Roo++ – Spring Roo Add-On

Manual del Usuario

Autor: Javier Menéndez Álvarez

Contenido

Introducción

Roo++ es un add-on que proporciona una serie de comandos adicionales a la plataforma de desarrollo de aplicaciones en Java, Spring Roo. Por tanto, no proporciona comandos para tareas tales como la creación de proyectos, puesto que estos comandos ya forman parte de Roo, y Roo++ lo tiene en cuenta (tomando, por ejemplo, el nombre del proyecto que se ha introducido desde el comando de Roo correspondiente).

El objetivo fundamental de Roo++ es permitir el desarrollo rápido de aplicaciones web complejas, multimódulo, que responden a una serie de características típicas. Dentro de ese tipo de aplicación, con una serie de módulos, Roo++ proporciona cierta flexibilidad: usar o no autenticación de usuarios (pudiendo elegir entre varios tipos), automatizar el uso de una base de datos (dentro de una serie de sistemas de gestión de bases de datos), etc.

Sin embargo, su objetivo principal no es proporcionar la mayor flexibilidad posible, ya que ésta ya la proporciona Roo. El principal cometido de Roo++ es permitir el desarrollo de aplicaciones de cierta complejidad, llevando a cabo todas las tareas de configuración necesarias (algunas de ellas muy complejas) en sólo un instante, ahorrando gran cantidad de tiempo, y de posibilidades de cometer errores.

Si la aplicación a desarrollar no se ajusta a estas características, o si se le quiere dar otro enfoque, o corregir errores, una vez comenzado su desarrollo, siempre sigue disponible la flexibilidad de los comandos de Roo, y la posibilidad de editar manualmente los ficheros del proyecto en desarrollo.

El uso de un conjunto amplio de archivos de configuración permite ajustar las características de las aplicaciones desarrolladas con Roo++ a las necesidades de la organización, y al momento de su desarrollo (por ejemplo, ajustar la versión del sistema de gestión de bases de datos a la más reciente existente en este momento, o a la más estable disponible, etc.). De esta forma, el ajuste de una serie de archivos (cuyo propósito está documentado, línea por línea) permite aplicar el mismo patrón a gran cantidad de pequeñas ubicaciones dentro del proyecto en desarrollo. Sin Roo++, no sólo el desarrollo de la aplicación sería mucho más largo, sino que, además, dentro de una organización, en muchos casos gran parte del desarrollo de cada proyecto consistiría en repetir pasos, complejos y propensos a errores, ya dados en el desarrollo de otros proyectos similares.

Este manual está dirigido a personas con conocimientos avanzados de programación en Java, y también con conocimientos sobre Spring Roo y Maven.

Instalación

Veamos ahora los diferentes pasos que se han de seguir para instalar el add-on Roo++ para Spring Roo.

En primer lugar, es necesario tener instalado en el sistema el siguiente software (cada uno de ellos aparece con un enlace a la página web desde la que se puede descargar; Roo++ puede ser utilizado en cualquier máquina y sistema operativo en que todo este software haya sido instalado; el software debe instalarse en el orden indicado):

1. JDK de Java EE (recomendado) o Java SE, implementación de Sun Microsystems/Oracle (recomendado), u otra compatible. Requerido Java 6 o posterior, recomendado Java 7. **Es necesario el JDK, no sólo el JRE.**
2. Apache Maven (recomendado elegir la versión estable más reciente; **instalar sólo si no se va a utilizar Spring Tool Suite, o si se desea usar una versión diferente a la que ésta proporciona).**
3. Spring Roo (recomendado elegir la versión estable más reciente; **instalar sólo si no se va a utilizar Spring Tool Suite, o si se desea usar una versión diferente a la que ésta proporciona).**
4. Spring Tool Suite. Alternativamente, si Eclipse ya se encuentra instalado en el sistema, se puede optar por añadir Spring Tool Suite a la instalación existente de Eclipse, buscando "spring tool suite" en Eclipse Marketplace, y descargando e instalando la versión que se ajuste a la versión de Eclipse utilizada.
5. Un sistema de gestión de bases de datos, a elegir entre los siguientes: MySQL, PostgreSQL, Oracle, o Microsoft SQL Server. Sólo estos cuatro funcionarán con Roo++ sin necesidad de hacer ajustes adicionales manualmente. Tener en cuenta que en las dos últimas opciones es necesario el pago de la licencia si se desea hacer uso de una versión estable con toda su funcionalidad.

NOTA 1: *la instalación de Spring Tool Suite actualmente incluye Spring Roo y Apache Maven; si en un futuro dejara de incluir alguno de ellos podría ser necesario llevar a cabo los pasos 2 y/o 3 a pesar de usar Spring Tool Suite.*

NOTA 2: *si futuras versiones de Spring Roo (sean descargadas directamente, o a través de Spring Tool Suite) fueran incompatibles con Roo++, y no fuera posible solucionar estas incompatibilidades, usar la versión compatible más reciente, teniendo en cuenta que esta versión de Roo++ fue diseñada para Spring Roo versión 1.2.3.*

NOTA 3: *si el sistema de gestión de bases de datos utilizado es MySQL, es necesario cambiar, en la aplicación MySQL Workbench (instalarla si no está incluida en la versión de MySQL instalada), el motor por defecto, de forma que éste sea InnoDB (ir a "Server Administration" -> "Configuration" -> "Options File", y en la pestaña "General" habilitar la casilla "default-storage-engine", verificando que el valor de este campo sea "InnoDB", "INNODB" o similar, además de establecer a "ON" el campo "innodb" en la pestaña de nombre "InnoDB"; recordar guardar los cambios; antes de hacer todo esto puede ser necesario crear una "New Server Instance" desde la pantalla principal de MySQL Workbench).*

Una vez todo este software se encuentra instalado en el sistema, deben llevarse a cabo los siguientes pasos, en el orden indicado:

1. Configurar en el diálogo de variables de entorno de Windows, o en el fichero *.bashrc* o equivalente de Unix/Linux, la variable de entorno *Path* o *$PATH*, respectivamente, para que de forma permanente incluya los siguientes directorios (en caso de que no estén ya incluidos tras la instalación del software asociado; en sistemas Linux varios de estos directorios pueden ser el mismo, si el software requerido ha sido instalado a través del sistema de gestión de paquetes de la distribución, y en ese caso el número de directorios a añadir sería menor; ver la jerarquía de directorios estándar de Linux):
    - Ruta del directorio *bin* en el que se encuentran los ficheros ejecutables *java* y *javac* de la JDK instalada.
    - Ruta del directorio *bin* en el que se encuentran los scripts ejecutables *mvn*, *mvn.bat*, *mvnDebug* y *mvnDebug.bat* de Apache Maven.
    - Ruta del directorio *bin* en el que se encuentran los scripts ejecutables *roo.sh* y *roo.bat* de Spring Roo.
2. Situar el archivo *rooplusplus.roo.addon-1.0.jar* en el lugar elegido para su ubicación definitiva.
3. Situar el directorio *RooAddonsData*, con todo su contenido, en el lugar elegido para su ubicación definitiva (salvo algún motivo especial, tal como su compartición por varios usuarios del sistema, su ubicación recomendada es la raíz del directorio personal del usuario).
4. **Sólo si se ha elegido otra ubicación para *RooAddonsData* que no sea la recomendada**, crear, del mismo modo que se ha hecho para modificar el *path* en el punto 1), la nueva variable de entorno *ROOADDON* (*%ROOADDON%* en Windows [si se configura desde el diálogo de variables de entorno, no se indican los *%*] o *$ROOADDON* en Unix), y asignarle como valor la ruta completa de la ubicación seleccionada, acabada en el carácter */*.
5. Ajustar todos los valores de los diferentes archivos de configuración situados en *RooAddonsData* para que se ajusten a las necesidades de la organización y del tipo de aplicaciones web que se van a desarrollar con Spring Roo y Roo++.
6. Abrir una ventana de línea de comandos e introducir el comando *roo*, en Windows, o *roo.sh* en Unix/Linux. Si todo se ha hecho de la forma correcta se deberá entrar en Spring Roo.
7. Introducir los siguientes comandos en Spring Roo (por orden):
    - *download accept terms of use*
    - *osgi install --url file:///ruta/correspondiente/ rooplusplus.roo.addon-1.0.jar*
    - *osgi start --url file:///ruta/correspondiente/ rooplusplus.roo.addon-1.0.jar*
    - *quit*

    De esta forma se permite la descarga de dependencias en Roo, y se instala e inicializa el add-on Roo++.

8. En el archivo *conf/settings.xml*, situado en el directorio en que se encuentre instalada la versión de Maven a utilizar, introducir el siguiente texto (dentro siempre del nodo

XML *settings/profiles* que ya viene en la versión por defecto de este archivo; puede ser necesario descomentar código XML preexistente):

```
   <profile>
      <id>rooaddon</id>

      <activation>
        <activeByDefault>true</activeByDefault>
      </activation>

      <repositories>
      </repositories>
      <pluginRepositories>
         <pluginRepository>
                <releases>
                    <enabled>true</enabled>
                </releases>
                <snapshots>
                    <enabled>true</enabled>
                </snapshots>
                <id>central</id>
                <url>http://repo1.maven.org/maven2</url>
            </pluginRepository>
            <pluginRepository>
                <releases>
                    <enabled>true</enabled>
                </releases>
                <snapshots>
                    <enabled>true</enabled>
                </snapshots>
                <id>sun</id>
                <url>http://java.sun.com/products/java-
media/jai</url>
            </pluginRepository>
        <pluginRepository>
          <id>docbkx.snapshots</id>
          <name>Maven Plugin Snapshots</name>
          <url>http://docbkx-
tools.sourceforge.net/repository/</url>
          <releases>
            <enabled>true</enabled>
          </releases>
          <snapshots>
            <enabled>false</enabled>
          </snapshots>
        </pluginRepository>
      </pluginRepositories>
    </profile>
```

Si todo ha ido bien, en este momento Spring Roo y su add-on Roo++ se encuentran correctamente instalados y configurados en el sistema, y pueden comenzar a ser utilizados.

Compilación de Roo++

Si Roo++ ha de ser compilado, debe hacerse desde una ventana de línea de comandos, situándose en el directorio *roo++-roo-addon*, y ejecutando el comando:

mvn clean package

Debido a la presencia de archivos *pom.xml* dentro del directorio de resources del proyecto, **nunca** se debe tratar de compilar Roo++ desde Spring Roo (comando *perform package*), Eclipse o Spring Tool Suite.

