gvNIX - Documentación de referencia

1.5.1.RC1

Parte I. Primeros pasos con gvNIX

Esta parte de la documentación contiene la información necesaria para entender qué es gvNIX y cómo empezar a utilizarlo.

Capítulo 1. Introducción

1.1. ¿Qué es gvNIX?

gvNIX es un entorno de trabajo Java de código abierto para el desarrollo rápido de aplicaciones web altamente productivo, flexible y que no compromete la calidad de los proyectos.

Está compuesto de un conjunto de herramientas de código abierto entre las que destaca su núcleo, Spring Roo. Esto le proporciona un amplio apoyo por parte de importantes organizaciones como Spring Source y VMWare. Añade funcionalidades de alto nivel a las prestaciones que ya se obtienen con Spring Roo para mejorar la productividad.

gvNIX es un proyecto subvencionado y liderado por la Consejería de Infraestructuras, Transporte y Medio Ambiente (CITMA) de la Generalidad Valenciana.

gvNIX está disponible dentro del proyecto de Migración a Software de fuentes abiertas gvPONTIS.

- Comunidad online de gvNIX
- Proyecto gvNIX en Google Code
- Página de gvNIX dentro del proyecto gvPONTIS
- Página oficial de Spring Roo

1.2. ¿Qué es Spring Roo?

Según su creador, Ben Alex: "Roo es un pequeño genio que observa desde un segundo plano y gestiona todo aquello de lo que no me quiero preocupar".

Spring Roo es un entorno de trabajo Java que permite el desarrollo de aplicaciones web de forma rápida y cómoda para el desarrollador. Sus principales características son:

- Generación de código en Java (lenguaje estático).
- Eliminar el trabajo tedioso centrando el desarrollo en la lógica de negocio.
- Convención sobre configuración.
- Desarrollo dirigido por el dominio (Domain-Driven Development):
  - Diseño dirigido por el modelo de entidades.
  - Lógica en las entidades (Real Object Oriented), eliminando capas redundantes.
  - Otras capas opcionales (servicios, DAOs, ...).
- Crea un proyecto en segundos.
- Realimentación: añade valor durante todo el ciclo de vida.
- No incorpora elementos adicionales al entorno de ejecución, por lo que no penaliza la velocidad ni la memoria de la aplicación.

- No requiere ningún IDE.
- Recibe instrucciones a través de una consola interactiva con autocompletado y ayuda en línea.
- Extensible usando *bundles OSGi*.
- Aprovecha el conocimiento: no necesita más conocimiento que el necesario para el desarrollo de aplicaciones JEE.

1.3. ¿Qué ofrece gvNIX?

gvNIX aprovecha las características de Spring Roo y aporta su propia filosofía, ofrecer al desarrollador componentes de alto valor funcional para aplicaciones corporativas:

- Control de concurrencia a nivel de aplicación sin campos en base de datos.
- Utilidades de generación de consultas.
- Utilidades de modificaciones múltiples de entidades.
- Gestión de temas visuales.
- Gestión de la visualización de excepciones.
- Gestión de mensajes de usuario en ventana modal.
- Gestión de la estructura de páginas en el menú.
- Gestión de patrones de visualización de entidades y sus relaciones.
- Gestión de transformación de cadenas vacías a valores nulos.
- Gestión de servicios locales y servicios web (importación y exposición).
- Control de acceso (autenticación y autorización).
- Gestión de configuraciones por entorno.
- Generación de documentos (reportes).

1.4. Arquitectura

gvNIX se centra en el desarrollo de aplicaciones Java para entornos corporativos.

La arquitectura de Roo y gvNIX se subdivide en 2 grandes bloques: el entorno de desarrollo y el entorno de ejecución

- El entorno de desarrollo incorpora distintas herramientas enfocadas al desarrollo rápido de aplicaciones Java. La característica más importante del entorno de desarrollo es que no introduce ningún tipo de librería propia, de tal forma que en tiempo de ejecución no añade sobre coste alguno al rendimiento a las aplicaciones.
- El entorno de ejecución es un entorno típico de aplicaciones basadas en Spring 3:
  - Acceso a bases de datos relacionales usando el API Java Persistence (JPA).

- Inyección de dependencias.
- Gestión de transacciones propia de Spring.
- Pruebas unitarias con JUnit.
- Configuración Maven para la construcción de las aplicaciones.
- Vistas JSP usando Spring MVC. Además, se pueden utilizar otras tecnologías de visualización tales como Flex, GWT, JSF y Vaadin.

Las aplicaciones basadas en Spring son probablemente las aplicaciones Java más populares seguidas de JSF, Struts y GWT según el Developer Productivity Report 2012.

Es importante destacar que Roo y gvNIX no imponen ninguna restricción sobre el tipo de aplicaciones que se pueden generar con este entorno. Algunos ejemplos de las funcionalidades que se pueden cubrir fácilmente con la versión actual son (notar que no están limitadas únicamente a estas):

- Intercambio de mensajes por JMS o envío por SMTP.
- Capa de servicios opcional con posibilidad de acceso remoto para clientes RIA.
- Ejecución de acciones predefinidas contra la base de datos.

Una de las mayores diferencias entre Roo/gvNIX y las aplicaciones tradicionales generadas a mano es que, por defecto, no añade capas de abstracción innecesarias. Las aplicaciones Java más tradicionales tienen una capa DAO, capas de servicios, capa de dominio y una capa de control. En una aplicación generada con Roo/gvNIX inicialmente solo se usa una capa de Entidad (que es similar a la capa de dominio) y una capa Web. Se podrá generar también una capa de Servicios y/o DAO si fuese necesario.

A continuación se puede ver un diagrama de los componentes de la arquitectura, capas y tecnologías relacionadas en Roo/gvNIX:

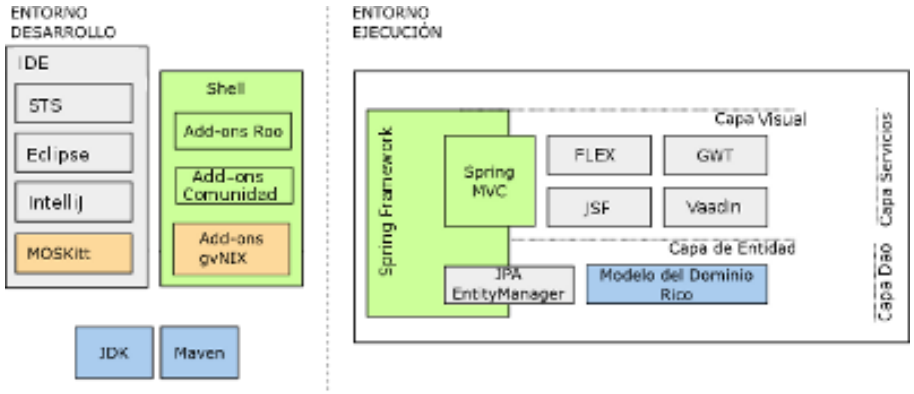

Figura 1.1. Diagrama Arquitectura

1.5. El intérprete de comandos

Para que el entorno empiece a funcionar, hay que arrancar un intérprete de comandos (*Roo shell*) mediante el que se interactuará con el entorno de desarrollo. Este intérprete de comandos o consola tiene dos funciones básicas:

- Ejecutar las órdenes indicadas por el desarrollador.

- Monitorizar cualquier cambio en el proyecto para reajustar el código generado.

La consola tiene el siguiente aspecto:

```
    ____  ____  ____
   / __ \/ __ \/ __ \
  / /_/ / / / / / / /
 / _, _/ /_/ / /_/ /   gvNIX x.x.x distribution
/_/ |_|\____/\____/    x.x.x-RELEASE [rev xxxxxxx]

Welcome to Spring Roo. For assistance press TAB or type "hint" then hit ENTER.
roo-gvNIX>
```

La usabilidad es un factor clave en el entorno de trabajo. Entre las principales características de usabilidad que proporciona la consola se encuentran:

- Comandos *hint* y *help*: Guían al usuario sobre la forma de proceder
- *TAB* para completar prácticamente cualquier comando.

  Pulsando *TAB* Roo sugiere que hacer a continuación, que comando se puede utilizar a continuación, incluso pulsando algún caracter qué comandos que empiezan por ese caracter se pueden utilizar.

- Si se produce algún error en alguna operación, Roo deshará todos los cambios generados dejando la aplicación como se encontraba inicialmente.
- Ejecución de *scripts*: Se puede guardar una secuencia de comandos en un archivo de texto y ejecutarlos en bloque cuantas veces se necesite. Por ejemplo, se puede hacer un guión para crear una aplicación completa y ejecutarlo en el intérprete de Roo.
- Funcionamiento predictivo y conservador: Al ejecutarse en segundo plano debe ser un entorno conservador en el sentido que si no cambia nada en el proyecto no debe tocar nada, el desarrollador no debe perder nunca el control del proyecto.

1.6. ¿Cómo funciona?

Este diagrama muestra a alto nivel el modelo funcional de Roo y gvNIX:

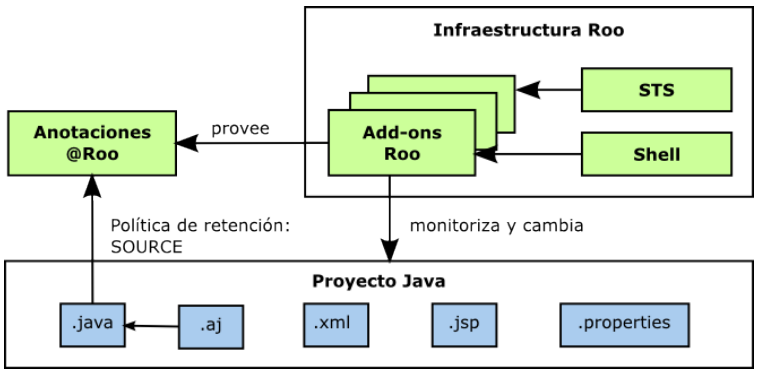

Figura 1.2. Diagrama de modelo funcional de Roo y gvNIX

- Anotaciones

  Conjunto de anotaciones JAVA que utilizan Roo y gvNIX para poder identificar un tipo de artefacto concreto y su configuración. Tienen una política de retención *SOURCE* por lo que el compilador Java no las incluirá en la clases compiladas.

- Proyecto Java

  Es el contenido de nuestra aplicación formada por todos los archivos necesarios.

- Infraestructura de Roo/gvNIX

  Toda la infraestructura de Roo/gvNIX que forma el entorno de desarrollo, autocontenida y separada del proyecto final. Se compone de:

  - Add-ons

    Componentes que automáticamente monitorizan y realizan cambios en nuestro proyecto.

    También ofrecen un conjunto de comandos para que los desarrolladores puedan realizar operaciones a través del intérprete de comandos.

  - Intérprete de comandos (Shell)

    Componente que permite la recepción y ejecución de órdenes del desarrollador

  - STS: SpringSource Tool Suite

    Plugin/Aplicación Eclipse de SpringSource para trabajar con proyectos Spring. Entre otras, dispone de herramientas para trabajar con Roo/gvNIX.

1.7. Generación de código

Roo y gvNIX es un generador de código híbrido, puesto que intenta combinar lo mejor de los modelos de generación pasiva y activa.

- Generación pasiva:

  El desarrollador indica a la herramienta qué es lo que debe generar, y una vez generado finaliza el trabajo de la herramienta. No se asume un mantenimiento a largo plazo del código generado. El desarrollador interactúa mediante el intérprete de comandos, desde ahí se realiza y termina el proceso de generación de código, que genera archivos xml y java.

- Generación activa:

  Es una realimentación automática que genera un modelo detallado de metadatos con la ayuda de las anotaciones @Roo y @GvNIX e incrementalmente actualiza archivos aj y jsp. Permite mantener automáticamente elementos del proyecto en respuesta a cambios, este tipo de generación nunca modifica archivos java, estos solo se modifican por indicación expresa del desarrollador y solo a través del intérprete de comandos.

Capítulo 2. Instalación de gvNIX

2.1. Requisitos previos

gvNIX es en si mismo una aplicación Java por lo que requiere tener una máquina virtual Java instalada en el sistema.

La lista de requisitos es la siguiente:

- Versión más actual de Java JDK 6 instalado en el sistema (Más información).
- Maven 3.0.x o superior (http://maven.apache.org/).
- Conexión a Internet para la descarga de las dependencias de los proyectos.

2.1.1. Configuración del entorno

La configuración del entorno solo será necesario si se va a utilizar gvNIX desde la consola del sistema. Por lo tanto, no será necesario realizar todos los pasos indicados en este punto si se va a utilizar gvNIX desde Eclipse o STS tal y como se detallará posteriomente.

Una vez descargado el archivo de instalación de Java JDK y Maven veamos como configurar el entorno de trabajo para usar estas herramientas.

Nota

Las siguientes notas sobre configuración son una recomendación de buenas prácticas. La variable *$HOME* hace referencia al path del directorio de raíz de usuario (por ejemplo: */home/usuario*). Se supondrñá instalado Java JDK y Maven en:

1. Java JDK 1.6 instalada en: */home/usuario/software/jdk1.6.0_35*
2. Maven 3 instalado en: */home/usuario/software/apache-maven-3.0.4*

- **Variables de entorno**.

  En entornos Unix, se puede utilizar *$HOME/.bashrc* para definir las variables de entorno y automáticamente se cargarán al abrir una nueva consola. Es una buena práctica definirlas en nuestro propio archivo y cargarlo desde el *$HOME/.bashrc*, en este caso dicho archivo se llamará *$HOME/.bash_devel*.

  En sistemas Windows, establecer en las propiedades de "Mi PC" únicamente las variables de entorno que se definen a continuación en el segundo punto.

  - **Modificar $HOME/.bashrc para que cargue el archivo $HOME/.bash_devel**:

    Añadir al final del archivo .bashrc las siguientes líneas:

    ```
    # Development settings
    if [ -f ~/.bash_devel ]; then
       . ~/.bash_devel
    fi
    ```

  - **Crear/Modificar $HOME/.bash_devel para añadir las variables de entorno**:

```
export JAVA_HOME=$HOME/software/jdk1.6.0_35

export M2_HOME=$HOME/software/apache-maven-3.0.4

export M2=$M2_HOME/bin

export MAVEN_OPTS="-Xmx1024m -XX:MaxPermSize=512"

export PATH=$JAVA_HOME/bin:$M2:$PATH
```

Con estos cambios cada vez que se abra una consola del sistema estarán cargadas las variables de entorno. También se puede cargar los cambios realizados en estos ficheros ejecutando el comando *source* en la consola de entornos Unix:

```
bash:~$ source .bashrc
```

Una vez realizados estos cambios se puede comprobar su funcionamiento: (Ejecutando los comandos siguientes veremos salidas similares a las que se muestran)

```
bash:~$ java -version
java version "1.6.0_35"
Java(TM) SE Runtime Environment (build 1.6.0_35-b04)
Java HotSpot(TM) 64-Bit Server VM (build 20.8-b03, mixed mode)
bash:~$ mvn -version
Apache Maven 3.0.4 (r1056850; 2012-01-09 17:58:10+0100)
Java version: 1.6.0_35, vendor: Sun Microsystems Inc.
Java home: /home/usuario/software/jdk1.6.0_35/jre
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "3.2.0-30-generic", arch: "amd64", family: "unix"
```

2.1.2. Acceso a través de un proxy

Si la conexión a Internet se realiza a través de un proxy, habrá que configurar tanto Java como Maven para que puedan acceder al exterior correctamente.

- **Java a través de proxy**:

  Es posible que la red en la que se esté trabajando requiera de la configuración de un Proxy para el acesso a internet.

  gvNIX/Roo dispone de un comando que indica si la instalación de Java está, o no, configurada para conectar a Internet a través de un proxy.

  ```
  roo-gvNIX> proxy configuration
                       *** Your system has no proxy setup ***
  http://download.oracle.com/javase/6/docs/technotes/guides/net/proxies.html offers
  useful information.
  For most people, simply edit /etc/java-6-openjdk/net.properties (or equivalent) and
  set the java.net.useSystemProxies=true property to use your operating system-defined
  proxy settings.
  ```

  En el ejemplo anterior indica que no hay ningún proxy configurado. Si es necesario hacerlo se debe seguir la guía tal y como indica la salida del comando.

  Si se han seguido las recomendaciones para instalar Java, ir a /home/usuario/software/jdk1.6.0_35/jre/lib. Aquí estará el archivo net.properties que se debe editar y configurarlo según las necesidades.

La opción más sencilla es establecer la propiedad *java.net.useSystemProxies=true*. Esto le indica a Java que debe utilizar la configuración del proxy que previamente se haya definido en el sistema operativo. Sin embargo puede no funcionar en todos los sistemas.

En caso de no funcionar lo anterior, definir la configuración específica del proxy buscando las propiedades descritas a continuación y añadiéndolas si no existen:

```
java.net.useSystemProxies=false

http.proxyHost=host.proxy.de.red (ejemplo: proxy.mired.com)
http.proxyPort=puerto (ejemplo: 8080)
http.nonProxyHosts=hosts.a.ignorar.1|host.a.ignorar.2
 (ejemplo: localhost|127.0.0.1|192.168.1.*|*.mired.com)

https.proxyHost=host.proxy.de.red (ejemplo: proxy.mired.com)
https.proxyPort=puerto (ejemplo: 8080)
https.nonProxyHosts=hosts.a.ignorar.1|host.a.ignorar.2
 (ejemplo: localhost|127.0.0.1|192.168.1.*|*.mired.com)
```

Si se configura alguna de estas propiedades el comando *proxy configuration* informará de los valores configurados.

- **Maven a través de proxy:**

Algunos comandos de gvNIX/Roo utilizan la herramienta Maven y dicha herramienta en algunos casos necesita conexión con Internet para, por ejemplo, descargar las dependencias de los proyectos generados.

Es por ello que si se accede a Internet a través de un proxy se deberá configurar en el fichero $M2_HOME/conf/settings.xml las siguientes secciones en el lugar adecuado del fichero. Revisar las secciones comentadas porque existirá ya una sección de este tipo como ejemplo:

```
<proxy>
        <id>Proxyhttp</id>
        <active>true</active>
        <protocol>http</protocol>
        <host>host.proxy.de.red (ejemplo: proxy.mired.com)</host>
        <port>puerto (ejemplo: 8080)</port>
</proxy>
<proxy>
        <id>Proxyhttps</id>
        <active>true</active>
        <protocol>https</protocol>
        <host>host.proxy.de.red (ejemplo: proxy.mired.com)</host>
        <port>puerto (ejemplo: 8080)</port>
</proxy>
```

2.2. Descarga e instalación de gvNIX

1. Descargar (https://code.google.com/p/gvnix/downloads/list) y descomprimir el fichero ZIP de la versión de gvNIX más actual, por ejemplo gvNIX-X.Y.Z.RELEASE.zip

   - Ejemplo para sistemas Unix y Apple:

     ```
     bash:~$ unzip gvNIX-X.Y.Z.RELEASE.zip
     ```

2. Al descomprimir el fichero ZIP se creará un nuevo directorio *gvNIX-X.Y.Z.RELEASE*. En el resto del documento se hará referencia a este directorio como *GVNIX_HOME*

3. Incluir el directorio *bin* de gvNIX en la variable de entorno *PATH*.

   Solo será necesaria si se utiliza gvNIX desde la consola del sistema. Por lo tanto, no será necesario realizar todos los pasos indicados en este punto si se utiliza gvNIX desde Eclipse o STS tal y como se detallará posteriomente.

   - Ejemplo para sistemas Unix y Apple:

     ```
     bash:~$ cd gvNIX-X.Y.Z.RELEASE
     bash:~/gvNIX-X.Y.Z.RELEASE$ export PATH=$PWD/bin:$PATH
     ```

     Se puede definir la variable *GVNIX_HOME* y su inclusión en el PATH del sistema mediante el archivo *.bash_devel* y así tenerla disponible de manera permanente.

     Recordar que en sistemas Windows, se pueden establecer en las propiedades de "Mi PC" las variables de entorno que se definen a continuación.

     Para ello modificamos el archivo para que quede como sigue:

     ```
     ...
     export GVNIX_HOME=$HOME/software/gvNIX-X.Y.Z.RELEASE

     export PATH=$JAVA_HOME/bin:$M2:$GVNIX_HOME/bin:$PATH
     ```

     Notar que la última línea es la modificación de la definición de la variable *PATH*. Recordar recargar el fichero *.bash_devel* mediante el comando *source* de la consola de entornos Unix.

Una vez hecho esto, ya se puede trabajar con el entorno gvNIX desde la línea de comandos. Sin embargo, se recomienda utilizar un IDE de desarrollo que permita integrar el entorno gvNIX como, por ejemplo STS o Eclipse tal y como se verá en la sección posterior. Aún así, si se desea abrir el intérprete de comandos desde la línea de comandos se puede hacer de la siguiente forma:

- Cambiar al directorio donde se encuentre el proyecto Java existente o a un directorio vacío en el caso de tratarse de un proyecto nuevo:

  ```
  bash:~$ cd ~/project-directory
  ```

- Ejecutar el intérprete de comando de gvNIX para interactuar con el proyecto Java:

  ```
  bash:~/project-directory$ gvnix.sh

  (En sistemas windows el intérprete se abrirá con gvnix.bat)
  ```

- Con esto se abrirá la consola de gvNIX y se mostrará algo similar a lo siguiente:

  ```
     ____  ____  ____
       / __ \/ __ \/ __ \
      / /_/ / / / / / / /
     / _, _/ /_/ / /_/ /   gvNIX x.x.x distribution
    /_/ |_|\____/\____/    x.x.x-RELEASE [rev xxxxxxx]


    Welcome to Spring Roo. For assistance press TAB or type "hint" then hit ENTER.
    roo-gvNIX>
  ```

Capítulo 3. Desarrollo con Eclipse

3.1. Instalación

Spring ha desarrollado una extensión para Eclipse llamada SpringSource Tool Suite o STS, que ofrece soporte para trabajar en Eclipse con aplicaciones de Spring, entre ellas Spring Roo y por tanto gvNIX. Entre otras funcionalidades incorpora un intérprete de comandos de Roo que permite ejecutar órdenes sin necesidad de salir de Eclipse.

Hay dos formas de instalar STS, la primera es como una distribución propia y la segunda es como un plugin de un Eclipse ya existente. Se recomienda encarecidamente utilizar la primera de ellas por su facilidad de instalación y mejor rendimiento.

3.1.1. STS como distribución

Para instalar el IDE como una distribución propia realizar los siguientes pasos:

- Descargar la última versión de Springsource Tool Suite desde http://spring.io/tools
- Descomprimir el archivo descargado y moverlo a una carpeta, por ejemplo $HOME/software.
- Ejecutar STS ($HOME/software/springsource-X.X.X/sts-X.X.X.RELEASE/STS). Tener en cuenta que STS es una aplicación Java por lo que requiere tener una máquina virtual Java instalada en el sistema.

