINT [PK_TiposDieta] PRIMARY KEY CLUSTERED
(
	[IDTipoDieta] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[ComplementosNoSalariales]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[ComplementosNoSalariales](
	[IDComplementoNoSalarial] [int] IDENTITY(1,1) NOT NULL,
	[IDTipoDieta] [int] NOT NULL,
	[Descripcion] [varchar](200) NOT NULL,
	[Valor] [decimal](18, 2) NOT NULL,
```

```
	[FechaInicio] [datetime] NOT NULL,
	[FechaFinal] [datetime] NOT NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_ComplementosNoSalariales] PRIMARY KEY CLUSTERED
(
	[IDComplementoNoSalarial] ASC,
	[IDTipoDieta] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[TiposIncidencia]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[TiposIncidencia](
	[IDTipoIncidencia] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](50) NOT NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_TipoIncidencia] PRIMARY KEY CLUSTERED
(
	[IDTipoIncidencia] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Prioridades]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Prioridades](
	[IDPrioridad] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](50) NOT NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_Prioridades] PRIMARY KEY CLUSTERED
(
	[IDPrioridad] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Servicios]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Servicios](
	[IDServicio] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](200) NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_Servicios] PRIMARY KEY CLUSTERED
(
	[IDServicio] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[TiposFacturacion]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[TiposFacturacion](
	[IDTipoFacturacion] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](200) NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_TiposFacturacion] PRIMARY KEY CLUSTERED
(
	[IDTipoFacturacion] ASC
```

```
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Clientes]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Clientes](
        [IDCliente] [int] IDENTITY(1,1) NOT NULL,
        [Nombre] [varchar](50) NULL,
        [Direccion] [varchar](200) NULL,
        [Poblacion] [varchar](50) NULL,
        [Provincia] [varchar](50) NULL,
        [Telefono] [varchar](50) NULL,
        [Fax] [varchar](50) NULL,
        [Logo] [varchar](max) NULL,
        [Eliminado] [bit] NULL,
 CONSTRAINT [PK_Cliente] PRIMARY KEY CLUSTERED
(
        [IDCliente] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Historial_backups]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Historial_backups](
        [ID_HISTORIAL_BACKUP] [int] NOT NULL,
        [ID_BACKUP] [int] NOT NULL,
        [ID_CLIENTE] [int] NOT NULL,
        [HOST_CLIENTE] [varchar](50) NOT NULL,
        [NOMBRE_BACKUP] [varchar](50) NOT NULL,
        [FECHA_INICIO] [datetime] NOT NULL,
        [FECHA_FIN] [datetime] NULL,
        [RESULTADO_BACKUP] [int] NOT NULL,
        [RESULTADO_COPIA] [varchar](50) NOT NULL,
        [RESULTADO_ENVIO] [varchar](50) NOT NULL,
 CONSTRAINT [PK_Historial_backups] PRIMARY KEY CLUSTERED
(
        [ID_HISTORIAL_BACKUP] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Historial_Jobs]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Historial_Jobs](
        [ID_HISTORIAL_JOB] [int] NOT NULL,
        [ID_BACKUP] [int] NOT NULL,
        [NOMBRE_JOB] [varchar](50) NOT NULL,
        [FECHA_INICIO] [datetime] NOT NULL,
        [FECHA_FIN] [datetime] NULL,
        [TAMANO] [int] NOT NULL,
        [RESULTADO_JOB] [int] NOT NULL,
        [RESULTADO_COPIA] [varchar](50) NOT NULL,
        [RESULTADO_ENVIO] [varchar](50) NOT NULL,
        [ID_CLIENTE] [int] NOT NULL CONSTRAINT [DF_Historial_Jobs_ID_CLIENTE]  DEFAULT
((-1)),
        [HOST_CLIENTE] [varchar](150) NOT NULL CONSTRAINT
[DF_Historial_Jobs_HOST_CLIENTE]  DEFAULT ('Oficina sin determinar'),
 CONSTRAINT [PK_HISTORIAL_JOBS] PRIMARY KEY CLUSTERED
(
        [ID_HISTORIAL_JOB] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
```

```
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[TiposPermisos]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[TiposPermisos](
	[IDTipoPermiso] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](200) NOT NULL,
	[NumTotalDias] [int] NOT NULL,
	[DiaCompleto] [bit] NOT NULL,
	[Laborables] [bit] NOT NULL,
	[RequiereAprobacion] [bit] NOT NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_TiposPermisos] PRIMARY KEY CLUSTERED
(
	[IDTipoPermiso] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Incidencias]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Incidencias](
	[IDIncidencia] [int] IDENTITY(1,1) NOT NULL,
	[IDInformador] [int] NOT NULL,
	[IDAsignada] [int] NOT NULL,
	[IDCliente] [int] NULL,
	[IDContrato] [int] NOT NULL,
	[IDTipoIncidencia] [int] NOT NULL,
	[IDPrioridad] [int] NOT NULL,
	[Titulo] [varchar](max) NOT NULL,
	[Descripcion] [varchar](max) NOT NULL,
	[RealizableRemoto] [bit] NULL,
	[TiempoEstimado] [float] NULL,
	[FechaAlta] [datetime] NULL,
	[Visibilidad] [bit] NOT NULL,
	[Estado] [varchar](50) NOT NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_Incidencias] PRIMARY KEY CLUSTERED
(
	[IDIncidencia] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Perfiles]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Perfiles](
	[IDPerfil] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](50) NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_Perfiles] PRIMARY KEY CLUSTERED
(
	[IDPerfil] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Provincias]') AND type in (N'U'))
BEGIN
```

```
CREATE TABLE [dbo].[Provincias](
	[IDProvincia] [int] NOT NULL,
	[Provincia] [varchar](255) NOT NULL,
 CONSTRAINT [PK_Provincias] PRIMARY KEY CLUSTERED
(
	[IDProvincia] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[Grupos]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Grupos](
	[IDGrupo] [int] IDENTITY(1,1) NOT NULL,
	[Descripcion] [varchar](50) NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_Grupos] PRIMARY KEY CLUSTERED
(
	[IDGrupo] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Gestores]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Gestores](
	[IDGestor] [int] IDENTITY(1,1) NOT NULL,
	[Nombre] [varchar](50) NULL,
	[URL] [varchar](max) NULL,
	[Login] [varchar](50) NULL,
	[Password] [varchar](50) NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_Gestores] PRIMARY KEY CLUSTERED
(
	[IDGestor] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[PermisosModuloUsuario]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[PermisosModuloUsuario](
	[IDUsuario] [int] NOT NULL,
	[IDModulo] [int] NOT NULL,
	[Escritura] [bit] NULL,
 CONSTRAINT [PK_PermisosModuloUsuario] PRIMARY KEY CLUSTERED
(
	[IDUsuario] ASC,
	[IDModulo] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[PermisosModuloPerfil]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[PermisosModuloPerfil](
	[IDPerfil] [int] NOT NULL,
	[IDModulo] [int] NOT NULL,
	[Escritura] [bit] NULL,
```

```
 CONSTRAINT [PK_PermisosModuloPerfil] PRIMARY KEY CLUSTERED
(
	[IDPerfil] ASC,
	[IDModulo] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[Log]') AND
type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Log](
	[IDLog] [int] IDENTITY(1,1) NOT NULL,
	[IDSesion] [int] NOT NULL,
	[Fecha] [datetime] NOT NULL,
	[Evento] [varchar](200) NOT NULL,
	[Pagina] [varchar](100) NULL,
	[Nivel] [int] NULL,
 CONSTRAINT [PK_Log] PRIMARY KEY CLUSTERED
(
	[IDLog] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[DietasDetalle]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[DietasDetalle](
	[IDDietaDetalle] [int] IDENTITY(1,1) NOT NULL,
	[IDDieta] [int] NOT NULL,
	[IDTipoDieta] [int] NOT NULL,
	[IDComplementoNoSalarial] [int] NULL,
	[Albaran] [varchar](50) NULL,
	[Valor] [decimal](6, 2) NULL,
	[ValorConvenio] [decimal](6, 2) NULL,
	[Descripcion] [varchar](200) NULL,
	[Observaciones] [varchar](500) NULL,
	[NombreImagen] [varchar](200) NULL
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[TarifasServicios]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[TarifasServicios](
	[IDTarifaServicio] [int] IDENTITY(1,1) NOT NULL,
	[IDServicio] [int] NULL,
	[Nombre] [varchar](50) NULL,
	[PrecioHora] [decimal](18, 2) NULL,
	[FechaInicio] [datetime] NULL,
	[FechaFin] [datetime] NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_TarifasServicios] PRIMARY KEY CLUSTERED
(
	[IDTarifaServicio] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Contratos]') AND type in (N'U'))
BEGIN
```

```
CREATE TABLE [dbo].[Contratos](
	[IDContrato] [int] IDENTITY(1,1) NOT NULL,
	[IDCliente] [int] NULL,
	[IDTipoFacturacion] [int] NULL,
	[Titulo] [varchar](150) NULL,
	[Descripcion] [varchar](500) NULL,
	[FechaInicio] [datetime] NULL,
	[FechaFin] [datetime] NULL,
	[FechaAlta] [datetime] NULL,
	[Estado] [varchar](150) NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_Contratos] PRIMARY KEY CLUSTERED
(
	[IDContrato] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[ClientesISPConfig]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[ClientesISPConfig](
	[IDCliente] [int] NOT NULL,
	[IDISPConfig] [int] NOT NULL,
 CONSTRAINT [PK_ClientesISPConfig] PRIMARY KEY CLUSTERED
(
	[IDCliente] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Dominios]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Dominios](
	[IDDominio] [int] IDENTITY(1,1) NOT NULL,
	[IDGestor] [int] NOT NULL,
	[IDCliente] [int] NOT NULL,
	[Nombre] [varchar](50) NULL,
	[ClaveFTP] [varchar](50) NULL,
	[Usuario] [varchar](50) NULL,
	[Contraseña] [varchar](50) NULL,
	[FechaAlta] [datetime] NULL,
	[FechaVencimiento] [datetime] NULL,
	[FechaRenovacion] [datetime] NULL,
	[EspacioTotalCuentas] [float] NULL,
	[Observaciones] [varchar](max) NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_Dominios] PRIMARY KEY CLUSTERED
(
	[IDDominio] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Usuarios]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Usuarios](
	[IDUsuario] [int] IDENTITY(1,1) NOT NULL,
	[IDCliente] [int] NULL,
	[Nombre] [varchar](50) NULL,
	[Login] [varchar](50) NULL,
	[Password] [varchar](50) NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_Usuarios] PRIMARY KEY CLUSTERED
(
```

```
	[IDUsuario] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[ContratoServicios]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[ContratoServicios](
	[IDContrato] [int] NOT NULL,
	[IDServicio] [int] NOT NULL,
	[NumHoras] [int] NULL,
	[PrecioHora] [decimal](18, 2) NULL,
	[Desplazamiento] [bit] NULL,
	[Descripcion] [varchar](500) NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_ContratoServicios] PRIMARY KEY CLUSTERED
(
	[IDContrato] ASC,
	[IDServicio] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id = OBJECT_ID(N'[dbo].[Dietas]')
AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Dietas](
	[IDDieta] [int] IDENTITY(1,1) NOT NULL,
	[IDUsuario] [int] NOT NULL,
	[Estado] [int] NULL,
	[Fecha] [datetime] NULL,
	[TotalKM] [int] NULL,
	[TotalImporte] [decimal](6, 2) NULL,
	[TotalHoras] [decimal](6, 2) NULL,
 CONSTRAINT [PK_Dietas] PRIMARY KEY CLUSTERED
(
	[IDDieta] ASC,
	[IDUsuario] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'1' , N'SCHEMA',N'dbo',
N'TABLE',N'Dietas', N'COLUMN',N'Estado'))
EXEC sys.sp_addextendedproperty @name=N'1', @value=N'Sin Validar' ,
@level0type=N'SCHEMA',@level0name=N'dbo', @level1type=N'TABLE',@level1name=N'Dietas',
@level2type=N'COLUMN',@level2name=N'Estado'
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'2' , N'SCHEMA',N'dbo',
N'TABLE',N'Dietas', N'COLUMN',N'Estado'))
EXEC sys.sp_addextendedproperty @name=N'2', @value=N'Validada por el usuario' ,
@level0type=N'SCHEMA',@level0name=N'dbo', @level1type=N'TABLE',@level1name=N'Dietas',
@level2type=N'COLUMN',@level2name=N'Estado'
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'3' , N'SCHEMA',N'dbo',
N'TABLE',N'Dietas', N'COLUMN',N'Estado'))
EXEC sys.sp_addextendedproperty @name=N'3', @value=N'Autorizada' ,
@level0type=N'SCHEMA',@level0name=N'dbo', @level1type=N'TABLE',@level1name=N'Dietas',
@level2type=N'COLUMN',@level2name=N'Estado'
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'4' , N'SCHEMA',N'dbo',
N'TABLE',N'Dietas', N'COLUMN',N'Estado'))
EXEC sys.sp_addextendedproperty @name=N'4', @value=N'Incompleta' ,
@level0type=N'SCHEMA',@level0name=N'dbo', @level1type=N'TABLE',@level1name=N'Dietas',
@level2type=N'COLUMN',@level2name=N'Estado'
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[PerfilesUsuario]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[PerfilesUsuario](
	[IDPerfil] [int] NOT NULL,
	[IDUsuario] [int] NOT NULL,
 CONSTRAINT [PK_PerfilesUsuario] PRIMARY KEY CLUSTERED
(
	[IDPerfil] ASC,
	[IDUsuario] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[GruposUsuarios]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[GruposUsuarios](
	[IDGrupo] [int] NOT NULL,
	[IDUsuario] [int] NOT NULL,
 CONSTRAINT [PK_GruposUsuarios] PRIMARY KEY CLUSTERED
(
	[IDGrupo] ASC,
	[IDUsuario] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[BolsaVacaciones]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[BolsaVacaciones](
	[IDUsuario] [int] NOT NULL,
	[DiasAcumulados] [smallint] NOT NULL
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[PermisosRetribuidos]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[PermisosRetribuidos](
	[IDPermisoRetribuido] [int] IDENTITY(1,1) NOT NULL,
	[IDUsuario] [int] NOT NULL,
	[IDTipoPermiso] [int] NOT NULL,
	[FechaInicio] [datetime] NOT NULL,
	[FechaFin] [datetime] NOT NULL,
	[Comentarios] [varchar](200) NOT NULL,
	[Estado] [smallint] NOT NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_PermisosRetribuidos] PRIMARY KEY CLUSTERED
(
	[IDPermisoRetribuido] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'Autorizado' , N'SCHEMA',N'dbo',
N'TABLE',N'PermisosRetribuidos', N'COLUMN',N'Estado'))
EXEC sys.sp_addextendedproperty @name=N'Autorizado', @value=N'2' ,
@level0type=N'SCHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'PermisosRetribuidos',
@level2type=N'COLUMN',@level2name=N'Estado'
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'Denegado' , N'SCHEMA',N'dbo',
N'TABLE',N'PermisosRetribuidos', N'COLUMN',N'Estado'))
EXEC sys.sp_addextendedproperty @name=N'Denegado', @value=N'3' ,
@level0type=N'SCHEMA',@level0name=N'dbo',
```