Actualización de Roo++

Para actualizar Roo++ a una versión más nueva, introducir, por orden, las siguientes órdenes en la línea de comandos de Roo (sustituyendo *numeroDeVersion* por su valor correspondiente):

- *osgi uninstall --bundleSymbolicName rooplusplus.roo.addon*
- *osgi install --url file:///ruta/correspondiente/ rooplusplus.roo.addon-numeroDeVersion.jar*
- *osgi start --url file:///ruta/correspondiente/ rooplusplus.roo.addon-numeroDeVersion.jar*

Desinstalación de Roo++

Si es necesario desinstalar Roo++, introducir la siguiente orden en la línea de comandos de Roo:

osgi uninstall --bundleSymbolicName rooplusplus.roo.addon

Uso

Si Spring Roo, su add-on Roo++, y el resto de software necesario se encuentran instalados correctamente, se puede entrar a Roo desde una ventana de línea de comandos, a través del comando ***roo***, en Windows, o ***roo.sh*** en Unix/Linux, después de haberse situado previamente en el directorio raíz del proyecto a desarrollar (es **altamente recomendable** que cada proyecto se desarrolle dentro de un directorio dedicado exclusivamente al mismo, en el cual hay que situarse con el comando *cd* y a continuación lanzar el comando *roo*). Una vez dentro de Roo, puede ser ejecutado cualquiera de sus comandos, así como cualquiera de los comandos de Roo++, facilitando de esta forma el desarrollo de la aplicación. **Para ejecutar cualquier comando de Roo++ siempre es necesario haber ejecutado Roo desde el directorio raíz del proyecto, y que, dentro del mismo Roo, el foco se encuentre en dicho directorio.**

Si se quiere tener una sugerencia sobre un posible orden de ejecución de estos comandos, ver el siguiente ejemplo.

Una vez definida la base de datos a utilizar (si se usa alguna, que será lo habitual en una aplicación del tipo de las desarrolladas con Roo++), con el comando ***roo++ project web jpa***, es necesario introducir en la misma las tablas y sus campos, y quizás también una carga de datos inicial. Ver el siguiente apartado para llevar a cabo este proceso.

Para el desarrollo de la aplicación propiamente dicho (escribir código Java, crear nuevas clases o introducir nuevos resources en el proyecto), es **altamente recomendable** el uso de Spring Tool Suite. Para poder hacer uso de este entorno de desarrollo, se debe antes ejecutar el siguiente comando desde la interfaz de comandos del sistema (es decir, fuera de Roo), situándose en el directorio raíz del proyecto en desarrollo:

mvn eclipse:clean eclipse:eclipse

Para compilar y empaquetar el proyecto una vez desarrollado (en todo o en parte), incluyendo todos y cada uno de sus módulos, y la resolución de sus dependencias, se ha de ejecutar el siguiente comando desde en el directorio raíz del proyecto en desarrollo:

mvn clean install -Dmaven.test.skip=true

Si lo que se desea es hacer pruebas con la interfaz web de la aplicación desarrollada, se debe, en primer lugar, haber ejecutado los comandos ***roo++ project web configuration*** y ***roo++ project web deploy*** (dentro de Spring Roo), y tras ellos ***mvn clean install -Dmaven.test.skip=true*** desde la interfaz de comandos del sistema operativo (fuera de Spring Roo), y, finalmente, desde esta misma interfaz, y situándose en el directorio raíz del módulo web del proyecto desarrollado, ejecutar:

mvn clean tomcat:run

o bien:

mvn clean jetty:run

según se quiera usar para las pruebas el servidor web Tomcat o el Jetty.

Finalmente, cuando ya ha concluido el proceso de desarrollo de la aplicación, esta puede ser distribuida (en forma de un archivo EAR, que se puede desplegar en un servidor web como cualquiera de los dos mencionados), junto con su documentación y definición de la base de datos utilizada, en el archivo ZIP contenido en el directorio target del módulo ***package*** del proyecto.

NOTA: *los usuarios avanzados de Spring Roo han de evitar en la medida de lo posible cambiar la ubicación y/o el nombre de archivos generados automáticamente en el proyecto, ya que pueden hacer que Roo++ no se comporte de la forma esperada, aun cuando Roo sí lo haga.*

Estructura de un proyecto

Veamos ahora cuáles son los diferentes módulos de que se compone un proyecto, tal y como los crea Roo++, y cuál es la función de cada uno. Cada uno de los módulos se encuentra en un subdirectorio diferente, dentro del directorio raíz del proyecto en desarrollo. El formato del nombre del directorio de un módulo siempre será ***nombre_del_proyecto-nombre_del_módulo*** (por ejemplo, *proyecto1-core*). Cada uno de estos subdirectorios siempre contendrá a su vez un subdirectorio ***src***, y, si el proyecto ya ha sido compilado en alguna ocasión, también un subdirectorio de nombre ***target***, en el que se almacenan los resultados de dicha compilación. Tan sólo se trabajará directamente sobre los archivos contenidos en el subdirectorio *src*.

Además, también es importante destacar la presencia de los archivos *pom.xml* (uno en cada módulo, en su directorio raíz, más otro adicional en el directorio raíz del proyecto completo), los cuales le indican a Maven cómo ha de construir el proyecto. En general, estos archivos son modificados de forma automática por Roo++ cuando se ejecutan sus diferentes comandos, aunque en ocasiones puede ser necesario editarlos manualmente cuando las necesidades del proyecto en desarrollo no se ajustan al tipo de proyecto para el que está pensado Roo++, o cuando se han producido errores a la hora de configurar el proyecto, de lo que han podido resultar *pom.xml* inconsistentes o con un contenido que no es el previsto.

core

La función de este módulo es implementar la lógica de negocio de la aplicación en desarrollo. Su subdirectorio *src* ha de contener los archivos *.java* con el código fuente para implementar dicha funcionalidad de la aplicación. En el subdirectorio *target* se empaquetará el resultado de compilar este módulo, en un archivo JAR.

docs

Este módulo contiene, en el subdirectorio *src*, los archivos fuente necesarios para generar la documentación del proyecto mediante DocBook XSL, que una vez creada se encontrará en el subdirectorio *target*. La documentación generada será empaquetada en un archivo ZIP.

ear

En el subdirectorio *target* de este módulo se generará un archivo EAR que contenga toda la aplicación, empaquetada, para su posterior despliegue en un servidor web apto para JSP (tal como Tomcat o Jetty).

package

En su subdirectorio *target* se generará un archivo ZIP con todos los archivos ZIP, EAR y WAR correspondientes a los diferentes módulos, y que han sido generados por Maven durante la compilación del proyecto.

scripts

Este módulo ha de contener, en el subdirectorio *src*, los scripts SQL (archivos *.sql*) necesarios para crear la base de datos que será utilizada por la aplicación, además de sus tablas, campos, relaciones entre ellas, carga inicial de datos (si es necesaria), etc. Estos scripts han de ser creados según el proceso indicado en este manual. Para crear realmente la base de datos y su contenido, en el momento de desplegar finalmente la aplicación, será necesario utilizar un software de gestión de bases de datos (el mismo que se haya usado durante el desarrollo), ejecutando estos scripts desde el mismo. Tras la compilación del proyecto, los scripts se encontrarán dentro de un archivo ZIP en el subdirectorio *target*.

web

Su subdirectorio *src* contiene tanto el código Java encargado de la interfaz web de la aplicación, como los elementos de que ésta se compone (incluyendo: imágenes, CSS, JavaScript, mensajes de localización, HTML, archivos de propiedades, archivos applicationContext.xml, y otros muchos archivos de configuración, la mayor parte en formato XML).Tras la compilación del proyecto, todo el módulo web y sus dependencias se encontrarán empaquetados en un archivo WAR dentro del subdirectorio *target*.

Proceso para la creación de la base de datos

Este proceso ha de llevarse a cabo siempre tras la ejecución del comando de Roo++ ***roo++ project web jpa***. Se han de seguir, por orden, los siguientes pasos:

1. Desde el sistema de gestión de bases de datos elegido (Oracle, MySQL, Microsoft SQL Server o PostgreSQL) crear una base de datos vacía (comando ***create database***), cuyo nombre coincida con el que se ha indicado como parámetro al ejecutar *roo++ project web jpa*.
2. En el directorio *src/main/resources/META-INF*, dentro del módulo web del proyecto, crear un archivo vacío de nombre *persistence.xml*. Este paso es necesario debido a un enfoque distinto entre Roo++ y Roo.
3. Desde la ventana de comandos del sistema operativo, situarse en el directorio raíz del módulo web (con el comando *cd*) del proyecto, y entrar en Roo con el comando ***roo*** (Windows) o ***roo.sh*** (Unix/Linux). Una vez dentro de Roo, es necesario definir las diferentes tablas y sus campos, ejecutando los comandos Roo necesarios para ello. Ver el siguiente ejemplo para más información.
4. Introducir el comando Roo ***web mvc all --package ~.web***.
5. Salir de Roo (comando ***quit***).
6. En el fichero *src/main/resources/META-INF/spring/database.properties*, dentro del mismo módulo web, buscar la línea *database.hibernate.hbm2ddl.auto=validate*, y sustituir el *validate* por *create* o *update*, según estemos creando o modificando la base de datos, respectivamente.

7. Volver desde la línea de comandos del sistema operativo al directorio raíz del proyecto (comando *cd* ..).
8. Ejecutar allí la orden ***mvn clean install -Dmaven.test.skip=true***, para compilar el proyecto (este paso no es necesario si el proyecto ya ha sido compilado anteriormente, y no se han hecho cambios en su módulo *core*).
9. Volver al directorio raíz del módulo web, haciendo uso del comando *cd*.
10. Ejecutar el comando ***mvn clean tomcat:run***, o bien ***mvn clean jetty:run*** (según qué servidor web queramos usar).
11. Exportar la base de datos que se acaba de crear (MySQL: *mysqldump -p -u username database_name > dbname.sql*; PostgreSQL: *pg_dump -U {user-name} {source_db} -f {dumpfilename.sql}*; Oracle: ver link; SQL Server: ver link).
12. Colocar el/los archivos SQL obtenidos en el paso anterior en el directorio *src/main/resources*, dentro del módulo scripts del proyecto.
13. Volver al archivo *src/main/resources/META-INF/spring/database.properties*, y sustituir la línea *database.hibernate.hbm2ddl.auto=create* o *database.hibernate.hbm2ddl.auto=update* por *database.hibernate.hbm2ddl.auto=validate.*
14. Eliminar el archivo *persistence.xml* creado anteriormente. Este paso es necesario para evitar problemas a la hora de desplegar la aplicación en ciertos tipos de servidores.
15. Si se quieren ejecutar más comandos de Roo++ a continuación, recordar volver al directorio raíz del proyecto en desarrollo (comando *cd* ..) antes de entrar en Roo.

Comandos

A continuación veremos los comandos adicionales que proporciona Roo++ a Spring Roo. Los comandos se muestran agrupados por tipos. Si un comando hace uso de parámetros, el valor de éstos (no el nombre) aparecerá subrayado. Cualquier nombre o valor de parámetro opcional aparecerá en *cursiva*. Los comandos se muestran dentro de sus correspondientes grupos de comandos, y estos bloques, a su vez, siempre ordenados alfabéticamente, salvo el bloque de ***roo++ project***, que aparece en primer lugar debido a su importancia y a que uno de sus comandos siempre ha de ser el primero en ejecutarse al desarrollar un proyecto en Roo++. Dentro de cada bloque, los comandos siempre se muestran según su orden natural de ejecución.

roo++ project

Bloque de comandos relacionados con aspectos generales de la creación y configuración de proyectos en Roo++.

roo++ project create-modules *--version version*

Convierte un proyecto Spring Roo en multimódulo, creando una serie de módulos estándar de Roo++, con un contenido simple predefinido, siendo un paso necesario para poder ejecutar cualquiera de los otros comandos proporcionados por este add-on de Spring Roo. También establece las dependencias necesarias entre estos módulos.