3.1.2. STS como plugin

Para instalar el IDE en un Eclipse existente seguir los siguientes pasos:

- Si no se dispone todavía de él, descargar Eclipse IDE for Java EE Developers de http://www.eclipse.org/downloads/
- Crear la variable de classpath de Eclipse M2_REPO, para ello:
  1. **`Window > Preferences > Java > Build Path > Classpath Variables`** , botón `New...` .
  2. Definir los siguientes valores:
     - `Name = M2_REPO` .
     - `Path` = directorio correspondiente al repositorio de Maven (normalmente en Unix: $HOME/.m2/repository)
  3. *Ok* para crear la variable, y *Ok* de nuevo para cerrar la ventana de preferencias y guardar cambios.
- Instalar plugin M2Eclipse para Eclipse (http://m2eclipse.sonatype.org/installing-m2eclipse.html)

  En el menú Help > Install New Software, en el campo de texto "Work with:" indicar que se usará el site: http://m2eclipse.sonatype.org/sites/m2e. A continución eleccionar "Maven Integration for Eclipse". Pulsar en siguiente y tras aceptar la licencia del plugin instalar.
- Activa la opción `Enable Window > Preferences > General > Workspace > Refresh Automatically` .

- Opcionalmente, instalar el plugin Subeclipse para trabajar sobre el sistema de control de versiones Subversion (SVN).

  Para ello en Eclipse, en el menú *Help > Install New Software*. En la ventana que se abre hacer click sobre el botón *Add* que hay a la derecha de *Work with*. Se abre un diálogo en el que se indicará como nombre Subeclipse por ejemplo y en la URL indicar *http://subclipse.tigris.org/update_1.6.x* guardar con *OK* y la ventana anterior empezará la carga de los plugins que hay disponibles. Seleccionarlos y seguir haciendo click en *Next*. La URL indicada es de la versión actual, revisar el sitio oficial de Subeclipse para más información y futuras versiones.

- Para instalar el plugin STS seguir las instrucciones del punto "UPDATE SITE INSTALLATION" del documento Spring Source Tool Suite Installation Instructions.

3.1.3. Integrar soporte ROO en STS

Spring Roo ya no está incluido en la distribución de STS a partir de su versión *3.6.0*, por ello necesitaremos instalar este soporte para poder trabajar con gvNIX.

Para incluir este soporte, sigue los siguientes pasos:

1. Abre tu STS
2. Abre el dashboard del STS y busca Spring ROO
3. Instala *Spring Roo (current production release)* y *Spring IDE - Roo Extension*
4. Reinicia tu STS

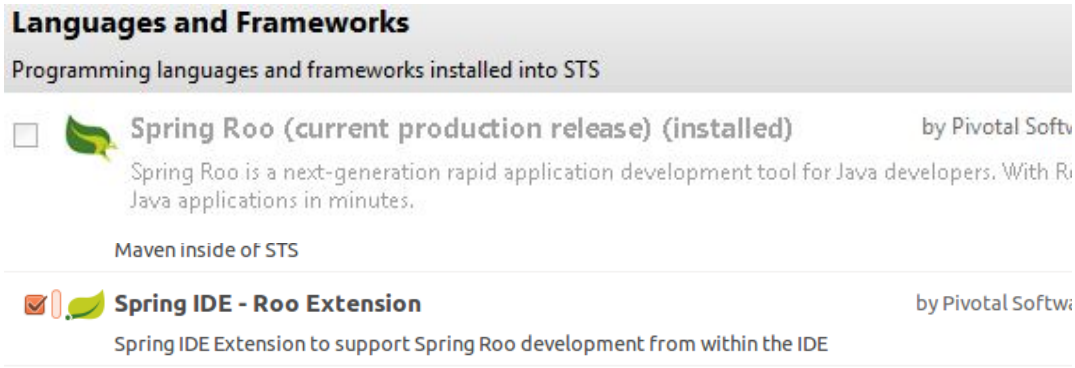

Figura 3.1. Integrando soporte ROO en STS

3.2. Consola gvNIX / Roo integrada en Eclipse

El plugin o distribución de STS instalado ofrece la posibilidad de incluir la consola de gvNIX en el propio entorno de desarrollo. Para ello antes hay que indicarle a Eclipse donde está instalada la nueva versión de gvNIX.

En el menú *Window > Preferences > Spring > Roo Support*. Pulsar sobre el botón Add y buscar en el navegador de archivos el lugar donde está instalado gvNIX. Aparecerá una pantalla similar a la siguiente:

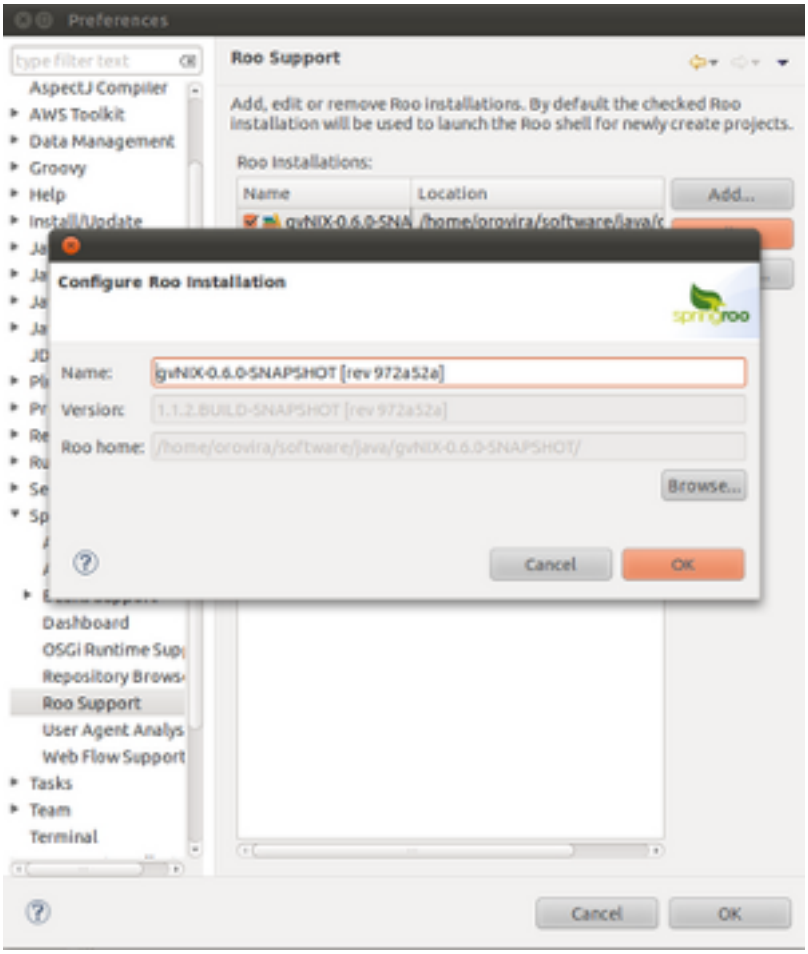

Figura 3.2. Configuración del soporte gvNIX en Eclipse

Presionando en OK sobre las dos pantallas que abiertas ya estará configurado el soporte para la linea de comandos de gvNIX. Para poder usarlo abrir la perspectiva de Spring con *Window > Open Perspective > Other > Spring*. En la parte inferior de la pantalla aparecerá una pestaña llamada Roo Shell. Si no apareciese la pestaña, puede abrirse seleccionando *Window > Show View > Roo Shell*.

Para abrir la consola pulsar en el botón que aparece en la parte superior derecha de la pestaña con el título *Open Roo Shell*. Entonces seleccionar el proyecto en el que se va a trabajar y ya se puede empezar a ejecutar comandos. Si no se dispone todavía de ningún proyecto, en el punto siguiente se verá como crear un nuevo proyecto.

La ejecución de comandos también se puede hacer de forma gráfica pulsando en el botón que aparece en la parte superior derecha de la pestaña con el título Open Roo Command Wizard.

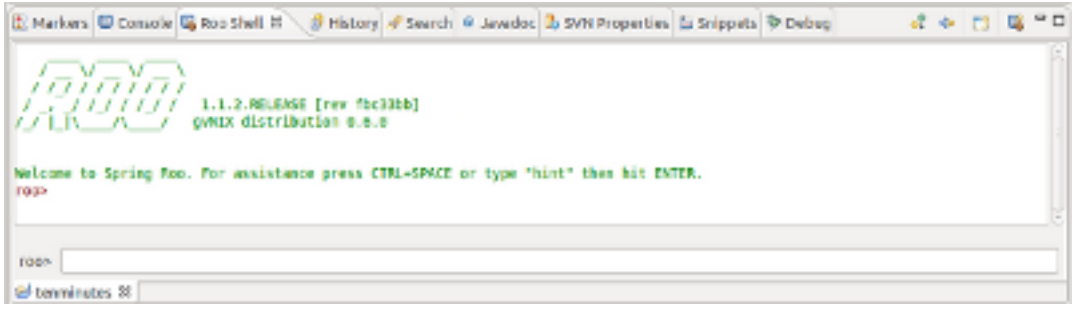

Figura 3.3. Consola gvNIX en Eclipse

Nota

Hay que tener en cuenta una pequeña diferencia a la hora de usar la línea de comandos desde Eclipse con respecto a la linea de comandos de gvNIX en la consola del sistema. Mientras en una consola del sistema se utiliza la tecla TAB para completar los comandos, en el IDE se utilizará Ctrl+Space que es el atajo por defecto usado en Eclipse.

3.3. Utilizar Eclipse como IDE

3.3.1. Crear proyecto gvNIX en Eclipse IDE

Crear un proyecto gvNIX desde Eclipse a través del menú: *File* > *New* > *Spring Roo Project*:

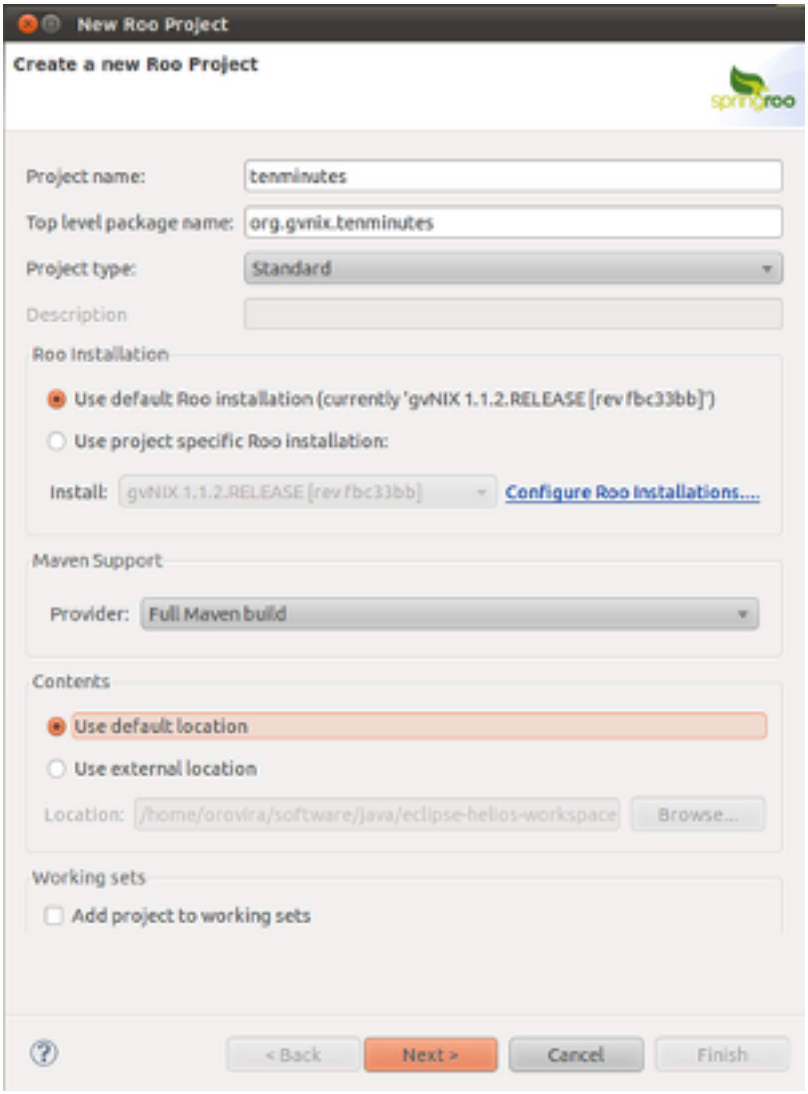

Figura 3.4. Nuevo proyecto Spring Roo con soporte gvNIX

Esta primera pantalla habrá ejecutado de forma automática el comando de creación de un nuevo proyecto:

```
project --topLevelPackage com.gvnix.tenminutes
```

En este punto ya se dispone de un proyecto Maven preparado.

A continuación, añadir el soporte para el manejo de la persistencia de datos a través de JPA. El siguiente ejemplo muestra los comandos que se deben usar para configurar el soporte JPA. Recordar que se puede usar Ctrl+Space como ayuda a la escritura en los comandos:

```
roo-gvNIX> jpa setup --provider HIBERNATE --database HYPERSONIC_IN_MEMORY
Created ...
roo-gvNIX> entity jpa --class ~.domain.Timer --testAutomatically
Created ...
~.domain.Timer roo-gvNIX> field string --fieldName message --notNull
Managed ...
```

El comando **jpa setup** permite configurar la capa de acceso a datos de manera sencilla. El comando **entitiy jpa** sirve para crear una nueva entidad y **field** crea atributos en la entidad. Estos comandos se verán con más detalle en la sección Persistencia de objetos y Crear una entidad con el intérprete de comandos, también se verá como generar todas las entidades de una aplicación mediante el proceso de ingeniería inversa contra un esquema de BBDD existente.

```
~.domain.Timer roo-gvNIX> web mvc setup
~.domain.Timer roo-gvNIX> web mvc all --package ~.web
Created ...
```

Los comandos **web mvc** crean la capa capa web que gestionaran la interacción del usuario con la aplicación. Se verán con más detalle en Crear la capa web con el intérprete de comandos.

```
~.web roo-gvNIX> selenium test --controller ~.web.TimerController
Created ...
```

Si se desea incluir en el proyecto los tests funcionales existe el comando **selenium test** que generará los scripts necesarios para probar la interfaz web de la aplicación. Más adelante se verá como poner en marcha la aplicación y como ejecutar estos tests.

```
~.web roo-gvNIX> theme install --id cit
~.web roo-gvNIX> theme set --id cit
```

Mediante el comando **theme**, del *Add-on Theme Manager* propio de gvNIX, se puede configurar el tema visual que presentará la aplicación.

3.3.2. Importar proyecto gvNIX en Eclipse

gvNIX/Roo permite crear proyectos desde la linea de comandos, aunque puede suceder que se necesite importar un proyecto ya existente en el IDE. Si se está ejecutando gvNIX en la línea de comandos, bastaría con ejecutar el comando **perform eclipse** para crear una estructura de proyecto Eclipse:

```
~.domain.PizzaOrder roo-gvNIX> perform eclipse
[Thread-6] Warning: JAVA_HOME environment variable is not set.
[Thread-6] [INFO] Scanning for projects...
[Thread-6] [INFO] ------------------------------------------------------------------------
[Thread-6] [INFO] Building pizzashop
[Thread-6] [INFO]    task-segment: [eclipse:clean, eclipse:eclipse]
[Thread-6] [INFO] ------------------------------------------------------------------------
[Thread-6] [INFO] [eclipse:clean {execution: default-cli}]
[Thread-6] [INFO] Deleting file: .project
[Thread-6] [INFO] Deleting file: .classpath
...
...
[Thread-6] [INFO] ------------------------------------------------------------------------
[Thread-6] [INFO] BUILD SUCCESSFUL
[Thread-6] [INFO] ------------------------------------------------------------------------
```

```
[Thread-6] [INFO] Total time: 4 seconds
[Thread-6] [INFO] Finished at: Wed Jun 16 21:21:49 CEST 2010
[Thread-6] [INFO] Final Memory: 36M/330M
[Thread-6] [INFO] ------------------------------------------------------------------------
~.domain.PizzaOrder roo-gvNIX>
```

Este comando termina ejecutando el comando del sistema**mvn eclipse:eclipse** en el proyecto, por lo que es indiferente el uso de cualquiera de los dos métodos. Tener en cuenta que este último comando necesita tener configurado Maven.

Ahora se puede importar en el entorno de trabajo mediante **File > Import > General > Existing Projects into workSpace**. Si aparece el mensaje *Turn Weaving Service on?*, marcar la casilla *Don't ask again until next upgrade* y clic en *Yes* (será necesario reiniciar Eclipse).

También es posible borrar la configuración de eclipse ejecutando el comando **mvn eclipse:clean** en una consola del sistema que tenga configurada Maven. Para regenerar la configuración de eclipse en un solo comando usar **mvn eclipse:clean eclipse:eclipse**.

Al importar un proyecto en Eclipse es importante comprobar que dicho proyecto tiene configuradas las características de proyecto Maven y proyecto de aspectos java. Esto aparece de forma visual en el proyecto mediante unas pequeñas letras M y AJ. Si no apareciesen se pueden añadir estas características seleccionando el proyecto y con el segundo botón del ratón eligiendo la opción *Configure* . Esto es muy importante ya que en el caso de no estar correctamente configurado nos aparecerán falsos errores en el proyecto.

3.4. Generación de pruebas de integración

El comando *entity jpa* dispone de la opción testAutomatically que al ser especificada generará los test de integración para dicha entidad usando JUnit.

Si los tests no fueron creados en el momento de la ejecución del comando *entity*, pueden ser generados con posterioridad mediante otro comando:

```
test integration --entity ~.domain.Timer
```

3.5. Arrancar la aplicación con Eclipse

Es posible arrancar la aplicación desde el propio Eclipse y, además de permitir depurarla, permite hacer cambios en caliente sobre la aplicación.

Para ello, debemos tener la aplicación importada en un *workspace* de Eclipse y configurar el servidor dónde ejecutarlo. Eclipse es capaz de gestionar distinto tipos de servidores, generando sus propios directorio y ficheros de configuración.

Para prepara un servidor seguir los siguiente pasos:

3.5.1. Mostrar la Vista de Servidores en Eclipse

Eclipse tiene una vista que permite ver los distintos servidores configurados y gestionarlos desde ahí. Estos son los pasos para mostrar la vista

1. Abra la opción del menú `Window > Show view > Other...`.

2. Busque la vista llamada `Servers` y pulse el botón `Ok`.

3.5.2. Crear un nuevo Servidor

Se pueden definir múltiples entornos de ejecución para los servidores. Esta configuración incluye el tipo de servidor (Tomcat, Jetty, JBoss, etc..), Máquina Virtual Java (JRE 1.5, JRE 1.6, etc...) y la ruta a los binarios del servidor.

Cabe destacar que eclipse **no usa la configuración que exista en la instalación del servidor** en la mayoría de los casos. Solo usa los ejecutables para arrancarlo con ficheros de configuración que almacena en el propio workspace.

En este caso, vamos a preparar el entorno para Tomcat 6 siguiendo los pasos indicados a continuación:

1. Dentro de la pestaña Servers, pinchar con el segundo botón del ratón y seleccionar New > Server.

2. Seleccionar `Apache Tomcat v6.0 Server` de la lista de tipos y pulsar el botón `Next`.

   En el caso en que no aparezca ningún tipo de servidor o no aparezcan los de Apache Tomcat se ha de realizar lo siguiente:

   - Ir a `Help > Install New Software...`.
   - Desmarcar la casilla "*Hide items that are already installed.*"
   - En "*Work With*" seleccionar el sitio "*Eclipse Web Tools Platform Repository - http://download.eclipse.org/webtools/updates/*" (si no aparece, añadirlo usando el botón `Add...`)
   - En el listado de software desplegar `Web Tools Platform Tests (WST Tests)` (si hay más de un *WST Tests* seleccionar el de la última versión).
   - Instalar (seleccionándolos) `WST Server tests` y `JST Server Tests` (si ya están instalado aparecerá su icono en gris)

3. Introducir un nombre para la configuración.

4. Seleccionar el directorio donde se encuentra una instalación de Tomcat 6. Si no se dispone de ninguna instalación de Tomcat 6 crear un nuevo directorio, seleccionarlo y pulsar sobre el botón Download and Install ... que pasados unos segundos terminará la descarga y permitirá continuar.

5. Seleccionar la máquina virtual Java con la que ejecutar el servidor.

6. Pulsar el botón Next.

7. Añadir las aplicaciones a ejecutar en este servidor de entre las disponibles.

3.5.3. Ajustar la configuración del Servidor

Puede ser necesario ajustar algunas opciones de la configuración del servidor. Para poder acceder a dichas opciones hay que seleccionar el servidor desde la pestaña de servidores y, con el botón derecho del ratón sobre él pulsar `Open`.

Desde el panel abierto podremos gestionar:

- Datos generales del servidor.
- Opciones de publicación y seguridad.
- TimeOuts (muy útil si se está depurando algún proceso de arranque).
- Puertos
- Configuración de tipos MIME.
- Las aplicaciones/modulos a lanzar (admite módulos externos).
- Configuración de arranque del servidor.

  El servidor Tomcat arranca con cierta cantidad de memoria que no soportar mas de 4 cambios en el proyecto antes de llenarse y dejar de funcionar. Para evitar esto, se puede incrementar la cantidad de memoria con la que arranca Tomcat.

Importante

Es muy interesante modificar los parámetros de memoría con los que trabaja el servidor. Para ello:

- En la pantalla de opciones de la configuración del servidor acceder a *Open launch configuration*
- En la nueva ventana, en la pestaña *Arguments* añadir al final del texo que aparece en el apartado *VM Arguments* lo siguiente: `-Xms64m -Xmx256m -XX:MaxPermSize=128m -XX:PermSize=128m`

3.5.4. Ejecución de la aplicación en el Servidor

Una vez configurado el servidor, se podrá añadir a este la aplicación que se desea ejecutar en él. Se puede ejecutar más de una a la vez. En la vista Servers, pulsar con el botón derecho sobre el servidor deseado y seleccionar la opción *Add and Remove ...* . En el diálogo que se abre seleccionar de la lista de la izquierda la aplicación y pulsar el botón *Add*. Por último, cerrar el diálogo con *Finish*.