```
@level1type=N'TABLE',@level1name=N'PermisosRetribuidos',
@level2type=N'COLUMN',@level2name=N'Estado'
GO
IF NOT EXISTS (SELECT * FROM ::fn_listextendedproperty(N'Solicitado' , N'SCHEMA',N'dbo',
N'TABLE',N'PermisosRetribuidos', N'COLUMN',N'Estado'))
EXEC sys.sp_addextendedproperty @name=N'Solicitado', @value=N'1' ,
@level0type=N'SCHEMA',@level0name=N'dbo',
@level1type=N'TABLE',@level1name=N'PermisosRetribuidos',
@level2type=N'COLUMN',@level2name=N'Estado'
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[DatosPersonales]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[DatosPersonales](
        [IDUsuario] [int] NOT NULL,
        [Direccion] [varchar](150) NULL,
        [Telefono] [varchar](20) NULL,
        [NIF] [varchar](10) NULL,
 CONSTRAINT [PK_DatosPersonales] PRIMARY KEY CLUSTERED
(
        [IDUsuario] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Sesiones]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Sesiones](
        [IDSesion] [int] IDENTITY(1,1) NOT NULL,
        [IDUsuario] [int] NOT NULL,
        [HoraInicio] [datetime] NOT NULL,
        [HoraFin] [datetime] NULL,
        [UltimoEvento] [datetime] NOT NULL,
        [Estado] [bit] NOT NULL,
 CONSTRAINT [PK_Sesiones] PRIMARY KEY CLUSTERED
(
        [IDSesion] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[Intervenciones]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[Intervenciones](
        [IDIntervencion] [int] IDENTITY(1,1) NOT NULL,
        [IDUsuario] [int] NOT NULL,
        [IDIncidencia] [int] NOT NULL,
        [Fecha] [datetime] NULL,
        [Parte] [varchar](50) NULL,
        [NumHoras] [float] NULL,
        [Motivo] [varchar](max) NULL,
        [Observaciones] [varchar](max) NULL,
        [Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_Intervenciones] PRIMARY KEY CLUSTERED
(
        [IDIntervencion] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
```

```
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[DocumentosIncidencia]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[DocumentosIncidencia](
	[IDDocumento] [int] IDENTITY(1,1) NOT NULL,
	[IDIncidencia] [int] NOT NULL,
	[Descripcion] [varchar](max) NULL,
	[Ruta] [varchar](max) NULL,
	[Eliminado] [bit] NOT NULL,
 CONSTRAINT [PK_DocumentosIncidencia] PRIMARY KEY CLUSTERED
(
	[IDDocumento] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
IF NOT EXISTS (SELECT * FROM sys.objects WHERE object_id =
OBJECT_ID(N'[dbo].[RelacionesIncidencias]') AND type in (N'U'))
BEGIN
CREATE TABLE [dbo].[RelacionesIncidencias](
	[IDRelacion] [int] IDENTITY(1,1) NOT NULL,
	[IDIncidenciaMain] [int] NOT NULL,
	[IDIncidenciaSub] [int] NOT NULL,
	[IDTipoRelacion] [int] NOT NULL,
	[Fecha] [datetime] NULL,
	[IDUsuario] [int] NULL,
	[Eliminado] [bit] NULL,
 CONSTRAINT [PK_RelacionesIncidencias] PRIMARY KEY CLUSTERED
(
	[IDRelacion] ASC
)WITH (PAD_INDEX  = OFF, IGNORE_DUP_KEY = OFF) ON [PRIMARY]
) ON [PRIMARY]
END
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PermisosModuloUsuario_Modulos]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PermisosModuloUsuario]'))
ALTER TABLE [dbo].[PermisosModuloUsuario]  WITH CHECK ADD  CONSTRAINT
[FK_PermisosModuloUsuario_Modulos] FOREIGN KEY([IDModulo])
REFERENCES [dbo].[Modulos] ([IDModulo])
GO
ALTER TABLE [dbo].[PermisosModuloUsuario] CHECK CONSTRAINT
[FK_PermisosModuloUsuario_Modulos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PermisosModuloUsuario_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PermisosModuloUsuario]'))
ALTER TABLE [dbo].[PermisosModuloUsuario]  WITH CHECK ADD  CONSTRAINT
[FK_PermisosModuloUsuario_Usuarios] FOREIGN KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[PermisosModuloUsuario] CHECK CONSTRAINT
[FK_PermisosModuloUsuario_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PermisosModuloPerfil_Modulos]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PermisosModuloPerfil]'))
ALTER TABLE [dbo].[PermisosModuloPerfil]  WITH CHECK ADD  CONSTRAINT
[FK_PermisosModuloPerfil_Modulos] FOREIGN KEY([IDModulo])
REFERENCES [dbo].[Modulos] ([IDModulo])
GO
ALTER TABLE [dbo].[PermisosModuloPerfil] CHECK CONSTRAINT
[FK_PermisosModuloPerfil_Modulos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PermisosModuloPerfil_Perfiles]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PermisosModuloPerfil]'))
ALTER TABLE [dbo].[PermisosModuloPerfil]  WITH CHECK ADD  CONSTRAINT
[FK_PermisosModuloPerfil_Perfiles] FOREIGN KEY([IDPerfil])
REFERENCES [dbo].[Perfiles] ([IDPerfil])
GO
ALTER TABLE [dbo].[PermisosModuloPerfil] CHECK CONSTRAINT
[FK_PermisosModuloPerfil_Perfiles]
GO
```

```
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Log_Sesiones]') AND parent_object_id = OBJECT_ID(N'[dbo].[Log]'))
ALTER TABLE [dbo].[Log]  WITH CHECK ADD  CONSTRAINT [FK_Log_Sesiones] FOREIGN
KEY([IDSesion])
REFERENCES [dbo].[Sesiones] ([IDSesion])
GO
ALTER TABLE [dbo].[Log] CHECK CONSTRAINT [FK_Log_Sesiones]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_DietasDetalle_TiposDieta]') AND parent_object_id =
OBJECT_ID(N'[dbo].[DietasDetalle]'))
ALTER TABLE [dbo].[DietasDetalle]  WITH CHECK ADD  CONSTRAINT
[FK_DietasDetalle_TiposDieta] FOREIGN KEY([IDTipoDieta])
REFERENCES [dbo].[TiposDieta] ([IDTipoDieta])
GO
ALTER TABLE [dbo].[DietasDetalle] CHECK CONSTRAINT [FK_DietasDetalle_TiposDieta]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_TarifasServicios_Servicios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[TarifasServicios]'))
ALTER TABLE [dbo].[TarifasServicios]  WITH CHECK ADD  CONSTRAINT
[FK_TarifasServicios_Servicios] FOREIGN KEY([IDServicio])
REFERENCES [dbo].[Servicios] ([IDServicio])
GO
ALTER TABLE [dbo].[TarifasServicios] CHECK CONSTRAINT [FK_TarifasServicios_Servicios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Contratos_TiposFacturacion]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Contratos]'))
ALTER TABLE [dbo].[Contratos]  WITH CHECK ADD  CONSTRAINT
[FK_Contratos_TiposFacturacion] FOREIGN KEY([IDTipoFacturacion])
REFERENCES [dbo].[TiposFacturacion] ([IDTipoFacturacion])
GO
ALTER TABLE [dbo].[Contratos] CHECK CONSTRAINT [FK_Contratos_TiposFacturacion]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_ClientesISPConfig_Clientes]') AND parent_object_id =
OBJECT_ID(N'[dbo].[ClientesISPConfig]'))
ALTER TABLE [dbo].[ClientesISPConfig]  WITH CHECK ADD  CONSTRAINT
[FK_ClientesISPConfig_Clientes] FOREIGN KEY([IDCliente])
REFERENCES [dbo].[Clientes] ([IDCliente])
GO
ALTER TABLE [dbo].[ClientesISPConfig] CHECK CONSTRAINT [FK_ClientesISPConfig_Clientes]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Dominios_Cliente]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Dominios]'))
ALTER TABLE [dbo].[Dominios]  WITH CHECK ADD  CONSTRAINT [FK_Dominios_Cliente] FOREIGN
KEY([IDCliente])
REFERENCES [dbo].[Clientes] ([IDCliente])
GO
ALTER TABLE [dbo].[Dominios] CHECK CONSTRAINT [FK_Dominios_Cliente]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Dominios_Dominios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Dominios]'))
ALTER TABLE [dbo].[Dominios]  WITH CHECK ADD  CONSTRAINT [FK_Dominios_Dominios] FOREIGN
KEY([IDDominio])
REFERENCES [dbo].[Gestores] ([IDGestor])
GO
ALTER TABLE [dbo].[Dominios] CHECK CONSTRAINT [FK_Dominios_Dominios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Usuarios_Clientes]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Usuarios]'))
ALTER TABLE [dbo].[Usuarios]  WITH CHECK ADD  CONSTRAINT [FK_Usuarios_Clientes] FOREIGN
KEY([IDCliente])
REFERENCES [dbo].[Clientes] ([IDCliente])
GO
ALTER TABLE [dbo].[Usuarios] CHECK CONSTRAINT [FK_Usuarios_Clientes]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_ContratoServicios_Contratos]') AND parent_object_id =
OBJECT_ID(N'[dbo].[ContratoServicios]'))
ALTER TABLE [dbo].[ContratoServicios]  WITH CHECK ADD  CONSTRAINT
[FK_ContratoServicios_Contratos] FOREIGN KEY([IDContrato])
REFERENCES [dbo].[Contratos] ([IDContrato])
GO
ALTER TABLE [dbo].[ContratoServicios] CHECK CONSTRAINT [FK_ContratoServicios_Contratos]
```

```
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Dietas_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Dietas]'))
ALTER TABLE [dbo].[Dietas]  WITH CHECK ADD  CONSTRAINT [FK_Dietas_Usuarios] FOREIGN
KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[Dietas] CHECK CONSTRAINT [FK_Dietas_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PerfilesUsuario_Perfiles]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PerfilesUsuario]'))
ALTER TABLE [dbo].[PerfilesUsuario]  WITH CHECK ADD  CONSTRAINT
[FK_PerfilesUsuario_Perfiles] FOREIGN KEY([IDPerfil])
REFERENCES [dbo].[Perfiles] ([IDPerfil])
GO
ALTER TABLE [dbo].[PerfilesUsuario] CHECK CONSTRAINT [FK_PerfilesUsuario_Perfiles]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PerfilesUsuario_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PerfilesUsuario]'))
ALTER TABLE [dbo].[PerfilesUsuario]  WITH CHECK ADD  CONSTRAINT
[FK_PerfilesUsuario_Usuarios] FOREIGN KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[PerfilesUsuario] CHECK CONSTRAINT [FK_PerfilesUsuario_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_GruposUsuarios_Grupos]') AND parent_object_id =
OBJECT_ID(N'[dbo].[GruposUsuarios]'))
ALTER TABLE [dbo].[GruposUsuarios]  WITH CHECK ADD  CONSTRAINT
[FK_GruposUsuarios_Grupos] FOREIGN KEY([IDGrupo])
REFERENCES [dbo].[Grupos] ([IDGrupo])
GO
ALTER TABLE [dbo].[GruposUsuarios] CHECK CONSTRAINT [FK_GruposUsuarios_Grupos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_GruposUsuarios_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[GruposUsuarios]'))
ALTER TABLE [dbo].[GruposUsuarios]  WITH CHECK ADD  CONSTRAINT
[FK_GruposUsuarios_Usuarios] FOREIGN KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[GruposUsuarios] CHECK CONSTRAINT [FK_GruposUsuarios_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_DiasAcumulados_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[BolsaVacaciones]'))
ALTER TABLE [dbo].[BolsaVacaciones]  WITH CHECK ADD  CONSTRAINT
[FK_DiasAcumulados_Usuarios] FOREIGN KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[BolsaVacaciones] CHECK CONSTRAINT [FK_DiasAcumulados_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PermisosRetribuidos_TiposPermisos]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PermisosRetribuidos]'))
ALTER TABLE [dbo].[PermisosRetribuidos]  WITH CHECK ADD  CONSTRAINT
[FK_PermisosRetribuidos_TiposPermisos] FOREIGN KEY([IDTipoPermiso])
REFERENCES [dbo].[TiposPermisos] ([IDTipoPermiso])
GO
ALTER TABLE [dbo].[PermisosRetribuidos] CHECK CONSTRAINT
[FK_PermisosRetribuidos_TiposPermisos]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_PermisosRetribuidos_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[PermisosRetribuidos]'))
ALTER TABLE [dbo].[PermisosRetribuidos]  WITH CHECK ADD  CONSTRAINT
[FK_PermisosRetribuidos_Usuarios] FOREIGN KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[PermisosRetribuidos] CHECK CONSTRAINT
[FK_PermisosRetribuidos_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_DatosPersonales_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[DatosPersonales]'))
```