Descripción de los parámetros:

- **version:** número de versión (puede contener cualquier carácter) que se le quiere dar al proyecto a desarrollar, y por tanto también a cada uno de sus módulos. Si no se indica, el número de versión será *1.0.BUILD-SNAPSHOT*.

Ficheros de configuración implicados en la ejecución de este comando:

- **parent.txt** (si no está presente o no tiene contenido se informará al usuario de que el proyecto no tendrá *parent*)
- **fonts.xml**
- **custom.xsl**
- **Archivos de fuentes**
- **Installation_Manual.pdf**
- **User_Manual.pdf**
- **Archivos XML con plug-ins de Maven para la generación de la documentación**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **src (directorio):** eliminado (con todo su contenido)
- **nombreDelProyecto-core (directorio):** creado
- **nombreDelProyecto-core/pom.xml:** creado
- **nombreDelProyecto-docs (directorio):** creado
- **nombreDelProyecto-docs/src (directorio):** creado
- **nombreDelProyecto-docs/src/assembly (directorio):** creado
- **nombreDelProyecto-docs/src/assembly/package.xml:** creado
- **nombreDelProyecto-docs/src/docbkx (directorio):** creado

- **nombreDelProyecto-docs/src/docbkx/custom.xsl:** copiado desde el archivo de configuración correspondiente.
- **nombreDelProyecto-docs/src/fonts (directorio):** creado, introduciendo además en el mismo una copia de cada uno de los archivos de fuentes situados en el directorio de configuración correspondiente.
- **nombreDelProyecto-docs/src/front_pages (directorio):** creado
- **nombreDelProyecto-docs/src/front_pages/Installation_Manual.pdf:** copiado desde el archivo de configuración Installation_Manual.pdf.
- **nombreDelProyecto-docs/src/front_pages/User_Manual.pdf:** copiado desde el archivo de configuración User_Manual.pdf.
- **nombreDelProyecto-docs/src/images (directorio):** creado
- **nombreDelProyecto-docs/src/InstallationManual.xml:** creado
- **nombreDelProyecto-docs/src/UserManual.xml:** creado
- **nombreDelProyecto-docs/pom.xml:** creado
- **nombreDelProyecto-ear (directorio):** creado
- **nombreDelProyecto-ear/src (directorio):** creado
- **nombreDelProyecto-ear/src /main (directorio):** creado
- **nombreDelProyecto-ear/src/main/application (directorio):** creado
- **nombreDelProyecto-ear/src/main/application/META-INF(directorio):** creado
- **nombreDelProyecto-ear/src/main/application/META-INF/weblogic-application.xml:** creado
- **nombreDelProyecto-ear/pom.xml:** creado
- **nombreDelProyecto-package (directorio):** creado
- **nombreDelProyecto-package/src (directorio):** creado
- **nombreDelProyecto-package/src /assembly (directorio):** creado
- **nombreDelProyecto-package/src /assembly/package.xml:** creado
- **nombreDelProyecto-package/pom.xml:** creado
- **nombreDelProyecto-scripts (directorio):** creado
- **nombreDelProyecto-scripts/src (directorio):** creado
- **nombreDelProyecto-scripts/src/assembly (directorio):** creado
- **nombreDelProyecto-scripts/src/assembly/compress-zip.xml:** creado
- **nombreDelProyecto-scripts/pom.xml:** creado
- **nombreDelProyecto-web (directorio):** creado
- **nombreDelProyecto-web/src (directorio):** creado
- **nombreDelProyecto-web/src/main (directorio):** creado
- **nombreDelProyecto-web/src/main /java (directorio):** creado
- **nombreDelProyecto-web/src/main /resources (directorio):** creado
- **nombreDelProyecto-web/src/main /resources/META-INF (directorio):** creado
- **nombreDelProyecto-web/src/main /resources/META-INF/spring (directorio):** creado
- **nombreDelProyecto-web/src/main /resources/META-INF/spring/applicationContext.xml:** creado
- **nombreDelProyecto-web/src/main /webapp (directorio):** creado, con todo su contenido
- **nombreDelProyecto-web/pom.xml:** creado

- **pom.xml:** modificado, reemplazando su contenido original por otro en el que se establecen los diferentes módulos de que se compone el proyecto, así como sus características y las de su proyecto padre.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.

Comandos que no pueden haber sido ejecutados anteriormente:

Ninguno

roo++ project license

Introduce en el archivo *pom.xml* correspondiente a la raíz del proyecto referencias a la licencia o licencias que se quiere aplicar al mismo, junto con un plugin de Maven para su posterior introducción a la hora de empaquetar la aplicación ya desarrollada.

Además, también introduce en el archivo *pom.xml* correspondiente al módulo web del proyecto referencias a una serie de licencias correspondientes a diferentes librerías de Roo introducidas habitualmente, de forma automática, como dependencias en los proyectos desarrollados con Roo++.

Ficheros de configuración implicados en la ejecución de este comando:

- **licenses.xml**
- **plugin-license.xml**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/pom.xml:** introduce como hijo de */project/build/plugins* un nodo XML con referencias a una serie de licencias usadas por las librerías de Spring.
- **pom.xml:** introduce como hijo de */project/build/plugins* el nodo XML contenido en el archivo de configuración plugin-license.xml, y como hijo de */project/licenses* el nodo XML contenido en el archivo de configuración licenses.xml.

Comandos que tienen que haber sido ejecutados previamente:

project, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.

roo++ project create-modules

Comandos que no pueden haber sido ejecutados anteriormente:

roo++ project license

roo++ project web configuration

Introduce una serie de propiedades en el fichero *src/main/resources/META-INF/spring/application.properties*, dentro del módulo web del proyecto en desarrollo, junto con otros ajustes para dar la configuración adecuada a dicho módulo, para el desarrollo de la aplicación en Roo++ (en particular, seleccionar las diferentes ubicaciones en que se encuentran los archivos de propiedades que serán utilizados para la configuración del módulo web del proyecto, la URL en que la aplicación será desplegada, o las de los archivos de internacionalización y localización).

Ficheros de configuración implicados en la ejecución de este comando:
locations-application-properties.xml

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/src/main/resources/META-INF/spring/application.properties:** creado
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext.xml:** introduce los beans *messageSource*, *applicationPropertiesBean* y *property-placeholder*.
- **nombreDelProyecto-web/pom.xml:** el campo *useProjectReferences* de *maven-eclipse-plugin* es establecido a *false*.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ project web configuration***

roo++ project web jpa --user user --password password *--database database --dbms dbms --server server --port port --hibernateversion hibernateversion*

Configura el módulo web de la aplicación en desarrollo para permitir el acceso a una base de datos.

Si ya se había ejecutado previamente ***roo++ project web deploy***, será necesario ejecutarlo de nuevo tras ejecutar ***roo++ project web jpa***, antes de poder desplegar el módulo web de la aplicación en los servidores Tomcat o Jetty.

Antes de ejecutar este comando, es **muy importante** asegurarse de que la versión del sistema de gestión de bases de datos y el dialecto SQL a utilizar que se encuentran en los archivos de configuración asociados a este comando se ajusten a la base de datos concreta que será usada.

NOTA: *salvo que más adelante se quiera llevar a cabo una configuración distinta a la indicada en los parámetros, ignorar el siguiente mensaje que se muestra durante la ejecución de este comando:*

```
Please update your database details in src/main/resources/META-INF/spring/database.properties.
```

Descripción de los parámetros:

- **user:** nombre de usuario con el que se accederá al sistema de gestión de bases de datos a utilizar. Es un parámetro obligatorio.
- **password:** contraseña con la que dicha cuenta de usuario accede al sistema de gestión de bases de datos a utilizar. Es un parámetro obligatorio.
- **database:** nombre de la base de datos (no confundir con sistema de gestión de bases de datos) a la que accederá la aplicación desarrollada. Si no se indica, el nombre de la base de datos será el mismo que el de la aplicación (excepto si el sistema de gestión de

bases de datos utilizado es Oracle). En el caso particular de Oracle, este parámetro no se debe indicar, ya que será ignorado; con Oracle el nombre de la base de datos siempre será el mismo que el nombre del usuario.

- **dbms:** sistema de gestión de bases de datos al que se conectará la aplicación a desarrollar para el acceso a su base de datos asociada. Sus posibles valores son: oracle, mysql, mssql, postgres. Si no se indica el parámetro, o se indica un valor no válido, el sistema de gestión de bases de datos utilizado será Oracle.
- **server:** máquina en la que se ejecuta el sistema de gestión de bases de datos utilizado (puede indicarse su dirección IP o su nombre de dominio; **no poner su URL completa**, sólo el nombre de dominio o la IP). Si no se indica este parámetro, se asume que la base de datos a utilizar se encuentra en la máquina local (*localhost*).
- **port:** número de puerto por el que la aplicación a desarrollar se comunicará con el correspondiente sistema de gestión de bases de datos. Si no se indica, o si se indica algo que no es un número entero, se utilizará el puerto por defecto de la instalación del sistema de gestión de bases de datos utilizado: 1521 (Oracle), 3306 (MySQL), 1433 (Microsoft SQL Server), ó 5432 (PostgreSQL).
- **hibernateversion:** versión de la librería Hibernate a utilizar. ***NOTA:*** debido a un bug de Hibernate, si la base de datos utilizada es **MySQL** es necesario asignar a este parámetro el valor ***3.6.9.Final***, hasta que el bug sea corregido. Salvo este caso, no es conveniente indicar este parámetro.

Ficheros de configuración implicados en la ejecución de este comando:

- **mssql.txt**
- **mysql.txt**
- **oracle_sid.txt**
- **oracle.txt**
- **postgresql.txt**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/pom.xml:** introduce la dependencia respecto al driver del sistema de gestión de bases de datos utilizado. Además, si se ha indicado utilizar una versión de Hibernate distinta de la usada por defecto, también modificará este campo en las dependencias *hibernate-core* y *hibernate-entitymanager*.
- **nombreDelProyecto-web/src/main/jetty (directorio):** creado, con todo su contenido.
- **nombreDelProyecto-web/src/main/resources/META-INF/persistenceInfo.xml:** creado
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext.xml:** introduce el bean *entityManagerFactory*, y el *dataSource* de *jndi*, necesarios para el uso de la base de datos.
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/database.properties:** creado
- **nombreDelProyecto-web/src/main/tomcat (directorio):** creado, con todo su contenido.

- **nombreDelProyecto-web/src/main/webapp/WEB-INF/web.xml:** introduce la referencia al *dataSource* correspondiente, según el nombre del proyecto, para que el servidor web Jetty funcione correctamente al desplegar la aplicación en él.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ project web jpa***

roo++ project web deploy

Configura el módulo web de la aplicación en desarrollo para permitir su despliegue mediante los servidores web Tomcat y Jetty, introduciendo los plugins de Maven correspondientes en archivo *pom.xml* de dicho módulo.