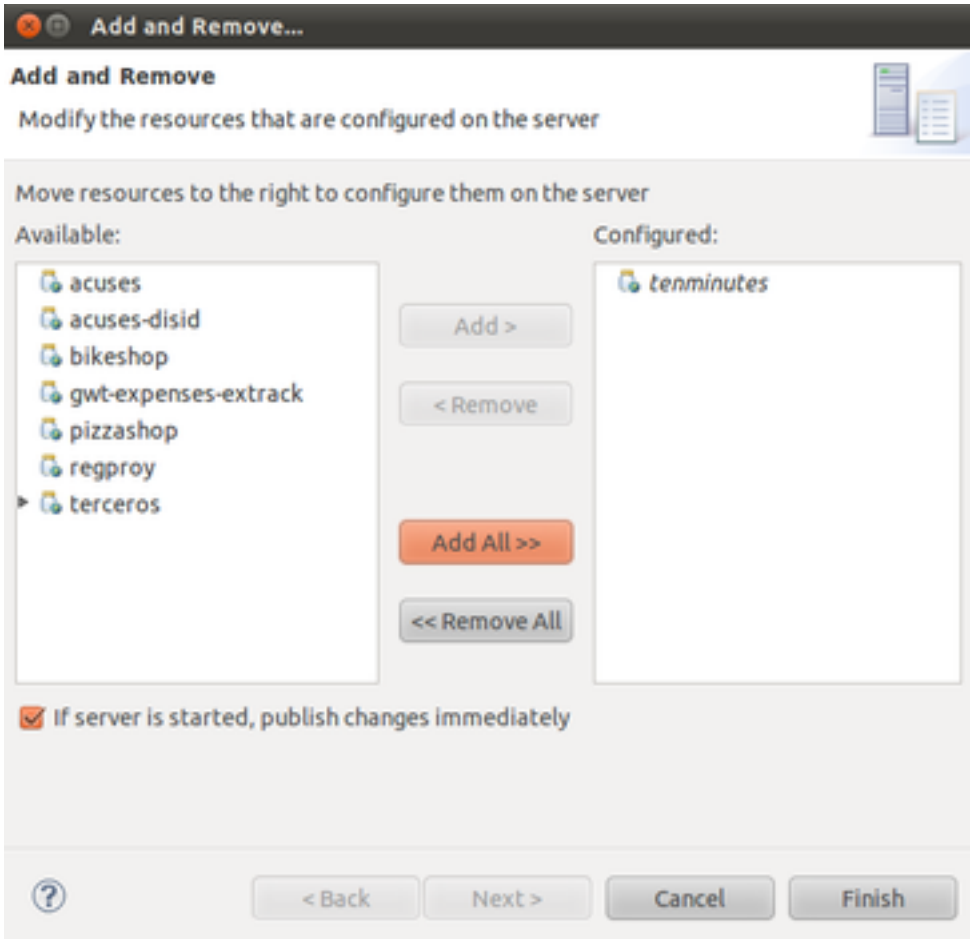

Figura 3.5. Añadir / Eliminar aplicaciones al Servidor

Ya es posible arrancar el servidor y probar la aplicación.

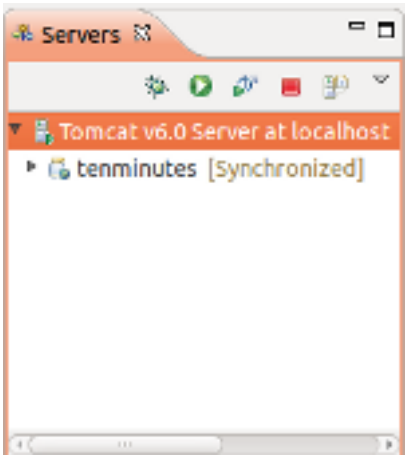

Figura 3.6. Vista de servidores disponibles en el workspace de Eclipse

En la imagen anterior se observa un servidor Tomcat 6 con la aplicación *tenminutes* disponible para ejecutar. Con los botones disponibles se puede arrancar la aplicación en modo debug, arrancar la aplicación de manera normal (botón verde con triángulo blanco en el centro), arrancar en modo profile, pararlo o re-publicar los cambios del proyecto en el servidor.

Figura 3.7. Botones de la vista servidores de Eclipse

Una vez arrancado el servidor se podrá navegar por la aplicación bien desde un navegador externo (Firefox) o bien desde un navegador propio que incorpora Eclipse accediendo a la dirección http://localhost:8080/tenminutes. Observar que el final de la dirección se corresponde con el nombre de la aplicación proporcionado al crear el proyecto. Para abrir el navegador interno de Eclipse utilizar *Window > Show view > Internal Web Browser*.

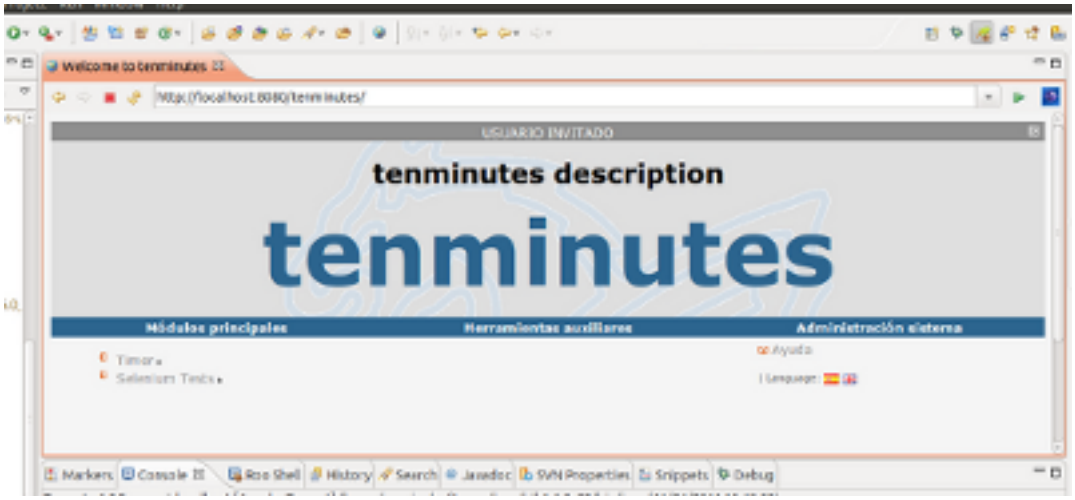

Figura 3.8. Navegador interno de Eclipse

Ejecutar la apliación desde Eclipse puede ser muy útil a la hora de hacer cambios en el proyecto, modificaciones en las vistas, en los objetos de aplicación, bien desde el propio IDE o desde la línea de comandos de gvNIX, ya que automáticamente se verán reflejados los cambios en el navegador.

3.6. Trabajando con el código de la aplicación en Eclipse

Al generar un proyecto con gvNIX/Roo habrá que trabajar sobre el código de la aplicación para añadir más funcionalidades y/o modificar alguna de las que se han generado. A continuación se explicarán un par de detalles que simplificaran esta tarea usando Eclipse y el plugin STS instalado.

Figura 3.9. Vista de editor y Cross References en Eclipse

En la imagen anterior se observa el editor de Eclipse con el archivo Timer.java que se ha generado al ejecutar los comandos *entity* y *field* anteriores. En la parte derecha de la captura se ve una ventana que muestra información de la clase Timer. Se trata de la vista *Cross Reference*. Si no está disponible en la perspectiva de trabajo, se puede incorporar desde el menú *Window > Show view > Cross References*. Esta vista es similar a la vista *Outline* que muestra los campos y métodos declarados en una clase.

La particularidad de la vista Cross References es que muestra los campos y métodos asociados a la clase Java mediante aspectos java que se verán en el Cápitulo sobre el código que genera gvNIX/Roo.

Esta información también está disponible desde un menú contextual accesible haciendo click con el botón derecho sobre la flecha que hay a la izquierda de la declaración de la clase. El menú contextual es el siguiente.

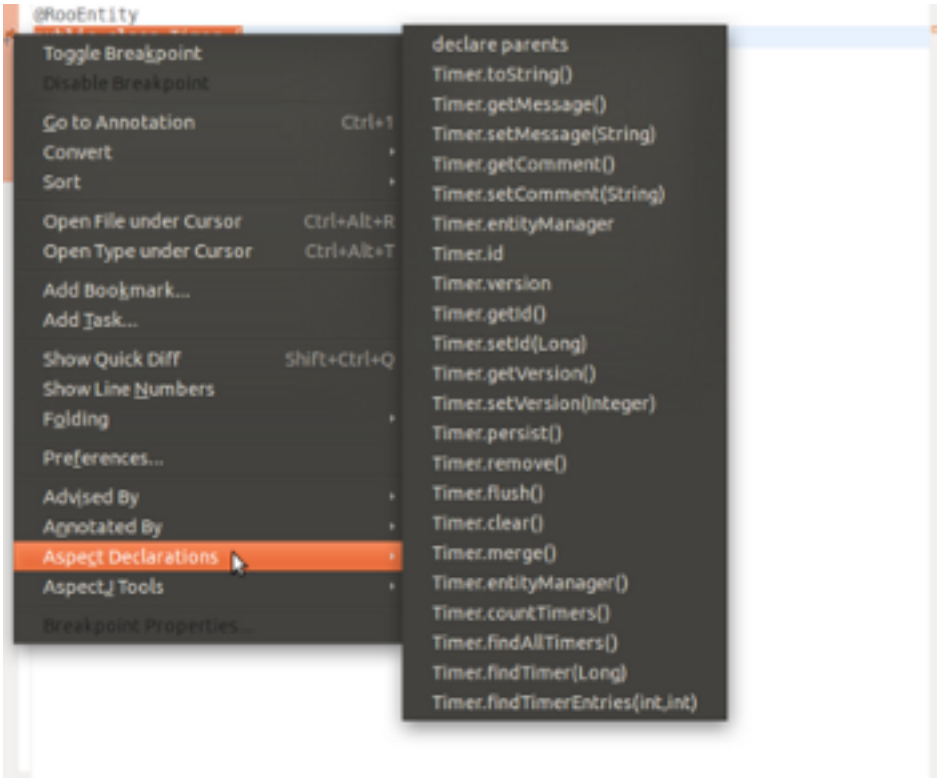

Figura 3.10. Contextual Cross References

Haciendo click sobre una de las entradas que aparecen en la ventana *Cross References* o en el menú contextual, se abrirá el fichero .aj que tiene declarado ese campo o método. Por ejemplo, si se accede al método *Timer.toString()* se abrirá el editor con el archivo *Timer_Roo_ToString.aj*.

Importante

Cabe destacar que para que aparezca esta información sobre el código asociado a una clase Java mediante aspectos java, la clase debe estar libre de errores de compilación.

3.6.1. Modificación del código generado

Como se ha comentado, seguramente será necesario añadir nuevas funcionalidades o modificar algunas de las que se han generado automáticamente para adaptar el código a los requisitos de la aplicación. En estos casos hay que tener en cuenta que **no se debe hacer ningún cambio sobre los aspectos java (ficheros aj) asociados a una clase Java**. gvNIX/Roo ya nos lo avisa en la primera línea del archivo .aj. El motivo de esto es, que teniendo la consola gvNIX funcionando, o al arrancar, gvNIX/Roo volvería a generar el código automáticamente perdiendo las modificaciones que hubiésemos realizado.

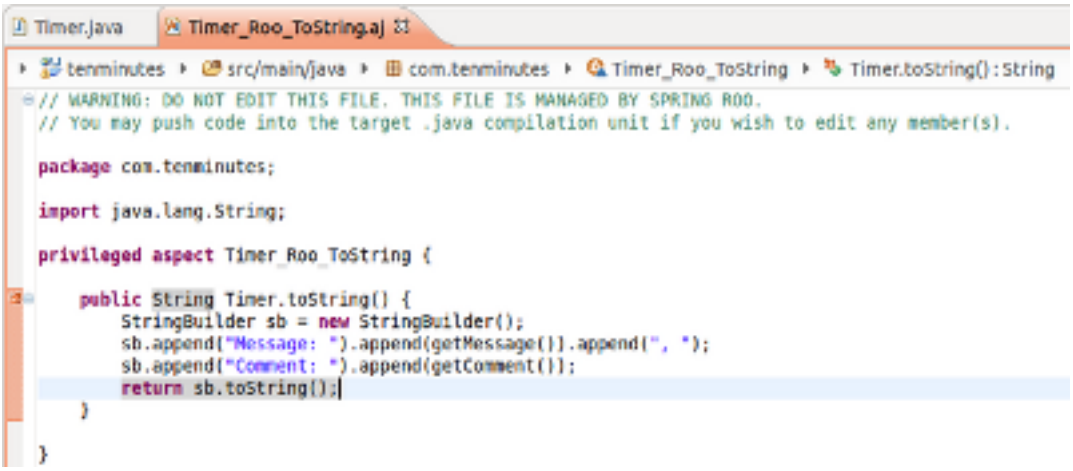

Figura 3.11. Edición archivo Timer_Roo_ToString.aj

Para modificar el código de un método o la declaración de un campo que se encuentre en un aspecto java, se puede mover el método o el campo a la clase Java, del que depende el aspecto java. En el ejemplo, se movería a la clase *Timer.java*. Puede realizarse cortando y pegando, pero Eclipse con el plugin STS ofrece una opción mucho más adecuada para esto.

Seleccionando el nombre del método a mover y a través del menú contextual *AspectJ Refactoring* > *Push In* ... Eclipse se ocupará de eliminar el código del .aj y moverlo al .java.

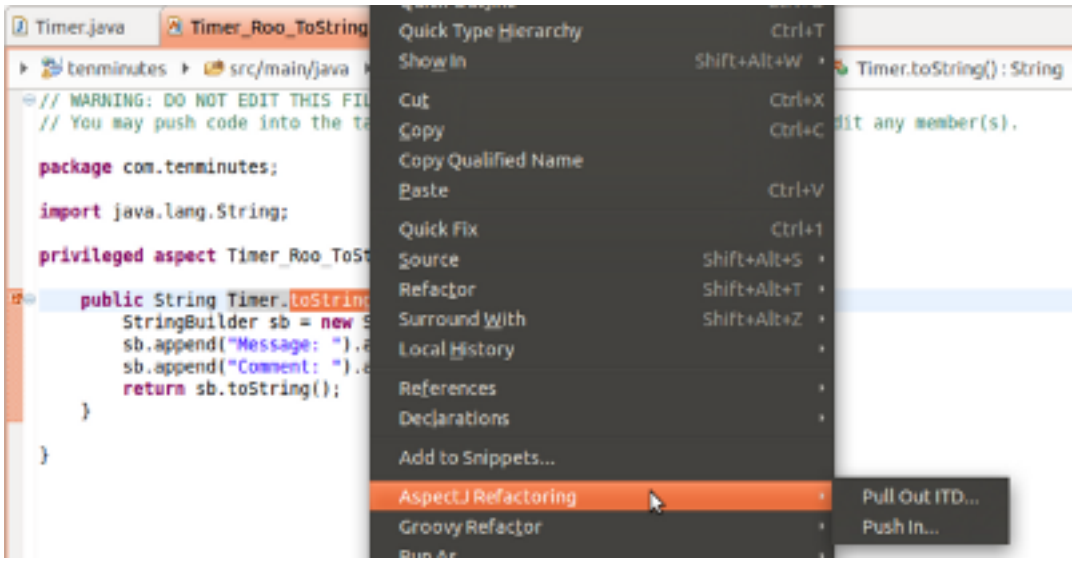

Figura 3.12. Menú contextual AspecJ Refactoring

Una vez en el .java, se podrá hacer cuantas modificaciones se necesiten sin peligro de que se pierdan por la acción de gvNIX/Roo.

Del mismo modo, si se necesita añadir nuevos métodos o campos a la clase Java, se debe hacer directamente en el .java.

Importante

Es interesante tener abierta la consola gvNIX para que los cambios realizados sobre los ficheros del proyecto disparen los cambios automáticos que realiza gvNIX/Roo.

Capítulo 4. Ejemplos gvNIX/Roo

En el fichero zip de gvNIX se incluye un directorio `samples` donde se puede encontrar algunos scripts que generan proyectos de aplicaciones de ejemplo.

Para ejecutarlo puede crearse un nuevo proyecto y ejecutarlo de la siguiente forma:

```
roo-gvNIX> script --file SCRIPT.roo
project ...
...
```

Donde `SCRIPT.roo` es el nombre del fichero de ejemplo que se desea ejecutar.

No todos los scripts descritos a continuación están en el directorio samples, pero todos sí pueden ser ejecutados tal y como se ha indicado.

clinic.roo
: Ejemplo de una aplicación de clínica veterinaria usando Hibernate, Registro Activo y Spring MVC

bikeshop.roo
: Ejemplo de una aplicación de tienda de bicicletas usando Eclipse Link, DAO y JSF.

expenses.roo
: Ejemplo de una aplicación de control de gastos usando Hibernate, Registro Activo y GWT.

multimodule.roo
: Mismo ejemplo que clinic.roo pero separando la aplicación en tres módulos: core, ui y mvc.

pizzashop.roo
: Ejemplo de una aplicación de venta de pizzas usando Eclipse Link, DAO y ofreciendo acceso remoto con JSON a través de Spring MVC.

embedding.roo
: Ejemplo de una aplicación sin persistencia con elementos web ricos (documentos, mapas, imágenes y videos) en Spring MVC.

vote.roo
: Ejemplo de una aplicación de voto usando Hibernate, Registro Activo, Spring MVC y Seguridad.

wedding.roo
: Ejemplo de aplicación con uso de correo electrónico.

report.roo
: Mismo ejemplo que clinic.roo con generación de informes mediante Jasper Reports.

theme.roo
: Mismo ejemplo que clinic.roo, pero aplicando distinto tema visual.

es-i18n.roo
: Mismo ejemplo que clinic.roo incluyendo la lengua Valenciana.

occ.roo
: Mismo ejemplo que clinic.roo añadiendo control de concurrencia sin columnas adicionales en base de datos.

menu.roo
: Mismo ejemplo que clinic.roo, pero utilizando un sistema de menús gestionable desde la consola.

service.roo
: Mismo ejemplo que clinic.roo incluyendo servicios web.

bing.roo
: Ejemplo de aplicación que utiliza el servicio web de búsqueda bing.

configuration.roo
: Mismo ejemplo que clinic.roo con dos perfiles de configuración distintos: uno para desarrollo y otro para producción.

dialog.roo
: Mismo ejemplo que clinic.roo con visualización de diálogos en ventana emergente y gestión de excepciones.

flex.roo
: Ejemplo de una aplicación de agenda de personas y direcciones usando Hibernate, Registro Activo y Flex. El acceso a la aplicación se debe realizar a través de la siguiente dirección: http://localhost:8080/flexrocks/flexrocks_scaffold.html

rootunes.roo
: Ejemplo de una aplicación de catálogo de canciones usando Hibernate, Registro Activo y Flex. El acceso a la aplicación se debe realizar a través de la siguiente dirección: http://localhost:8080/rootunes/rootunes_scaffold.html

binding.roo
: Mismo ejemplo que clinic.roo configurando el guardado de nulo en lugar de cadenas vacías en base de datos.

pattern.roo
: Ejemplo de una aplicación aplicando distintos patrones de visualización de la información en pantalla de entidades y sus relaciones.

typicalsecurity.roo
: Mismo ejemplo que clinic.roo configurando la seguridad de acceso a la aplicación mediante usuarios almacenados en base de datos.

bootstrap.roo
: Mismo ejemplo que clinic.roo modificando la apariencia para que utilice Bootstrap 3

datatables.roo
: Mismo ejemplo que clinic.roo aplicando datatables en todas las pantallas de listado del proyecto.

gvasecurity.roo
: Mismo ejemplo que clinic.roo configurando como proveedor de seguridad SAFE y APLUSU

La mayoría de esto ejemplos utilizan bases de datos en memoria por lo que no necesitan configuración adicional.

Parte II. Desarrollo de aplicaciones con gvNIX

En esta sección se verá paso a paso cómo desarrollar una aplicación web desde cero demostrando cómo utilizar gvNIX en cualquier proyecto web. Algunas de las características de gvNIX que se verán son:

- Creación de un nuevo proyecto.
- Configuración del acceso a datos.
- Código genereado automáticamente.
- Gestión del modelo de entidades.
- Gestión de la capa web.

Capítulo 5. Crear una nueva aplicación

El proyecto de ejemplo que se utilizará en esta parte de la documentación consiste en una aplicación web para la venta de pizzas, **Pizza Shop**.

Requerimientos:

- Los clientes pueden hacer pedidos por la web.
- Los empleados crean pedidos recibidos por teléfono.
- Las pizzas son un conjunto de ingredientes sobre una base.

gvNIX es especialmente potente si usamos los principios del *diseño dirigido por el dominio* (DDD). Básicamente son un conjunto de patrones para construir aplicaciones a partir del modelo del dominio. En el ámbito de gvNIX destacan las siguientes dos características:

- **Diseño dirigido por el modelo**. Representar fielmente los conceptos seleccionados del dominio del problema, por ejemplo utilizando UML (diagramas de clases).
- **Arquitectura por capas**. Separar la lógica de negocio de la lógica de aplicación (transacciones, seguridad, etc), de la lógica de presentación y de la lógica de infraestructura (acceso a datos, acceso a servicios externos, etc).

  La arquitectura por capas es un patrón que gvNIX se encarga de aplicar por nosotros, sin embargo, debemos conocer las capas generadas por gvNIX porque este conocimiento será de utilidad a la hora de hacer nuestros desarrollos.

Por tanto, lo primero que se debe hacer es definir el modelo del dominio:

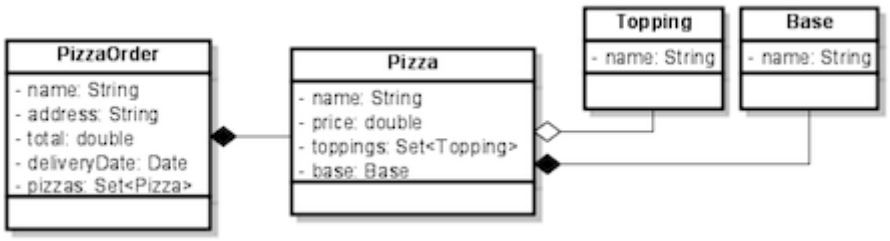

Figura 5.1. Pizza Shop Domain Model

5.1. Crear un nuevo proyecto

La creación de un nuevo proyecto se puede realizar de dos formas, en función de si se hace desde la consola del sistema operativo o desde el IDE. Para ello, leer las secciones Crear proyecto gvNIX en Eclipse IDE para la creación desde el IDE o Descarga e instalación gvNIX para la creación desde la consola del sistema.

Acceder a la consola y teclear **hint** para mostrar la información del asistente que cambiará durante el proceso de desarrollo en función del estado en el que se encuentre el proyecto:

```
roo-gvNIX> hint
Welcome to Roo! We hope you enjoy your stay!

Before you can use many features of Roo, you need to start a new project.
```

```
To do this, type 'project' (without the quotes) and then hit TAB.

Enter a --topLevelPackage like 'com.mycompany.projectname' (no quotes).
When you've finished completing your --topLevelPackage, press ENTER.
Your new project will then be created in the current working directory.

Note that Roo frequently allows the use of TAB, so press TAB regularly.
Once your project is created, type 'hint' and ENTER for the next suggestion.
You're also welcome to visit http://forum.springframework.org for Roo help.
```

Muestra las instrucciones para continuar y crear un proyecto. El proyecto ya estará creado si se hizo desde la consola del IDE y se podrá pasar a la siguiente sección, en caso contrario teclear **project**. Pulsar la tecla TAB (tabulador) para que se muestre el parámetro *--topLevelPackage* y definir el paquete principal del proyecto, en este caso com.springsource.roo.pizzashop. También se puede indicar el JDK con el que se va a trabajar en el proyecto añadiendo el atributo *--java 6*, el número 6 corresponde a la versión del JDK utilizado, en este caso JDK 1.6, si no se especifica Roo por defecto utiliza el JDK 1.6

```
roo-gvNIX> project --topLevelPackage com.springsource.roo.pizzashop
Created ROOT/pom.xml
Created SRC_MAIN_RESOURCES
Created SRC_MAIN_RESOURCES/log4j.properties
Created SPRING_CONFIG_ROOT
Created SPRING_CONFIG_ROOT/applicationContext.xml
```

Una vez creado el proyecto observar que el foco de la consola ha cambiado, ahora se encuentra apuntando al paquete base del proyecto que se ha indicado al crearlo:

```
com.springsource.roo.pizzashop roo-gvNIX>
```

Por otro lado, la estructura de directorios creada esta basada en las recomendaciones de Maven.