```
ALTER TABLE [dbo].[DatosPersonales]  WITH CHECK ADD  CONSTRAINT
[FK_DatosPersonales_Usuarios] FOREIGN KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[DatosPersonales] CHECK CONSTRAINT [FK_DatosPersonales_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Sesiones_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Sesiones]'))
ALTER TABLE [dbo].[Sesiones]  WITH CHECK ADD  CONSTRAINT [FK_Sesiones_Usuarios] FOREIGN
KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[Sesiones] CHECK CONSTRAINT [FK_Sesiones_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Intervenciones_Incidencias]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Intervenciones]'))
ALTER TABLE [dbo].[Intervenciones]  WITH CHECK ADD  CONSTRAINT
[FK_Intervenciones_Incidencias] FOREIGN KEY([IDIncidencia])
REFERENCES [dbo].[Incidencias] ([IDIncidencia])
GO
ALTER TABLE [dbo].[Intervenciones] CHECK CONSTRAINT [FK_Intervenciones_Incidencias]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_Intervenciones_Usuarios]') AND parent_object_id =
OBJECT_ID(N'[dbo].[Intervenciones]'))
ALTER TABLE [dbo].[Intervenciones]  WITH CHECK ADD  CONSTRAINT
[FK_Intervenciones_Usuarios] FOREIGN KEY([IDUsuario])
REFERENCES [dbo].[Usuarios] ([IDUsuario])
GO
ALTER TABLE [dbo].[Intervenciones] CHECK CONSTRAINT [FK_Intervenciones_Usuarios]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_DocumentosIncidencia_Incidencias]') AND parent_object_id =
OBJECT_ID(N'[dbo].[DocumentosIncidencia]'))
ALTER TABLE [dbo].[DocumentosIncidencia]  WITH CHECK ADD  CONSTRAINT
[FK_DocumentosIncidencia_Incidencias] FOREIGN KEY([IDIncidencia])
REFERENCES [dbo].[Incidencias] ([IDIncidencia])
GO
ALTER TABLE [dbo].[DocumentosIncidencia] CHECK CONSTRAINT
[FK_DocumentosIncidencia_Incidencias]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_RelacionesIncidencias_Incidencias]') AND parent_object_id =
OBJECT_ID(N'[dbo].[RelacionesIncidencias]'))
ALTER TABLE [dbo].[RelacionesIncidencias]  WITH CHECK ADD  CONSTRAINT
[FK_RelacionesIncidencias_Incidencias] FOREIGN KEY([IDIncidenciaMain])
REFERENCES [dbo].[Incidencias] ([IDIncidencia])
GO
ALTER TABLE [dbo].[RelacionesIncidencias] CHECK CONSTRAINT
[FK_RelacionesIncidencias_Incidencias]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_RelacionesIncidencias_Incidencias1]') AND parent_object_id =
OBJECT_ID(N'[dbo].[RelacionesIncidencias]'))
ALTER TABLE [dbo].[RelacionesIncidencias]  WITH CHECK ADD  CONSTRAINT
[FK_RelacionesIncidencias_Incidencias1] FOREIGN KEY([IDIncidenciaSub])
REFERENCES [dbo].[Incidencias] ([IDIncidencia])
GO
ALTER TABLE [dbo].[RelacionesIncidencias] CHECK CONSTRAINT
[FK_RelacionesIncidencias_Incidencias1]
GO
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_RelacionesIncidencias_RelacionesIncidencias]') AND
parent_object_id = OBJECT_ID(N'[dbo].[RelacionesIncidencias]'))
ALTER TABLE [dbo].[RelacionesIncidencias]  WITH CHECK ADD  CONSTRAINT
[FK_RelacionesIncidencias_RelacionesIncidencias] FOREIGN KEY([IDRelacion])
REFERENCES [dbo].[RelacionesIncidencias] ([IDRelacion])
GO
ALTER TABLE [dbo].[RelacionesIncidencias] CHECK CONSTRAINT
[FK_RelacionesIncidencias_RelacionesIncidencias]
```

Con la estructura de tablas ya establecida necesitamos introducir los datos básicos que la aplicación necesita para funcionar. Este paso no se puede incluir en el script de generación

anterior puesto que la versión 2005 de Microsoft SQL Server no dispone de esta función. Estos datos son:

9.1.3.1 Tabla Modulos:

IDModulo	Nombre
1	Gestión de Usuarios
2	Gestión de Dominios
3	Gestión de Vacaciones
4	Gestión de Dietas
5	Gestión de Contratos
6	Gestión de Servicios
7	Gestión de Incidencias
8	Gestión de Clientes
10	Gestion de Backups

9.1.3.2 Tabla Perfiles:

IDPerfil	Descripcion	Eliminado
1	Administrador	0
2	Desarrollador	0
3	Técnico	0
4	Administración	0
5	Organizador	0

9.1.3.3 Tabla Usuarios

IDUsuario	IDCliente	Nombre	Login	Password	Eliminado
1	1	Administrador	admin	admin	0

9.1.3.4 Tabla PerfilesUsuario

IDPerfil	IDUsuario
1	1

9.1.3.5 Tabla PermisosModuloUsuario

IDUsuario	IDModulo	Escritura
1	1	1
1	2	1
1	3	1
1	4	1
1	5	1
1	6	1
1	7	1
1	8	1
1	10	1

9.1.3.6 Tabla PermisosModuloPerfil

IDPerfil	IDModulo	Escritura
1	1	1
1	2	1
1	3	1
1	4	1
1	5	1
1	6	1

```
1           10          1
```

9.1.3.7 Tabla TiposDieta

```
IDTipoDieta Descripcion          Unidades
----------- -------------------- -------------
1           Media Dieta          €
2           Dieta Completa       €
3           Parking              €
4           Desplazamiento       km
5           Otros                €
6           Horas Extra          horas
```

9.1.3.8 Tabla TiposPermisos

```
IDTipoPermiso Descripcion                                NumTotalDias DiaCompleto
Laborables RequiereAprobacion Eliminado
------------- ---------------------------------------- ------------ ----------- ------
---- ------------------ ---------
7             Matrimonio                                 18           1           0
0                  0

8              Fallecimiento de familiar hasta 2° grado  5            1           0
0                  0

9              Enfermedad de familiar hasta 2° grado     5            1           0
0                  0

10             Ingreso UVI/UCI familiar hasta 2° grado   10           1           0
0                  0

11            Muerte familiar de 3° grado                1            1           1
0                  0

12            Traslado de domicilio habitual             1            1           1
0                  0

13            Asuntos propio                             5            1           1
0                  0

14            Nacimiento de hijo                         2            1           1
0                  0

15             Matrimonio de parientes hasta 2° grado    1            1           1
0                  0

16             Consulta médica: tiempo indispensable     -1           0           1
0                  0

17            Consulta médica de familiares              -1           0           1
0                  0

18            Prenatal                                   -1           0           1
0                  0

19            Libre disponibilidad                       2            1           1
0                  0

20            Exámenes oficiales                         -1           0           1
0                  0

25            Vacaciones                                 30           1           0
1                  0
```

9.1.3.9 Tabla ComplementosNoSalariales

```
IDComplementoNoSalarial IDTipoDieta Descripcion                Valor       FechaInicio
FechaFinal              Eliminado
----------------------- ----------- ------------------------ ---------- ----------------
------- ----------------------- ---------
1                         1           Media dieta 2013          15.69       2012-01-01
00:00:00.000 2013-12-31 00:00:00.000 0

2                         2            Dieta completa 2013      45.33       2012-01-01
00:00:00.000 2013-12-31 00:00:00.000 0

3                         4            Desplazamiento 2013      0.31        2012-01-01
00:00:00.000 2013-12-31 00:00:00.000 0

4                         6           Horas extra 10€           10.00       2012-01-01
00:00:00.000 2013-12-31 00:00:00.000 0

5                         6           Horas extra 20€           20.00       2012-01-01
00:00:00.000 2013-12-31 00:00:00.000 0