Ficheros de configuración implicados en la ejecución de este comando:

Ninguno

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/pom.xml:** introduce los nodos de configuración en los plug-ins de Maven para los servidores web Tomcat y Jetty, teniendo en cuenta si es necesaria o no la conexión con una base de datos, según se haya ejecutado o no *web jpa* con anterioridad.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***

Comandos que no pueden haber sido ejecutados anteriormente:

Ninguno

roo++ breadcrumb

Comandos relacionados con la configuración de las breadcrumbs o "migas de pan" en la aplicación desarrollada, de acuerdo a las diferentes tablas y campos de la base de datos que ésta utiliza.

roo++ web breadcrumb setup

Introduce las breadcrumbs o "migas de pan" en la interfaz del módulo web de la aplicación desarrollada. Tener en cuenta que si se quiere utilizar un *layout* (apariencia y distribución de los diferentes elementos en la interfaz) diferente del creado por defecto (*default.jspx*), será necesario editar manualmente el fichero *.jspx* asociado a dicho *layout* para que haga uso de las "migas de pan", de forma similar a como haya quedado hecho en *default.jspx*.

Ficheros de configuración implicados en la ejecución de este comando:

- **breadcrumb-dependency.xml**

- **breadcrumbs-beans.xml**
- **breadcrumbs-url.txt**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/pom.xml:** introduce la dependencia contenida en el archivo de configuración breadcrumb-dependency.xml.
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext-platform.xml:** introduce los *beans* necesarios para el uso de las "migas de pan", según el contenido del archivo de configuración breadcrumbs-beans.xml**.** Además, a cada uno de ellos se le añade la propiedad *scanPackage* según el nombre del paquete a que pertenezca la aplicación.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/layouts/default.jspx:** modificado para permitir el uso de las "migas de pan".

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***
- ***roo++ project web jpa***
- ***roo++ platform setup***
- ***roo++ platform configuration***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ web breadcrumb setup***

roo++ context

Comandos para la creación de contextos (diferentes modos de realizar una separación lógica de los datos de la aplicación; cada usuario accederá a uno de ellos a través de la interfaz web, según cuáles sean sus permisos).

roo++ context setup

Transforma el módulo web de la aplicación de forma que cada usuario, según cuáles sean sus permisos, accederá a un conjunto de datos determinado, de entre todos los manejados por la aplicación.

Tras la ejecución de este comando, es necesario realizar manualmente una serie de tareas (las 3 primeras, **sin salir de Roo**, para que los cambios sean detectados por éste):

1. Ir al directorio ***nombreDelProyecto-web/src/main/java/ruta/interna/correspondiente/nombreDelProyecto/domain***.
2. Para cada archivo con extensión ***.java***, introducir lo siguiente dentro de la clase Java que contiene:
   ```
   @PersistenceContext(unitName = "persistenceUnitNombredelproyecto")
   transient EntityManager entityManager;
   ```
   (sustituyendo *Nombredelproyecto* por lo que corresponda).
3. En cada uno de estos mismos archivos, importar las siguientes clases:
   ```
   import javax.persistence.PersistenceContext;
   import javax.persistence.EntityManager;
   ```

4. Recordar, entre los pasos 10 y 11 del proceso para la creación de la base de datos (cuando sea el momento de llevar a cabo este proceso), ejecutar, desde el sistema de gestión de bases de datos utilizado, el archivo SQL *nombreDelProyecto-scripts/src/main/resources/context-dataload.sql* (que se ha creado copiando el archivo de configuración de igual nombre). Esto es necesario para que la base de datos de la aplicación contenga la información necesaria relativa a los contextos.

Ficheros de configuración implicados en la ejecución de este comando:

- **context-dataload.sql**
- **context-dependencies.txt**
- **context-messages.properties**
- **context-portal.properties**
- **node_to_add.txt**
- **portal_bean.xml**
- **authenticationSuccessHandler.txt**
- **security_bean.xml**
- **add-to-portal-header.txt**
- **directive-pages.jspx**
- **scriptlets.jspx**
- **vista.jspx**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-scripts/src/main/resources/context-dataload.sql:** creado, copiando el archivo de configuración *context-dataload.sql*.
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext-portal.xml:** introduce los *beans* necesarios para el uso de los contextos, según el contenido de los archivos de configuración contenidos en *contextBeans/portal*, e introduce además nodos adicionales en algunos de los *beans* ya existentes, según el contenido de los archivos de configuración situados en el directorio *nodesToAdd*.
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext-security.xml:** introduce los *beans* necesarios para el uso de los contextos, según el contenido de los archivos de configuración contenidos en *contextBeans/security*; asigna a *authenticationSuccessHandler* el valor contenido en el archivo de configuración *authenticationSuccessHandler.txt*, y elimina la *logout-success-url*, ya que esta URL han de proporcionarla los *beans* introducidos.
- **nombreDelProyecto-web/src/main/resources/META-INF/portal.properties:** introduce el contenido de *context-portal.properties* al final del archivo, con el fin de añadir las propiedades del portal necesarias para el uso de los contextos.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/i18n/messages.properties:** introduce los mensajes de localización relacionados con los contextos, contenidos en el archivo de configuración *context-messages.properties*.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/contexts (directorio):** creado, con todo su contenido, consistente en una copia de los archivos situados en el directorio de configuración *contextsViews*.

- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/header.jspx:** modificado, de acuerdo al contenido de los archivos de configuración *add-to-portal-header.txt*, *directive-pages.jspx* y *scriptlets.jspx*.
- **nombreDelProyecto-web/pom.xml:** introduce las dependencias Maven necesarias para el uso de los contextos, según lo indicado en el archivo *context-dependencies.txt*.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***
- ***roo++ project web configuration***
- ***roo++ project web jpa***
- ***roo++ platform setup***
- ***roo++ platform configuration***
- ***roo++ usermgmt setup***
- ***roo++ portal setup***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ context setup***

roo++ layout

Comandos relacionados con la modificación de la apariencia de la interfaz web de la aplicación desarrollada.

roo++ layout setup

Crea un nuevo *layout* (apariencia y distribución de los diferentes elementos en la interfaz) para el módulo *web* de la aplicación desarrollada, de acuerdo a lo indicado en el directorio *layout* y sus subdirectorios (destacando el archivo *load-scripts.jspx*, que siempre ha de estar presente y debe ajustarse al contenido de los subdirectorios *scripts* y *styles*). Si este comando ya había sido ejecutado anteriormente, no creará un nuevo *layout*, sino que actualizará el que se había creado en la primera ejecución del comando, de acuerdo al contenido actual de los archivos de configuración utilizados.

El comando sólo crea (o actualiza) el *layout;* para que éste sea el utilizado por la aplicación es necesario editar el archivo *src/main/webapp/WEB-INF/layouts/layouts.xml*, dentro del módulo *web*, sustituyendo en el campo *template* de cada nodo XML *tiles-definitions/definition* el valor *default.jspx* por *nombre_del_nuevo_layout.jspx*.

Sin embargo, la ejecución de este comando también introduce (si se encontraban en sus correspondientes directorios de configuración) una serie de hojas de estilos y JavaScripts que modificarán la apariencia y/o el comportamiento de la aplicación web, y esto siempre, se use el *layout* por defecto o el que ha creado o modificado la ejecución del comando.

Tener en cuenta que volver a ejecutar el comando no elimina CSS ni JavaScript introducidos por anteriores ejecuciones, salvo que la nueva ejecución introduzca otros con el mismo nombre. Si se desea eliminar los antiguos ha de hacerse manualmente. Además, tras ejecutar varias veces este comando es conveniente revisar el contenido del archivo

src/main/webapp/WEB-INF/tags/util/load-scripts.tagx, dentro del módulo *web*, para que se ajuste a lo realmente deseado.

Para seleccionar cuál de las CSS se va a utilizar como tema en la aplicación, se ha de editar el archivo *nombreDelProyecto-web/src/main/resources/META-INF/spring/application.properties*, sustituyendo *application.theme=standard* por *application.theme=estiloAUtilizar*, siendo *estiloAUtilizar.properties* el archivo contenido en el directorio *classes* del layout que hace referencia a la hoja de estilos que se desea aplicar.

Las imágenes introducidas en el módulo web del proyecto por la ejecución de este comando tan sólo serán utilizadas si se indica su uso editando manualmente el archivo .jspx correspondiente, según el lugar y situación en que se desee se muestre cada imagen.

Ficheros de configuración implicados en la ejecución de este comando:

- Todos los contenidos en el directorio ***layout*** y sus subdirectorios, destacando el archivo ***load-scripts.jspx***, que siempre ha de estar presente y debe ajustarse al contenido de los subdirectorios *scripts* y *styles*.

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/src/main/resources/META-INF/spring/application.properties:** introduce la variable *application.theme*, si no había sido introducida previamente.
- Dentro de **nombreDelProyecto-web/src/main/webapp,** directorios **images, scripts, styles, WEB-INF/classes, WEB-INF/i18n, WEB-INF/tags**: copiados archivos desde los directorios de configuración *images, scripts, styles, classes, properties, y tags*, respectivamente.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/spring/webmvc-config.xml:** reemplaza el bean *CookieThemeResolver* por *FixedThemeResolver*.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/tags/util/load-scripts.tagx:** modificado para permitir la carga correcta de los CSS y JavaScript que forman parte del nuevo *layout*.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/footer.jspx:** modificado, eliminando el nodo *<util:theme/>*.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***
- ***roo++ project web configuration***

Comandos que no pueden haber sido ejecutados anteriormente:

Ninguno (se informará al usuario si ha ejecutado varias veces ***roo++ layout setup*** sin cambiar el contenido del fichero *load-scripts.jspx*, ya que en ese caso la ejecución del comando no produce ningún efecto)

roo++ platform

Comandos relacionados con la configuración del uso de una plataforma de utilidades por parte de la aplicación desarrollada.

roo++ platform setup

Introduce en los ficheros *pom.xml* del proyecto y de sus módulos *core* y *web* todo lo necesario para establecer dependencias con respecto a una plataforma que proporcione utilidades para ser usadas desde la aplicación en desarrollo.

Ficheros de configuración implicados en la ejecución de este comando:

- **platform.txt**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-core/pom.xml:** introduce dependencias respecto al módulo *core* y al módulo de test de la plataforma.
- **nombreDelProyecto-web/pom.xml:** introduce dependencias respecto al módulo *web* y al módulo de test de la plataforma.
- **pom.xml:** introduce el campo *platform.version*, con el número de versión de la plataforma.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ platform setup***
- ***roo++ platform configuration***

roo++ platform configuration

Introduce en el módulo *web* del proyecto que tiene el foco los beans necesarios para el uso de la plataforma anteriormente seleccionada.

Ficheros de configuración implicados en la ejecución de este comando:

- **applicationContext-platform.xml**
- **platform.txt**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext-platform.xml:** creado (copiado desde el archivo de configuración applicationContext-platform.xml).
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext.xml:** introduce la plataforma como *base-package* en *component-scan*.
- **nombreDelProyecto-web/ src/main/webapp/WEB-INF/spring/webmvc-config.xml:** introduce la plataforma como *base-package* en *component-scan*.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***
- ***roo++ platform setup***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ platform configuration***

roo ++ portal

Comandos relacionados con la integración en un portal de la interfaz web de la aplicación desarrollada.

roo++ portal setup

Integra el módulo *web* de la aplicación en desarrollo dentro de un portal. De este portal se hará uso a través de la plataforma utilizada por dicha aplicación, y siempre usando un sistema de gestión de usuarios proporcionado por dicha plataforma.