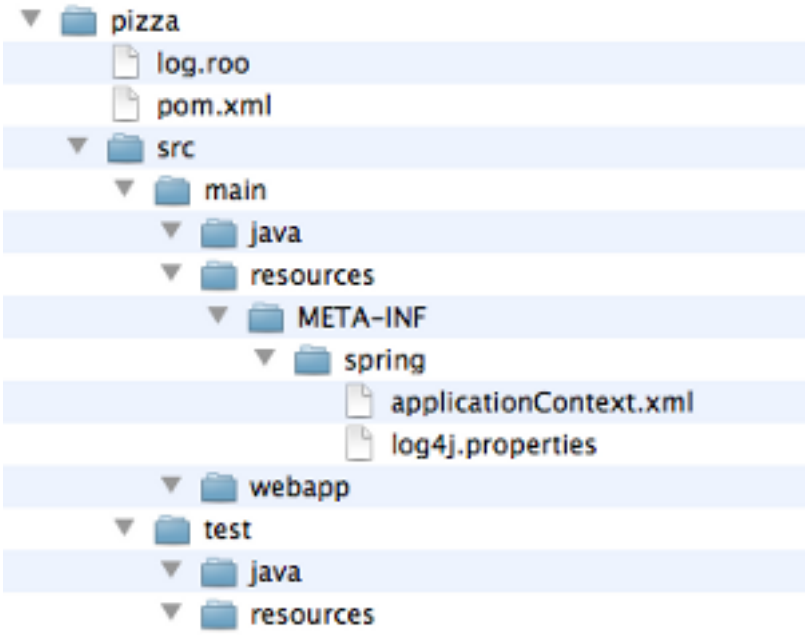

Figura 5.2. Estructura de carpetas

Capítulo 6. Gestión del acceso a datos

6.1. Persistencia de objetos

Trabajar en un entorno orientado a objetos con acceso a fuentes de datos relacionales (lo habitual en tecnología Java) puede suponer un incremento de los tiempos de desarrollo porque se debe incluir la lógica necesaria para hacer corresponder el modelo relacional de la base de datos, que trata con relaciones y conjuntos, con el paradigma orientado a objetos de las entidades del proyecto, que trata con objetos, atributos y asociaciones de unos con otros. Esta problemática es la que se conoce como *desajuste objeto-relacional* (object-relational impedance mismatch).

La técnica y herramientas *ORM* (Object-Relational Mapping) surgen para reducir el coste de desarrollo que supone el desajuste objeto-relacional. Las técnicas de ORM permiten automatizar procesos que trasladan objetos a formas almacenables en tablas y viceversa, preservando los atributos de los objetos. Para esto se basan en la utilización de metadatos de correspondencia que especifican la información necesaria para que se pueda efectuar de forma automática la conversión de datos entre el sistema relacional y el sistema orientado a objetos.

Esta técnica supuso que empezaran a aparecer multitud de librerías y herramientas ORM para Java. Para estandarizar esta técnica en el mundo Java y hacer nuestras aplicaciones independientes de la herramienta surgió JPA (Java Persistence API), actualmente en su versión 2 (JSR 317).

gvNIX (herencia del núcleo Spring Roo) fundamenta su interacción con la base de datos en un sistema de persistencia *JPA 2*. Gracias a esto las aplicaciones desarrolladas con gvNIX se benefician de las mejoras de tiempos de desarrollo que aportan la técnica y herramientas ORM.

Regresando a la creación del nuevo proyecto, el primer paso es instalar y configurar el sistema de persistencia. Teclear **hint** que sugerirá los siguientes pasos a dar:

```
roo-gvNIX> hint
Roo requires the installation of a persistence configuration,
for example, JPA or MongoDB.

For JPA, type 'jpa setup' and then hit TAB three times.
We suggest you type 'H' then TAB to complete "HIBERNATE".
After the --provider, press TAB twice for database choices.
For testing purposes, type (or TAB) HYPERSONIC_IN_MEMORY.
If you press TAB again, you'll see there are no more options.
As such, you're ready to press ENTER to execute the command.

Once JPA is installed, type 'hint' and ENTER for the next suggestion.

Similarly, for MongoDB persistence, type 'mongo setup' and ENTER.

roo-gvNIX>
```

En el ámbito de esta documentación es suficiente con conectar la aplicación con una base de datos que no necesita ningún servidor instalado como Hypersonic e Hibernate como implementación del sistema de persistencia:

```
roo-gvNIX> jpa setup --provider HIBERNATE --database HYPERSONIC_PERSISTENT
Created SPRING_CONFIG_ROOT/database.properties
Updated SPRING_CONFIG_ROOT/applicationContext.xml
Created SRC_MAIN_RESOURCES/META-INF/persistence.xml
Updated ROOT/pom.xml [added dependencies org.hsqldb:hsqldb:2.2.9,
org.hibernate:hibernate-core:4.2.2.Final, org.hibernate:hibernate-entitymanager:4.2.2.Final,
```

```
org.hibernate.javax.persistence:hibernate-jpa-2.0-api:1.0.1.Final,
commons-collections:commons-collections:3.2.1,
org.hibernate:hibernate-validator:4.3.1.Final,
javax.validation:validation-api:1.0.0.GA,
javax.transaction:jta:1.1, org.springframework:spring-jdbc:${spring.version},
org.springframework:spring-orm:${spring.version},
commons-pool:commons-pool:1.5.6, commons-dbcp:commons-dbcp:1.4]
roo-gvNIX>
```

Aviso

Este tipo de bases de datos no es la mejor opción para un entorno de producción, utilizarla sólo para entornos de desarrollo, demostraciones y formación.

Aviso

Con la ingeniería directa, es decir, la creación de entidades y campos mediante los comandos de la consola, se configura el sistema de persistencia para crear la base de datos cada vez que se arranca la aplicación. Por eso, los datos de la demo se perderán cada vez que se inicie la aplicación. Esto no es así cuando se utiliza la ingeniería inversa a partir de un esquema de base de datos ya existente.

Este comportamiento se puede cambiar en la configuración del sistema de persistencia: `src/main/resources/META-INF/persistence.xml`. Consultar la sección enlazar base de datos existente para aprender cómo cambiar el comportamiento por defecto.

6.2. Configurar la conexión con la base de datos

En cualquier punto del ciclo de vida de una aplicación puede surgir la necesidad de cambiar de base de datos (distinta tecnología de bases de datos, cambio de la clave de acceso, cambio del equipo donde se encuentra la base de datos, etc), para estos casos gvNIX ofrece una serie de comandos que simplificarán la actualización de la configuración de acceso a la base de datos:

- **jpa setup**
- **database properties**

6.2.1. jpa setup

Crea o actualiza la configuración del acceso a datos (base de datos y el proveedor de persistencia o ORM). Entre los parámetros requeridos están la herramienta ORM y la base de datos a utilizar.

Los parámetros obligatorios que requiere este comando son:

--provider
: Identificador de la herramienta ORM. Los valores posibles son:

  DATANUCLEUS
  : Data Nucleus

  ECLIPSELINK
  : EclipseLink

  HIBERNATE
  : Hibernate

OPENJPA
: OpenJPA

--database
: Identificador de la base de datos. Los valores posibles son:

- ORACLE. *Debido a que los drivers JDBC no están disponibles en ningún repositorio Maven, en caso de utilizar Oracle habrá que instalar manualmente el driver JDBC.*
- MSSQL.
- MYSQL.
- POSTGRES.
- SYBASE.
- DB2. *Debido a que los drivers JDBC no están disponibles en ningún repositorio Maven, en caso de utilizar DB2 habrá que instalar manualmente el driver JDBC.*
- DERBY.
- GOOGLE_APP_ENGINE.
- H2_IN_MEMORY.
- HYPERSONIC_IN_MEMORY.
- HYPERSONIC_PERSISTENT.
- FIREBIRD.
- DATABASE_DOT_COM.

Los parámetros opcionales que permite este comando son:

--applicationId
: Identificación de aplicación en Google App Engine (sólo utilizado para esta base de datos).

--databaseName
: Nombre de la base de datos con la que conectar.

--hostName
: El nombre DNS o la dirección IP del ordenador en el que se encuentra la base de datos a utilizar.

--jndiDataSource
: Fuente de datos JNDI a utilizar.

--password
: Contraseña del usuario para la conexión con la base de datos.

--persistenceUnit
: Nombre de la unidad de persistencia a utilizar en el fichero persistence.xml.

--transactionManager
: Nombre del gestor de transacciones a utilizar.

--userName
: Nombre del usuario para la conexión con la base de datos.

Dependiendo del tipo de base de datos pueden ser necesarios unos u otros parámetros. Se recomienda usar el completado de comandos con *TAB* (en la consola del sistema) o *CTRL+SPACE* (en el IDE) para que se indique lo que hace falta en función del contexto. Cuando el autocompletado no proporciona más opciones para un comando, se recomienda escribir los caracteres -- e ir auto completando para que aparezcan los parámetros opcionales.

6.2.2. database properties

Permite la consulta, eliminación, actualización y creación de las propiedades de conexión con la base de datos (a diferencia del comando anterior que configura el sistema de persitencia en su totalidad). Estas propiedades se almacenan el fichero `src/main/resources/META-INF/spring/database.properties` de la aplicación.

Se dispone del siguiente grupo de comandos:

database properties list
: Lista las propiedades de conexión con la base de datos.

database properties remove
: Elimina una propiedad de la configuración. Requiere un único parámetro `--key` para especificar la clave a eliminar.

database properties set
: Crea o actualiza una propiedad en la configuración. Requiere un parámetro `--key` con el nombre de la propiedad y un `--value` con el valor a establecer.

6.2.3. Ejemplo: Conectar con PostgreSQL

En este ejemplo se va a configurar la conexión a un servidor PostgreSQL situado en una máquina en de la red local. Los datos de conexión de ejemplo son:

- Servidor de base de datos: `dbserver`
- Puerto de conexión: `5438`
- Nombre de la base de datos: `my_db`
- Nombre de usuario de la base de datos: `user1`
- Contraseña del usuario: `1234`

Recordar que estos comando se pueden ejecutar en cualquier momento y tantas veces como se desee.

Los pasos a seguir para la configuración son:

1. Usar el comando **persistence setup** con los parámetros que disponemos:

```
roo-gvNIX> jpa setup --provider HIBERNATE --database POSTGRES --databaseName my_db
             --userName user1 --password 1234 --hostName dbserver
Created SPRING_CONFIG_ROOT/database.properties
Please update your database details in
```

```
 src/main/resources/META-INF/spring/database.properties.
Updated SPRING_CONFIG_ROOT/applicationContext.xml
Created SRC_MAIN_RESOURCES/META-INF/persistence.xml
Updated ROOT/pom.xml [added dependencies postgresql:postgresql:9.1-901-1.jdbc4,
 org.hibernate:hibernate-core:4.2.2.Final, org.hibernate:hibernate-entitymanager:4.2.2.Final,
 org.hibernate.javax.persistence:hibernate-jpa-2.0-api:1.0.1.Final,
 commons-collections:commons-collections:3.2.1,
 org.hibernate:hibernate-validator:4.3.1.Final, javax.validation:validation-api:1.0.0.GA,
 javax.transaction:jta:1.1, org.springframework:spring-jdbc:${spring.version},
 org.springframework:spring-orm:${spring.version}, commons-pool:commons-pool:1.5.6,
 commons-dbcp:commons-dbcp:1.4]
roo-gvNIX>
```

2. Comprobar como ha quedado la configuración de la conexión. Para ello usar el comando **database properties list**:

```
roo-gvNIX> database properties list
database.driverClassName = org.postgresql.Driver
database.password = 1234
database.url = jdbc:postgresql://dbserver:5432/my_db
database.username = user1
roo-gvNIX>
```

   Observar que lo único que falta es configurar el puerto de la base de datos en la propiedad `database.url`. Observar que el puerto configurado automáticamente es el puerto por defecto de la base de datos, sin embargo en este ejemplo el servidor de base de datos utiliza uno distinto.

3. Configurar la URL de conexión al servidor correctamente usando el comando **database properties set**:

```
roo-gvNIX> database properties set --key database.url --value
    jdbc:postgresql://dbserver:5438/my_db
Updated SPRING_CONFIG_ROOT/database.properties
roo-gvNIX>
```

4. Volver a comprobar el estado de los parámetros con el comando **database properties list**:

```
roo-gvNIX> database properties list
database.driverClassName = org.postgresql.Driver
database.password = 1234
database.url = jdbc:postgresql://dbserver:5438/my_db
database.username = user1
roo-gvNIX>
```

Una vez hecho, la siguiente vez que se arranque la aplicación usará la nueva conexión de base de datos configurada.

6.2.4. Actualización automática del esquema

Al crear las entidades y los campos desde la consola mediante los comandos de ingeniería directa, en el arranque de la aplicación se crea automáticamente el esquema de base de datos que se corresponde con el modelo de entidades. Esto no es así cuando se realiza la ingeniería inversa a partir de un esquema de base de datos ya existente.

Si se desea cambiar este comportamiento para, por ejemplo, que los datos almacenados no desaparezca en cada arranque seguir los siguientes pasos:

- Editar el archivo `src/main/resources/META-INF/persistence.xml`

- Cambiar el valor de la propiedad *hibernate.hbm2ddl.auto*:

  Existen cinco valores *validate, update, create, create-drop y none*. Estos valores deben utilizarse con precauciñón por el peligr" puede suponer para la integridad de la BBDD.

  - *validate*: valida que el modelo de objetos y el modelo relacional de base de datos son equivalentes. En caso de no ser iguales no se permitirá el arranque de la aplicación.
  - *update*: actualiza el esquema al arrancar la aplicación. Esto es, modificará los elementos del modelo relacional de base de datos que no se correspondan con el modelo de objetos.
  - *create*: crea el esquema en el arranque de la aplicación destruyendo la información existente.
  - *create-drop*: hace un *drop* de las tablas al parar la aplicación y en el arranque las crea.
  - none: no hace absolutamente ninguna validación ni modificación del esquema.

6.2.5. Múltiples fuentes de datos

La conexión con distintas bases de datos está soportada, aunque su configuración no se genera automáticamente. Distinguir de la conexión con distintos esquemas de la misma base de datos que está tanto soportado como contemplado por la generación automática que realiza la ingeniería inversa.

A continuación se muestran los pasos a seguir para configurar la conexión con distintas bases de datos en una misma aplicación mediante el proveedor de persistencia Hibernate:

- En *src/main/resources/META-INF/persistence.xml* incluir la siguiente configuración reemplazando los puntos suspensivos por los valores adecuados para la nueva conexión de base de datos:

```
<persistence-unit name="persistenceUnit2" transaction-type="RESOURCE_LOCAL">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <properties>
        <property name="hibernate.dialect" value="..."/>
        <property name="hibernate.hbm2ddl.auto" value="..."/>
        <property name="hibernate.ejb.naming_strategy" value="..."/>
        <property name="hibernate.connection.charSet" value="..."/>
    </properties>
</persistence-unit>
```

- En *src/main/resources/META-INF/spring/applicationContext.xml* incluir la siguiente configuración reemplazando los puntos suspensivos por los valores adecuados para la nueva conexión de base de datos:

```
<bean class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close"
  id="dataSource2">
    <property name="driverClassName" value="..."/>
    <property name="url" value="..."/>
    <property name="username" value="..."/>
    <property name="password" value="..."/>
    <property name="testOnBorrow" value="..."/>
    <property name="testOnReturn" value="..."/>
    <property name="testWhileIdle" value="..."/>
    <property name="timeBetweenEvictionRunsMillis" value="..."/>
    <property name="numTestsPerEvictionRun" value="..."/>
    <property name="minEvictableIdleTimeMillis" value="..."/>
    <property name="validationQuery" value="..."/>
```

```
    </bean>
      <bean class="org.springframework.orm.jpa.JpaTransactionManager"
        id="transactionManager2">
          <property name="entityManagerFactory" ref="entityManagerFactory2"/>
      </bean>
      <tx:annotation-driven mode="aspectj" transaction-manager="transactionManager2"/>
      <bean class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
        id="entityManagerFactory2">
          <property name="persistenceUnitName" value="persistenceUnit2" />
          <property name="dataSource" ref="dataSource2"/>
      </bean>
```

- En la clase Java de la entidad, por ejemplo, *src/main/java/.../domain/Entidad.java* añadir en la anotación *@RooJpaActiveRecord* el atributo *persistenceUnit = "persistenceUnit2"* haciendo referencia a la nueva conexión configurada:

```
    @RooJpaActiveRecord(persistenceUnit = "persistenceUnit2")
    class Entity
```

  De esta forma, para cada entidad de datos se puede asociar la conexión de base de datos que se desee.

Capítulo 7. Código generado por gvNIX/Roo

7.1. Clases Java y ficheros AJ

7.1.1. Archivos Java

Por lo general, la norma que se sigue es mantener la clase Java, el archivo .java, lo más sencillo posible y por ello, muchas veces, se puede observar que el archivo .java está prácticamente vacio. Contiene poco más que la declaración de la clase y anotaciones asociadas a la misma. El resto del código, campos y métodos se declaran en sus aspectos java de los que se hablará a continuación.

7.1.2. Archivos AJ

Estos son los ficheros que genera automáticamente el framework y que contienen las operaciones básicas, los campos definidos con sus getter y setter, el identificador, etc. En general contine el código que un programador haria de manera rutinaria. Estos ficheros tienen nombres como *_Roo_*.aj y *_gvNIX_*aj. Se trata de aspectos java que no son más que porciones de código que al compilar el proyecto, se asocian como código de la propia clase Java a la que están vinculadas. Si se desea personalizar alguna de las operaciones básicas, un getter determinado o añadir nuevas funcionalidades, estas deberán realizarse en los ficheros .java. En los ficheros aj no se debe de hacer ningún cambio. El motivo de esto es, que teniendo la consola funcionando, o al arrancarla, se volvería a generar el código automáticamente perdiendo las modificaciones que se hubiesen realizado. Y al contrario si en el fichero .java se ha creado el getter personalizandolo, automáticamente lo borrara del fichero .aj.

Capítulo 8. Gestión de entidades

El acceso a datos de una aplicación contiene la infraestructura necesaria para obtener y almacenar la información en, por ejemplo, una base de datos. Este acceso se puede estructurar en una sola capa en la que las entidades definen tanto el modelo de los datos, como sus métodos de interacción con la base de datos y la lógica de negocio utilizando entonces un modelo llamado *Registro Activo*. Existen otros casos más clásicos como la estructuración en una capa de entidades, otra de acceso a datos (DAO) y una última de servicios con la lógica de negocio; todo ello llamado *Servicios y Repositorios*. Este capítulo se centrará en la generación de la capa de entidades ya que es el método que se utiliza habitualmente en las aplicaciones desarrolladas con gvNIX y para el cual ha añadido la mayoría de sus nuevas funcionalidades.

Las entidades del modelo del dominio son uno de los elementos más importantes del framework. En Java una entidad se corresponde con una clase, definida en un archivo `.java`.

Además de la lógica de negocio, cada clase contiene una serie de anotaciones de clase y de atributos que forman parte de la metainformación que permite a gvNIX realizar acciones automáticas.

Ejemplo 8.1. Una clase Java

```
                                                                                    ❶
@RooJavaBean
@RooToString
@RooJpaActiveRecord
public class Entity1 {

        @Size(max = 20)                                                             ❷
        @NotNull
        private String name;

        @Size(max = 20)
        private String apellidos;
}
```

❶ Anotación de clase

❷ Anotación de atributo

El framework añadirá todos los métodos estándar necesarios para completar la definición de entidad, como los de persistencia (almacenamiento y recuperación), gets/sets y métodos auxiliares como toString. Como se vió en la sección de generación de código se crearán una serie de archivos `.aj` que en tiempo de compilación se incluyen en la clase de forma transparente.

Si por algún motivo es necesario *personalizar el comportamiento* de alguno de los *métodos estándar que se generan*, simplemente hay que *definirlos en la clase java*. Cuando el framework detecta el cambio actualiza el fichero `aj` correspondiente para eliminar esta definición.

Existen dos formas de crear una entidad:

1. Usando el intérprete de comandos.

2. Usando el IDE de desarrollo.

8.1. Crear una entidad con el intérprete de comandos

Desde el intérprete de comandos se puede crear entidades de forma sencilla con los comandos: **entity jpa** y **field**.

8.1.1. Comando entity jpa

Este comando crea una nueva entidad en el proyecto. Requiere como mínimo definir el parámetro `--class` para especificar el nombre y el paquete de la nueva clase. En este parámetro se puede utilizar el carácter ~ como abreviatura del paquete base del proyecto.

Entre los parámetros obligatorios se encuentran :

--class
: Nombre de la clase java para la entidad (incluido su paquete) (recordar que se puede usar el ~ para especificar el paquete base de la aplicación).

Entre los principales parámetros opcionales (notar que hay más) se encuentran:

--activeRecord
: Generar los métodos de creación, lectura, actualización y borrado (CRUD) para esta entidad. Esto implica el uso de un patrón de entidades del tipo registro activo.

--entityName
: Nombre a usar para la entidad. Diferenciar del parámetro *tabla* que define el nombre de su tabla asociada.

--identifierType
: Permite definir el tipo (clase java) de identificador (clave primaria) que usará esta entidad.

--identifierField
: Nombre del *campo en la entidad* (no en BBDD) que se usará para el identificador.

--identifierColumn
: Nombre de la *columna de la BBDD* donde almacenar el valor de clave primaria.

--persistenceUnit
: Unidad de persistencia a utilizar para gestionar la interacción de la entidad con la base de datos. Como cada unidad de persistencia puede estar asociada a una base de datos distinta, es la forma de definir que una entidad se encuentra almacenada en otra base de datos.

--schema
: Esquema de base de datos donde se encontrará la tabla asociada con la entidad.

--table
: Nombre de la *tabla de la BBDD* donde almacenar los datos para esta entidad.

--testAutomatically
: Genera los test unitarios para esta entidad usando JUnit.

--versionColumn
: Nombre de la columna en base de datos que se comporta como campo de control de concurrencia optimista basado en número de versión.

--versionType
: Tipo de la columna de base de datos que se comporta como campo de control de concurrencia optimista basado en número de versión.

El resto de las opciones podemos verlas con el autocompletado del comando escribiendo --, en la documentación oficial de Roo o usando el comando **help entity**.

Eliminar y modificar entidades

La consola no se utiliza para eliminar o modificar las entidades. Esto se debe realizar usando el IDE. Para más información ver la sección Modificar una entidad.