6                         6           Horas extra 30€           30.00       2012-01-01
00:00:00.000 2014-01-05 00:00:00.000 0
```

Con estos datos básicos introducidos ya tenemos la base de datos totalmente preparada para recibir cualquier orden de la aplicación.

9.1.4 Servidor web

El siguiente paso es preparar el entorno en el que se ejecutará la aplicación, es decir, el servidor web Apache Tomcat. Dado que es necesario modificar una serie de parámetros de configuración y adjuntar librerías adicionales, el proceso se torna muy tedioso de explicar y aplicar paso a paso. Por eso se incluye en el directorio de software del CD ya comprimido todo el entorno del Apache Tomcat.

Para ponerlo en funcionamiento basta con descomprimirlo en una carpeta cualquiera del disco duro (por ejemplo, `C:\apache-tomcat\`). Cabe destacar que está configurado para que las aplicaciones se ejecuten sobre el puerto 8080, por lo que el firewall del equipo debe permitir las conexiones entrantes en dicho puerto.

También es importante indicar que el servidor está preparado para ejecutar sus aplicaciones usando la base de datos local, instalada en el mismo equipo. Los parámetros de conexión por defecto son los indicados en el proceso de instalación de SQL Server 2005, es decir, instancia “PortalEmpleado”, puerto 1433, usuario “sa” y contraseña “Portal.Empleado”.

9.2 Manual de Ejecución

9.2.1 Inicio del servidor web

El proceso de arranque del servidor web es tan sencillo como ejecutar el archivo de lotes `bin\startup.bat` del directorio de Tomcat, siempre y cuando el equipo tenga Java instalado.

9.2.2 Desplegar la aplicación

La configuración que ya tiene preparada el servidor Tomcat hacen que desplegar la aplicación se realice simplemente copiando los archivos `Intranet.war`, `GestionDietas.war` y `GestionVacaciones.war` en la carpeta `webapps\` del directorio donde hayamos descomprimido tomcat en el apartado anterior. Tras unos segundos, el servidor leerá automáticamente los archivos de distribución de aplicaciones .war y los desplegará.

Requiere mención especial que el archivo Intranet.war no pertenece a este proyecto, pero que es necesario para poder tener acceso a las otras dos aplicaciones que sí pertenecen, puesto que se encarga de realizar el inicio de sesión del usuario.

Una vez desplegado, podemos acceder a la aplicación desde cualquier navegador web a través de la URL http://localhost:8080/Intranet, o bien desde el exterior, siempre y cuando el cortafuegos acepte conexiones externas, desde http://IP-de-la-máquina:8080/Intranet.

9.2.3 Parada del servidor web

De manera similar al arranque del servidor, detenerlo pasa por ejecutar el archivo de lotes `bin\shutdown.bat` del directorio donde previamente descomprimimos Tomcat.

9.3 Manual de Usuario

9.3.1 Inicio de sesión en la aplicación

Para poder comenzar a utilizar las funcionalidades de este proyecto antes es necesario iniciar sesión en la aplicación. Para ello debemos acceder desde el navegador web a la URL http://localhost:8080/Intranet si estamos trabajando desde el ordenador local, o bien a http://IP-de-la-máquina:8080/Intranet.

Para ofrecer un ejemplo completo de uso, podemos acceder a la aplicación desde http://156.35.98.47:8080/Intranet, el servidor de desarrollo utilizado por el alumno que solo contiene los módulos concernientes a este proyecto, o http://85.152.38.139:8080/Intranet/, el servidor de producción controlado por Ecocomputer donde toda la funcionalidad se encuentra desplegada. Al acceder nos encontraremos con una pantalla como esta, en la que deberemos escribir nuestras credenciales de acceso (usuario "admin" y contraseña "admin", por defecto):

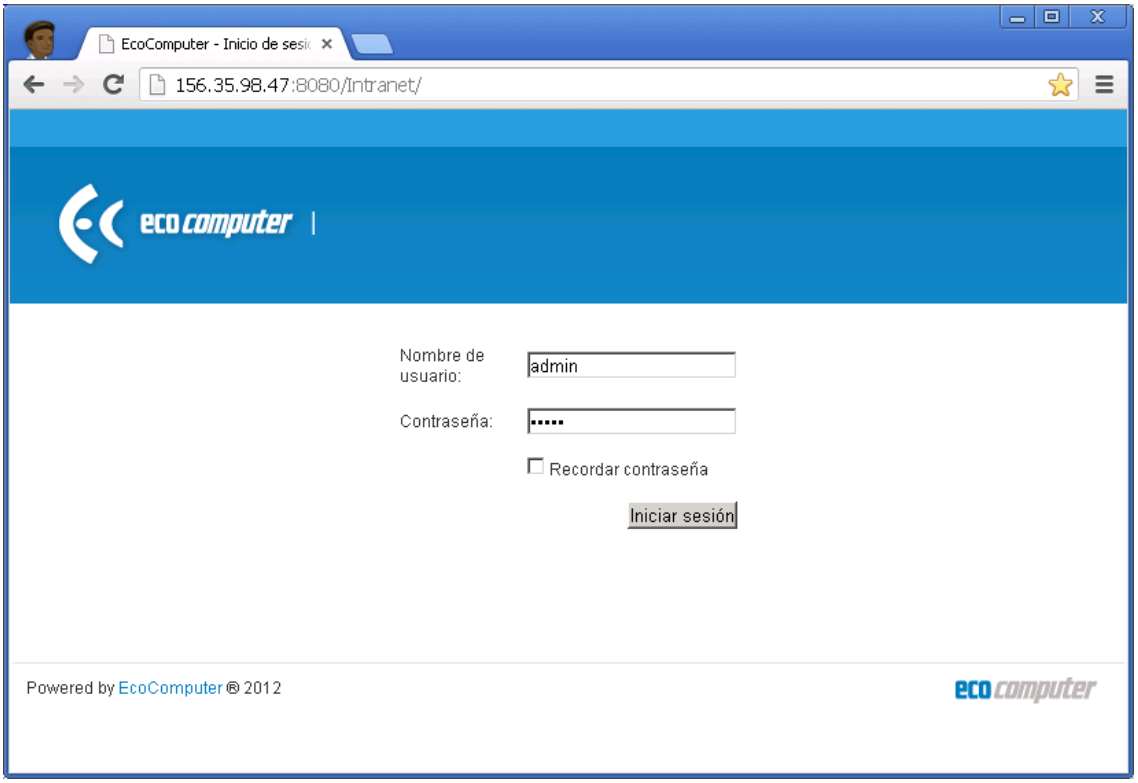


Figura 9.6 - Manual de usuario: inicio de sesión

Una vez identificados llegaremos a la pantalla de bienvenida donde los datos del usuario se muestran, permite cambiar la contraseña y demás. Como estos aspectos no pertenecen a este proyecto se han ignorado en este manual de usuario. Pasemos, por tanto, a explicar el funcionamiento de los módulos que sí pertenecen.

9.3.2 Módulo de gestión de dietas

Para acceder a este módulo es necesario seleccionar "Dietas" en la barra de navegación superior. Para que esto aparezca, el usuario debe tener permisos de escritura y de lectura sobre el módulo. Consúltese con el administrador del sistema si esto no es así.

9.3.2.1 Funcionalidad de usuario

9.3.2.1.1 Visualizar dietas

Lo primero que nos encontramos al acceder al módulo es una vista de las dietas declaradas por el usuario en el mes y año actual. Si no hay ninguna, la tabla aparece vacía:

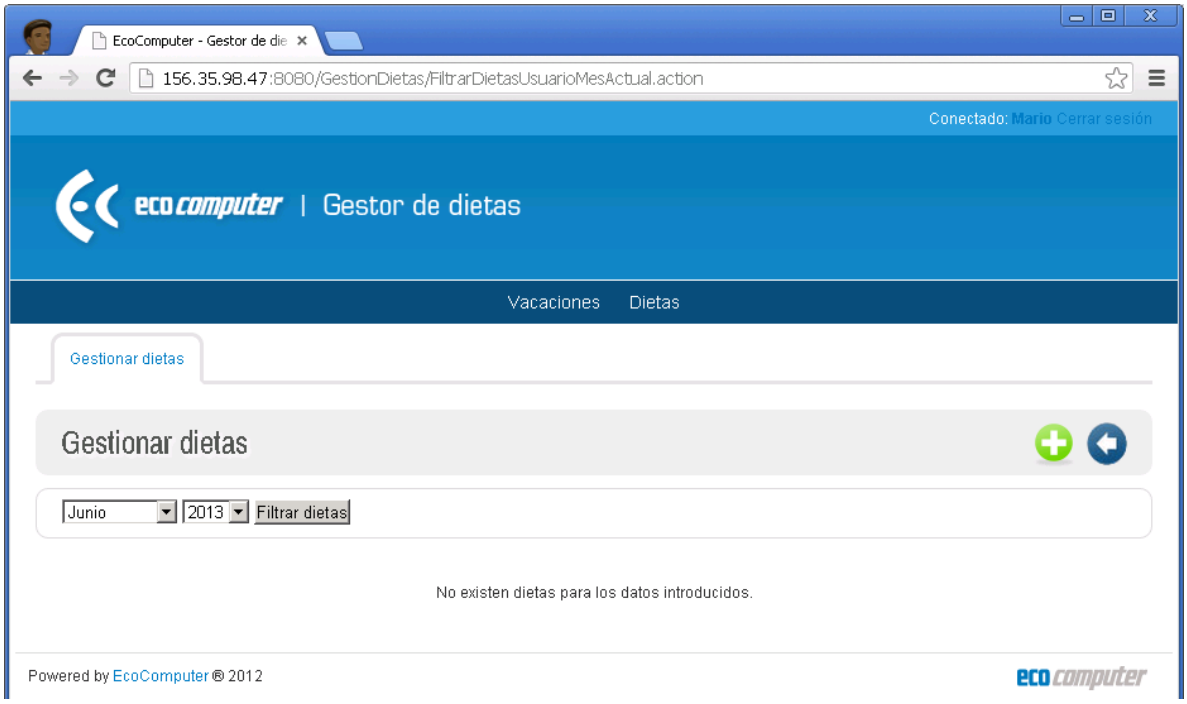


Figura 9.7 - Manual de usuario: sin dietas declaradas

Para cambiar el mes y año a visualizar basta con seleccionarlo en los menús desplegables correspondientes y pulsar el botón "Filtrar dietas".

9.3.2.1.2 Insertar dieta

Desde la ventana de visualización de dietas anterior pulsamos el botón denotado por una cruz blanca sobre fondo verde para acceder a la interfaz de inserción de dietas.

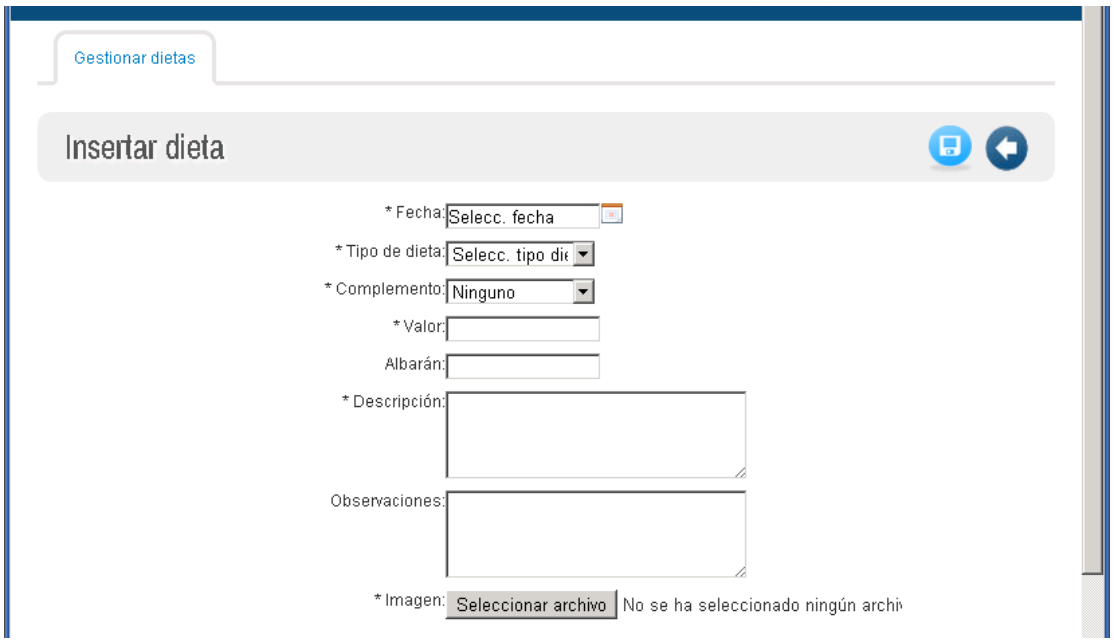


Figura 9.8 - Manual de usuario: inserción de dieta

Será necesario rellenar los datos de la dieta. Téngase en cuenta que tras seleccionar la fecha y el tipo de la misma el desplegable "Complemento" se actualizará para mostrar las opciones que el usuario tiene disponible para esos criterios. Los campos marcados con un asterisco (*) son obligatorios. La imagen debe estar en formato .png, .gif o .jpg y pesar menos de 2 MB para ser aceptada por el sistema. Una vez rellenos los datos debemos pulsar sobre el botón del disquete blanco sobre fondo azul:

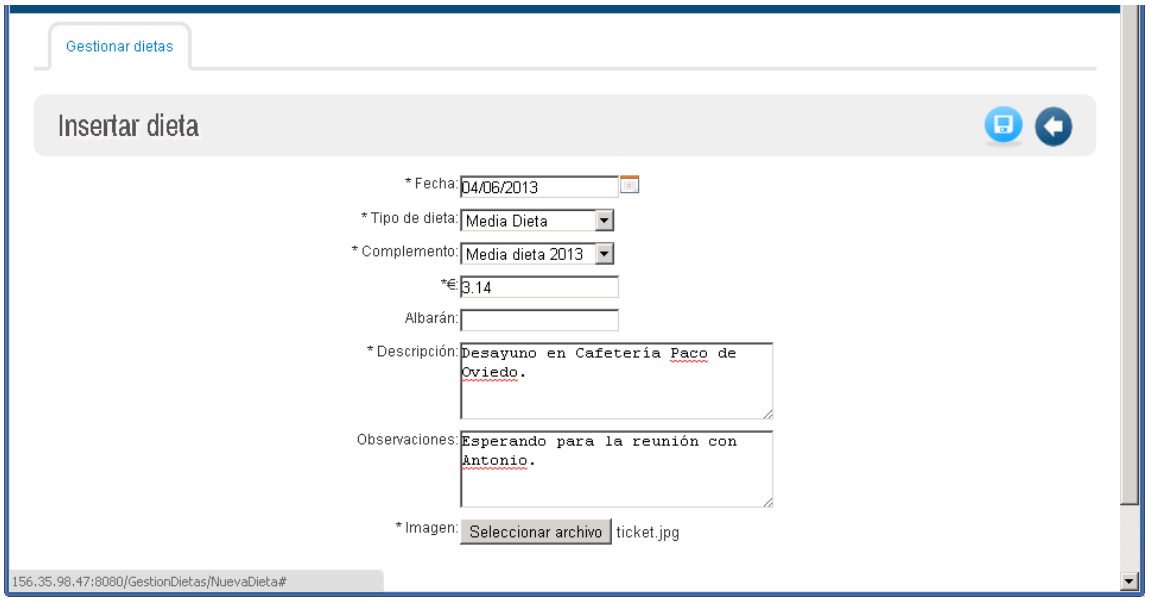


Figura 9.9 – Manuel de usuario: insertando dieta

Tras aceptar, volveremos a la interfaz de visualización y filtrado de dietas. Por motivos ilustrativos añadiremos alguna dieta más:

Figura 9.10 - Manual de usuario: visualización de dietas

Podemos ver que tenemos tres inserciones realizadas: la media dieta del 4 de junio que vimos en la pantalla anterior y dos más, una para el mismo día y otra para el 19 de junio. Puede contemplarse que las dietas se agrupan por días, y que todas ellas muestran su valor (en euros, kilómetros u horas) además de su importe tras aplicar el complemento no salarial correspondiente. Cada una de ellas muestra su descripción, observaciones y albarán, así como las acciones disponibles para cada una de ellas. Nótese que la primera columna contiene un icono representando el estado de la dieta.

Finalmente, en la última fila de la tabla podemos ver el cálculo total del importe que suponen las dietas indicadas.

9.3.2.1.3 Visualizar imagen adjunta a dieta

Para poder ver la imagen que en su momento se adjuntó con la dieta debemos pulsar sobre el primero de los botones de la derecha, denotado por una fotografía, y se nos abrirá una ventana emergente con dicha imagen:

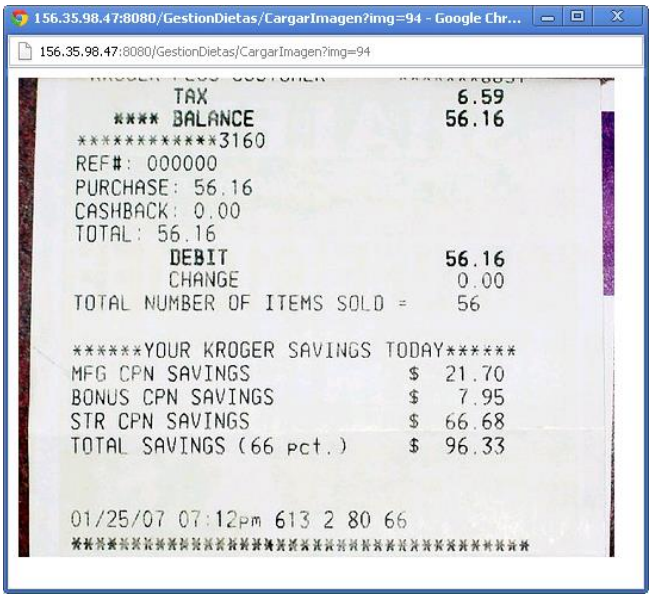


Figura 9.11 - Manual de usuario: imagen asociada a dieta

9.3.2.1.4 Editar dieta

El segundo de los botones nos permite realizar una modificación sobre la dieta a la que haga referencia. Pulsándolo llegamos a la misma interfaz anterior de inserción de dieta, pero con los datos ya preparados para editarlos cuando sea posible:

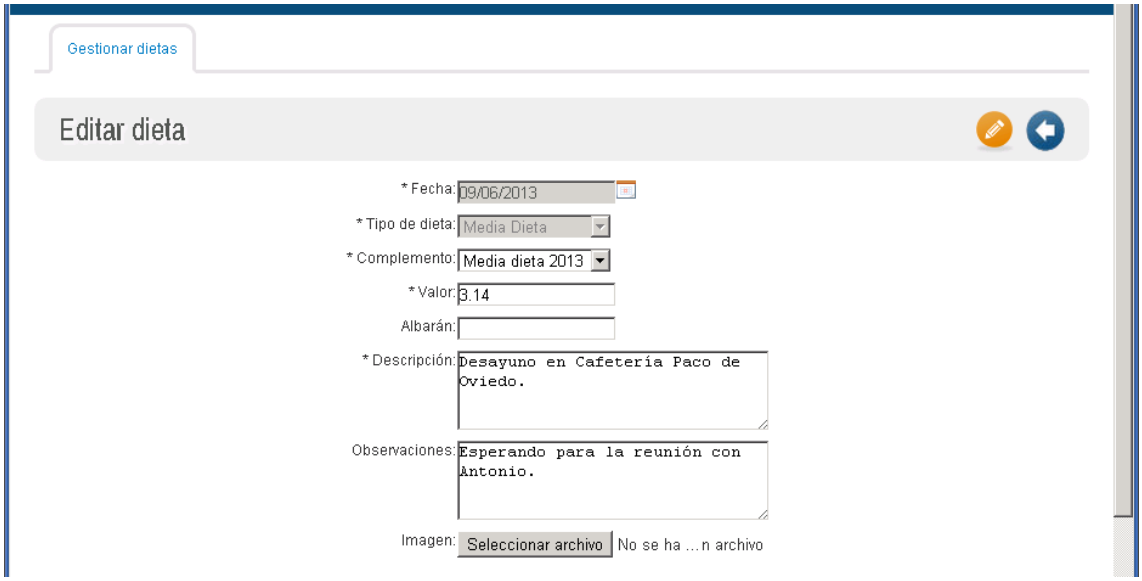


Figura 9.12 - Manual de usuario: editar dieta

Puede editarse cualquiera de los campos activos. Nótese que el campo imagen ya no es obligatorio, por lo que se si se deja vacío ningún cambio se hará en la imagen subida anteriormente. Pulsando el botón superior denotado por un lápiz blanco sobre fondo naranja se realiza la modificación.

9.3.2.1.5 Eliminar dieta

El último de los botones de la derecha de cada detalle, denotado por una equis blanca sobre fondo rojo, permite eliminarlo del sistema. Basta con presionarlo y aceptar el diálogo de confirmación consecuente para hacerlo.

9.3.2.1.6 Validar dieta

Una vez el usuario esté conforme con todos los conceptos de las dietas indicadas, puede pulsar el botón en forma de uve verde para validarla y poder ser así revisada por el administrador del sistema. Tras hacerlo, veremos cómo el icono de estado de la dieta cambia pare reflejar cómo se encuentra, además de que desaparecen los controles sobre la edición de los detalles:

Figura 9.13 - Manual de usuario: dieta validada

9.3.2.1.7 Generar informe PDF

Una vez que un mes dispone de dietas, el botón de generación de informe en formato PDF aparece en la barra de herramientas, identificado como una hoja de papel blanca sobre fondo naranja. Al pulsarlo y esperar que el programa lo genere (puede tardar más en función de la cantidad de dietas), el documento se descarga en el equipo del usuario:

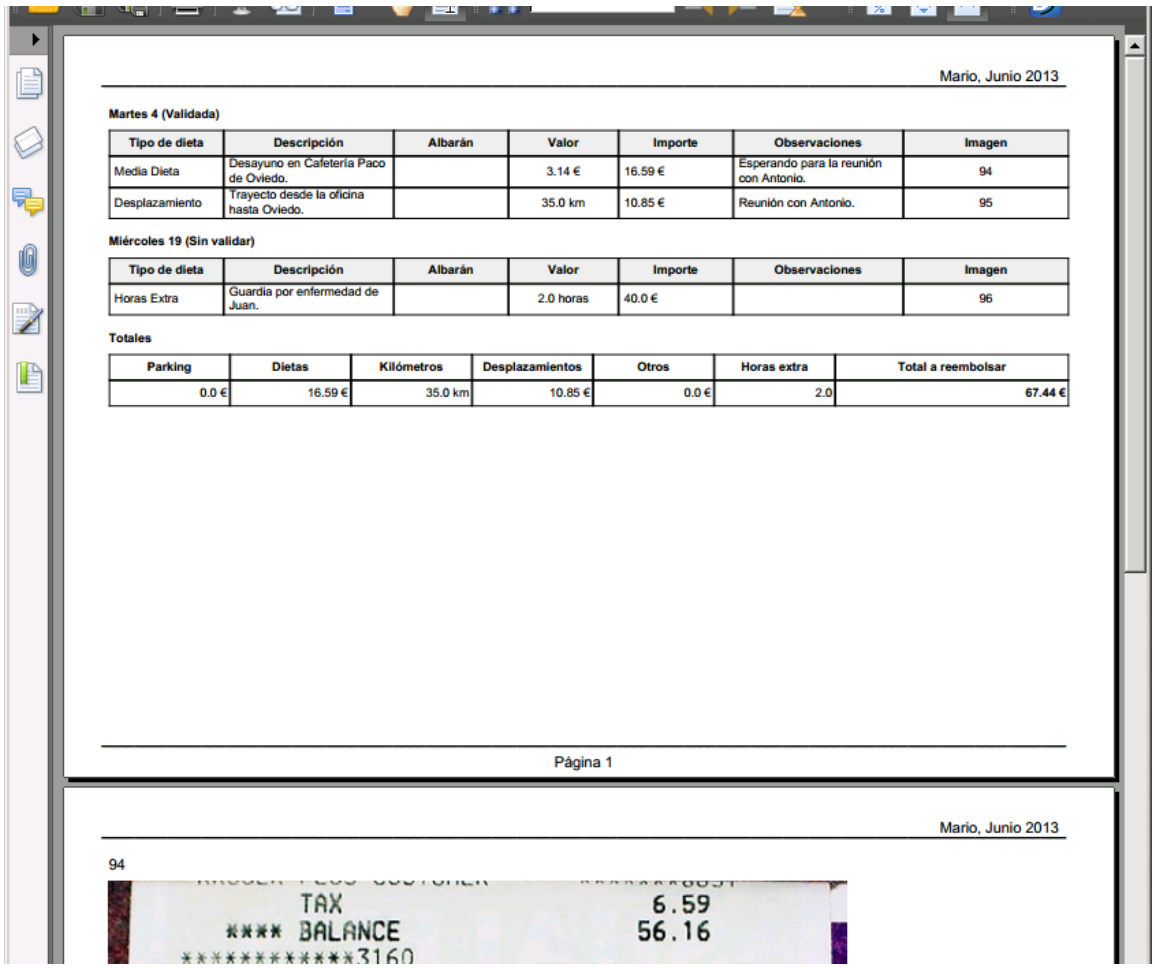


Figura 9.14 - Manual de usuario: informe PDF

Podemos ver que el informe contiene las dietas y sus detalles distribuidos en tablas, así como una última tabla donde se muestran los totales. Se adjuntan también las imágenes asociadas a dichas dietas, encabezadas por su identificador para una mejor identificación y localización de las mismas.

9.3.2.2 Funcionalidad adicional de administrador

Pese a que el administrador es un usuario más del sistema y por tanto puede realizar las mismas tareas que éstos, dispone de tareas extra de más alto nivel que el resto de los usuarios.

9.3.2.2.1 Visualizar dietas

En la interfaz de visualización de dietas podemos ver cómo el administrador dispone de un criterio de filtrado más: usuario. Esto significa que puede ver las dietas del resto de los usuarios de la aplicación:

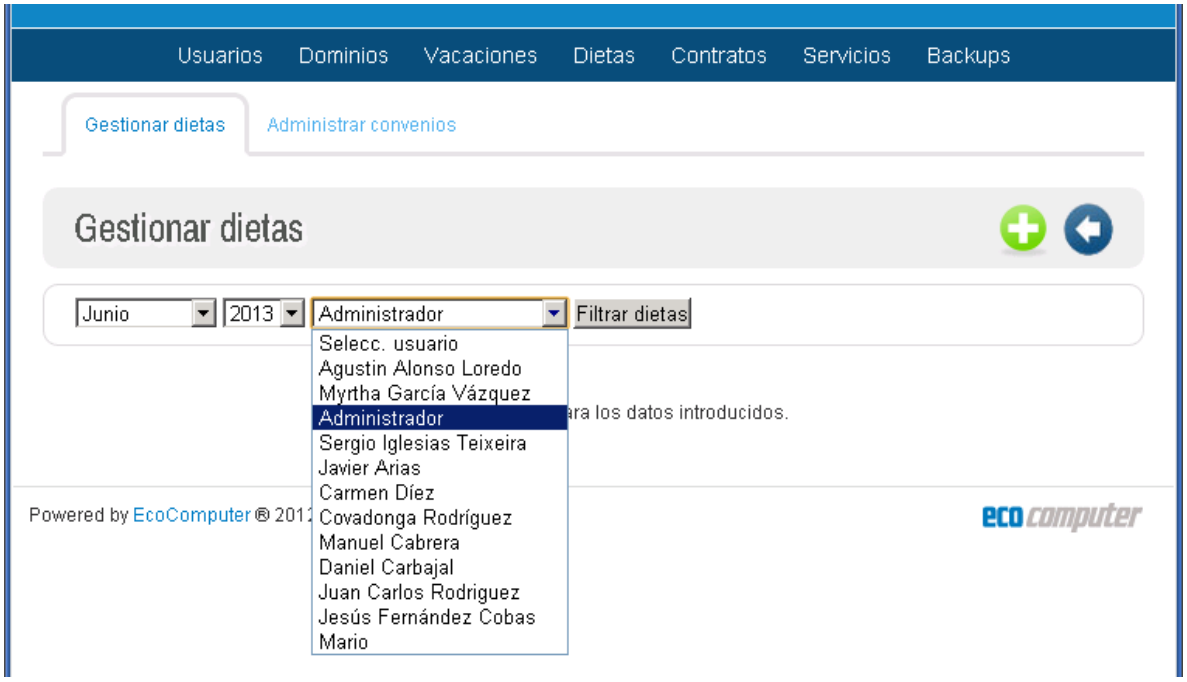


Figura 9.15 - Manual de usuario: visualizar dietas como administrador

9.3.2.2.2 Gestión de dietas de otros usuarios

Visualizando las dietas del usuario anterior, podemos ver que el administrador tiene controles de edición, eliminado y validación de dietas, aunque no sean suyas, así como poder generar el informe PDF. Adicionalmente posee dos controles más una vez que la dieta haya sido validada: autorizar y rechazar. Pinchar sobre cada uno de ellos cambiará el estado (y por tanto el icono) de la dieta a la que haga referencia:

Figura 9.16 - Manual de usuario: autorizar y rechazar dieta

9.3.2.2.3 Administración de convenio

Bajo el menú general de navegación, el administrador tendrá la opción de administrar los complementos no salariales del convenio no laboral pinchando sobre el enlace "Administrar convenios". Lo primero que vemos tras hacerlo es el listado de los mismos:

Figura 9.17 - Manual de usuario: listado de complementos no salariales

La tabla se explica por sí sola: cada complemento no salarial tiene un valor determinado, y un intervalo de fechas en los que dicho complemento está activo o vigente. Véase por ejemplo el último de la tabla, "complemento antiguo", que se encuentra inactivo puesto que su vigencia expiró el pasado 4 de junio. Este complemento, por tanto, no podrá ser seleccionado durante la inserción de dietas.

9.3.2.2.4 Inserción de complemento no salarial

De la misma manera que con las dietas, presionando el botón de la cruz blanca sobre fondo verde llegamos a la interfaz de inserción de nuevo complemento no salarial:

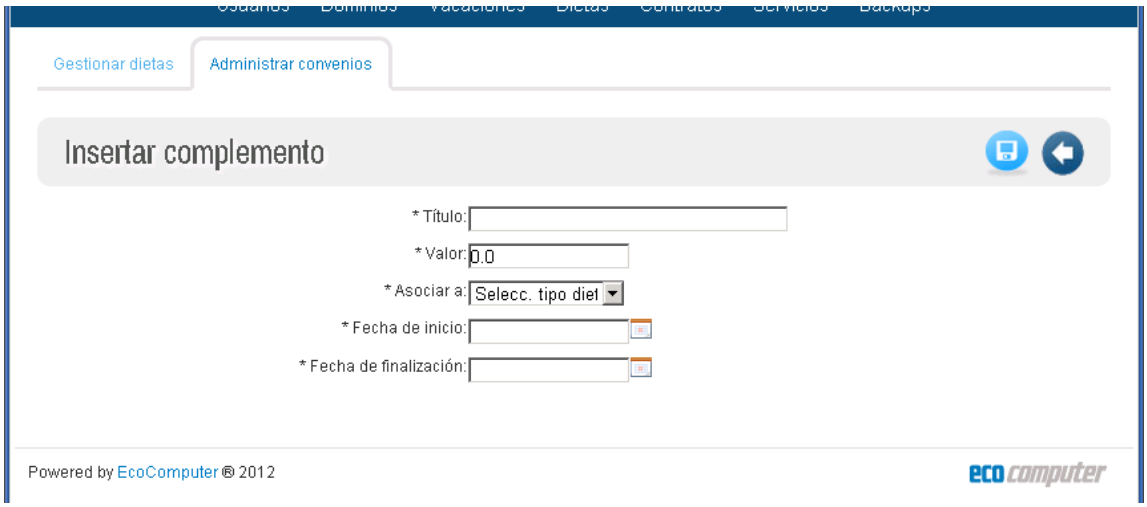


Figura 9.18 - Manual de usuario: insertar complemento no salarial

Una vez rellenados todos los campos, obligatorios, se guarda el complemento pulsando sobre el disquete de la barra de herramientas. Ahora aparecerá en la lista anterior.

9.3.2.2.5 Edición de complemento no salarial

Los complementos no salariales pueden ser editados pulsando sobre el icono de edición correspondiente. Veremos la interfaz de edición con los campos precargados y listos para ser modificados:

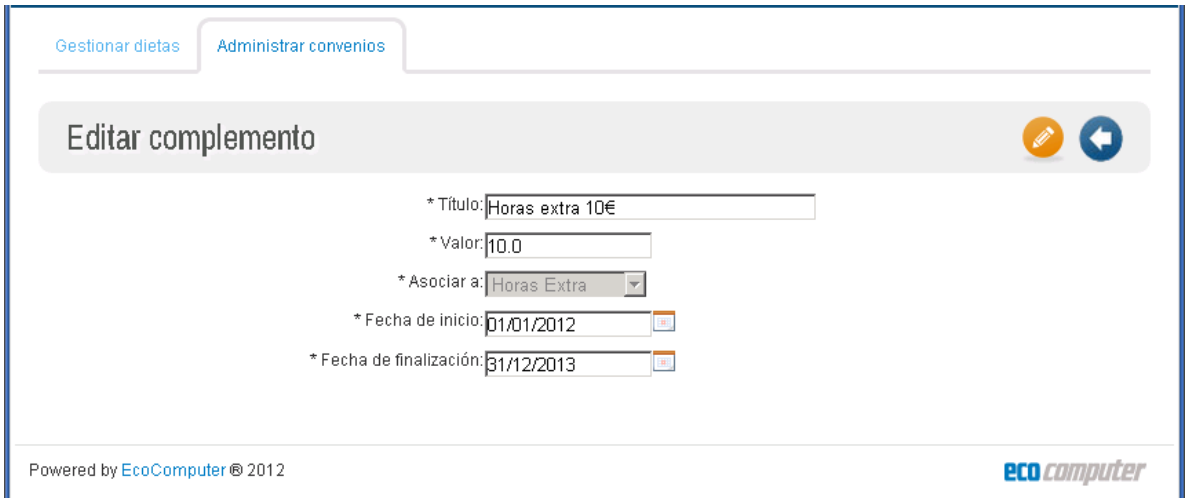


Figura 9.19 - Manual de usuario: editar complemento no salarial

9.3.2.2.6 Eliminación de complemento no salarial

El último de los botones de la derecha de cada complemento no salarial, denotado por una equis blanca sobre fondo rojo, permite eliminarlo del sistema. Basta con presionarlo y aceptar el diálogo de confirmación consecuente para hacerlo.

9.3.3 Módulo de gestión del calendario laboral

Para acceder a este módulo es necesario seleccionar “Vacaciones” en la barra de navegación superior. Para que esto aparezca, el usuario debe tener permisos de escritura y de lectura sobre el módulo. Consúltese con el administrador del sistema si esto no es así.

9.3.3.1 Funcionalidad de usuario

Lo primero que vemos nada más acceder al módulo es el resumen del año en curso de los permisos solicitados, sus datos y estados. Al no haber información dicho contenido aparece vacío:

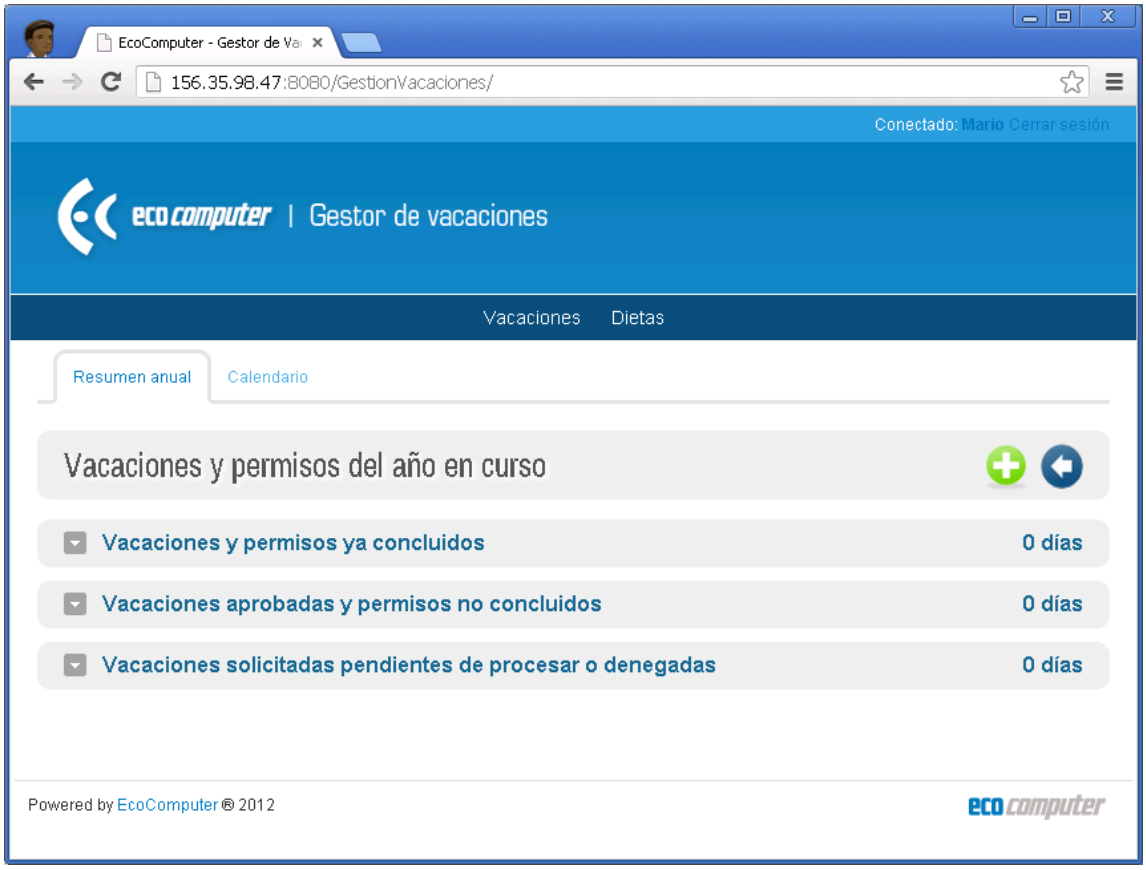


Figura 9.20 - Manual de usuario: resumen de permisos vacío

9.3.3.1.1 Solicitar permiso

De manera análoga al módulo de gestión de dietas anterior, pulsando sobre la cruz blanca sobre fondo verde llegamos a la interfaz de inserción de permisos, en la que todos los campos son obligatorios:

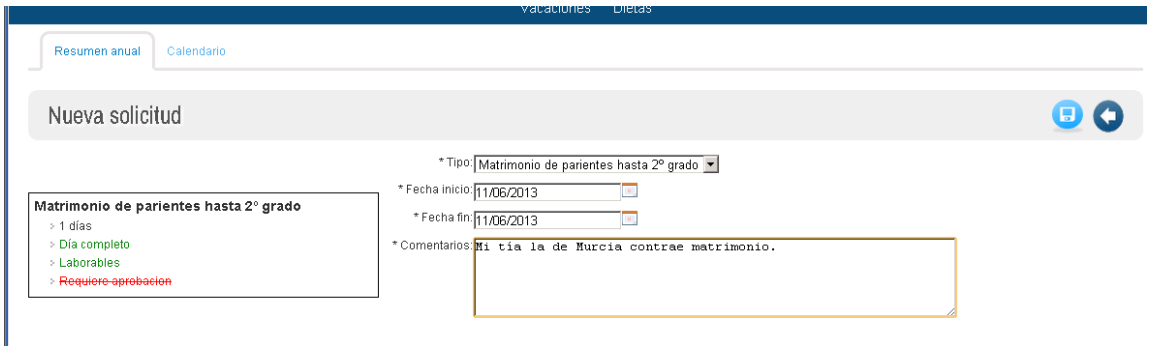


Figura 9.21 - Manual de usuario: insertar permiso

Al seleccionar el tipo de permiso, un cuadro informativo aparece indicando los datos del mismo: nombre, número de días y si es de día completo, laborable y/o requiere aprobación del administrador. Para insertarlo basta con pulsar el botón del disquete ya conocido.

Ahora podremos visualizarlo en el resumen anterior, junto con otros dos ejemplos que hemos añadido:

Figura 9.22 - Resumen de permisos

Podemos ver cómo los permisos y vacaciones se han repartido a lo largo de las tres secciones diferentes: los que ya han terminado se introducen en “Vacaciones y permisos ya concluidos”,

los aprobados aún no disfrutados en "Vacaciones aprobadas y permisos no concluidos", mientras que las solicitudes de vacaciones aún no aprobadas por el administrador, o ya rechazadas, van al apartado "Vacaciones solicitadas pendientes de procesar o denegar".

9.3.3.1.2 Editar permiso

En la botonera derecha de cada permiso, si estos no han concluido ya, tenemos la posibilidad de editarlos pulsando sobre el botón a tal efecto. Esto nos lleva a la pantalla de inserción anterior pero debidamente rellenada y modificada para una mayor facilidad de edición:

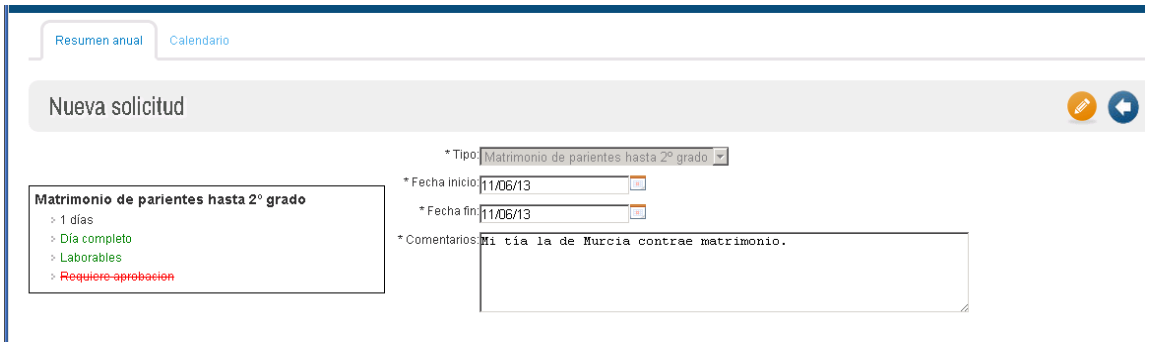


Figura 9.23 - Manual de usuario: editar permiso

Tras realizar los cambios pertinentes, realizamos la modificación tras pulsar el botón denotado por un lápiz blanco sobre fondo naranja.

Téngase en cuenta que si se edita un permiso de vacaciones ya autorizado o rechazado, este volverá al estado inicial, por lo que tendrá que volver a ser revisado por un usuario administrador.

9.3.3.1.3 Eliminar permiso

Además del botón de edición, disponemos del botón de borrado de permiso, si este no ha concluido ya. Pulsarlo hará aparecer un cuadro de diálogo solicitando confirmación y, tras aceptarlo, eliminará el permiso en cuestión.

9.3.3.1.4 Visualizar permisos en calendario

Bajó el menú general de navegación encontramos la pestaña "Calendario", en la que, tras pulsarla, podremos visualizar un calendario tradicional con los permisos relativos al mes y año indicado (por defecto, mes y año en curso):

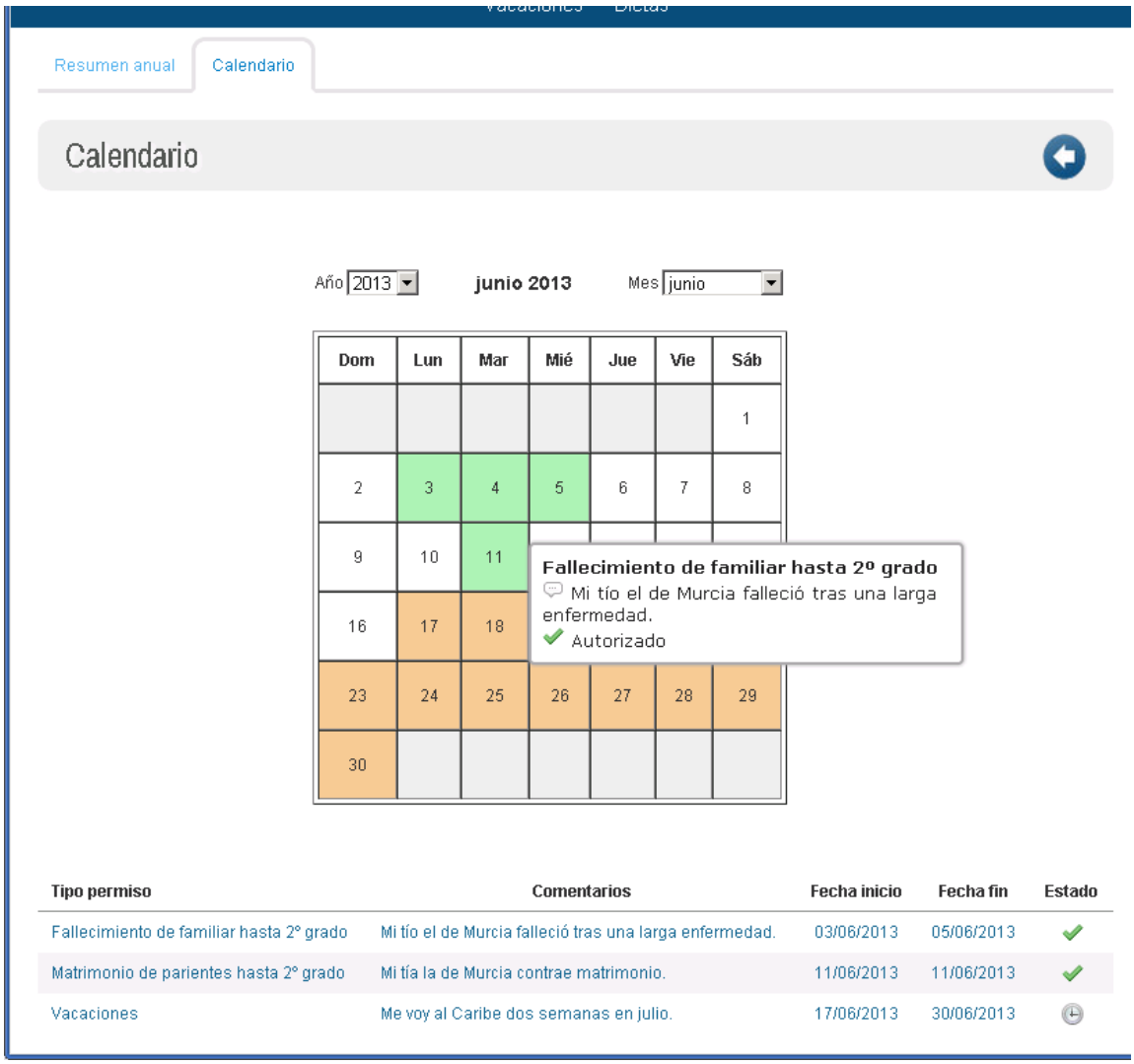


Figura 9.24 - Manual de usuario: calendario de permisos

El calendario marca los días en los que hay un permiso con un fondo de color verde cuando ya ha sido aprobado (o no requiere aprobación), naranja cuando está pendiente de ser aprobado y rojo si ha sido denegado. Además, pasando el puntero del ratón sobre uno de los días sombreados aparece un cuadro de ayuda con información sobre el permiso en cuestión.

Adicionalmente, en la parte inferior del calendario se incluye un listado de los permisos para el mes indicado, sus datos y estado, para que sea más fácil la visualización de contenidos.

Para cambiar de mes y año basta simplemente con pinchar en los desplegables correspondientes y seleccionar el mes y año deseado. La interfaz se recargará sola.

9.3.3.2 Funcionalidad adicional de administrador

Pese a que el administrador es un usuario más del sistema y por tanto puede realizar las mismas tareas que éstos, dispone de tareas extra de más alto nivel que el resto de los usuarios.

9.3.3.2.1 Buscador y filtro de permisos y vacaciones

La pestaña "Buscador" ofrece al administrador la posibilidad de buscar un permiso concreto o filtrar en base a los criterios especificados. Indicando fecha de inicio, fecha de finalización, tipo de permiso, estado del permiso y/o usuario podrá consultar los permisos que se correspondan con dichos criterios. Si alguno de los menús desplegables se deja con el valor por defecto ignorará ese criterio y buscará todos.

En el siguiente ejemplo se visualizan los permisos comprendidos entre el 1 de enero de 2013 y el 31 de diciembre de 2013, para todos los tipos de permisos, estados y usuarios:

Figura 9.25 - Manual de usuario: buscador y filtro de permisos

Podemos ver que todos los datos de los permisos se muestran en la tabla, así como las acciones disponibles para cada uno de ellos. Obsérvese que en relación al estado o conclusión del permiso se pueden realizar unas acciones u otras al igual que se vio en el resumen de permisos y vacaciones del usuario.

9.3.3.2.2 Aprobar permiso

Desde la vista del buscador el administrador dispone de la posibilidad de aprobar un permiso, siempre y cuando este requiera aprobación y esté en estado solicitado. Basta con pulsar el botón a tal efecto en la botonera derecha de cada permiso.