Ficheros de configuración implicados en la ejecución de este comando:

- **applicationContext-portal.xml**
- **messages.properties**
- **portal.properties**
- **portal.txt**
- **portal-header.jspx**
- **portal-url.txt**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-web/src/main/resources/META-INF/portal.properties:** creado (copiado desde el archivo de configuración portal.properties).
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext-portal.xml:** creado (copiado desde el archivo de configuración applicationContext-portal.xml).
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext.xml:** modificado para incluir todas las ubicaciones de archivos *.properties* en el *classpath*.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/i18n/messages.properties:** creado, copiando el archivo de configuración messages.properties.
- **nombreDelProyecto-web-src/main/webapp/WEB-INF/spring/webmvc-config.xml:** modificado, asociando la URL */index* con la página principal de la aplicación web.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/tags/util/load-scripts.tagx:** modificado para el uso del portal, según lo configurado en el archivo portal-url.txt, cambiando también la posición del nodo *portal:load-scripts*, para que las CSS utilizadas funcionen adecuadamente, e introduciendo también el nodo *<portal:load-scripts noIncludeDojo="true"/>*.

- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/header.jspx:** contenido reemplazado por el del archivo de configuración portal-header.jspx, que está adaptado al uso del portal.
- **nombreDelProyecto-web/pom.xml:** introduce la dependencia con respecto al portal.
- **pom.xml:** introduce el campo *portal.version*, con el número de versión del portal.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***
- ***roo++ project web configuration***
- ***roo++ platform setup***
- ***roo++ platform configuration***
- ***roo++ usermgmt setup***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ portal setup***

roo++ usermgmt

Comandos relacionados con el uso de un sistema de autenticación de usuarios para el acceso a la aplicación.

roo++ usermgmt setup

Configura el módulo *web* de la aplicación desarrollada para el uso de un sistema de autenticación de usuarios. Este sistema podrá ser proporcionado por la plataforma de la que haga uso la aplicación (si hace uso de alguna, y puede proporcionar esta funcionalidad), o bien a través de cualquier sistema CAS, a través de las librerías de Spring Security. La existencia o no del directorio de configuración ***usermgmt*** determinará el uso de la primera técnica o la segunda, respectivamente.

Ficheros de configuración implicados en la ejecución de este comando:

usermgmt: la existencia o no de este directorio determinará qué tipo de gestión de usuarios se llevará a cabo, y también qué grupo de archivos de configuración será tenido en cuenta en cada caso. Si este directorio existe, serán utilizados los siguientes archivos de configuración (contenidos en el mismo directorio):

- **application.properties**
- **applicationContext-security.xml**
- **artifactItems.txt**
- **changePassword.jspx**
- **userInfo.jspx**
- **usermgmt.txt**

Si el directorio no existe, los ficheros de configuración utilizados serán los siguientes:

- **usermgmt.txt**
- **usermgmt_application.properties**

- **usermgmt_users.txt**

Archivos del proyecto creados o modificados por la ejecución de este comando:

- **nombreDelProyecto-package/pom.xml:** si se usa la gestión de usuarios proporcionada por la plataforma, introduce los nodos *artifactItem* necesarios para el uso del módulo de gestión de usuarios, a partir del contenido del archivo de configuración artifactItems.txt.
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/application.properties:** introduce las propiedades correspondientes relacionadas con la gestión de usuarios, sea cual sea el tipo de gestión de usuarios que se lleve a cabo (dependiendo de cuál sea el tipo, las nuevas propiedades serán obtenidas del archivo de configuración usermgmt/application.properties o del usermgmt_application.properties).
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext.xml:** introduce referencias a las diferentes ubicaciones de los mensajes de localización, si la gestión de usuarios se hace a través de la plataforma. Además, también sólo en ese mismo caso, introduce el *groupId* del módulo de gestión de usuarios como "*base-package*", junto con el *groupId* del propio proyecto, y el de la plataforma, que ya aparecían como tales anteriormente.
- **nombreDelProyecto-web/src/main/resources/META-INF/spring/applicationContext-security.xml:** creado (en el caso de usar la gestión de usuarios proporcionada por la plataforma en lugar de las librerías de Spring Security, es copiado desde el archivo de configuración applicationContext-security.xml).
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/spring/webmvc-config.xml:** en el caso de usar la gestión de usuarios proporcionada por la plataforma en lugar de las librerías de Spring Security, introduce el *groupId* del módulo de gestión de usuarios como "*base-package*", junto con el *groupId* del propio proyecto, y el de la plataforma, que ya aparecían como tales anteriormente. Además, en ese mismo caso, también introduce referencias a las diferentes ubicaciones de los mensajes de localización, y añade una nueva propiedad para el bean *tilesConfigurer*, de forma que el *classpath* incluya el fichero *views.xml*.
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/users (directorio):** creado (sólo en el caso de usar la plataforma y no Spring Security para la gestión de usuarios).
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/users/changePassword.jspx:** creado copiando el archivo de configuración changePassword.jspx (sólo en el caso de usar la plataforma y no Spring Security para la gestión de usuarios).
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/users/userInfo.jspx:** creado copiando el archivo de configuración userInfo.jspx (sólo en el caso de usar la plataforma y no Spring Security para la gestión de usuarios).
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/casfailure.jspx:** creado
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/footer.jspx:** modificado, haciendo que la URL de *logout* sea */j_spring_security_logout* en lugar de */resources/j_spring_security_logout*.

- **nombreDelProyecto-web/src/main/webapp/WEB-INF/views/views.xml:** modificado, introduciendo la referencia a la página de fallo de autenticación con CAS (que se mostrará cuando un usuario autenticado intenta acceder a un recurso sin estar autorizado para su rol).
- **nombreDelProyecto-web/src/main/webapp/WEB-INF/web.xml:** modificado, delegando la implementación de una cadena de filtros en el bean *springSecurityFilterChain*.
- **nombreDelProyecto-web/pom.xml:** si la gestión de usuarios se realiza a través de la plataforma, introduce la dependencia respecto al módulo de gestión de usuarios a utilizar; en el caso de que la gestión de usuarios se haga directamente mediante Spring Security, introduce las dependencias respecto a sus módulos *spring-security-core*, *spring-security-config*, *spring-security-web* y *spring-security-cas*.
- **pom.xml:** introduce (en el caso de usar la gestión de usuarios proporcionada por la plataforma en lugar de las librerías de Spring Security) la versión del módulo de gestión de usuarios y el nombre de la aplicación a través de la cual se accede al mismo, en los nodos *usermgmt.version* y *usermgmt.delivery*, respectivamente.

Comandos que tienen que haber sido ejecutados previamente:

- ***project***, comando estándar de Roo (no es, por tanto, parte de Roo++), que crea un proyecto básico para comenzar a trabajar sobre él.
- ***roo++ project create-modules***
- ***roo++ project web configuration***

Además, en el caso de realizar la gestión de usuarios a través de la plataforma, en lugar de mediante Spring Security, también tendrán que haber sido ejecutados:

- ***roo++ platform setup***
- ***roo++ platform configuration***

Comandos que no pueden haber sido ejecutados anteriormente:

- ***roo++ usermgmt setup***
- ***roo++ portal setup***

Ficheros de configuración

Todos los archivos de configuración de Roo++ han de encontrarse en un directorio llamado *RooAddonsData*, ubicado en el directorio raíz del usuario que está usando la aplicación (por ejemplo, *C:\Users\Usuario* o */home/usuario*). Alternativamente, dicho directorio de configuración podrá encontrarse, a elección del usuario o administrador, en cualquier otra ubicación, permitiendo de esta forma, por ejemplo, su uso por múltiples usuarios dentro de una misma máquina. Si se desea esta última opción, ha de definirse una variable de entorno en el sistema operativo, con el nombre *ROOADDON (%ROOADDON%* en Windows o *$ROOADDON* en Unix/Linux), y cuyo contenido sea la ruta completa de la carpeta (incluyendo su nombre, que en este caso no tiene que ser necesariamente *RooAddonsData*), acabado en el carácter */*. Si esta variable de entorno no está definida, Roo++ asume que el directorio es *RooAddonsData* en el directorio *home* del usuario, y si éste no existe o no tiene el contenido que debe no podrá funcionar adecuadamente.

Cualquiera de estos ficheros puede ser editado con el fin de ajustar el comportamiento de Roo++ a sus necesidades. En varios de los casos, una correcta configuración realizada por el usuario es necesaria antes de poder ejecutar muchos de los comandos de Roo++ de la forma esperada.

Este directorio de configuración contiene una serie de subdirectorios; la existencia de alguno de ellos (y de alguno de los ficheros) es opcional, y su eliminación o creación cambiará el comportamiento de Roo++ ante determinados comandos. Veamos a continuación cuáles son los ficheros y directorios de configuración, y cuál es el cometido y el formato de cada uno de ellos (asumiendo que *RooAddonsData* es el nombre del directorio raíz de la configuración; sustituirlo por el nombre correspondiente en caso de que se utilice otro). Todos los ficheros y directorios se muestran por orden alfabético. Los nombres (o partes del nombre) que son variables aparecen en *cursiva*. Cuando se hace referencia a un tipo de fichero que puede encontrarse, en su directorio correspondiente, ninguna, una, o varias veces, su nombre aparecerá entre [corchetes]. Si la existencia de un directorio o fichero es opcional, su nombre aparecerá entre (paréntesis).

NOTA: *el directorio "RooAddonsData/layout" será tratado de forma diferente al resto, por sus características particulares.*

RooAddonsData

applicationContext-platform.xml

Fichero XML que contiene los beans que sean necesarios para que la aplicación desarrollada pueda hacer uso de una plataforma.

Se trata de un fichero de tipo *applicationContext.xml*, utilizados en muchas ocasiones en Spring. Un ejemplo de la estructura de uno de estos archivos (sin relación con este caso particular de la plataforma) sería el siguiente:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE beans PUBLIC "-//SPRING//DTD BEAN//EN"
"http://www.springframework.org/dtd/spring-beans.dtd">
```

```
<beans>

  <bean id="fileEventType"
class="com.devdaily.springtest1.bean.FileEventType">
    <property name="eventType" value="10"/>
    <property name="description" value="A sample description here"/>
  </bean>

  <bean id="fileEventDao"
class="com.devdaily.springtest1.dao.FileEventDao">
    <property name="dataSource" ref="basicDataSource"/>
  </bean>

  <bean id="basicDataSource"
class="org.apache.commons.dbcp.BasicDataSource">
    <property name="driverClassName" value="com.mysql.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost/my_database" />
    <property name="username" value="my_username" />
    <property name="password" value="my_password" />
    <property name="initialSize" value="3" />
    <property name="maxActive" value="10" />
  </bean>

</beans>
```

applicationContext-portal.xml

Fichero XML que contiene los beans que sean necesarios para que el módulo web de la aplicación desarrollada pueda ser integrado en un portal.

Se trata de un fichero de tipo *applicationContext.xml*, utilizados en muchas ocasiones en Spring. Un ejemplo de la estructura de uno de estos archivos (sin relación con este caso particular del portal) sería éste.

breadcrumb-dependency.xml

Contiene el nodo XML con la dependencia respecto al módulo de la plataforma utilizada que introduce las breadcrumbs o "migas de pan" en la interfaz web de la aplicación desarrollada; por ejemplo:

```
<dependency>
      <groupId>com.empresa.platform</groupId>
      <artifactId>platform-readcrumb-module</artifactId>
      <version>${platform.version}</version>
</dependency>
```

breadcrumbs-beans.xml

Contiene uno o varios nodos XML, cada uno de los cuales corresponde a un bean necesario para introducir las "migas de pan" en el módulo web de la aplicación desarrollada.

Cada *bean* debe tener nodos diferentes de apertura y cierre, incluso aunque no tenga nodos hijos; esto es necesario para que Roo++ pueda introducir ciertas propiedades en los mismos.

No ha de ser necesariamente un archivo XML válido en sí mismo (si contiene varios beans, no tendrá nodo raíz).

breadcrumbs-url.txt

Indica la URL a introducir en la cabecera *de src/main/webapp/WEB-INF/layouts/default.jspx*, dentro del módulo web del proyecto en desarrollo, para el uso correcto de las las breadcrumbs o "migas de pan".

context-dataload.sql

Contiene la lista de comandos SQL que se han de ejecutar para introducir en ciertas tablas de la base de datos asociada al proyecto todo lo necesario para el uso de los contextos.

context-dependencies.txt

Contiene, en líneas diferentes, el valor de los campos *groupId*, *artifactId* y *version*, respectivamente, de cada dependencia que se ha de añadir al archivo *pom.xml* del módulo *web* del proyecto para el uso de los contextos (repitiéndose una secuencia de estas 3 líneas para cada una de las dependencias que se han de introducir).

context-messages.properties

Contiene los mensajes de localización necesarios para el uso de los contextos.

context-portal.properties

Contiene las propiedades que es necesario aplicar al portal utilizado para permitir el uso de los contextos.

licenses.xml

Contiene un nodo XML que define la/s licencia/s que se quiere/n aplicar a la aplicación en desarrollo.

Su contenido ha de ser similar a:

```
<licenses>
  <license>
    <name>Software license</name>
    <url>${license.url}softwarelicense/license.txt</url>
    <distribution>repo</distribution>
    <comments>This is a software license</comments>
  </license>
</licenses>
```

locations-application-properties.xml

Contiene una serie de nodos XML (uno en cada línea), que serán introducidos en el bean *applicationPropertiesBean*, dentro del *pom.xml* del módulo web del proyecto que se está desarrollando. Estos nodos indican ubicaciones adicionales en que se han de buscar los ficheros *.properties* a tener en cuenta durante la construcción del proyecto (además de las que ya incluye el bean tal y como es introducido por defecto). Las ubicaciones configuradas tendrán siempre mayor prioridad sobre las demás que las incluidas por defecto, y cuanto más abajo aparezcan en *locations-application-properties.xml*, mayor prioridad tendrán.

Este fichero no es un fichero XML válido por sí mismo, pues no tiene nodo raíz; si no es necesario introducir nuevos nodos de este tipo en el bean, el fichero debe estar vacío.

Un ejemplo del posible contenido de este archivo es:

```
<value>file:///${TEST_HOME}/test-core.properties</value>

<value>file:///${TEST_HOME}/test-web.properties</value>
```

messages.properties

Indica el texto que se mostrará en la interfaz web de la aplicación desarrollada, cuando ésta haga uso de un portal. Su contenido ha de ser el siguiente, reemplazando los nombres por los que sean necesarios:

```
application_name=Aplicación

#Informacion de los modulos de la aplicacion en el menu general
application.name=Nombre de la Aplicación
application.description=Descripción de la aplicación


# Modulo de configuracion de la aplicacion
application.configuration.label.name=Administración
application.configuration.label.description=Menú de administración


#Casfailed
label.authorization.failure.title=Error de autenticación
label.authorization.failure.access=Error de acceso
label.global.volver=Volver
label.global.salir=Salir
label.global.footer=Texto del footer

portal.configuration.default.module.name=Administrar
portal.configuration.default.module.description=Módulo de administración
```

mssql.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver JTDS (por ejemplo, 1.2.4), y la versión del dialecto SQL (por ejemplo, SQLServer2008Dialect). Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de Microsoft SQL Server.

mysql.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver de MySQL para Java (por ejemplo, 5.1.18), y la versión del dialecto SQL (por ejemplo, MySQL5InnoDBDialect). Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de MySQL.

oracle_sid.txt

SID a utilizar cuando la aplicación desarrollada hace uso de una base de datos Oracle. En otro caso, el contenido de este archivo no será utilizado.

oracle.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver de bases de datos Oracle para Java (por ejemplo, 16), y la versión del dialecto SQL (por ejemplo, Oracle10gDialect). Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de Oracle.

(parent.txt)

Contiene 3 líneas, que indican, respectivamente, los campos *groupId*, *artifactId* y *version* del módulo que actuará como padre de la aplicación desarrollada.

Si este fichero no existe o no tiene el contenido completo, el proyecto desarrollado carecerá de padre.

platform.txt

Incluye, respectivamente, en diferentes líneas, los campos *groupId* y *version*, y el nombre del módulo que contiene clases de utilidades para llevar a cabo los test de la aplicación desarrollada, correspondientes a la plataforma en la que se integrará la aplicación.

plugin-license.xml

Contiene un plugin para Maven que será añadido al fichero *pom.xml* raíz del proyecto en desarrollo, y que establece como obtener y procesar la licencia de dicho proyecto.

Su contenido ha de ser similar a:

```
<plugin>
  <groupId>org.codehaus.mojo</groupId>
  <artifactId>license-maven-plugin</artifactId>
  <version>1.0</version>
    <configuration>
      <licenseName>license</licenseName>
      <licenseResolver>${license.url}</licenseResolver>
      <bundleLicensePath>META-INF/LICENSE.txt</bundleLicensePath>
      <generateBundle>false</generateBundle>
    </configuration>
    <executions>
      <execution>
        <id>update-project-license</id>
        <goals>
          <goal>update-project-license</goal>
        </goals>
        <phase>process-sources</phase>
     </execution>
  </executions>
```

```
</plugin>
```

portal.properties

Propiedades necesarias para integrar en un portal el módulo web de la aplicación que se está desarrollando. Su contenido ha de ser similar al siguiente, aunque puede variar dependiendo de las características concretas del portal utilizado:

```
# Beans que proporcionan la información de la aplicación
portal.application.provider.name=portalApplicationProvider
portal.configuration.provider.name=portalConfigurationProvider

# Properties de configuración del portal
portal.application.id=${APP.NAME}
portal.application.url=${REMOTE.HOST}/${APP.NAME}
portal.application.priority=1
portal.application.default.name=${APP.NAME}
portal.application.default.description=${APP.NAME}

portal.default.module.id=Default
portal.default.module.url=${portal.application.url}/index

portal.configuration.logout.url=/j_spring_security_logout
portal.configuration.user.url=${portal.application.url}/userInfo
portal.configuration.security.role=ROLE_ADMIN

portal.configuration.default.module.id=Configuration
portal.configuration.default.module.url=${portal.application.url}/index
portal.configuration.default.module.role=ROLE_ADMIN

portal.default.module.default.name=Default
portal.default.module.default.description=Default
```

portal.txt

Contiene los campos *groupId*, *artifactId* y *version* de la dependencia del proyecto desarrollado con respecto al módulo correspondiente al portal en el que se ha de integrar su módulo web.

portal-header.jspx

Archivo JavaServer Pages (*.jspx*) que introduce en la interfaz de la aplicación web desarrollada la cabecera correspondiente al portal en que ésta se integra.

Su contenido puede ser algo similar a:

```
<div id="header" xmlns:jsp="http://java.sun.com/JSP/Page"
        xmlns:fn="http://java.sun.com/jsp/jstl/functions"
        xmlns:c="http://java.sun.com/jsp/jstl/core"
        xmlns:spring="http://www.springframework.org/tags"
        xmlns:portal="http://www.empresa.com/portal" version="2.0">

      <jsp:directive.page
import="org.springframework.web.servlet.support.RequestContextUtils" />
      <jsp:directive.page
import="org.springframework.context.i18n.LocaleContextHolder" />
      <jsp:directive.page
import="com.empresa.portal.handler.PortalClientHandler" />

      <jsp:scriptlet>
```

```
            pageContext.setAttribute("applications",
PortalClientHandler.getApplications(), PageContext.PAGE_SCOPE);
        pageContext.setAttribute("configuration",
PortalClientHandler.getConfiguration(), PageContext.PAGE_SCOPE);
        pageContext.setAttribute("currentLocale",
LocaleContextHolder.getLocale(), PageContext.PAGE_SCOPE);
        pageContext.setAttribute("requestUri",
org.springframework.security.web.util.UrlUtils
            .buildFullRequestUrl(pageContext.getRequest().getScheme(),
pageContext.getRequest().getServerName(),
                    pageContext.getRequest().getServerPort(),

pageContext.getAttribute("javax.servlet.forward.request_uri", 2).toString(),
null), 2);
      </jsp:scriptlet>

      <portal:portal-header applications="${applications}"
currentLocale="${currentLocale}"
            configuration="${configuration}" requestUri="${requestUri}" />

</div>
```

portal-url.txt

URL correspondiente al portal a utilizar en el módulo web de la aplicación en desarrollo (por ejemplo, *http://www.empresa.com/portal*).

postgresql.txt

Contiene 2 líneas, indicando, respectivamente, la versión del driver de PostgreSQL para Java (por ejemplo, 9.0-801.jdbc4), y la versión del dialecto SQL (por ejemplo, PostgreSQLDialect). Este fichero tan sólo será utilizado cuando la aplicación desarrollada haga uso de una base de datos de PostgreSQL.

usermgmt.txt

Este fichero sólo será usado en caso de que se utilicen las librerías de Spring Security para la gestión de usuarios, en lugar de un módulo personalizado. Contiene 6 líneas, cuyo contenido es, por orden:

- Nombre de la máquina en la que se ejecutará la aplicación en desarrollo (por ejemplo, *localhost* o *maquina.dominio.com*).
- Puerto en el que se accederá a la interfaz web de la aplicación en desarrollo (típicamente será 8080, aunque puede ser otro si así se establece en el servidor web utilizado).
- URL de la página de autenticación (*login*) del sistema CAS (Central Authentication Service) utilizado, incluyendo el número de puerto (por ejemplo, *https://localhost:9443/cas/login*).
- URL del sistema CAS que valida el *ticket* (indicando que el usuario se haya autenticado correctamente y que su sesión no haya caducado), incluyendo el número de puerto (coincide con la URL base del CAS; por ejemplo, *https://localhost:9443/cas*).
- Un identificador único para el proveedor de autenticación (puede tener cualquier valor, tal como el nombre de la aplicación o de la organización).

- URL de la página de *logout* del sistema CAS, incluyendo el número de puerto (por ejemplo, *https://localhost:9443/cas/logout*).

Un sistema CAS sencillo de instalar y utilizar es el de Jasig, que después de ser compilado con Maven puede ser desplegado en una instalación local de Tomcat.

usermgmt_application.properties

Todas las propiedades que sea necesario añadir al archivo de este mismo nombre, dentro del módulo web de la aplicación en desarrollo, para llevar a cabo la gestión de usuarios desde la propia aplicación, mediante las librerías de Spring Security. Si dicha gestión de usuarios se realiza a través de un módulo externo, configurado mediante el directorio *usermgmt*, este archivo no será utilizado. Por ejemplo, su contenido podría ser algo como:

```
# Role security
application.security.role=ROLE_ADMIN
```

Dependiendo siempre del nivel de protección deseado.

usermgmt_users.txt

Este fichero sólo será usado en caso de que se utilicen las librerías de Spring Security para la gestión de usuarios, en lugar de un módulo personalizado. Contiene 3 líneas por cada uno de los usuarios de la aplicación desarrollada: nombre de usuario, contraseña, y roles del usuario (por ejemplo, *ROLE_USER* o *ROLE_USER, ROLE_ADMIN*). Las líneas han de encontrarse en este orden, y a continuación de ellas las mismas 3 líneas correspondientes al siguiente usuario de la lista, si es que existe. En este fichero deben figurar estos 3 atributos para cada uno de los usuarios que deseen acceder a la aplicación protegida, **y sus nombres de usuario, contraseñas y roles han de ser los mismos que existan en el sistema CAS** (Central Authentication Service) utilizado para la autenticación.

RooAddonsData/contextBeans/portal

[*portal_bean.*xml]

Archivos que contienen, cada uno de ellos, un nodo XML, referente a un bean que es necesario introducir en el archivo applicationContext-portal.xml (no en el archivo de configuración, si no en su copia creada en el proyecto tras ejecutar el comando portal setup) para el uso de los contextos.

RooAddonsData/contextBeans/portal/nodesToAdd

[*node_to_add.*txt]

Cada uno de estos archivos contiene los datos sobre un nodo XML a introducir en el archivo applicationContext-portal.xml (no en el archivo de configuración, si no en su copia creada en el proyecto tras ejecutar el comando portal setup), con el fin de permitir el uso de los contextos. Ha de contener, en líneas diferentes, respectivamente, el XPath del nodo al que se le introducirá el nuevo elemento como hijo, el nombre del nuevo nodo a introducir como hijo de éste, y los diferentes atributos, cada uno de ellos seguido, en la siguiente línea, por su correspondiente valor.

RooAddonsData/contextBeans/security

authenticationSuccessHandler.txt

Contiene el nombre del bean introducido en el archivo applicationContext-security.xml (situado en el módulo web del proyecto tras la ejecución del comando usermgmt setup) que será el encargado de gestionar la situación de éxito en la autenticación de usuarios, en lugar del que lo hace por defecto (*authenticationSuccessHandler*), para posibilitar el uso de los contextos.

[*security_bean*.xml]

Archivos que contienen, cada uno de ellos, un nodo XML, referente a un bean que es necesario introducir en el archivo applicationContext-security.xml (situado en el módulo web del proyecto tras la ejecución del comando usermgmt setup) para hacer posible el uso de los contextos.

RooAddonsData/contextHeader

add-to-portal-header.txt

Contiene las asignaciones de variables que sea necesario añadir al nodo *portal:portal-header*, en el archivo *src/main/webapp/WEB-INF/views/header.jspx* (dentro del módulo web del proyecto), para hacer posible el uso de contextos por parte de la aplicación.

directive-pages.jspx

Contiene los nodos "*<jsp:directive.page>*"que se han de introducir en el archivo *src/main/webapp/WEB-INF/views/header.jspx* (dentro del módulo web del proyecto) para posibilitar el uso de los contextos. No tiene por qué ser un archivo jspx válido (puede no tener nodo raíz).

scriptlets.jspx

Contiene los nodos "*<jsp:scriptlet>*"que se han de introducir en el archivo *src/main/webapp/WEB-INF/views/header.jspx* (dentro del módulo web del proyecto) para posibilitar el uso de los contextos. No tiene por qué ser un archivo jspx válido (puede no tener nodo raíz).

RooAddonsData/contextViews

[*vista*.jspx]

Cada uno de estos archivos contiene una vista diferente, necesaria para la visualización de la aplicación según los contextos utilizados.

RooAddonsData/doc

fonts.xml

Lista de fuentes que serán utilizadas en la documentación del proyecto en desarrollo, incluyendo la ubicación de sus archivos correspondientes. Una misma fuente puede aparecer en varias ocasiones, cuando se hace uso de la misma con varios estilos diferentes (negrita, cursiva, etc).

Dicha lista debe formar parte de un nodo XML de nombre *fonts*. Un ejemplo de posible contenido para este archivo sería:

```
<fonts>
	<font>
		<name>Calibri</name>
		<style>normal</style>
		<weight>normal</weight>
		<embedFile>${basedir}/src/fonts/calibri.ttf</embedFile>
		<metricsFile>${project.build.directory}/fonts/calibri-
metrics.xml</metricsFile>
	</font>
	<font>
		<name>Calibri</name>
		<style>italic</style>
		<weight>normal</weight>
		<embedFile>${basedir}/src/fonts/calibrii.ttf</embedFile>
		<metricsFile>${project.build.directory}/fonts/calibrii-
metrics.xml</metricsFile>
	</font>
	<font>
		<name>Calibri</name>
		<style>normal</style>
		<weight>bold</weight>
		<embedFile>${basedir}/src/fonts/calibrib.ttf</embedFile>
		<metricsFile>${project.build.directory}/fonts/calibrib-
metrics.xml</metricsFile>
	</font>
	<font>
		<name>Georgia</name>
		<style>normal</style>
		<weight>normal</weight>
		<embedFile>${basedir}/src/fonts/georgia.ttf</embedFile>
		<metricsFile>${project.build.directory}/fonts/georgia-
metrics.xml</metricsFile>
	</font>
</fonts>
```

En este ejemplo , *${basedir} /src/fonts/georgia.ttf* le indica a Maven que localice el archivo que contiene esa fuente en esa ubicación, dentro del módulo *docs* (a cuyo directorio raíz se refiere *${basedir}*, ya que este nodo *fonts* será introducido en el fichero *pom.xml* de dicho módulo), mientras que *${project.build.directory}/fonts/georgia-metrics.xml* le indica que el archivo de métricas de dicha fuente se generará en ese lugar dentro del directorio *target* (el directorio en que se guardan los resultados durante la "construcción" del proyecto) de dicho módulo *docs*.

RooAddonsData/doc/docbkx

custom.xsl

Hoja de estilos XSL que se utilizará cuando se genere la documentación del proyecto en desarrollo mediante DocBook XSL, para determinar la presentación de la misma.

RooAddonsData/doc/fonts

[*fontname.ttf*]

Archivos conteniendo las fuentes (TrueType, OpenType, o cualquier otro tipo) que serán utilizadas en la documentación del proyecto en desarrollo.

RooAddonsData/doc/front_pages

(Installation_Manual.pdf)

Archivo PDF que contiene una serie de páginas prefijadas (probablemente portadas , contraportadas, u otras páginas especiales que no se vayan a introducir mediante el fichero XML de DocBook XSL *src/InstallationManual.xml* (dentro del módulo *docs* del proyecto).

Las reglas para su introducción en el documento *Installation_Manual.pdf* (manual de implantación de la aplicación desarrollada) final están en RooAddonsData/docPlugins.

Si este archivo no está presente, RooAddonsData/docPlugins debe estar vacío, o los plug-ins allí contenidos no obtener ninguna página del archivo *front_pages/Installation_Manual.pdf*.

(User_Manual.pdf)

Archivo PDF que contiene una serie de páginas prefijadas (probablemente portadas, contraportadas, u otras páginas especiales que no se vayan a introducir mediante los ficheros XML de DocBook XSL *src/UserManual.xml* (dentro del módulo *docs* del proyecto).

Las reglas para su introducción en el documento *User_Manual.pdf* (manual de implantación de la aplicación desarrollada) final están en RooAddonsData/docPlugins.

Si este archivo no está presente, RooAddonsData/docPlugins debe estar vacío, o los plug-ins allí contenidos no obtener ninguna página del archivo *front_pages/User_Manual.pdf*.

RooAddonsData/docPlugins

[*pluginname*.xml]

Ninguno, uno, o varios ficheros XML conteniendo plug-ins de Maven que serán incluidos en el fichero *pom.xml* correspondiente al módulo *docs* del proyecto. La función de estos plug-ins será determinar el modo en que se genere la documentación del proyecto en desarrollo a partir de lo que se haya colocado en los archivos de configuración custom.xsl, Installation_Manual.pdf, User_Manual.pdf y fonts.xml, y de lo que, una vez se haya generado el proyecto, se haya introducido en los archivos *src/InstallationManual.xml* y *src/UserManual.xml*, dentro del módulo *docs* del proyecto. El objetivo de este/estos plug-in/s es el de generar los correspondientes *Installation_Manual.pdf* y *User_Manual.pdf*, que contendrán la documentación final del software en desarrollo orientada al implantador y al usuario, respectivamente. Si no se incluye ningún plug-in, los dos manuales serán solamente los documentos PDF generados por DocBook XSL a partir de *src/InstallationManual.xml* y *src/UserManual.xml*.

La estructura de un fichero XML que contenga un plug-in básico para este propósito ha de ser la siguiente:

```
<plugin>
	<groupId>com.organizacion.maven.plugins</groupId>
	<artifactId>maven-documentation-plugin</artifactId>
	<version>1.0</version>
	<configuration>
		<builds>
			<BuildDocument>
				<inputFiles>
					<PdfConfiguration>
	<file>src/front_pages/Installation_Manual.pdf</file>
		<pagesPattern>1-2</pagesPattern>
					</PdfConfiguration>
					<PdfConfiguration>
						<file>target/help-
pdf/InstallationManual.pdf</file>
						<pagesPattern>3-*</pagesPattern>
					</PdfConfiguration>
					<PdfConfiguration>

<file>src/front_pages/
Installation_Manual.pdf</file>
	<pagesPattern>3-3</pagesPattern>
					</PdfConfiguration>
				</inputFiles>

	<outputFile>target/Installation_Manual.pdf</outputFile>
			</BuildDocument>
			<BuildDocument>
				<inputFiles>
					<PdfConfiguration>

	<file>src/front_pages/User_Manual.pdf</file>
						<pagesPattern>1-2</pagesPattern>
					</PdfConfiguration>
					<PdfConfiguration>
						<file>target/help-
pdf/UserManual.pdf</file>
						<pagesPattern>3-*</pagesPattern>
					</PdfConfiguration>