8.1.2. Comando field

No existe un comando **field** propiamente dicho, sino que es el nombre de un grupo de comandos que permiten añadir campos a una entidad. Los comandos finales se forman con el formato **field {tipo}** donde *`{tipo}`* es el tipo de dato que va almacenar. Así tenemos los siguientes:

field string
: Crea un campo de tipo cadena alfanumérica.

field boolean
: Crea un campo de tipo booleano (valores cierto o falso).

field date
: Crea un campo de tipo fecha.

field email template
: Crea un campo con una plantilla preparada para almacenar direcciones de correo electrónico.

field embedded
: Añade un campo del tipo *@Embedded*

field enum
: Crea un campo de tipo enumerado. El tipo enumerado debe haber sido creado previamente con el comando *enum type*.

field file
: Crea un campo de tipo fichero.

field number
: Crea un campo de tipo numérico (es posible especificarle el tipo mediante el parámetro `--type`).

field reference
: Crea un campo que será una referencia a otra entidad.

field set
: Crea un campo que devolverá una lista de elementos de una entidad relacionados con este elemento.

field other
: Crea un campo de otro tipo no registrado.

Cada uno de estos comandos admiten bastantes parámetros que permiten definir múltiples comportamientos especiales, algunos aplicables a todos los comandos y otros específicos solo para algunos de ellos. Entre los generales que aplicar a la mayoría podemos encontrar los siguientes:

--class
: Clase entidad dondecrear el campo. Admite el uso de ~ para identificar al paquete base del proyecto y tiene autocompletado. No es necesario de especificar si acabamos de crear una entidad.

--column
: Nombre de la columna en BBDD donde se almacenará el valor.

--fieldName
: Nombre del campo que vamos a generar.

--notNull y --nullRequired
: Especifica el comportamiento de campo ante los valores nulos.

--regexp
: Permite definir una expresión regular que debe cumplir el valor que se establezca en este campo.

--sizeMax y --sizeMin
: Limitaciones de tamaño.

--transient
: Indica que el campo no tiene asociada ninguna columna en la base de datos, es decir, no se almacena ni obtiene de ella.

--unique
: Indica que el campo tiene la restricción de unicidad para los valores que puede almacenar.

Existen más opciones dependiendo del tipo de dato, para más información ver la ayuda de la consola o acceder a la documentación oficial de Roo en http://static.springsource.org/spring-roo/reference/html/

Enlazar modelo de entidades con base de datos existente

Notar que el parámetro **--column** permiten que en tiempo de creación podemos indicar que la entidad sea persistente sobre un modelo relacional creado con anterioridad.

Eliminar y modificar campos

No se pueden eliminar o modificar los campos desde línea de comandos. Esto se debe realizar usando el IDE. Para más información ver la sección Modificar una entidad.

8.1.3. Proyecto ejemplo

Crear el modelo de dominio de la aplicación de venta de pizzas con la siguiente secuencia de comandos a partir del mismo ejemplo iniciado en capítulos anteriores:

1. Topping:

```
roo-gvNIX> entity jpa --class ~.domain.Topping --testAutomatically
~.domain.Topping roo-gvNIX> field string --fieldName name --notNull --sizeMin 2
```

2. Base:

```
~.domain.Topping roo-gvNIX> entity jpa --class ~.domain.Base --testAutomatically
~.domain.Base roo-gvNIX> field string --fieldName name --notNull --sizeMin 2
```

3. Pizza:

```
~.domain.Base roo-gvNIX> entity jpa --class ~.domain.Pizza --testAutomatically
~.domain.Pizza roo-gvNIX> field string --fieldName name --notNull --sizeMin 2
  --comment "Nombre de la pizza" --column pizza_nombre --regexp [A-Za-z0-9].*
~.domain.Pizza roo-gvNIX> field number --fieldName price --type java.lang.Float
~.domain.Pizza roo-gvNIX> field set --fieldName toppings --type ~.domain.Topping
~.domain.Pizza roo-gvNIX> field reference --fieldName base --type ~.domain.Base
```

Esta secuencia de comandos genera la siguiente clase Java:

```
@RooJavaBean
@RooToString
@RooJpaActiveRecord
public class Pizza {

    @NotNull
    @Column(name = "pizza_nombre")
    @Size(min = 2)
    @Pattern(regexp = "[A-Za-z0-9].*")                                    ❶
    private String name;

    private Float price;

    @ManyToMany(cascade = CascadeType.ALL)
    private Set<Topping> toppings =                                       ❷
      new HashSet<Topping>();

    @ManyToOne(targetEntity = Base.class)
    @JoinColumn
    private Base base;                                                    ❸
}
```

❶ Validación del texto

❷ Asociación con un conjunto de entidades

❸ Asociación con otra entidad

4. PizzaOrder:

```
~.domain.Pizza roo-gvNIX>entity jpa --class ~.domain.PizzaOrder --testAutomatically
~.domain.PizzaOrder roo-gvNIX>field string --fieldName name --notNull --sizeMin 2
~.domain.PizzaOrder roo-gvNIX>field string --fieldName address --notNull --sizeMax 30
~.domain.PizzaOrder roo-gvNIX>field number --fieldName total --type java.lang.Float
~.domain.PizzaOrder roo-gvNIX>field date --fieldName deliveryDate --type java.util.Date
~.domain.PizzaOrder roo-gvNIX>field set --fieldName pizzas --type ~.domain.Pizza
```

En este punto, se ha completado la versión inicial del modelo del dominio

8.2. Crear una entidad con un IDE

Como se ha visto en el punto anterior, el código Java generado no tiene ningún elemento particular del framework, por lo que crear el modelo del dominio con cualquier IDE consiste en codificar las clases Java con las anotaciones que permiten a la consola monitorizar y generar el código.

Es importante destacar que el objetivo de utilizar anotaciones es permitir que el proyecto se pueda gestionar codificando las clases con cualquier IDE. Entonces al arrancar la consola se generará toda la infraestructura necesaria para la persistencia, validación, etc

Entre las anotaciones que se pueden utilizar en la codificación del modelo de entidades, se pueden destacar:

@RooJavaBean
: Anotación que implica la generación de los getter/setter que falten para acceder/modificar los valores de las propiedades. Si alguno ya estuviese generado, no se volverá a generar.

@RooToString
: Anotación que implica la generación del método `toString()` para esta clase. Este método se genera a partir de las propiedades de la clase.

@RooJpaActiveRecord
: Anotación que identifica esta clase como persistente, es decir, la marca como una entidad. Esta anotación genera todos lo métodos de persistencia necesarios para crear, leer, actualizar y borrar (CRUD) la entidad siguiendo un patrón de registro activo.

Importante

Notar que para usar las anotaciones hay que importar sus clases:

```
import org.springframework.roo.addon.javabean.RooJavaBean;
import org.springframework.roo.addon.jpa.activerecord.RooJpaActiveRecord;
import org.springframework.roo.addon.tostring.RooToString;
```

Consultar la documentación oficial de Roo para un detalle exhaustivo de las anotaciones.

8.3. Modificar una entidad

Si se desea añadir un atributo a una entidad existente puede hacerse desde la consola. Cuando se crea o modifica una entidad, la consola se situa en el contexto de dicha entidad. Por eso en el ejemplo anterior, al añadir campos tras crear la entidad PizzaOrder, no es necesario especificar en cada comando la entidad sobre la que se están incluyendo.

Añadir un atributo
: Para añadir un nuevo campo deberemos usar el mismo comando **field** indicando sobre que entidad queremos trabajar mediante el parámetro *--class.*

```
field string --fieldName refCode --notNull --sizeMin 2 --class ~.domain.Pizza
~.domain.Pizza roo-gvNIX>
```

Notar que tras ejecutar el comando *field* indicando el parámetro *--class* la consola cambia al contexto de la entidad Pizza y por tanto si se sigue ejecutando el comando field sin indicar *--class* los campos se añadirán a esta entidad.

Nota

Es posible situarse en el contexto de una entidad (o de cualquier clase del proyecto) mediante el comando **focus** indicando el parámetro *--class*.

```
roo-gvNIX> focus --class ~.domain.PizzaOrder
~.domain.PizzaOrder roo-gvNIX>
```

A diferencia de la creación de entidades, la modificación del modelo del dominio debe realizarse modificando la clase Java de la entidad, por ejemplo, mediante un IDE de desarrollo.

Eliminar una entidad
: Borrar el fichero `.java` de que define la entidad y automáticamente se detectará el cambio y se eliminarán todos los elementos relacionados: ficheros `{entidad}_*.aj`, controladores asociados, elementos de menú, etc.

Eliminar un atributo
: Borrar el atributo de la clase Java y todos los componentes relacionados `{entidad}_*.aj` serán actualizados automáticamente.

Modificar un atributo
: Modificar el atributo deseado en la clase Java y se actualizarán todos los ficheros `{entidad}_*.aj` relacionados para contemplar los cambios.

8.4. Identificadores compuestos

En el caso de necesitar que alguna de las entidades tenga definida una clave primaria compuesta de varios campos se debe hacer de la siguiente forma.

```
entity jpa --class ~.domain.Entity --identifierType ~.domain.EntityPK
```

De esta forma se genera una clase *EntityPK* sobre la que definir los campos que conformarán el identificador compuesto y que se corresponderán en el modelo relacional de base de datos con una clave primaria compuesta. La entidad usará la la clase que representa la clave compuesta, en el ejemplo: *Entity*.

```
field number --fieldName campo1 --type java.lang.Long
...
field string --fieldName campo2
...
```

Se obtendrá una entidad en la que el tipo del campo identificador es un objeto con distintos campos. Este mecanismo es el mismo que se emplea al realizar la ingeniería inversa de una BBDD en el caso de que se encuentre una estructura de este tipo, es decir, una clave primaria compuesta.

Otra forma de hacer lo mismo sería definir primero la clase que servirá de identificador (en el ejemplo *EntityPK*) y luego definir la entidad indicando que el identificador es del tipo de la clase que hemos creado previamente.

```
embeddable --class ~.domain.Entity2PK
...
field number --fieldName campo1 --type java.lang.Long
...
field string --fieldName campo2
```

```
...
entity jpa --class ~.domain.Entity2 --identifierType ~.domain.EntityPK2
...
```

8.5. Definir características específicas para el modelo relacional

Una vez construido el modelo de entidades de la aplicación mediante ingeniería directa, tal y como se ha visto hasta ahora, al arrancar la aplicación se creará automáticamente el modelo relacional de base de datos a partir del modelo de entidades. Puede ocurrir que se desee que el modelo relacional de base de datos tenga algunas características específicas, en estos casos se pueden modificar las clases Java y plasmar las características específicas deseadas. Observar que todas las características específicas que se van a adaptar modificando la clase Java de las entidades pueden ser definidas en la definición inicial realizada con los comandos *entity* y *field*.

8.5.1. Definir un nombre de tabla

Con la anotación de clase `@RooJpaActiveRecord(..., table = "table_name")` se puede definir el nombre de la tabla donde se guardará la información asociada con la entidad.

Ejemplo 8.2. Uso de la anotación

```
@RooJavaBean
@RooToString
@RooJpaActiveRecord(table = "entity_1")
public class Entity1 {

    private String field1;
}
```

8.5.2. Definir un nombre de columna

Existen varias anotaciones que permiten definir la correspondencia entre atributos de una entidad y las columnas de la tabla.

Las más importantes son:

`@Column`
: Indica el nombre de la columna de la base de datos en donde se almacenará el valor del atributo. Si se omite esta anotación se aplicará una convención por la que el nombre de la columna será el mismo que el del atributo de la clase.

`@Transient`
: Indica que el atributo de la clase Java no tiene representación en la tabla (no será cargado ni almacenado).

`@JoinColumn`, `@JoinColumns`, `@OneToOne`, `@ManyToOne`
: Define la correspondencia de una asociación *N-1*.

`@JoinTable`, `@OneToMany`, `@ManyToMany`, `@OrderBy`
: Define la correspondencia de una asociación *1-N* o *N-M*.

Ejemplo 8.3. Uso de las anotaciones

```
@RooJavaBean
@RooToString
@RooJpaActiveRecord(name = "ENT1")
public class Entity1 {

    private String field1;
                                                                                    ❶
    @Transient
    private String tmpInfo;
                                                                                    ❷
    @Column(name="ORDER_COST", precision=12, scale=2)
    private BigDecimal cost;                                                        ❸

    @ManyToOne(targetEntity = Customer.class)
    @JoinColumn(name = "CUSTOMER_ID")                                               ❹
    private Customer customer;                                                      ❺

    @OneToMany(cascade = CascadeType.ALL, mappedBy = "entity1")
    private Set<Price> prices = new java.util.HashSet<Price>();                     ❻
}
```

❶ Este campo se cargará y almacenará en la columna `field1` de la tabla `ENT1`

❷ Este campo no se tendrá en cuenta en la carga y guardado de la base de datos.

❸ Este campo numérico se cargará y almacenará en la columna `ORDER_COST` de la tabla `ENT1`, usando los valores de *precisión* y *escala* especificados.

❹ Declara que la propiedad tiene origen en una relación *n-1*, con la entidad `Customer`. La tabla relacionada y sus columnas se obtendrán de las declaraciones que se hayan hecho en dicha entidad.

❺ Especifica que la columna de clave ajena en la tabla `ENT1` para esta relación (con `Customer`) es `CUSTOMER_ID`

❻ Declara que el campo `prices` es una relación *1-n* con la entidad `Price`. También declara que los cambios (actualización, borrado, creación, ...) de `Entity1` se aplican en cascada sobre `Price`. El campo (propiedad en la clase java) que estable la clave ajena en `Price` se llama `entity1`.

8.5.3. Campos calculados

Los campos *@Transient* son de utildad cuando necesitamos que en el Objeto de la entidad se almacene o se calcule algún dato y que este no se tenga en cuenta al guardar (sentencias insert o update) la instancia en la BBDD. Por ejemplo, una entidad Persona puede tener la siguiente declaración de campos:

```
public class Persona {

    private String name;
    private String lastName;

    @Transient
    private String fullName;
}
```

Con esta declaración existe la posibildad de guardar temporalemente en el campo *fullName* la concatenación de *name* y *lastName*, por motivos de simplicidad, y sin tener que modificar la estructura de la tabla de la BBDD para ello.

Usando Hibernate como proveedor de persistencia, existe una anotación que puede ser útil para que un registro dado se calcule de manera automática haciendo una cosulta sobre la BBDD. Se trata de la anotación *org.hibernate.annotations.Formula*.

@Formula tiene como atributo una cadena que se usa como sentencia HQL. Esta se ejecuta cada vez que se carga el registro de la BBDD y el resultado se almacena en el campo con esta anotación. Siguiendo con el ejemplo anterior, se podría tener lo siguiente:

```
public class Persona {

    private String name;
    private String lastName;

    @Formula("(SELECT name || ' ' || lastName from persona)")
    private String fullName;
}
```

En el ejemplo cuando se carga una instancia de la entidad persona se ejecutará la consulta definida y en el campo fullName estará disponible el resultado de la consulta. La cosulta puede ser tan compleja como sea necesario, incluso incluir subconsultas. El campo anotado con *@Formula* es tratado como solo lectura y, por tanto, no se tendrá en cuenta en las operaciones de escritura de la BBDD.

Otra forma de calcular valores, sin el uso de campos de la BBDD, consiste en declarar un método en la entidad que realice las operaciones necesarias para devolver el valor calculado. De esta forma, se podrá acceder al cálculo sin tener que declarar un campo (con sus respectivos get/set).

8.5.4. Clave primaria

Por defecto, una clave primaria se representa en la clase java como una propiedad con nombre *id* y tipo `java.lang.Long`. A continuación se muestra un ejemplo de cómo se pueden cambiar distintas características.

Modificar la anotación @RooJpaActiveRecord para ajustar el nombre y el tipo de la columna que hace de clave primaria en la tabla o el nombre de la propiedad identificadora en la clase.

Ejemplo 8.4. Uso de las anotaciones

```

@RooJavaBean
@RooToString
@RooJpaActiveRecord(identifierField="ent1Id",
        identifierType=Long.class, identifierColumn="ENT_ID")
public class Entity1 {

    private String field1;

}
```

Observar que todas estas características se pueden especificar en el momento de la creación de la entidad mediante el comando *entity*.

Si se utiliza Hibernate como proveedor ORM, usará la secuencia *hibernate_sequence* para obtener de forma incremental el identificador al crear un nuevo registro en la entidad. Por tanto, esta secuencia debe existir en la BBDD sobre la que se ejecuta la aplicación, de lo contrario se producirá un error al no poder obtener el siguiente valor de la sequencia antes de crear un nuevo registro. Por defecto, todas las entidades utilizarán la misma secuencia para obtener los identificadores de los registros que se vayan creando.

Si se necesita especificar una sequencia distinta para generar los identificador para cada Entidad se puede utilizar el atributo *--sequenceName* del comando *entity*. Si la entidad ya fue creada con anterioridad y no se desea volver a crearla mediante el comando con la nueva característica, modificar la anotación @RooJpaActiveRecord de la siguiente forma:

```
@RooJavaBean
@RooToString
@RooJpaActiveRecord(sequenceName="ent1_seq_generator")
public class Entity1 {

    private String field1;

}
```

Como se ha comentado, las secuencias que se usan en las Entidades han de existir en la BBDD. Se deberán crear mediante la sintaxis propia de la BBDD o bien delegarlo en Hibernate modificando la propiedad del archivo *persistence.xml*, *hibernate.hbm2ddl.auto* con valor *update* de forma que, al arrancar la aplicación, creará las secuencias necesarias. **Consultar con el administrador de BBDD si se permite la creación de nuevas estructuras.**.

Cuando se usa una secuencia como identificador, antes de insertar un nuevo registro, el ORM consulta el valor de la secuencia que debe asiganarle. Esto implica una cosulta por cada inserción, con el consecuente sobrecoste. Hibernate, por defecto, intenta evitar este sobrecoste reservando un conjunto de valores de la secuencia la primera vez que se usa tras arrancar la aplicación (por defecto reserva los siguientes 50). Mientras no consuma estos valores, no volverá a solicitar el siguiente valor de la secuencia. Para más información sobre la optimización de generadores de secuencia consultar la documentación de Hibernate. Es posible modificar este comportamiento haciendo push-in de la propiedad id y añadiendo a la anotación *@SequenceGenerator* el atributo *allocationSize = 1*.

8.5.5. Campo para el control de concurrencia optimista.

En la ingeniería directa, por defecto, para cada entidad se genera un campo de control de concurrencia optimista con nombre `version` y de tipo `java.lang.Integer`

El control por versión utiliza números de versión, o sellos de fecha (timestamps), para detectar actualizaciones en conflicto y prevenir la pérdida de actualizaciones. Cuando 2 usuarios actualizan el mismo registro de forma simultánea, sin control de concurrencia, los cambios del segundo usuario en actualizar sobreescriben los datos actualizados por el primer, dando la sensación al primer usuario que sus datos se perdieron o nunca existieron.

En el control de concurrencia por versión hay una columna *Version* en la tabla de base de datos que se mapea a un atributo de la entidad anotado con *@Version*. Al hacer update o delete se añade al WHERE la clave primaria y la comparación de versión, *si version != VERSION_DB* no se actualiza nada.

Observar que todas estas características se pueden especificar en el momento de la creación de la entidad mediante el comando entity. Para ajustarlo posteriormente existen otras opciones:

1. Configurar la anotación @RooJpaActiveRecord para ajustar el nombre del campo, el nombre de la columna y el tipo de datos.

```
@RooJavaBean
@RooToString
```

```
@RooJpaActiveRecord(versionField="version",
        versionType=Long.class, versionColumn="ENT_ID")
public class Entity1 {

    private String field1;

}
```

2. Otro ejemplo desactivando el control de concurrencia optimista:

```
@RooJavaBean
@RooToString
@RooJpaActiveRecord(versionField="")
public class Entity1 {

    private String field1;

}
```

Mas adelante se verán otros mecanismos para realizar un control de concurrencia optimista. Para más información consultar el comando **occ checksum set**.

8.5.6. Regeneración de la Base de datos en cada arranque.

Recordar que por defecto, en la ingeniería directa, la configuración que se genera **reconstruye la base de datos en cada ejecución de la aplicación**. Si se desea modificar este comportamiento consultar la sección Actualización automática del esquema .

8.5.7. Creación de una nueva entidad sin comandos

En el ejemplo que se está siguiendo a lo largo de la documentación, se puede definir una nueva entidad *Repartidor* creando la siguiente clase Java. También se puede realizar su creación mediante los comandos *entity* y *field*.

```
package com.springsource.roo.pizzashop.domain;

import org.springframework.roo.addon.javabean.RooJavaBean;
import org.springframework.roo.addon.jpa.activerecord.RooJpaActiveRecord;
import org.springframework.roo.addon.tostring.RooToString;

@RooJavaBean
@RooToString
@RooJpaActiveRecord
public class Repartidor {

        private String name;
}
```

Observar que al salvar la clase Java con las anotaciones correspondientes, en la consola aparecen mensajes informando de la creación de distintos ficheros de aspectos java (*.aj). A continuación, se puede generar los test de integración para la nueva entidad y definir una relación entre PizzaOrder y el Repartidor del pedido.

```
test integration --entity ~.domain.Repartidor
...
field reference --class ~.domain.PizzaOrder --fieldName repartidor
  --type ~.domain.Repartidor
...
```

Finalmente, lanzar los test de integración para confirmar que todo se ha generado y funciona correctamente.

```
roo-gvNIX> perform tests
...
[INFO] BUILD SUCCESSFUL
```

8.6. Ingeniería inversa de entidades

El framework permite realizar una ingeniería inversa multiesquema e incremental de un modelo relacional de base de datos existente, de forma que basándose en la metainformación recopilada de la BBDD se generarán todas las entidades con sus respectivos campos de manera automática. El encargado de esta funcionalidad es el Add-on *Database Reverse Engineering* disponible en el framework de forma automática.

8.6.1. Instalación del driver JDBC

El Add-on DBRE soporta las principales bases de datos relacionales del mercado (MySQL, MS SQL, PostgreSQL, Oracle, ...). El driver necesario será detectado y se ofrecerá su descarga e instalación automática la primera vez que lo requiera alguno de los comandos *database introspect* o *database reverse engineer*.

Aviso

Si se está trabajando en el entorno de la CITMA, antes de seguir adelante es recomendable añadir el repositorio Maven de la CITMA al pom.xml del proyecto. En la sección de recetas existe un ejemplo de como hacerlo.

Aviso

Si se lanza este comando con la configuración actual del proyecto que usa HYPERSONIC_PERSISTENT como BBDD indicará que no hay driver disponible para HSQLDB.