9.3.3.2.3 Rechazar permiso

Desde la vista del buscador el administrador dispone de la posibilidad de rechazar un permiso, siempre y cuando este requiera aprobación y esté en estado solicitado. Basta con pulsar el botón a tal efecto en la botonera derecha de cada permiso.

9.3.3.2.4 Visualizar tipos de permiso

Otra de las pestañas disponibles para el usuario administrador es "Administrar tipos". Pinchando en ella llegamos a la interfaz que los lista todos:

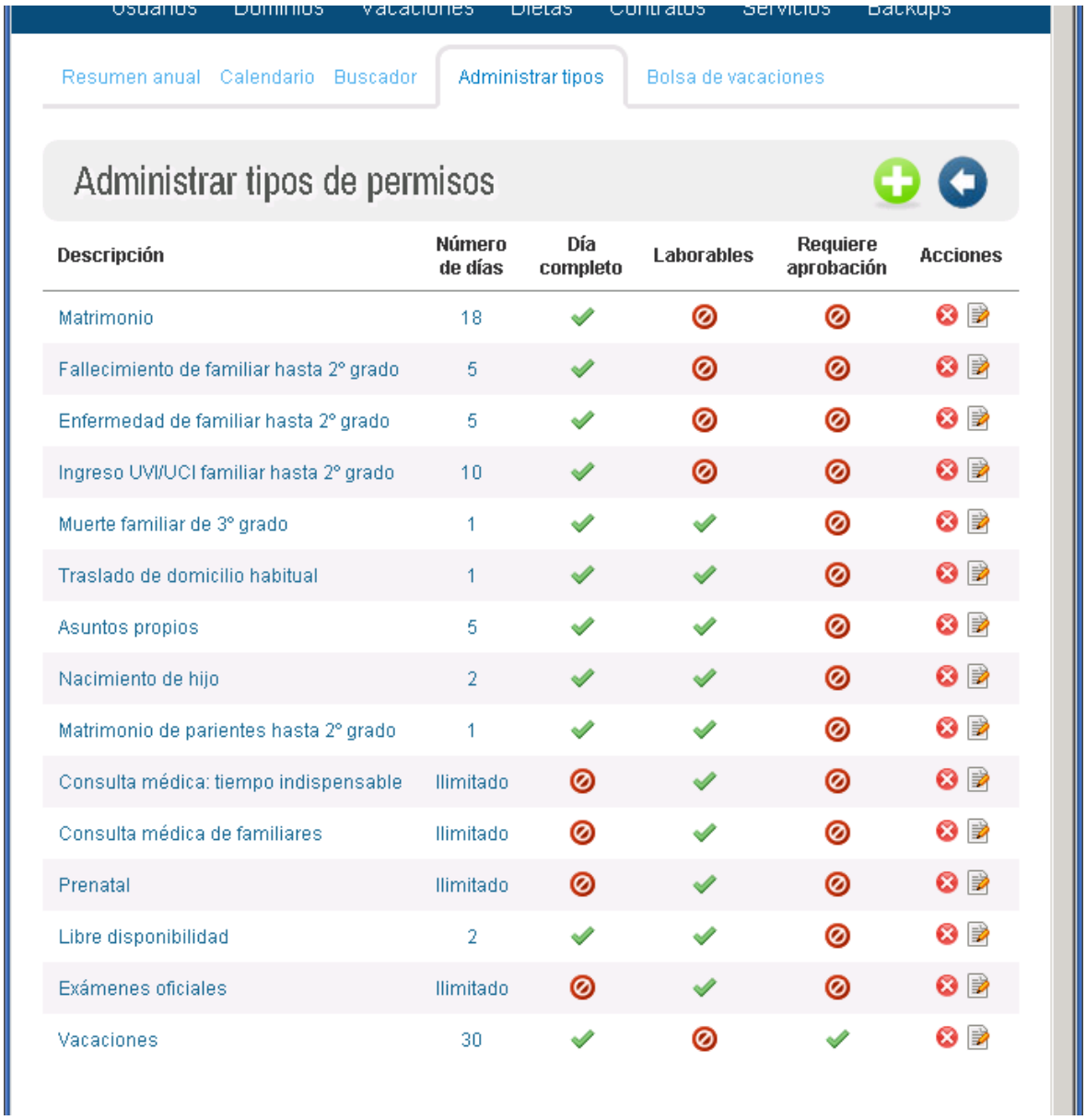


Figura 9.26 - Manual de usuario: tipos de permiso

La distribución de la información viene siendo la habitual, e igualmente cada uno de los tipos de permiso contiene botoneras con las que realizar acciones.

9.3.3.2.5 Insertar nuevo tipo de permiso

Pulsando la cruz habitual llegamos a la interfaz de inserción de tipos de permiso:

Figura 9.27 - Manual de usuario: insertar nuevo tipo de permiso

Deberemos rellenar los campos y seleccionar las características del permiso: si es día completo, laborable y/o requiere aprobación por parte del administrador.

Para guardar el tipo de permiso se debe pulsar en el icono de disquete habitual.

9.3.3.2.6 Editar tipo de permiso

Pulsando el botón de edición de tipo de permiso en la botonera correspondiente del listado de tipos de permisos llegamos a la interfaz de edición, en la que deberemos introducir los cambios que queremos realizar y confirmarlos pulsando el botón del lápiz.

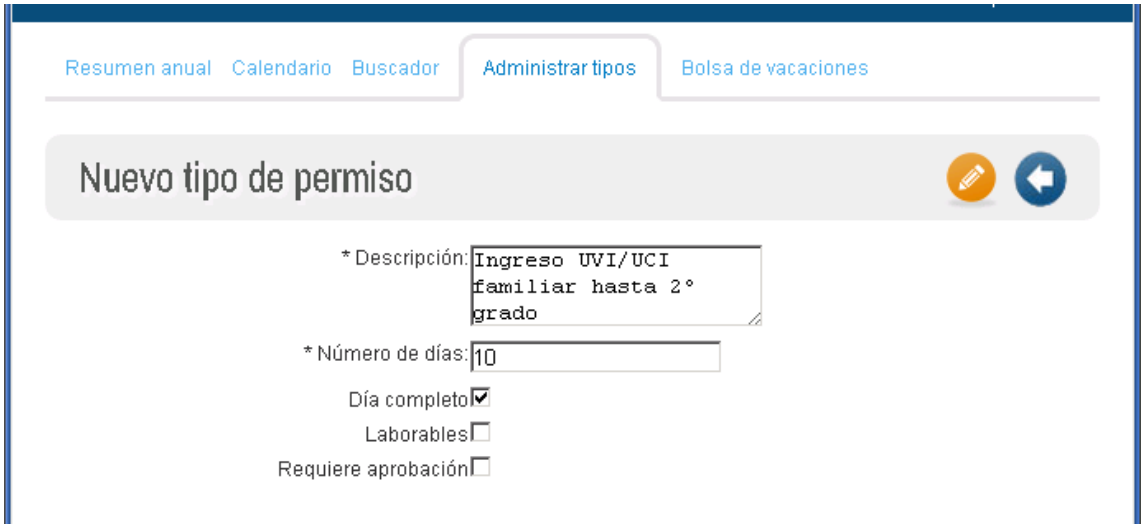


Figura 9.28 - Manual de usuario: editar tipo de permiso

9.3.3.2.7 Eliminar tipo de permiso

Además del botón de edición, disponemos del botón de borrado de tipo de permiso. Pulsarlo hará aparecer un cuadro de diálogo solicitando confirmación y, tras aceptarlo, eliminará el tipo de permiso en cuestión.

9.3.3.2.8 Editar la bolsa de días de vacaciones de los usuarios

La última de las pestañas que el administrador puede visitar es "Bolsa de vacaciones". Al entrar en ella veremos el listado de todos los usuarios de la aplicación con su número de días disponibles:

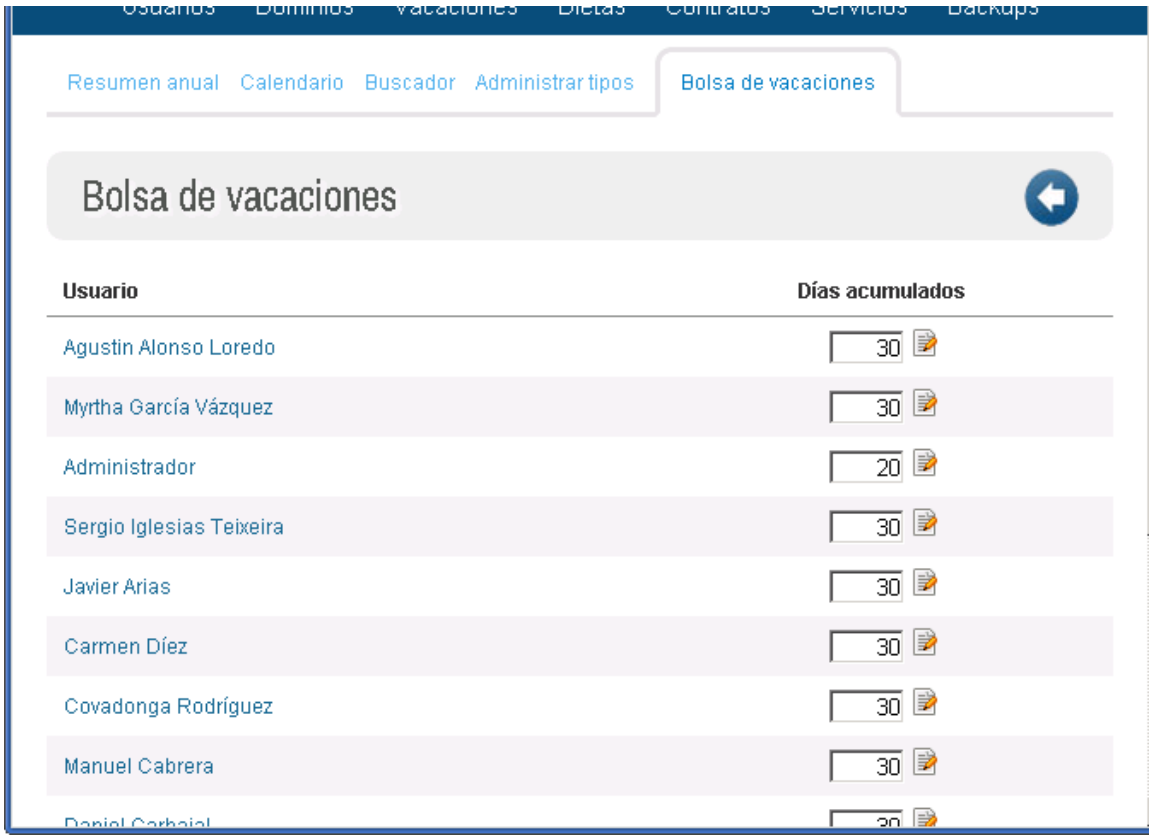


Figura 9.29 - Manual de usuario: bolsa de vacaciones

Podemos observar que el número de días acumulados está denotado por un campo de texto editable. Escribiendo el número de días a los que se quiere actualizar y pulsando el botón de edición adyacente cambiaremos dicho valor.

Considérese que autorizar y eliminar o modificar permisos de tipo vacaciones incrementan y decrementan, respectivamente, el número de días acumulados para el usuario automáticamente. Por tanto no es necesario editar a mano la bolsa de vacaciones cada vez que unas vacaciones se autoricen, modifiquen o eliminen.

9.4 Manual del Programador

En este manual se describen cualquier aspecto que pueda ayudar a otros programadores a ampliar, modificar o entender aspectos de la construcción de nuestra aplicación. No se explica la arquitectura general de una aplicación struts2 o el funcionamiento de Spring, si no aquellos aspectos de quizá mayor dificultad de comprensión.

Como ya hemos ido viendo a lo largo de la documentación, este proyecto y sus dos aplicaciones diferentes están construidos sobre una base común pero independiente: la intranet, una librería común y la base de datos. Esto se transforma en una serie de facilidades para el programador de los módulos, puesto que no tiene que gestionar algunos aspectos importantes. Concretamente:

- La Intranet se encarga de colocar al usuario en sesión; los Action de los módulos pueden recuperar la información del usuario implementando la interfaz SessionAware y utilizando el método `session.get("usuario")`.

- La clase Usuario y asociadas (Perfil, Cliente…) se encuentran en la librería Java ecoclases.jar, incluida dentro del directorio de librerías de Apache Tomcat.

- La librería también contiene lo necesario para gestionar el log o registro de la aplicación. Para ello basta con declarar la variable `Logger log = new Logger(this.getClass().getName(), session)`. Escribir entradas del registro se hace mediante los métodos `log.info()`, `log.error()` y `log.warning()`, recibiendo como parámetro un String con el mensaje a escribir en la base de datos.

- También se encarga de la comunicación con la base de datos, realizada mediante un *pool de conexiones* para gestionar mejor los recursos y reducir tiempos de espera innecesarios. La conexión con la base de datos se establece declarando `Connection con = ConnectionManager.getConnection()`.

- Para crear un nuevo elemento del modelo se recomienda seguir la misma jerarquía de clases e interfaces vista en apartados anteriores (ManagerService, DataService, DAO…). Recuérdese declarar la inyección de dependencias necesarias en el fichero applicationContext.xml.

- Algunos Action tienen tipos de retorno especiales. Tal es el caso, por ejemplo, de ObtenerDatosPermisoAction, que se invoca tras una llamada AJAX desde nuevaSolicitud.jsp y que retorna un objeto JSON para obtener nueva información sin necesidad de recargar toda la interfaz. ImageAction, que se encarga de devolver la imagen de dieta asociada, devuelve un objeto CustomImageBytesResult. Ambas definiciones de “result-types” pueden encontrarse en struts.xml.

- Las JavaServer Pages se componen mediante el uso de “tiles”, unas librerías Java que se encargan de crear la interfaz como si de un árbol se tratase, compartiendo atributos de padres a hijos para ahorrar código. Veamos esto en un ejemplo:

- El fichero struts.xml llama a la ejecución de un Action cuyo resultado apunta a un "tiles" (por ejemplo, `<result type="tiles">Buscador.tiles</result>`).

- La definición de dicho resultado se encuentra en el fichero tiles.xml, que declara sus propios atributos y a su vez los hereda de su padre:

```
<definition name="ModuloVacaciones" extends="AccesoLayout">
        <put-attribute name="title"
                       value="EcoComputer - Gestor de Vacaciones" />
        <put-attribute name="logo" value="images/logo_vacaciones.png" />
        <put-attribute name="tabs"
                       value="/WEB-INF/tiles/tabs/TabsGestionVacaciones.jsp" />
</definition>

<definition name="Buscador.tiles" extends="ModuloVacaciones">
        <put-attribute name="toolbar"
                       value="/WEB-INF/tiles/toolbars/ToolbarBuscador.jsp" />
        <put-attribute name="content"
                       value="/WEB-INF/pages/vacationsManagement/buscador.jsp" />
</definition>
```

- Este ejemplo muestra la definición de "Buscador.tiles" en el que declara la localización de sus elementos "toolbar" y "content". A su vez, al extender de "ModuloVacaciones", recibe de él los elementos "title", "logo" y "tabs". Este módulo a su vez recibe otros atributos de "AccesoLayout", puesto que extiende de él.

Capítulo 10. Conclusiones y Ampliaciones

10.1 Conclusiones

El objetivo principal de este proyecto ha sido alcanzado: realizar una aplicación que ayude a una empresa a gestionar ciertos aspectos de su día a día tales como las dietas o el calendario laboral de sus empleados.

El realizar un proyecto para un cliente real fue otro de los aspectos que se persiguieron en la ejecución del mismo. Especificación de requisitos, reuniones de seguimiento, cambios imprevistos de funcionalidad… Todo lo relacionado con el trato con un cliente sucedió. Y aunque en algunos casos resultase más inconveniente que ventaja, se esperaba que esto sucediese para intentar hacer de este proyecto un caso lo más parecido posible a la realidad del mercado laboral actual.

Atarse a los deseos y necesidades de un cliente tiene también sus inconvenientes, principalmente centrados en no poder aplicar lo que se quiere y debe en ciertos aspectos del desarrollo. Por ejemplo, todo lo relacionado con el diseño de la interfaz de usuario y sus aspectos de usabilidad y accesibilidad fueron eliminados de las responsabilidades del programador, puesto que era uno de los requisitos del cliente. De no haber sido así los resultados podrían haber sido muy diferentes, para bien o para mal, pero al menos bajo los deseos del programador. Aunque, como ya se ha mencionado, con este proyecto se buscaba satisfacer las necesidades de un cliente y no las del desarrollador.

Si bien es cierto que algunos aspectos del planteamiento inicial se han visto mermados o eliminados, también cabe destacar la labor colaborativa con el resto de participantes para mejorar la experiencia de usuario. La implantación de la intranet y entorno de trabajo común no era uno de los requisitos iniciales del cliente, pero su implantación solo aporta ventajas a desarrolladores y usuarios. El haber logrado esto satisfactoriamente se considera, sin ningún tipo de duda, un logro.

No obstante, debido a este y otros aspectos ya comentados en la planificación del proyecto, los plazos manejados en un principio no fueron muy acordes a la realidad, lo que conllevó a retrasos indeseados. El no haber sabido evitar o acotar mejor el impacto de estos riesgos es uno de los aspectos que se consideran, al menos parcialmente, un fracaso.

En términos generales, es una satisfacción haber realizado un proyecto de esta envergadura, investigando tecnologías nuevas y utilizando otras que ya se consideraban atractivas. El trato con el cliente, la colaboración con otros equipos de trabajo, el interés personal en la tecnología utilizada… Todo ello hace que se termine el desarrollo de esta etapa con un alto nivel de satisfacción del trabajo logrado.

10.2 Ampliaciones

10.2.1 Usabilidad y accesibilidad

Como ya se ha comentado numerosas veces, adaptarse a los requisitos del cliente era la prioridad. Uno de ellos era el diseño de las interfaces de usuario, proporcionadas por él y a las que había que adaptarse. Es por eso que se ha dejado en segundo plano cumplir con los estándares y normas dictados establecidos y, por tanto, no poder garantizar la usabilidad y accesibilidad de la aplicación.

Una de las posibles ampliaciones pasa por subsanar esta carencia del proyecto y rediseñar las interfaces de manera que cumpla con las normas.

10.2.2 Lógica de grupos de trabajo

Uno de los requisitos iniciales de la aplicación que se descartó por falta de tiempo fue la incorporación de la lógica de grupos de trabajo. Los usuarios de la aplicación pueden pertenecer a uno o varios grupos de trabajo y, aunque esta funcionalidad está disponible en el módulo de gestión de usuarios, no se encuentra desarrollada en el de gestión de dietas.

La idea se basa en implementar un sistema de avisos para que cuando un usuario vaya a insertar un permiso o vacaciones, el sistema notifique al usuario si alguno de los miembros de sus grupos de trabajo también ha realizado una petición. De esta manera puede evitarse que un grupo de trabajo no se vea demasiado afectado por la falta de sus miembros.

10.2.3 Lógica de jefe de departamento

Directamente relacionado con los grupos de trabajo existía la figura del jefe de departamento, que tendría las funciones de administrador únicamente sobre los miembros de su grupo de trabajo.

De esta manera un jefe de departamento podría aprobar, denegar, editar o simplemente visualizar las dietas o permisos de los integrantes del grupo de trabajo del que sea responsable, en lugar de necesitar ser administrador completo del sistema para hacerlo.

10.2.4 Manejo de festivos

La aplicación de gestión de dietas no permite marcar permisos de tipo “laborable” en fines de semana, puesto que son permisos solo seleccionables de lunes a viernes. Pese a esto, el concepto de “no laborable” también abarca a festivos nacionales y locales, por lo que debería ser imposible, por ejemplo, solicitar un permiso el día de Navidad.

Para solucionar esta carencia se plantea la posibilidad de comunicarse con alguna API pública que especifique si un día determinado es laborable o no en la localidad donde tenga sede la empresa, para impedir que los usuarios soliciten permisos dichos días.

10.2.5 Aplicación para Android

Otra de las ideas iniciales que se descartó fue el diseño de la aplicación bajo la plataforma móvil Android. Pese a que la aplicación fue diseñada para funcionar bajo cualquier tipo de navegador, no todos son capaces de realizar todas las funciones. Android no es una excepción.

Es por esto que podría resultar interesante realizar una versión simplificada de la aplicación para dicha plataforma, de manera que los usuarios pudiesen hacer uso de la misma a través de su dispositivo móvil y no tener que recurrir a un equipo de escritorio.

Capítulo 11. Presupuesto

11.1 Desarrollo de Presupuesto Simplificado

Debido a que el proyecto se ha realizado con y para una empresa de desarrollo informático, se ha creído conveniente realizar el presupuesto de la manera más aproximada posible a los estándares de dicha empresa. Es por eso que se han sus precios establecidos a la hora de desarrollar el presupuesto.

Considérese que la empresa no realiza distinciones entre las diferentes fases del trabajo realizado, si no que el precio por hora es el mismo para todos. Igualmente, en dicho precio se consideran todos los gastos directos e indirectos que el empleado ocasiona durante sus horas de trabajo (sueldo, seguro médico, agua, luz, mantenimiento, uso de los equipos y licencias de software…).

Concepto	Cantidad	Precio unitario	TOTAL
Horas de desarrollo web	436	39,00 €	17.004,00 €
		Subtotal	17.004,00 €
		I.V.A. (21%)	3.570,84 €
		TOTAL	**20.574,84 €**

Capítulo 12. Referencias Bibliográficas

12.1 Referencias en Internet

12.1.1 Documentaciones oficiales

[Oracle] "Java™ Platform, Standard Edition 7". http://docs.oracle.com/javase/7/docs/api/. 2013.

[W3C] "HTML 4.01 Specification". http://www.w3.org/TR/html401/. 1999.

[W3C] "Cascading Style Sheets home page". http://www.w3.org/Style/CSS/. 2013.

[Oracle] "JavaServer Pages API Documentation". http://docs.oracle.com/cd/E17802_01/products/products/jsp/2.1/docs/jsp-2_1-pfd2/. 2005.

[Microsoft MSDN] "Transact-SQL Reference (Database Engine)". http://msdn.microsoft.com/en-us/library/bb510741.aspx. 2007.

[The Apache Software Foundation] "Apache Tomcat 6.0 documentation". http://tomcat.apache.org/tomcat-6.0-doc/index.html. 2013.

[The Apache Software Foundation] "Apache Struts 2 Documentation". http://struts.apache.org/release/2.2.x/docs/home.html. 2011.

[The Apache Software Foundation] "Tiles API". http://tiles.apache.org/framework/apidocs/index.html. 2012

[iText] "iText, a Free Java-PDF library 5.4.2 API". http://api.itextpdf.com/itext/. 2013.

[Joda.org] "Joda time 2.2 API". http://joda-time.sourceforge.net/apidocs/index.html. 2013.

[jQuery] "jQuery API Documentation". http://api.jquery.com/. 2013.

[jQueryUI] "jQuery UI 1.9 API Documentation". http://api.jqueryui.com/1.9/. 2013.

12.1.2 Otras documentaciones no oficiales

[W3Schools] "HTML". http://www.w3schools.com/html/. 2013.

[W3Schools] "CSS". http://www.w3schools.com/css/. 2013.

[W3Schools] "JavaScript". http://www.w3schools.com/js/. 2013.

[W3Schools] "SQL". http://www.w3schools.com/sql/. 2013.

[W3Schools] "jQuery". http://www.w3schools.com/jquery/default.asp. 2013.

12.1.3 Tutoriales y guías varias

[RoseIndia] "Struts 2 Tutorial". http://www.roseindia.net/struts/struts2/. 2008.

[Mkyong] "Struts 2 Tutorial". http://www.mkyong.com/tutorials/struts-2-tutorials/. 2010.

[DZone] "Struts 2 Tutorials". http://www.dzone.com/tutorials/java/struts-2. 2012.

[viralpatel] "Struts 2 Tutorials, Tips & Tricks". http://viralpatel.net/blogs/category/j2ee/struts-2/. 2009.

12.1.4 Foros de ayuda

[stackoverflow] "Stack Overflow". http://stackoverflow.com/. 2013.

[CodeRanch] "Java Forums at the Big Moose Saloon". http://www.coderanch.com/forums. 2013.

Capítulo 13. Apéndices

13.1 Glosario

- **Bolsa [de días de vacaciones]:** Los días de vacaciones de los usuarios no se gastan si no se utilizan, si no que se van acumulando año a año para que cada empleado pueda organizarse las vacaciones como más conveniente le venga. Es por esto que es necesario tener una bolsa de días de vacaciones donde controlar cuántos días tiene aún sin utilizar cada uno de los empleados, así como editar el valor por parte del administrador.

- **Complemento no salarial:** Según el convenio laboral de la empresa, algunos gastos relacionados con dietas están supeditados a un complemento no salarial, o lo que es lo mismo, su equivalencia monetaria. Estos complementos no salariales, que tienen fecha de inicio y finalización para verificar su vigencia, actualmente están definidos como:

    - **Desplazamiento 2013:** Cada kilómetro realizado para las dietas tipo "desplazamiento" supone un importe de 0,31€.

    - **Dieta completa 2013:** Cualquier dieta catalogada como "dieta completa" supone un gasto de 45,33€, independientemente de su cuantía real.

    - **Horas extra 10€:** Cada hora extra realizada bajo este tipo de dieta supone un importe de 10€.

    - **Horas extra 20€:** Cada hora extra realizada bajo este tipo de dieta supone un importe de 20€.

    - **Horas extra 30€:** Cada hora extra realizada bajo este tipo de dieta supone un importe de 30€.

    - **Media Dieta 2013:** Cualquier dieta catalogada como "media dieta" supone un gasto de 16,59€, independientemente de su cuantía real.

- **Convenio [laboral]:** En algunas ocasiones se utiliza el término "convenio" para hacer referencia al conjunto de complementos no salariales, puesto que es quien las dicta.

- **Detalle [de dieta]:** Se refiere a cada entrada independiente de una dieta concreta. Cada detalle puede ser de un tipo, tener asociado un complemento, un valor monetario concreto, sus propios comentarios…

- **Dieta:** Agrupa a un conjunto de uno o más detalles de dieta. Es quien posee el estado, la fecha, los totales de los detalles… En ocasiones y por cuestiones prácticas se utiliza indistitantemente el término "dieta" para hacer referencia tanto a la dieta como a sus detalles.

- **Permiso:** Se trata de la solicitud realizada para marcar días no laborables, ya sean permisos (matrimonio, enfermedad de familiar…) como vacaciones. Entre sus propiedades se destaca la posibilidad de que sean:
    - **Día completo:** Esta característica denota a aquellos permisos que por su naturaleza suponen todo el día. Está directamente asociado con el concepto de "permiso ilimitado".
    - **Ilimitado:** Cuando un permiso no es de día completo si no de unas horas determinadas (por ejemplo, visitas al médico), el permiso se considera ilimitado. El número de días permitidos para este caso es "-1".
    - **Laborable:** Hace referencia a los permisos que deben solicitarse únicamente en días laborables, de lunes a viernes no festivos.
    - **Requiere aprobación:** Los permisos marcados con esta característica requieren supervisión de un administrador para ser aprobados o denegados. Si desactivado, los permisos pasan directamente a estar autorizados.
- **Tipo de dieta:** Son los diferentes motivos que generan gastos relacionados con dietas en la empresa. Pueden ser medias dietas, dietas completas, parking, desplazamiento, otros y horas extra. Pueden llevar o no uno o varios complementos no salariales asociados.
- **Tipo de permiso:** Son los diferentes motivos que generan la necesidad de solicitar un permiso laboral (matrimonio, fallecimiento o enfermedad de familiar, asuntos propios, vacaciones…).

13.2 Contenido Entregado en el CD-ROM

Directorio	Contenido
./	Contiene un fichero leeme.txt explicando esta misma estructura.
./Portal del empleado	Contiene toda la estructura de directorios del proyecto desarrollado.
./Instalación	Ficheros utilizados para la instalación del proyecto.
./Documentación	Contiene toda la documentación asociada al proyecto.
./Documentación/img	Directorio que contiene las imágenes utilizadas en la documentación.
./Documentación/uml	Directorio que contiene los ficheros de diagramas del proyecto.
./Herramientas	Contiene los ficheros de instalación de las herramientas utilizadas para el desarrollo o puesta en marcha del proyecto.
./Herramientas/Desarrollo	Ficheros de instalación de las herramientas utilizadas en el desarrollo.
./Herramientas/Explotación	BD, servidor Web y herramientas en general.