<PdfConfiguration>
					<file>src/front_pages/User_Manual.pdf</file>
					<pagesPattern>3-3</pagesPattern>
					</PdfConfiguration>
				</inputFiles>
				<outputFile>target/User_Manual.pdf</outputFile>
			</BuildDocument>
		</builds>
	</configuration>
	<executions>
		<execution>
			<id>generate-full-am-help</id>
			<phase>generate-resources</phase>
			<goals>
				<goal>compose-pdf</goal>
			</goals>
		</execution>
	</executions>
</plugin>
```

Este ejemplo asume que los dos archivos *Installation_Manual.pdf* y *User_Manual.pdf*, situados en *RooAddonsData/doc/front_pages*, tienen ambos 3 páginas, correspondientes, respectivamente, a las 2 primeras y a la última de los documentos finales de sus mismos nombres que se generarán , siendo el resto de su contenido creado automáticamente a partir de lo indicado en los ficheros de configuración *fonts.xml* y *custom.xsl*, y del texto incluido en

los archivos *src/InstallationManual.xml* y *src/UserManual.xml*, dentro del módulo *docs* del proyecto en desarrollo.

RooAddonsData/layout

Directorio que contiene diversos elementos que serán copiados al módulo web del proyecto en desarrollo (concretamente, dentro de *src/main/webapp/WEB-INF*) para que éste adquiera el *layout* (apariencia y distribución de los elementos en la interfaz) deseado. Su contenido se encuentra estructurado en una serie de subdirectorios:

- **classes:** archivos *.properties* que seleccionar estilos diferentes para modificar la apariencia de la interfaz web de la aplicación. Un ejemplo de su posible contenido es: `styleSheet=resources/styles/hojaDeEstilos.css`
- **images:** archivos de imágenes (JPEG, PNG, etc.).
- **layouts:** archivos JavaServer Pages (*.jspx*), cada uno de los cuales contiene un modo de distribuir los diferentes elementos en la página web.
- **properties:** archivos *.properties* que serán introducidos en el directorio de *i18n*, que indican diferentes propiedades relacionadas con la internacionalización y localización de la aplicación.
- **scripts:** archivos JavaScript (*.js*) que se desea se ejecuten dentro de la aplicación web desarrollada.
- **styles:** archivos que contienen Cascading Style Sheets (*.css*) que determinarán la apariencia de la interfaz de la aplicación web.
- **tags:** archivos de tags de JavaServer Pages (*.tagx*), que contienen definiciones de elementos a mostrar en la interfaz web. Habitualmente se encontrarán a su vez en subdirectorios, dentro de *tags*, según el tipo de elemento (menú, formulario, etc.) de que se trate. Ver los archivos de este creados automáticamente junto con el módulo web para obtener más información.

Además, en *RooAddonsData/layout* se encuentra el fichero **load-scripts.jspx**, que ha de contener, por cada JavaScript que haya en el directorio *scripts*:

```
<spring:url value="/resources/scripts/script.js" var="script_url" />
<script src="${script_url}" type="text/javascript"><!-- required for FF3 and
Opera --></script>
```

Y por cada CSS que haya en el directorio *styles*:

```
<spring:url value="/resources/styles/estilo.css" var="estilo_url" />
<link rel="stylesheet" type="text/css" href="${estilo_url}" />
```

(No se trata de un archivo JSPX (JavaServer Pages) completo, tan sólo de estas líneas que serán introducidas en el archivo del mismo nombre, dentro del proyecto en desarrollo).

(RooAddonsData/usermgmt)

Este directorio sólo ha de existir en caso de que se desee hacer uso de un módulo externo de gestión de usuarios, y nunca si lo que se quiere es llevar a cabo la gestión de usuarios directamente desde la propia aplicación en desarrollo, mediante Spring Security y CAS (Central Authentication Service). Su función es configurar cómo se ha de llevar a cabo la gestión de

usuarios mediante el módulo externo. **La existencia o no existencia de este directorio determinará el tipo de gestión de usuarios que se llevará a cabo cuando se ejecute el comando Roo++ correspondiente.**

application.properties

Todas las propiedades que sea necesario añadir al archivo de este mismo nombre, dentro del módulo web de la aplicación en desarrollo, para hacer uso del módulo de gestión de usuarios. Por ejemplo, su contenido podría ser algo como:

```
# Remote security
AUTHENTICATION.REQUIRED=true

# Autenticacion para servicios remotos
REMOTE.USER.USERNAME=user
REMOTE.USER.PASSWORD=password
PASSWORD.ENCRYPT.TYPE=SIMPLE

# Properties file with server URL settings for remote access
cas.securityContext.service.url=${REMOTE.HOST}/${APP.NAME}
cas.securityContext.serviceProperties.service=${REMOTE.HOST}/${APP.NAME}/j_spr
ing_cas_security_check
cas.securityContext.ticketValidator.casServerUrlPrefix=${REMOTE.USERMODULE.HOS
T}/${USERMGMT.NAME}
cas.securityContext.casAuthenticationFilterEntryPoint.loginUrl=${cas.securityC
ontext.ticketValidator.casServerUrlPrefix}/login
cas.securityContext.casAuthenticationFilterEntryPoint.logoutUrl=${cas.security
Context.ticketValidator.casServerUrlPrefix}/j_spring_security_logout
cas.securityContext.remoteUsersService=${REMOTE.USERMODULE.HOST}/${USERMGMT.NA
ME}/remoting/remoteUsersService
cas.securityContext.remotePrincipalService=/${USERMGMT.NAME}/remoting/remotePr
incipalService
cas.securityContext.remoteUserDetailsService=${REMOTE.USERMODULE.HOST}/${USERM
GMT.NAME}/remoting/remoteUserDetailsService
cas.securityContext.remoteAuthenticationManagerService=${REMOTE.USERMODULE.HOS
T}/${USERMGMT.NAME}/remoting/remoteAuthenticationService

# Role security
application.security.role=ROLE_ADMIN
```

Este contenido dependerá de aspectos tales como el nivel de seguridad que se le quiera dar a la aplicación, o las características particulares que tenga el módulo de gestión de usuarios utilizado.

applicationContext-security.xml

Fichero XML que contiene los beans que sean necesarios para que el módulo web de la aplicación desarrollada utilice la gestión de usuarios proporcionada por un módulo externo.

Se trata de un fichero de tipo *applicationContext.xml*, utilizados en muchas ocasiones en Spring. Un ejemplo de la estructura de uno de estos archivos (sin relación con este caso particular de la gestión de usuarios) sería éste.

artifcactItems.txt

Contiene los diferentes elementos artifactItem que han de ser introducidos en el fichero *pom.xml* del módulo *package* del proyecto, con el fin de empaquetar diferentes elementos relativos al de gestión de usuarios, cuando ésta se hace a través de un módulo externo en lugar de con las librerías de Spring Security.

Su contenido ha de ser similar al siguiente:

```
usermgmt-application
war
${usermgmt.delivery}.war
usermgmt-doc
zip
${usermgmt.delivery}-docs.zip
usermgmt-scripts
zip
${usermgmt.delivery}-scripts.zip
```

En este ejemplo vemos que hay 3 *artifactItems* diferentes, y para cada uno de ellos se indica en una línea diferente el valor de los campos *artifactId*, type y destFileName.

changePassword.jspx

Archivo JavaServer Pages (*.jspx*) que introduce en la interfaz web de la aplicación en desarrollo la posibilidad permitir usuario que la está usando cambiar su contraseña.

Su contenido podría ser similar a:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div id="user_password" xmlns:jsp="http://java.sun.com/JSP/Page"
	xmlns:userInfo="http://www.empresa.com/usermgmt/userInfo"
	xmlns:spring="http://www.springframework.org/tags"
	class="body_info"
	version="2.0">
	<jsp:output omit-xml-declaration="yes" />

		<fieldset>
			<legend>
				<spring:message
					code="label_usermgmt_user_username"
					arguments="${userPasswordChange.username}" />
			</legend>
			<userInfo:changePassword />
		</fieldset>

</div>
```

userInfo.jpsx

Archivo JavaServer Pages (*.jspx*) que introduce en la interfaz web de la aplicación en desarrollo la posibilidad de mostrar una página con el perfil del usuario que la está usando en cada momento.

Su contenido podría ser similar a:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div id="user_info" xmlns:jsp="http://java.sun.com/JSP/Page"
	xmlns:userInfo="http://www.empresa.com/usermgmt/userInfo"
	xmlns:spring="http://www.springframework.org/tags"
	class="body_info"
	version="2.0">
	<jsp:output omit-xml-declaration="yes" />
```

```
            <fieldset>
                  <legend>
                        <spring:message
                              code="Label_usermgmt_user_username"
                              arguments="${user.username}" />
                  </legend>
                  <userInfo:userInfo />
            </fieldset>
</div>
```

usermgmt.txt

Contiene 5 líneas, que definen, respectivamente, los campos *groupId*, *artifactId* y *version* del módulo de gestión de usuarios, la URL completa (incluyendo en la misma el número de puerto correspondiente) en la que se encuentra dicho módulo, y el nombre de la aplicación de gestión de usuarios que este módulo proporciona (este último valor será asignado a la variable `${usermgmt.delivery}`).

Su contenido ha de ser algo tal como:

```
com.empresa.usermgmt
usermgmt-client
1.0
http://maquina.empresa.com:8080
appname
```

ANEXO I: Ejemplo de orden de ejecución de los comandos de Roo++

Los comandos de Roo++, respetando siempre el orden de ejecución indicado en la documentación de cada uno de ellos, pueden ejecutarse en cualquier orden. Roo no permite la ejecución de un comando en un instante en que ésta no es posible.

Un posible orden de ejecución habitual, en un caso en que se ejecuten todos los comandos, sería:

1. *project* (comando estándar de Roo; no forma parte de Roo++)
2. *roo++ project create-modules*
3. *roo++ project web configuration*
4. *roo++ project web jpa*
5. *roo++ project web deploy*
6. *roo++ platform setup*
7. *roo++ platform configuration*
8. *roo++ usermgmt setup*
9. *roo++ portal setup*
10. *roo++ layout setup*
11. *roo++ web breadcrumb setup*
12. *roo++ context setup*

ANEXO II: Ejemplo de Base de Datos

Cuando la aplicación desarrollada con Roo++ haga uso de una base de datos, será necesario definir sus tablas y sus campos desde Roo.

Si la aplicación desarrollada se ha de encargar de la gestión de una clínica veterinaria, la estructura de su base de datos podría ser:

TABLE: owners
PRIMARY KEY id
name

TABLE: types
PRIMARY KEY id
name

TABLE: pets
PRIMARY KEY id
FOREIGN KEY type_id references the types table id field
FOREIGN KEY owner_id references the owners table id field
name

TABLE: vets
PRIMARY KEY id
name

TABLE: specialties
PRIMARY KEY id
name

TABLE: vet_specialties - a link table for vets and their specialties
FOREIGN KEY vet_id references the vets table id field
FOREIGN KEY specialty_id references the specialties table id field

TABLE: visits
PRIMARY KEY id
FOREIGN KEY pet_id references the pets table id field
description

En este caso, los comandos de Roo correspondientes para definir esta base de datos serían:

```
entity jpa --class ~.domain.Owner --testAutomatically
field number --fieldName id --type int --notNull
field string --fieldName name --notNull --sizeMin 2
entity jpa --class ~.domain.Type --testAutomatically
field number --fieldName id --type int --notNull
field string --fieldName name --notNull --sizeMin 2
entity jpa --class ~.domain.Pet --testAutomatically
field number --fieldName id --type int --notNull
field string --fieldName name --notNull --sizeMin 2
```

```
field reference --fieldName type --type ~.domain.Type
field reference --fieldName owner --type ~.domain.Owner
entity jpa --class ~.domain.Vet --testAutomatically
field string --fieldName name --notNull --sizeMin 2
entity jpa --class ~.domain.Specialty --testAutomatically
field string --fieldName name --notNull --sizeMin 2
entity jpa --class ~.domain.Vet_specialties --testAutomatically
field reference --fieldName vet_id --type ~.domain.Vet
field reference --fieldName specialty_id --type ~.domain.Specialty
entity jpa --class ~.domain.Visit --testAutomatically
field reference --fieldName pet_id --type ~.domain.Pet
field string --fieldName description --notNull --sizeMin 2
```