Por ejemplo, si se ejecuta la ingeniería inversa en un proyecto con PostgreSQL configurado como sistema de base de datos:

```
roo-gvNIX> database introspect --schema unable-to-obtain-connection
Located add-on that may offer this JDBC driver
1 found, sorted by rank; T = trusted developer; R = Roo 1.2 compatible
ID T R DESCRIPTION ------------------------------------------------------------
01 Y Y 9.1.0.901_0001 Postgres #jdbcdriver...
-------------------------------------------------------------------------------
[HINT] use 'addon info id --searchResultId ..' to see details about a search result
[HINT] use 'addon install id --searchResultId ..' to install a specific search result, or
[HINT] use 'addon install bundle --bundleSymbolicName TAB' to install a specific
  add-on version
JDBC driver not available for 'org.postgresql.Driver'
```

Esta salida del comando *database introspect* informa que hay disponible un driver para el soporte de DBRE y sugiere qué hacer a continuación (lineas que empiezan con [HINT]).

Se puede obtener más información sobre el driver mediante:

```
roo-gvNIX> addon info id --searchResultId 01
Name.........: Spring Roo - Wrapping - postgresql-jdbc4
```

```
BSN.........: org.springframework.roo.wrapping.postgresql-jdbc4
Version.....: 9.1.0.901_0001 [available versions: 9.0.801.0001,
              9.1.0.901_0001]
Roo Version..: 1.2
Ranking.....: 0.0
JAR Size.....: 513073 bytes
PGP Signature: 0xEC67B395 signed by Alan Stewart (stewarta@vmware.com)
OBR URL.....: http://spring-roo-repository.springsource.org/repository.xml
```

Donde se está pidiendo más información (--searchResultId) del Add-on 01 que ha devuelto el comando anterior.

Para instalarlo usar el id del Add-on o bien su nombre simbólico.

```
addon install id --searchResultId 01
```

o

```
addon install bundle
        --bundleSymbolicName org.springframework.roo.wrapping.postgresql-jdbc4
```

Al instante estará disponible el driver JDBC y se podrá empezar a usar los comandos *database introspect* y *database reverse.*

8.6.2. Comandos de la ingeniería inversa

DBRE ofrece dos comandos: *database introspect* y *database reverse engineer*.

```
database introspect --schema <nombre del esquema>
        --file <nombre del fichero> --enableViews
```

El comando mostrará la estructura de la BBDD en formato XML. Es obligatorio indicar el esquema, para ello presionado la tecla TAB se mostrará la lista de esquemas de la BBDD. Algunas BBDD no usan el concepto de esquema, MySQL entre ellas, aun así la opción --schema es necesaria y por ello el asistente mostrará como esquema "no-schema-required". La opción --file indica que guarde la información también en un fichero determinado y --enableViews especifica que también debe obtener información sobre las vistas.

```
database reverse engineer --schema
        --package --testAutomatically --enableViews --includeTables
        --excludeTables --includeNonPortableAttributes
```

Este comando crea las entidades JPA del proyecto representando las tablas y columnas de la BBDD. Al igual que antes --schema es obligatorio. Se debe usar --package para indicar el nombre del paquete del proyecto donde crear las clases Java, como convención se recomienda utilizar el subpaquete domain dentro del paquete base, ese decir, *~.domain*.

Podemos indicar también que se generen automáticamente los test de integración para cada una de las nuevas entidades creadas con --testAutomatically.

Si se especifica la opción --enableViews en este comando, se crearan entidades JPA que representen las vistas que se encuentren en la inspección de la BBDD.

Se pueden filtrar las tablas que se van a considerar en la operación. --includeTables indicará cuales son las tablas que deben ser tenidas en cuenta. Se pueden indicar uno o varios nombres de tabla, si se indican más de uno, se deberán especificar separadas por espacios y todas ellas entre comillas dobles. Se permite el uso de * para hacer matching de uno o varios caracteres en la búsqueda o el uso de ?

para indicar un único caracter. De manera opuesta --excludeTables indica qué tablas no han de ser tenidas en cuenta durante la ingeniería inversa y admite el mismo uso de * y ?. Las exclusiones de tablas son requeridas especialmente en aquellas ocasiones en las que tablas incluidas en la ingeniería inversa tienen relaciones con otras tablas no incluidas, por lo que estas segundas deberán excluirse. Un ejemplo:

```
database reverse engineer --schema no-schema-required
        --package ~.domain --includeTables "foo* bar?"
```

Con este comando se generarían las entidades de aquellas tablas de la BBDD que tengan como nombre foo<cualquier cosa> y bar<otro caracter más>, por ejemplo 'foo_tabla_cool' y 'bars'.

Aviso

Al excluir tablas, además de evitar que se creen las entidades de estas tablas se evita también que se creen las asociaciones y relaciones en otras entidades. Esto se hace para evitar problemas de compilación en el código del proyecto.

Como DBRE ofrece ingeniería inversa incremental, se puede ejecutar este comando tantas veces como sea necesario y de esta forma se mantendrán todas las entidades JPA de manera automática.

El parámetro **--includeNonPortableAttributes** puede ser de utilidad. En versiones anteriores de la funcionalidad de ingeniería inversa de BBDD este parámetro no existía y, por defecto, cuando se generaban las entidades se incluía en las anotaciones JPA un atributo que rompía la portabilidad del código Java para trabajar con BBDD distintas a la de origen. El atributo en cuestión es *columnDefinition* en la anotación *@Column*, que sirve para indicar el tipo de dato usado para crear la columna en la BBDD. Este atributo se utiliza en dos fases: cuando se delega en JPA la creación de las tablas de la BBDD (propiedad *hibernate.hbm2ddl.auto* con valor *create* de Hibernate), de esta forma se indica el tipo de dato físico utilizado en la BBDD y también se utiliza en la fase de validación del esquema de BBDD en el arranque de la aplicación (propiedad *hibernate.hbm2ddl.auto* con valor *validate*), que comprueba que las entidades del proyecto cumplen con el esquema de BBDD y por tanto son compatibles.

Por lo tanto, hay que tener en cuenta este detalle al hacer la ingeniería inversa de la BBDD. Si se va a seguir trabajando con la misma BBDD, se puede optar por requerir que se incluyan los atributos "no portables" en la generación de las entidades. Por contra, si la aplicación se va a ejecutar en distintas bases de datos, es mejor no incluir estos atributos y no usar este parámetro del comando *database reverse engineer*.

Importante

Cuando se utiliza el Add-on de ingeniería inversa se modifica el fichero META-INF/persistence.xml cambiando el valor de la propiedad *hibernate.hbm2ddl.auto* por el de *none* para evitar poner en peligro la integridad de la BBDD a la que se está conectando desde el proyecto. Si se ha utilizado la opción --includeNonPortableAttributes el valor de esta propiedad será *validate* de modo que en el arranque de la aplicación se realizará la fase de validación del esquema.

Importante

Si se realiza la ingeniería inversa de alguna vista, es recomendable leer la sección Ingeniería inversa de vistas ya que puede ser necario realizar alguna personalización.

8.6.3. Anotación @RooDbManaged

Tal y como se ha comentado en la sección Crear una entidad con un IDE se hace uso de una serie de anotaciones Java para que el framework gestione el código del proyecto. A las anotaciones descritas anteriormente se suma ahora la anotación @RooDbManaged que se añade a todas las entidades creadas durante el proceso de ingeniería inversa.

```
@RooJavaBean
@RooToString
@RooDbManaged(automaticallyDelete = true)
@RooJpaActiveRecord
public class Pizza {
}
```

El atributo *automaticallyDelete* que acompaña a la anotación y que toma el valor *true* indica que se debe eliminar la entidad en el caso de que la tabla de la BBDD a la cual hace referencia sea eliminada. Si el atributo toma el valor *false* la entidad no se eliminará en caso de que la tabla desaparezca de la BBDD.

8.6.4. Soporte de funcionalidades JPA 2.0

El Add-on DBRE crea y mantiene claves primarias simples, claves primarias compuestas, relaciones entre entidades, restricciones tamaño, gestión de valores nulos, etc. A continuación, se verá como realiza la gestión de algunos de estos elementos.

8.6.4.1. Claves primarias simples

Para el caso de claves primarias simples DBRE genera un campo identificador en el aspecto java de la entidad marcado con la anotación @Id de forma similar a como se hace al ejecutar el comando *entity*.

8.6.4.2. Claves primarias compuestas

En este caso DBRE utiliza una solución más elaborada. Crea una clase Java que representa la clave primaria anotándola con @RooIdentifier(dbManaged = true) y añade el atributo "identifierType" con el nombre de la clase del identificador a la anotación @RooJpaActiveRecord en la clase de la entidad. Por ejemplo:

```
@RooJavaBean
@RooToString
@RooDbManaged(automaticallyDelete = true)
@RooJpaActiveRecord(identifierType = LineItemPK.class, table = "line_item",
  schema = "order")
public class LineItem {
}
```

```
@RooIdentifier(dbManaged = true)
public class LineItemPK {
}
```

En el aspecto java de la entidad LineItem existirá un campo anotado con @EmbeddedId del tipo LineItemPK:

```
privileged aspect LineItem_Roo_Jpa_Entity {

    declare @type: LineItem: @Entity;

    declare @type: LineItem: @Table(name = "line_item", schema = "order");
```

```
    @EmbeddedId
    private LineItemPK LineItem.id;

    public LineItemPK LineItem.getId() {
        return this.id;
    }

    public void LineItem.setId(LineItemPK id) {
        this.id = id;
    }

    ...
}
```

Y en el aspecto java de la clase que representa la clave compuesta estarán los campos de la clave primaria con la anotación @Embeddable:

```
privileged aspect LineItemPK_Roo_Identifier {

    declare @type: LineItemPK: @Embeddable;

    @Column(name = "line_item_id", nullable = false)
    private BigDecimal LineItemPK.lineItemId;

    @Column(name = "order_id", nullable = false)
    private BigDecimal LineItemPK.orderId;

    public LineItemPK.new(BigDecimal lineItemId, BigDecimal orderId) {
        super();
        this.lineItemId = lineItemId;
        this.orderId = orderId;
    }

    private LineItemPK.new() {
        super();
    }

    ...
}
```

8.6.5. Ingeniería inversa multi esquema

La ingeniería inversa proporcionada por el framework ofrece soporte para la conexión con múltiples esquemas de la misma base de datos de forma automática.

El opción *--schema* de los comandos de la ingeniería inversa permiten especificar una lista de esquemas separados por espacios y englobados todos ellos por dobles comillas.

La ingeniería inversa también se puede hacer de distintas fuentes de datos utilizando para ello la documentación de la sección Múltiples fuentes de datos.

8.6.6. Ingeniería inversa incremental

La ingeniería inversa proporcionada por el framework permite ejecutarse todas las veces que sea necesario de forma que se pueden modificar las características y elementos incluidos en ingenierías inversas anteriores.

Si al hacer una ingeniería inversa no se incluyó alguna tabla que posteriormente se ha visto necesario, basta con volver a ejecutar el comando de ingeniería inversa incluyendo la nueva tabla. Todas las modificaciones realizadas sobre las entidades que ya existían no se perderán.

8.6.7. Ingeniería inversa de vistas

Tras realizar la ingeniería inversa de una vista de la base de datos es muy probable que se tengan que realizar algunas modificaciones manuales para que funcione correctamente y que no son posibles realizar de manera automática ya que se requiere tomar ciertas decisiones de diseño.

En una vista, por definición, no existe definida ninguna clave primaria. Sin embargo, al convertir esta estructura relacional al modelo orientado a objetos es necesario definir uno o varios campos que identifiquen de forma unívoca cada resultado que se vaya a obtener de la vista.

La ingeniería inversa de vistas, por defecto, configura todas las columnas de la vista como clave primaria compuesta. Es por ello que todas las propiedades serán creadas en la clase con sufijo PK, por ejemplo EntidadPK, y no se creará ninguna propiedad en la clase base, por ejemplo Entidad.

Habitualmente algunas columnas de la vista serán opcionales. Sin embargo, en el modelo orientado a objetos es incorrecto que alguna de las propiedades que representan a la clave primaria compuesta tenga valores nulos.

Es por ello que deberemos analizar el modo en el que está construida la vista en la base de datos para así identificar uno o varios campos que representen de forma unívoca cada resultado de la vista. En algunos casos en los que la vista de alimenta de otra tabla, la clave primaria de la vista suele ser la misma que la de la tabla.

Una vez detectadas las propiedades que conforman la clave primaria de la vista deberán realizarse las siguientes modificaciones:

Copiar propiedades en la entidad.
: Copiar las propiedades de la clase EntidadPK que no representan a la clave primaria en la clase Entidad. Las propiedades de la clase EntidadPK se encuentran en el fichero EntidadPK_Roo_Identifier.aj. Al copiar las propiedades, recordar eliminar el prefijo "EntidadPK." que tiene cada una de las propiedades.

Modificar propiedades en la clave primaria.
: Hacer push-in de las propiedades de la clase EntidadPK que no representan a la clave primaria compuesta, añadirles la anotación @Transient y eliminar cualquier otra anotación que tengan. Las propiedades de la clase EntidadPK se encuentran en el fichero EntidadPK_Roo_Identifier.aj y al hacer push-in pasarán a estar en el fichero EntidadPK.java.

Desactivar operaciones en la capa de entidades.
: Añadir a la anotación @RooJpaActiveRecord de la clase Entidad los siguientes atributos que desactivarán el borrado, creación y actualización en la capa de entidades: removeMethod = "", persistMethod = "", mergeMethod = "".

Desactivar operaciones en la capa web.
: Si se ha generado la capa web asociada a la entidad que representa a la vista, añadir a la anotación @RooWebScaffold de la clase EntidadController los siguientes atributos que desactivarán el borrado, creación y actualización en la capa web: removeMethod = "", persistMethod = "", mergeMethod = "".

8.6.7.1. Tests de la ingeniería inversa de vistas

En las vistas que no realizamos push-in e identificamos un elemento como clave primaria hay que realizar los siguientes cambios si deseamos mantenerlos ejecutándose con éxito. Como ejemplo pondremos VaumAplusuPerfiles.

Realizar push-in

Realizar push-in de todos los métodos: persist, remove, los finder y count.

Modificar count y persist

Al count y persist le eliminaremos la siguiente línea, que es la primera que tiene en código: Assert.assertNotNull("Data on demand for 'VaumAplusuPerfiles' failed to initialize correctly", dod.getRandomVaumAplusuPerfiles()); Esto no afecta a la comprobación de si el count funciona correctamente.

Modificar finder un resultado

Al finder de un objeto de la entidad en concreto (testFindVaumAplusuPerfiles) lo que haremos será insertar un objeto que sabemos que todos los valores no son null utilizando dod.getNewTransientVaumAplusuPerfiles(Integer.MAX_VALUE); y posteriormente buscaremos dicho elemento: Sustituimos VaumAplusuPerfiles obj = dod.getRandomVaumAplusuPerfiles(); por VaumAplusuPerfiles obj = dod.getNewTransientVaumAplusuPerfiles(Integer.MAX_VALUE); obj.persist();

Modificar finder de múltiples resultados

Al finder All y ...Entries tendremos que eliminar las líneas assertNotNull situadas al principio del método y justo debajo de obtener el listado. Assert.assertNotNull("Data on demand for 'VaumAplusuPerfiles' failed to initialize correctly", dod.getRandomVaumAplusuPerfiles()); Assert.assertNotNull("Find all method for 'VaumAplusuPerfiles' illegally returned null", result);

Al remove le sustetuiremos VaumAplusuPerfiles obj = dod.getRandomVaumAplusuPerfiles(); por VaumAplusuPerfiles obj = dod.getNewTransientVaumAplusuPerfiles(Integer.MAX_VALUE); obj.persist();

Capítulo 9. Buscadores de entidades

En la sección de recetas se puede encontrar información adicional para desarrollar buscadores con gran cantidad de campos y también para hacer los campos opcionales en los buscadores .

9.1. Descripción

El framework permite generar buscadores sobre cualquier entidad del modelo de la aplicación mediante el comando finder add. El comando espera el nombre del buscador a generar en un formato determinado. Para conocer el formato se puede utilizar como ayuda el comando finder list.

En los siguientes puntos se muestra un ejemplo de como elegir y generar un buscador sobre una de las entidades del proyecto del script de ejemplo *clinic.roo* que viene incluido con el framework. Se utilizará la entidad Owner para generar un buscador sobre ella.

9.2. Listar buscadores

Ejecutar el comando finder list sobre la entidad Owner

```
roo-gvNIX> finder list --class ~.domain.Owner --depth 2 --filter LastName
  ...
  findOwnersByLastNameLikeAndCityLike(String lastName, String city)
  ...
```

En el comando se indica la entidad sobre la que se desea consultar sus buscadores con *--class*. El resto de parámetros son opcionales. El parámetro *--depth* indica el nivel de profundidad a tener en cuenta, es decir la cantidad de propiedades a combinar en el buscador. Debido a la enorme cantidad de combinaciones de propiedades que puede existir para una entidad, está limitado a un máximo de 3. El parámetro *--filter* aplica un filtro de forma que solo se mostrarán los posibles buscadores que contengan la cadena indicada.

El comando devuelve una lista de resultados con todos los posibles buscadores que coinciden con los términos de búsqueda, de entre ellos se va a escoger findOwnersByLastNameLikeAndCityLike que genera un buscador para la entidad Owners cuyo *lastName* y *city* contenga la cadena que se le pase como parámetro respectivamente, ambas utilizando para ello el operador *like*.

Consultar la documentación del comando finder list para más información.

9.3. Creación de un buscador

Es importante resaltar que no es necesario utilizar el comando *finder list* antes de utilizar el comando *finder add*. Una vez conocida la nomenclatura utilizada para definir los buscadores no es complicado definir directamente el nombre del buscador deseado. A continuación, se muestra una pequeña guía con el formato que sigue la definición del nombre de un buscador:

```
findEntitiesByPropertyFilterOperator
```

- Entities: Nombre de la entidad sobre la que generar el buscador. Su nombre debe ser escrito en plural, utilizando para su pluralización las normas del lenguage Inglés. Por ejemplo, la entidad *Entity* tomará como plural *Entities*.

- La porción del buscador *PropertyFilter* puede repetirse tantas veces como propiedades se desee añadir al filtro siempre y cuando se utilice un *Operator* para separar el filtro sobre una propiedad y el siguiente.
  - Property: Nombre de la propiedad de la entidad a incluir como filtro en el buscador. Debe tener el mismo nombre que la propiedad definida en la entidad, pero comenzando por mayúscula.
  - Filter: Tipo de filtro a aplicar sobre la propiedad anterior de entre los siguientes posibles valores en función del tipo de dato de la propiedad.
    - Cadenas (String, ...): Equals, IsNotNull, IsNull, Like, NotEquals.
    - Numéricos (Float, Integer, Long, ...): Between, Equals, GreaterThan, GreaterThanEquals, IsNotNull, IsNull, LessThan, LessThanEquals, NotEquals.
    - Fechas (Date, Calendar, ...): Between, Equals, GreaterThan, GreaterThanEquals, IsNotNull, IsNull, LessThan, LessThanEquals, NotEquals.
    - Booleanos (Boolean, ...): Not.
    - Enumerados (Enum, ...): No aplica. El filtro ya viene definido por el valor de la enumeración que se le proporcionará al buscador para esta propiedad.
    - Relación simple con otra entidad (Pet, Vet, ...): No aplica. El filtro ya viene definido por la entidad que se le proporcionará al buscador para esta propiedad.
    - Relaciones múltiples (Set, ...): No aplica. El filtro ya viene definido por la lista de valores que se le proporcionará al buscador para esta propiedad.
  - Operator: And o Or. Es el operador que se utilizará para relacionar el filtro sobre una propiedad y el siguiente.

Algunos ejemplos serían: findOwnersByFirstNameLike, findPetsByWeightGreaterThan, findVetsByBirthDayBetween, findPetsBySendRemindersNot, findPetsByType, findVisitsByPet, findOwnersByPets, findOwnersByTelephoneIsNotNullAndBirthDayEquals, findPetsByWeightLessThanEqualsOrTypeOrSendRemindersNot.

Al ejecutar el comando *finder add* con el nombre del buscador deseado se generará el buscador en la capa de las entidades, pero no se generará todavía ningún código en la capa de la vista.

```
roo-gvNIX> finder add --class ~.domain.Owner
      --finderName findOwnersByLastNameLikeAndCityLike
  Updated SRC_MAIN_JAVA/com/springsource/petclinic/domain/Owner.java
  Created SRC_MAIN_JAVA/com/springsource/petclinic/domain/Owner_Roo_Finder.aj
```

Consultar la documentación del comando finder add para más información.

9.4. Código generado

La ejecución del comando *finder add* desencadenará la modificación de una anotación en la clase Java de la entidad especificada.

```
...
  @RooJpaActiveRecord(finders = { "findOwnersByLastNameLikeAndCityLike" })
```

```
  public class Owner extends AbstractPerson {
  }
```

La aparición del atributo *finders* en la anotación *RooJpaActiveRecord* de la entidad provocará la generación del aspecto Java correspondiente que implementará el buscador solicitado.

```
privileged aspect Owner_Roo_Finder {

   public static TypedQuery<Owner> Owner.findOwnersByLastNameLikeAndCityLike(
     String lastName, String city) {
       if (lastName == null || lastName.length() == 0)
         throw new IllegalArgumentException("The lastName argument is required");
       lastName = lastName.replace('*', '%');
       if (lastName.charAt(0) != '%') {
           lastName = "%" + lastName;
       }
       if (lastName.charAt(lastName.length() - 1) != '%') {
           lastName = lastName + "%";
       }
       if (city == null || city.length() == 0) throw new IllegalArgumentException(
         "The city argument is required");
       city = city.replace('*', '%');
       if (city.charAt(0) != '%') {
           city = "%" + city;
       }
       if (city.charAt(city.length() - 1) != '%') {
           city = city + "%";
       }
       EntityManager em = Owner.entityManager();
       TypedQuery<Owner> q = em.createQuery("SELECT o FROM Owner AS o WHERE " +
         "LOWER(o.lastName) LIKE LOWER(:lastName) AND LOWER(o.city) LIKE LOWER(:city)",
         Owner.class);
       q.setParameter("lastName", lastName);
       q.setParameter("city", city);
       return q;
   }
  }
```

Capítulo 10. Pruebas de integración

10.1. Creación de pruebas de integración

En la creación de entidades, al especificar el parámetro **--testAutomatically** se generan automáticamente las pruebas de integración que permitirán validar el buen funcionamiento del código de persistencia de la entidad.

Si no se solicitó la generación de las pruebas de integración en el momento de la creación de una entidad, se pueden generar posteriormente con el comando *test integration*.

Para ejecutar los test utilizar el comando **perform tests**:

```
~.domain.PizzaOrder roo-gvNIX> perform test
  [Thread-4] [INFO] Scanning for projects...
  [Thread-4] [INFO] ------------------------------------------------------------------------
  [Thread-4] [INFO] Building pizzashop
  [Thread-4] [INFO]    task-segment: [test]
  [Thread-4] [INFO] ------------------------------------------------------------------------
  [Thread-4] [INFO] [aspectj:compile {execution: default}]
  ...
  ...
  [Thread-4] -------------------------------------------------------
  [Thread-4]  T E S T Soo>
  [Thread-4] -------------------------------------------------------
  [Thread-4] Running com.springsource.roo.pizzashop.domain.PizzaOrderIntegrationTest
  [Thread-4] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 4.389 sec
  [Thread-4] Running com.springsource.roo.pizzashop.domain.ToppingIntegrationTest
  [Thread-4] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.174 sec
  [Thread-4] Running com.springsource.roo.pizzashop.domain.PizzaIntegrationTest
  [Thread-4] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.145 sec
  [Thread-4] Running com.springsource.roo.pizzashop.domain.BaseIntegrationTest
  [Thread-4] Tests run: 9, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.137 sec
  [Thread-4] zzaOrder roo-gvNIX>
  [Thread-4] Results :roo-gvNIX>
  [Thread-4] zzaOrder roo-gvNIX>
  [Thread-4] Tests run: 36, Failures: 0, Errors: 0, Skipped: 0
  [Thread-4] zzaOrder roo-gvNIX>
  [Thread-4] [INFO] ------------------------------------------------------------------------
  [Thread-4] [INFO] BUILD SUCCESSFUL
  [Thread-4] [INFO] ------------------------------------------------------------------------
  [Thread-4] [INFO] Total time: 17 seconds
  [Thread-4] [INFO] Finished at: Wed Jun 16 21:10:22 CEST 2010
  [Thread-4] [INFO] Final Memory: 45M/299M
  [Thread-4] [INFO] ------------------------------------------------------------------------
  ~.domain.PizzaOrder roo-gvNIX>
```

Capítulo 11. Gestión de la capa web

La capa web de la aplicación contiene la infraestructura necesaria para gestionar las peticiones HTTP y visualizar cierta información al usuario como respuesta. Actualmente esta capa puede desarrollarse con Spring MVC, GWT, Flex, JSF y Vaadin. Cada una de estas tecnologías está integrada a distinto nivel con la generación de código del framework por lo que cada una de ellas ofrecerá ciertas características. Este capítulo se centrará en la generación de la capa web con Spring MVC ya que se trata de la tecnología más evolucionada dentro de la generación de código del framework y para la que gvNIX ha añadido la mayoría de sus nuevas funcionalidades.

La gestión de peticiones en Spring MVC se basa en los controladores, clases en las que, mediante anotaciones, se definen los métodos que atienden peticiones HTTP.

Un controlador es una clase Java anotada con `@Controller` a la que le llegan las peticiones que realiza el usuario vía el navegador web. Identifica las URLs que atiende mediante la anotación `@RequestMapping`. Esta última anotación se puede usar tanto en la clase como en los métodos del controlador.

Actualmente se pueden generar automáticamente los controladores necesarios para gestionar los objetos de la aplicación: crear, leer, actualizar, borrar y buscar (CRUDS).

11.1. Crear la capa web con el intérprete de comandos

Desde el intérprete de comandos se puede crear la capa web en Spring MVC de forma sencilla, con los comandos siguientes:

- **web mvc setup**
- **web mvc scaffold**
- **web mvc all**
- **web mvc controller**

11.1.1. web mvc setup

Este comando crea toda la configuración necesaria para el funcionamiento de Spring MVC en el proyecto. Esto implica varios ficheros de configuración preparados para su correcto funcionamiento en el proyecto actual, recursos gráficos, hojas de estilo, soporte multidioma, páginas iniciales, tags, menú, etc.

11.1.2. web mvc scaffold

Este comando crea un controlador con el CRUD (creación, lectura, actualización y borrado) de la entidad especificada. No hay restricciones para crear varios controladores para la misma entidad.

Este comando, además de generar el controlador, genera también las vistas para la creación, visualización, listado y actualización de los registros de la entidad y toda la configuración necesaria para que las peticiones del usuario lleguen al controlador y se visualize al usuario la respuesta (página) correspondiente.

Las URL de las peticiones que acepta el Controller generado siguen el convenio RESTful (en español). Toda la lógica se genera en un fichero `*_Roo_*.aj` asociado al controller, de forma que la clase Java queda limpia de los métodos generados automáticamente.

Los parámetros obligatorios para este comando son:

--class
: Nombre de la clase java para el controller (incluido su paquete). Recordar que se puede usar el ~ para especificar el paquete base de la aplicación. Por convención, estas clases se suelen emplazar en el paquete `~.web` y con el formato `EntidadController`.

Los parámetros opcionales para este comando son:

--backingType
: Nombre de la clase Entidad que manejará el controlador. Si no se especifica este parámetro, se tomará como valor la clase en la cual se encuentre el contexto de la consola.

--path
: Ruta base para la generación de la URL para las peticiones basadas en RESTful.

--disallowedOperations
: Lista de operaciones, separada por comas, de las operaciones NO permitidas en este controller. Los elementos de esta lista deben estar entre estos: `create update delete`.

11.1.3. web mvc all

Este comando ejecuta un web mvc scaffold para cada una de las entidades registradas en la aplicación.

Los parámetros para este comando son:

--package
: Nombre del paquete donde se generarán los controladores. Por convención, se suele utilizar el nombre de paquete `~.web`.

Un ejemplo:

```
roo-gvNIX> web mvc setup
roo-gvNIX> web mvc all --package ~.web
```

11.1.4. web mvc controller

Este comando permite crear un controlador de uso general no vinculado a ninguna entidad. Servirá para realizar implementaciones a medida.

Los parámetros para este comando son:

--class
: Nombre de la clase java para el controlador (incluido su paquete). Recordar que se puede usar el ~ para especificar el paquete base de la aplicación.Por convención, estas clases se suelen emplazar en el paquete `~.web`.

--preferredMapping
: Permite definir la ruta de la petición para este controller.

Un ejemplo:

```
roo-gvNIX> web mvc setup
roo-gvNIX> web mvc controller --class ~.web.MyController
```

Genera una clase como esta:

```
@RequestMapping("/my/**")
@Controller
public class MyController {

    @RequestMapping
    public void get(ModelMap modelMap, HttpServletRequest request,
      HttpServletResponse response) {
    }

    @RequestMapping(method = RequestMethod.POST, value = "{id}")
    public void post(@PathVariable Long id, ModelMap modelMap, HttpServletRequest request,
      HttpServletResponse response) {
    }

    @RequestMapping
    public String index() {
        return "my/index";
    }
}
```

En este ejemplo la clase atenderá cualquier petición cuya URL comience por `{URLBase}/my` definido por la anotación `@RequestMapping` asociada a la clase. El método receptor, tal cual están definidas las anotaciones `@RequestMapping` en los métodos, cumplirán las siguientes reglas:

1. Las peticiones de tipo POST con una petición cuya URL cumpla `{URLBase}/myController/{entero}` entrarán por el método `post` de la clase.

2. Las peticiones a `{URLBase}/my/index` se atenderán en el método `index` que mostrará la vista `my/index`.

3. El resto de peticiones serán atendidas en el método `get` de la clase.

Para más información sobre la anotación `@RequestMapping` ver la documentación de Spring MVC.

Este comando, además, generará la vista la vista `WEB-INF/views/my/index.jspx`, añadirá una entrada de menú para ella y actualizará el fichero `WEB-INF/i18n/application.properties` con entradas de internacionalización.

11.1.5. Proyecto ejemplo

Continuando con el proyecto del tutorial, la capa web de la aplicación de venta de pizzas se creará con la siguiente secuencia de comandos:

```
~.domain.PizzaOrder roo-gvNIX> web mvc setup
~.domain.PizzaOrder roo-gvNIX> web mvc all --package ~.web
```

11.2. Crear la capa web con un IDE

Para que una clase `Java` sea un controlador de Spring MVC debe estar anotada con la anotación `@Controller` y añadir la anotación `@RequestMapping` como convenga para definir la correspondencia

entre una URL y los métodos del controlador. Los controladores pueden ser a medida, para realizar las operaciones específicas o se pueden crear controladores para la gestión del CRUD (creación, lectura, actualización y borrado) de las entidades.

11.2.1. Controlador a medida

Para crear un controlador a medida, además de generar la clase con la anotaciones pertinentes, se debe tener en cuenta las siguientes cuestiones:

- Si se utiliza una vista, crearla dentro de `WEB-INF/views/{path}` junto con un fichero `WEB-INF/views/{path}/views.xml` de configuración del `Tiles`.
- Si se utilizan cadenas de literales susceptibles de internacionalización, darlas de alta en los ficheros `WEB-INF/i18n/*.properties` para los idiomas disponibles.

11.2.2. Controlador CRUD

Se puede crear un controlador en Spring MVC que realice de forma automática la creación, lectura, actualización y borrado de registros de la entidad. Estas clases tienen el siguiente aspecto:

```
@RequestMapping("/bases")
@Controller
@RooWebScaffold(path = "bases", formBackingObject = Base.class)
public class BaseController {
}
```

Estas clases contienen, además de las anotaciones ya explicadas, la anotación `@RooWebScaffold` que admite los siguientes atributos obligatorios:

Y también admite los siguientes atributos opcionales:

path
: Ruta base del controlador. Debe coincidir con la ruta de la anotación `@RequestMapping` de la clase. También se usará como ruta base para crear las vistas automáticas a partir de `WEB-INF/views/` en la aplicación.

formBackingObject
: Objeto que maneja el controlador. Debe ser una clase java anotada como entidad.

update, delete y create
: Define si deben permitirse estas operaciones en el controlador. Por defecto `true`.

populateMethods
: Indica si deben generarse los métodos encargados de publicar las listas de valores que sean necesarias en las vistas. Por defecto `True`.

Cuando el framework detecte la anotación realizará las siguientes operaciones:

1. Creará un fichero *_Roo_Controller.aj donde generará todos los métodos para gestionar las peticiones que lleguen al controller.
2. Añadirá las vistas necesarias en `WEB-INF/views/{nombre_controller}` para gestionar las acciones configuradas en la anotación (creación, actualización y/o borrado).

3. Añadirá las entradas de menú necesarias.

4. Añadirá en el fichero `WEB-INF/i18n/application.properties` las entradas multidioma necesarias.

11.2.3. Código generado en las vistas de la capa web

Cuando se genera capa web de la aplicación lo hace generando ficheros jspx y tagx. Los ficheros jspx no son más que JSPs que tienen un formato XML válido. A su vez las jspx hacen uso de tags de las librerías JSTL, Spring Framework y otras que se generán también como parte de la aplicación ubicadas en WEB-INF/tags.

Ya que es una buena práctica la reutilización de código o de componentes de la vista web, se utiliza el motor de plantillas Apache Tiles. Este motor de plantillas permite definir fragmentos que pueden ser integrados en una página completa en tiempo de ejecución, facilitando el desarrollo de la capa de presentación de una forma consistente en toda la aplicación.

11.2.3.1. Estructura de archivos de las vistas en la capa web

En el proyecto de ejemplo PizzaShop, se habrá creado hasta el momento la estructura de directorios y archivos que se puede ver en la siguiente imagen:

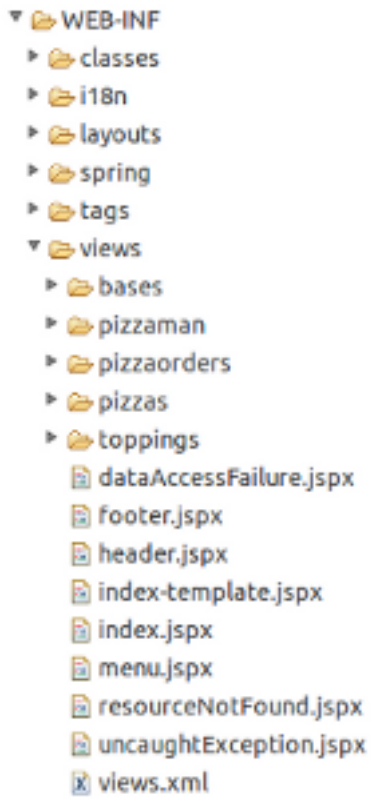

Figura 11.1. Estructura archivos Pizza Shop

De todas las carpetas anteriores el desarrollo se centrará en *layouts, tags y views*.

11.2.3.2. Motor de plantillas: Tiles

Dentro de *views* hay varios ficheros jspx y varias carpetas. Entre los ficheros se encuentra *views.xml*. Este fichero establece la configuración de vistas que Tiles deberá generar. Tiles se basa en que todas las vistas de una aplicación web son similares, tienen el mismo diseño y estructura, pero cada página está

compuesta de distintos trozos que conforman el contenido pero siempre colocados de la misma forma. Al mismo tiempo permite modificar ciertos aspectos del diseño para adaptarlos a las necesidades de una vista concreta. Por ejemplo, es común que una página web tenga una cabecera, donde se puede ver el título de la misma, un menú con las distintas secciones que hay en el site, un pie de página y una parte principal con el contenido. Tiles nos brinda la posibilidad de especificar que una disposición de la página contiene estos cuatro bloques, pero por contra en alguna sección o situación (por ejemplo: el usuario de la web está o no logado en la aplicación) se desea que el menú no se muestre.

```
<definition name="index" extends="default">
  <put-attribute name="body" value="/WEB-INF/views/index.jspx" />
</definition>
```

El código anterior, extraido de *view.xml*, define una vista llamada *index* que se basa en, extiende de, otra vista *default* y a la que se le pasa como atributo *body* la página definida por el fichero *index.jspx*. La definición de la vista *default* ayudará a entender esta otra. *default* se define en el fichero *WEB-INF/layouts/layouts.xml*.

```
<definition name="default" template="/WEB-INF/layouts/default.jspx">
  <put-attribute name="header" value="/WEB-INF/views/header.jspx" />
  <put-attribute name="menu" value="/WEB-INF/views/menu.jspx" />
  <put-attribute name="footer" value="/WEB-INF/views/footer.jspx" />
</definition>
```

En la definición de esta vista se indica que la plantilla (*template*) que va a dibujar está parte de la web está programada en la JSP *default.jspx*. Además, aquí ya se observan otros fragmentos que van a definir la composición de la vista: la cabecera (*header*), menú (*menu*) y pie de página (*footer*), los cuales serán dibujados por las respectivas páginas jspx. Volviendo a la definición de la vista *index* del fragmento de código anterior, se observa cómo uniendo los atributos de las dos definiciones se obtiene que la vista *index* se compone de los cuatro fragmentos que comentados anteriormente: cabecera, menú, pie de página y cuerpo. Así pues, es sencillo definir una nueva vista con esta misma estructura simplemente indicando que el contenido principal, el cuerpo, viene definido por otra JSP cualquiera.

Como aclaración, se puede analizar el contenido del fichero *default.jspx*:

```
<html xmlns:jsp="http://java.sun.com/JSP/Page"
 xmlns:c="http://java.sun.com/jsp/jstl/core"
 xmlns:tiles="http://tiles.apache.org/tags-tiles"
 xmlns:spring="http://www.springframework.org/tags"
 xmlns:util="urn:jsptagdir:/WEB-INF/tags/util" >

  <jsp:output doctype-root-element="HTML" doctype-system="about:legacy-compat" />

  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <jsp:directive.page pageEncoding="UTF-8" />

  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=8" />

    <util:load-scripts />

    <spring:message code="application_name" var="app_name" htmlEscape="false"/>
    <title><spring:message code="welcome_h3" arguments="${app_name}" /></title>
  </head>

  <body class="tundra spring">
    <div id="wrapper">
      <tiles:insertAttribute name="header" ignore="true" />
      <tiles:insertAttribute name="menu" ignore="true" />
```

```
      <div id="main">
        <tiles:insertAttribute name="body"/>
        <tiles:insertAttribute name="footer" ignore="true"/>
      </div>
    </div>
  </body>
</html>
```

En el fragmento de código anterior aparecen etiquetas de HTML y algunas otras etiquetas resaltadas en negrita. **<util:load-scripts />** es una etiqueta definida dentro del proyecto, luego se verá su finalidad. Las etiquetas **<spring:message .../>** son de Spring Framework y sirven para mostrar texto traducido a los distintos idiomas soportados por la aplicación. Por último las etiquetas **<tiles:insertAttribute ... ignore="true" />** son propias de Tiles y permiten insertar en esa posición de la página JSP el contenido del fichero que se ha especificado en la definición de las vistas como parámetro (*head, menu, body, footer*). Cabe destacar el atributo *ignore* que acompaña a alguna de estas etiquetas. El atributo *ignore* con valor *true* indica que, si el bloque que debe insertarse en el lugar de esa etiqueta no está establecido en la definición de la vista, lo ignore y siga con el dibujado del resto de la página, por contra, en ausencia de este atributo o con valor *false*, al dibujar la página, si el bloque a insertar no está definido, se produciría una excepción en tiempo de ejecución y por tanto no se dibujaría la página.

Con esto ya se conoce el funcionamiento del motor de plantillas usado en el framework, a continuación se verá el resto de directorios y ficheros que componen la capa web de la aplicación.

11.2.3.3. Vistas CRUD (Create, Read, Update, Delete)

Observando las carpetas contenidas en la carpeta views, se puede observar que hay una carpeta por cada una de las entidades de la lógica de negocio. Cada una de estas carpetas contiene las páginas JSP que dibujarán las vistas para las operaciones de Creación (Create), Lectura (Read), Modificación (Update) y Borrado (Delete). Tomando como ejemplo la entidad *bases*:

```
bases
|-- create.jspx
|-- list.jspx
|-- show.jspx
|-- update.jspx
`-- views.xml
```

Cabe destacar que también existe un archivo *views.xml*. También se trata de un archivo de definiciones de vistas. Mirando su contendo se observa que define varias vistas que se basan en *default* tal y como se ha comentado anteriormente pero cada una de ellas define un *body* con una JSP distinta.

Parece bastante evidente cual es el cometido de cada una de las JSP:

- *create.jspx* muestra el formulario de creación de un nuevo registro de la entidad Base,
- *list.jspx* muestra el listado de todas las bases de pizza existentes,
- *show.jspx* mostrará la información detallada de la base de pizza seleccionada
- y por último *update.jspx* muestra el formulario de actualización del registro.

No hay una JSP dedicada al borrado de registros porque ya se incluye un botón para el borrado de un registro tanto en *show* como en *list*.

11.2.3.4. Componentes de las JSP de las vistas CRUD

Se van a analizar los componentes que forman las vistas CRUD del proyecto.

Nota

Se listan algunos componentes que pueden aparecer en cada vista aunque cada uno de estos componentes puede no aparecer, aparecer varias veces o aparecer otros distintos dependiendo de los campos a informar.

create.jspx

```
<form:create>
  <field:input/>
  <field:datetime/>
  <field:select/>
</form:create>
<form:dependency/>
```

<form:create/> define un formulario en el que los campos vienen definidos por las etiquetas *<field:xxxx/>*. Las etiquetas *form* generan el código HTML correspondiente a la infraestructura de un formulario de creación siguiendo la definición RESTful. Las etiquetas *field* generan el código HTML necesario para mostrar campos de texto, introducción de fechas o desplegables de selección. En este caso cada campo puede utilizar una etiqueta distinta en función del tipo de datos que contenga y la forma en la que se desee presentar su información en pantalla.

list.jspx

```
<page:list>
  <table:table>
    <table:column/>
  </table:table>
</page:list>
```

<table:table/> y *<page:list/>* generarán el HTML de una página y una tabla donde las distintas columnas serán los campos indicados a través de la etiqueta *<table:column/>* y en cada fila se emplazará cada uno de los registros de la entidad paginados. Puesto que esta página solo proporciona la visualización de los valores de cada campo sin edición, no es necesario especificar distintos tipos de columna para cada campo.

show.jspx

```
<page:show>
  <field:display/>
</page:show>
```

Muestra los campos de un registro concreto de la entidad en formato texto en la que cada campo se representa mediante la etiqueta *<field:display/>*. Esta página también solo proporciona visualización de valores, por lo que tampoco es necesario especificar distintos tipos de columna para cada campo.

update.jspx

```
<form:update>
  <field:input/>
  <field:datetime/>
  <field:select/>
</form:update>
```

Se comporta de forma muy parecida a la página *create.jspx* pero esta página permite la actualización de los datos de un registro. Las etiquetas *field* que aparecen sirven en este caso para la modificación de un registro y son exáctamente las mismas que en el caso del alta de un nuevo registro.

11.3. Visualización de entidades en la capa web

Con los componentes <field:display/>, <table:column/> y <field:.../> se consigue mostrar datos de las entidades de la aplicación, es decir, en el proyecto PizzaShop de ejemplo, tomando la definición de la página jspx de listado de como la entidad PizzaOrder:

```
<field:display field="pizzas"
  id="s_com_disid_roo_pizzashop_domain_PizzaOrder_pizzas" object="${pizzaorder}"
  z="WSxjsiiEWhA5vo8ecxRoox5EjKA="/>
```

En la página se mostrará información de las Pizzas ligadas a un PizzaOrder. El framework utilizará para la visualización en la capa web de estos objetos relacionados los llamados **Converters**. Los Converters son clases java que, como su nombre indica, convierten un objeto dado en otro. En este caso particular de los Converters, el framework los utiliza para convertir un objeto del modelo (cada Pizza) en una cadena de forma que sea legible para un humano.

Este mecanismo está implementado en la clase java *ApplicationConversionServiceFactoryBean.java* que se puede encontrar en el mismo paquete donde están los controladores de la aplicación. Esta clase la genera el framework de manera automática al generar la capa web. Tal y como se ha comentado en el apartado de generación de código, asociado al *ApplicationConversionServiceFactoryBean.java* existe un Aspecto Java (fichero aj) con la declaración de los Converters y un método que los registra para que estén disponibles a la hora de "convertir" una entidad del modelo a la cadena que se mostrara en página.

Si la cadena que se visualiza en la página web no muestra la información deseada para una entidad, se puede modificar el Converter de esa entidad y adaptarlo a sus requerimientos. En la sección sobre modificación de código se explica como hacerlo.

11.4. Mejoras de renimiento

Se pueden implementar algunas mejoras de rendimiento definidas en la sección Evitar la carga de listas de valores innecesarias.

Capítulo 12. Buscadores en la capa web

En la sección de recetas se puede encontrar información adicional para desarrollar buscadores con gran cantidad de campos y también para hacer los campos opcionales en los buscadores .

12.1. Descripción

Si está definida la capa web con Spring MVC, se podrá generar el código necesario en el controlador y la página jspx con el formulario que invocará a uno de los buscadores que se encuentren definidos en la capa de entidades.

12.2. Creación de buscadores

Al ejecutar el comando *web mvc finder add* indicando la entidad que contiene algún buscador definido a nivel de entidad y el controlador encargado de gestionar dicha entidad, se generará el buscador en la capa de vista.

```
roo-gvNIX> web mvc finder add --formBackingType ~.domain.Owner
    --class ~.web.OwnerController
Upd SRC_MAIN_JAVA/com/springsource/petclinic/web/OwnerController.java
Upd SRC_MAIN_WEBAPP/WEB-INF/views/owners/views.xml
Upd SRC_MAIN_WEBAPP/WEB-INF/i18n/application.properties
Cre SRC_MAIN_JAVA/com/springsource/petclinic/web/OwnerController_Roo_Controller_Finder.aj
Cre SRC_MAIN_WEBAPP/WEB-INF/views/owners/findOwnersByLastNameLikeAndCityLike.jspx
Upd SRC_MAIN_WEBAPP/WEB-INF/views/menu.jspx
```

Consultar la documentación del comando web mvc finder add para más información.

12.3. Código generado

La ejecución del comando *web mvc finder add* desencadenará la inclusión de una anotación en la clase Java del controlador especificado.

```
...
  @RooWebFinder
  public class OwnerController {
  }
```

La aparición de la anotación *RooWebFinder* en el controlador provocará la generación del aspecto Java correspondiente que implementará la parte web de los buscadores definidos en su entidad relacionada.

```
privileged aspect OwnerController_Roo_Controller_Finder {

      @RequestMapping(params = { "find=ByLastNameLikeAndCityLike", "form" },
        method = RequestMethod.GET)
      public String OwnerController.findOwnersByLastNameLikeAndCityLikeForm(
        Model uiModel) {
          return "owners/findOwnersByLastNameLikeAndCityLike";
      }

      @RequestMapping(params = "find=ByLastNameLikeAndCityLike",
        method = RequestMethod.GET)
      public String OwnerController.findOwnersByLastNameLikeAndCityLike(
        @RequestParam("lastName") String lastName,
        @RequestParam("city") String city, Model uiModel) {
          uiModel.addAttribute("owners",
            Owner.findOwnersByLastNameLikeAndCityLike(lastName, city).getResultList());
```

```
            return "owners/list";
        }
    }
```

Se generará también la página jspx correspondiente y los recursos adicionales necesarios para su visualización.

Capítulo 13. Arranque y pruebas de la aplicación web

13.1. Pruebas funcionales

Una vez creada la capa web es posible generar automáticamente un conjunto de pruebas funcionales que permitirán realizar pruebas de aceptación de la aplicación.

La creación de las pruebas funcionales se realizan a través del comando **selenium test**

En el siguiente ejemplo se está generando las pruebas funcionales para el controlador de la entidad *Pizza*:

```
selenium test --controller ~.web.PizzaController
```

Los test generados estarán en *src/main/webapp/selenium*. La opción *--controller* especifica la clase controladora para la que se desea generar el test. Existen dos parámetros opcionales, uno es *--serverUrl* que permite indicar la dirección base de la aplicación sobre la que deben ejecutarse los tests y el otro es *--name* que permite dar un nombre específico al test. Habitualmente, no será necesario especificar estos parámetros opcionales.

13.2. Arrancar la aplicación web

Desde que se creó la capa web, ya era posible poner en marcha la aplicación y visualizarla en un navegador web. La puesta en marcha de la aplicación se puede realizar de dos formas distintas.

Si la consola del framework se está ejecutando desde un Eclipse o un STS, consultar la sección de la documentación Arrancar la aplicación con Eclipse en la que se explica tanto la instalación de un servidor en el IDE como el modo de arrancar la aplicación.

Si la consola del framework se está ejecutando desde la consola del sistema, salir de la consola del framework con el comando **quit** y ejecutar la orden **mvn tomcat:run**. Observar que en la consola del sistema debe estar configurado y por lo tanto disponible la herramienta Maven.

Para poder ver en funcionamiento la aplicación se debe acceder a la dirección: `http://localhost:8080/pizzashop` con un navegador web.

13.3. Ejecutar los tests funcionales

Para lanzar las pruebas, la aplicación debe estar en marcha y entonces pueden ejecutarse utilizando el plugin M2Eclipse de STS o Eclipse. Para ello, seleccionar la opción **Run As > Maven build ...** del menú contextual del proyecto o del menú superior e introducir como **Goals** el valor **selenium:selenese**. Darle un nombre a esta configuración y a partir de entonces utilizar esta configuración guardada para ejecutar los test funcionales mediante el botón **Run As...**.

También es posible lanzar las pruebas funcionales con la orden **mvn selenium:selenese** desde la consola del sistema.

De este modo se ejecutará el conjunto de tests sobre una nueva instancia del navegador FireFox (obviamente, es necesario tener instalado dicho navegador en la máquina).

Los resultados de los tests se guardarán en *target/selenium.html* en formato HTML.

Capítulo 14. Mejoras de rendimiento

???

14.1. Descripción

En las aplicaciones generadas en algún caso pueden requerir la implementación de alguna pequeña mejora para mejorar su rendimiento.

Las mejoras de rendimiento no serán siempre necesarias ya que dependen de las características de la aplicación en desarrollo.

14.2. Patrones de conversión óptimos en el log

Se puede mejorar el rendimiento del sistema de log (log4j) evitando que se muestre información innecesaria. Para ello modificar el fichero *src/main/resources/log4j.properties* modificando el *ConversionPattern* del appender que esté utilizando la aplicación, por ejemplo *log4j.appender.stdout.layout.ConversionPattern=*.

El *ConversionPattern* adecuado depende del entorno en el que nos encontremos. Será interesante disponer de mayor información en el log en entornos de desarrollo aunque se reduzca algo el rendimiento y ofrecer una menor información en los logs en entornos productivos. A continuación se recomiendan unos valores adecuados para cada tipo de entorno:

```
Entornos de desarrollo e integración: %d{dd/MM/yyyy HH:mm:ss.SSS} [%t] %p %c{1}.%M(%L) | %m%n
Entornos de preproducción y producción: %d{dd/MM/yyyy HH:mm:ss.SSS} [%t] %p %c{1} | %m%n
```

Los distintos valores en función del entorno se pueden establecer de forma automática mediante el add-on de configuración dinámica.

14.3. Evitar la carga de listas de valores innecesarias

Las vistas (páginas jspx) y los controladores (clases Java) son generadas con toda la información necesaria para que funcionen todas las acciones de listado, creación, actualización, borrado, búsqueda, etc. sobre una entidad.

En algunos casos, estas pantallas son personalizadas para ocultar campos que no son necesarios. Esto es especialmente importante en el caso de las relaciones con otras entidades y que se representan en página como campos desplegables.

```
... field="propertyname" render="false" ...
```

En dicho caso es recomendable evitar que se carguen las listas de datos que no se van a utilizar ya que realizan consultas a la base de datos innecesarias. Para ello, se puede personalizar los métodos del controlador llamados *populate* y comentar aquellas líneas relacionadas con el campo que se ha ocultado.

```
```

```
void populateEditForm(Model uiModel, Entity entity) {
...
      uiModel.addAttribute("propertyname", OtherEntity.findAllOtherEntities());
      ...
}
```

Parte III. Desarrollo avanzado con gvNIX

En esta se va a ver cómo ampliar una aplicación web básica creada con el framework para cumplir con los requerimientos de cualquier proyecto real.

Se revisará las características distintivas que ofrece gvNIX y que van enfocadas al desarrollo de funcionalidades de alto nivel para facilitar el desarrollo y proporcionar una mejor experiencia de uso al usuario:

- Gestión de distintos temas visuales.
- Organización de la estructura de páginas en el menú.
- Gestión de patrones de visualización de entidades y sus relaciones.
- Control de concurrencia a nivel de aplicación sin campos en base de datos.
- Gestión de la visualización de excepciones y gestión de mensajes de usuario en ventana modal.
- Control de acceso (autenticación y autorización) y seguridad.
- Generación de documentos (reportes).
- Gestión de servicios locales y servicios web (importación y exposición).
- Soporte de idiomas oficiales para el estado Español (inicialmente incluido el Valenciano).
- Gestión de configuraciones por entorno.
- Gestión de transformación de cadenas vacías a valores nulos.

Cada uno de estos elementos son añadidos (add-ons) que gvNIX incluye de forma adicional a todas las funcionalidades que ya proporciona su núcleo Spring Roo. Cada uno de estos addons pueden ser también utilizados en cualquier distribución de Spring Roo estándar.

Estos add-ons han sido inicialmente diseñados para ser aplicados en un modelo de dominio del tipo registro activo y una capa visual del tipo spring MVC. Sin embargo, pueden funcionar también con otros modelos de dominio y visuales y en futuras versiones se irá incrementando dicha compatibilidad. .

Capítulo 15. Add-on Web Menu

Sistema de gestión del menú web

15.1. Descripción

Este Add-on instala un nuevo sistema de gestión del menú web de la aplicación sobre la que se está trabajando para facilitar la organización de su estructura de páginas y grupos.

Las principales características de este sistema son:

Modelo de datos del menú separado de su formato de visualización.
: El modelo de datos del menú no está ubicado en la misma página que lo visualiza como ocurre en el modelo de menú que se proporciona por defecto. El modelo del menú estará definido en un archivo XML independiente.

Múltiples niveles en el menú
: Soporte para crear elementos jerarquizados con múltiples niveles. Permite añadir, mover y anidar en varios niveles los elementos del menú.

Gestión de elementos desde consola
: Permite añadir, quitar, mover, actualizar, definir permisos, ocultar, definir enlaces, etc. para cada uno de los elementos del menú desde la consola del framework.

Integración con Spring Security
: Si está instalado el sistema de seguridad, se puede establecer que ciertos elementos del menú no sean visibles si el usuario no está entre los roles permitidos para ello.

Gestiona las peticiones de Roo
: Aunque el sistema de gestión del menú es distinto al que se proporciona por defecto, este nuevo sistema también gestiona las peticiones de generación de elementos del menú realizadas por el resto de comandos. Es decir, si se encuentra instalada la nueva gestión del menú y cualquier otro comando solicita crear o actualizar un elemento del menú, la solicitud se realizará correctamente utilizando el nuevo sistema de gestión.

15.2. Definiciones

Antes de continuar explicando las funcionalidades del add-on es interesante tener clara las siguientes definiciones de los elementos que conforman un menú.

Página
: Representación de una URL. El destino puede ser interno o externo. Los destinos internos apuntan dentro de la propia aplicación mediante una ruta relativa, por ejemplo */entity1s/entity1?id=1*. Los enlaces externos apuntan a cualquier dirección web mediante una ruta absoluta, por ejemplo *https://www.cit.gva.es*.

Grupo
: Contenedor de páginas. El contenido de un grupo puede ser también otros grupos. También puede tener asociado una URL pero con las mismas restricciones que la página.

Menú
: Estructura en árbol de páginas y grupos.

Ítem del menú
: Un elemento del menú que puede ser una página o un grupo.

15.3. Instalación de la gestión del menú

El uso del sistema de gestión del menú aportado por esta herramienta es opcional. Por ello, para instalarlo y activarlo es necesario ejecutar en primer lugar el comando **menu setup**.

```
roo-gvNIX> menu setup
Component org.springframework.roo.addon.web.mvc.jsp.menu.MenuOperationsImpl disabled

Created SRC_MAIN_WEBAPP/WEB-INF/views/menu.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/views/menu.xml
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu/MenuItem.java
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu/Menu.java
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu/MenuLoader.java
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu/ContextMenuStrategy.java
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu/BaseURLContextMenuStrategy.java
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu/URLBrothersContextMenuStrategy.java
Created SRC_MAIN_JAVA/org/gvnix/test/web/menu/URLChildrenContextMenuStrategy.java
Created SRC_MAIN_WEBAPP/WEB-INF/tags/menu/gvnixmenu.tagx
Created SRC_MAIN_WEBAPP/WEB-INF/tags/menu/gvnixitem.tagx
Updated SRC_MAIN_WEBAPP/WEB-INF/views/menu.jspx
Updated ROOT/pom.xml [added property 'gvnix.version' = 'X.Y.Z']
```

Este comando añadirá los siguientes elementos:

1. Clases que representan el modelo de datos del menú. Cada una de estas clases estarán anotadas con `@GvNIXMenu*` para poder ser identificadas. Estas clases se crearán en el subpaquete `web.menu`.

2. Una Clase para la carga del menú y hacerlo disponible para su pintado en la vista. Esta clase se creará en el subpaquete `web.menu`.

3. Archivo XML para almacenar el modelo de datos del menú, rellenado previamente con la información del menú de Roo que existiese en ese momento. Por defecto se creará en `WEB-INF/views/menu.xml`.

4. *Tagx* necesarios para el pintado del menú, almacenados en `WEB-INF/tags/menu`.

5. Una nueva página jspx de la gestión del menú en la que se utilizan los *tagx* de pintado del menú, creada en `WEB-INF/views/menu.jspx`.

6. Clases para el uso de los menús contextuales. Estas clases se crearán en el subpaquete `web.menu`.

Para más información sobre el comando ver **menu setup** en el apéndice de comandos de este add-on.

El nombre del fichero XML que contiene el modelo de datos del menú se toma de la constante `MENU_CONFIG_FILE` de la clase anotada como `@GvNIXMenuLoader`.

15.4. Modificación del menú.

Para hacer cambios en el menú se deben utilizar los comandos disponibles en la consola, que serán accesibles después de haber instalado la nueva gestión del menú.

Para mas información sobre los comandos ver el apéndice de comandos de este add-on.

15.5. Futuras versiones

Mejoras a incluir en futuras versiones del add-on.

- El modelo de datos del menú actualmente se define en un archivo XML y en un futuro podrá estar almacenada en la base de datos.

Capítulo 16. Add-on JPA

Add-on de utilidades enfocadas a la persistencia y consulta de las entidades.

Este add-on incluye las siguientes funcionalidades:

- Servicios de persistencia de entidades en bloque.
- Registro de información adicional para búsquedas por relaciones.
- Auditoría y registro de histórico de cambios de las endidades.
- Persistencia de entidades con campos de tipo geográfico.

Para poder utilizar estas funcionalidades hay que ejecutar previamente el comando **jpa gvnix setup**.

16.1. Servicios persistencia en bloque

Esta funcionalidad permite generar *servicios* que permiten realizar operaciones de creación, actualización y eliminación de múltiples instancias de una entidad en una sola petición y de forma transaccional.

Estos *servicios* se generan como un *bean* de Spring, de forma que, para utilizarlos, solo es necesario declararlos como propiedad en la clase que los requiera y serán inyectados en el momento de la construcción de la instancia por Spring.

Un ejemplo de este servicio generado sería este:

```
package com.springsource.petclinic.domain;
import org.gvnix.addon.jpa.batch.GvNIXJpaBatch;
import org.springframework.stereotype.Service;

@Service
@GvNIXJpaBatch(entity = Owner.class)
public class OwnerBatchService {
}
```

En el correspondiente fichero `OwnerBatchService_Roo_GvNIXJpaBatch.aj` se generarán los siguientes métodos:

deleteAll()
: Elimina todos los registro de la entidad.

deleteIn(List<Long> ids)
: Elimina los registro de la entidad cuyos indentificadores se encuentren en la lista *ids*.

deleteNotIn(List<Long> ids)
: Elimina los registro de la entidad cuyos indentificadores *no* se encuentren en la lista *ids*.

create(List<Owner> owners)
: Persiste como nuevos todos los elementos de la lista *owners*.

update(List<Owner> owners)
: Persiste todos los elementos de la lista *owners*.

Nota

Importante: Si algún elemento de la lista no tiene establecido su campo de clave primaria o su campo de control de concurrencia (este último sólo si está definido) el registro *se guardará como registro nuevo*.

findByValues(Map<String,Object> propertyValues)
: Devuelve una lista de elemento cuyas propiedades coincidan (se usa el operador *igual a*) con *todos* valores recibidos en *propertyValues*.

deleteByValues(Map<String,Object> propertyValues)
: Elimina todos elementos cuyas propiedades coincidan (se usa el operador *igual a*) con *todos* valores recibidos en *propertyValues*.

Para poder usar este servicio en cualquier clase solo es necesario declarar una propiedad con la anotación @Autowire. Spring se encarga de inyectar una instancia del servicio en la propiedad. Este sería un ejemplo de la declaración para usarlo:

```
class MyClass {

    @Autowire
    private OwnerBatchService ownerBatch;
```

Todas las operaciones generadas (excepto *findByValues*) son *transaccionales*. Por lo que, si se produce un error en cualquier elemento afectado, *la operación entera* será abortada (no se guardarán los cambios)

Esta funcionalidad es requerida por otros add-ons como Web Mvc datatables para poder realizar operaciones sobre múltiples registros a la vez.

Para generar estos servicios se pueden utilizar los siguientes comandos:

jpa batch add
: Genera el servicio para una entidad.

jpa batch all
: Genera el servicio para todas las entidades de la aplicación.

16.2. Información adicional para búsquedas por relaciones

Esta funcionalidad permite definir información adicional sobre las propiedades de la entidad que declaran una relación que permiten realizar búsquedas y ordenaciones sobre ella.

Esta información se utiliza actualmente para permitir realizar búsquedas de texto dentro de entidades relacionadas y su ordenación cuando se usa el add-on web mvc datatables.

No tiene implementado ningún comando. Funciona añadiendo la anotación *GvNIXJpaQuery* a la propiedad. Por ejemplo, para permitir que, desde el listado de *Pet*, sea posible buscar por el nombre y apellido de su *Owner* y cuando se ordene por su columna se utilicen los mismos campos, deberíamos añadir la siguiente anotación:

```
....
```

```
public class Pet {
    ....
    ....

    @GvNIXJpaQuery(filterBy={"lastName","firstName"}, orderBy={"lastName","firstName"})
    @ManyToOne
    private Owner owner;
```

16.3. Auditoría y resgistro de cambios de entidades

Esta funcionalidad añade soporte para hacer auditoría (registrar quíen y cuándo se crea o modifica un resigtro) de entidades y, opcionalmente, registrar cada uno de los cambios de las modificaciones sufridas por los registros de la entidad.

16.3.1. Configurar detalles de usuario

Es posible configurar la clase que proveerá el nombre de usuario que realiza un cambio. Para ello se ejecuta el siguiente comando:

jpa audit setup
: Configura la auditoría de historico creando la clase que proveerá del nombre de usuario que realiza el cambio. Sólo se puede ejecutar una vez.

16.3.2. Auditoría básica de entidades

Para añadir la auditoría a las entidades se pueden utilizar los siguientes comandos:

jpa audit add
: Añade auditoría para una entidad.

jpa audit all
: Añade auditoría para todas las entidades de la aplicación.

Al instalar la auditoría en un proyecto gvNIX se creará una clase (con el nombre facilitado en el parámetro *--service*) anotada con *@GvNIXJpaAuditUserService*, se incluirá el siguiente método (en su correspondiente fichero *.aj*) para obtener los datos del Usuario:

getUser()
: Devolverá el tipo facilitado en el parámetro *--userType*. En caso de no definir ninguno, devolverá un tipo *String*

Al activar al auditoría sobre una entidad, que serán marcadas con la anotación *GvNIXJpaAudit*, se le incluirán las siguientes propiedades (en su correspondiente fichero *.aj*) para almacenar los datos de auditoría:

auditCreation
: Fecha de creación del elemento.

auditCreatedBy
: Usuario que creó el elemento.

auditLastUpdate
: Fecha de la última modificación del elemento.

auditLastUpdatedBy
: Último usuario que modificó el elemento.

Hay que tener en cuenta que este add-on no provee lógica de pintado, pero estas propiedades serán añadidas de forma automática a las correspondientes vistas si se han generado, o se generan, utilizando las funcionalidades de generación automática.

Nota

En las vistas generadas de forma automática para la creación y actualización de elementos puede ser interesante realizar cambios de forma manual para que estos campos no sean rellenados o modificado por el usuario.

Nota

Para evitar la pérdida de los datos de creación, en las vistas generadas de forma automática para la actualización *incluir los datos de auditoría como campos ocultos en el formulario*. Si no se incluyen estos campos en las peticiones puede perderse sus valores en el proceso de *binding* de los objetos recibidos

Nota

Es muy importante *no utilizar* actualizaciones/eliminaciones del estilo *em.createQuery("UPDATE Country SET population = 0, area = 0");* ya que los cambios aplicados no serán registrados por la auditoría.

Para mantener esta información acutalizada se genera un clase, anotada con *GvNIXJpaAuditListener*, que será registrada como *EntityListener* de la librería *JPA*. Una instancia de esta clase será llamada cada vez que un elemento de la entidad sea creado/modificado.

Esta clase tendrá implementados los siguientes métodos:

onCreate
: Método llamado antes de la creación de un registro. Rellena todos los campos de auditoría de la entidad (creación y actualización).

onUpdate
: Método antes de la actualización de un registro. Rellena los campos de auditoría correspondientes a la última actualización.

Nota

Este método *sólo será llamado* después de un ***merge*** *si el registro a sufrido modificaciones en sus datos*.

Las clases *EntityListener* requeridas por esta funcionalidad se registran de forma automática en el fichero `src/main/resources/orm.xml` del proyecto.

16.3.3. Auditoría y registro de cambios de entidades

Esta funcionalidad almacena todos los cambios sufridos por las entidades auditadas de forma que sea posible identificar qué, quién y cuándo se produjeron. Esto incluye las eliminaciones de los registros. Esta funcionalidad sólo se aplica a aquellas entidades marcadas con la anotación *GvNIXJpaAudit* (ver Auditoría básica de entidades)

Nota

En caso de *actualizar* o *eliminar* registros mediante el uso de *executeQuery* no se almacenarán los cambios sufridos por las entidades auditadas. Esto se debe a que no se dispararán los *listeners* necesarios para llevar a cabo este proceso.

Ya que esta funcionalidad puede implementarse de distinta forma, incluso dependiendo de la implementación de JPA que se esté utilizando en el proyecto, para empezar a utilizarla es necesario seleccionar un *proveedor*. Esto proveedores deben de estar instalados como add-on en el framework.

Para seleccionar el proveedor de registro de cambios se debe utilizar el siguiente comando:

jpa audit revisionLog
: Selecciona el proveedor de registro de cambios a usar.

Al activar el proveedor, se instalarán las librerías requeridas y se creará una clase que representará el registro índice de cambios en la aplicación. Esta clase será anotada con *GvNIXJpaAuditRevisionEntity* sus métodos y propiedades serán generados por el proveedor.

Para aquellas entidades anotadas con *GvNIXJpaAudit* y cuyo valor *storeRevisionLog* sea el adecuado (*YES* o *PROVIDER_DEFAULT/null* y la opción por defecto del proveedor sea activar el registro) se generarán en el *.aj* los siguientes métodos:

findAll*Entidad*
: Devuelve la lista de todos elementos de la entidad, al estado en el que estuviesen en una fecha en concreto o en un número revisión.

find*Entidad*
: Devuelve una entidad por código en el estado que estuviese en un fecha en concreto o en un número de revisión

get*Entidad*Revisions
: Devuelve una lista de *elementos de revisión* de la entidad entre fechas o números de revisión para un elemento en concreto, pudiendo especificar números de registros a devolver.

getRevisionNumberForDate
: Devuelve el identificador de revisión activo a una fecha.

find*Entidad*RevisionsByDates
: Devuelve una lista de *elementos de revisión* de la entidad entre fechas, pudiendo especificar filtros, ordenación y números de registros a devolver.

find*Entidad*Revisions
: Devuelve una lista de *elementos de revisión* de la entidad entre números de revisión, pudiendo especificar filtros, ordenación y números de registros a devolver.

Varios de los métodos arriba descritos devuelven *elementos de revisión*. Este elemento es una clase declarada para añadir información adicional a la entidad sobre los cambios producidos en una revisión del elemento de la entidad. Esta clase se generará en el fichero *.aj* de la entidad y tendrá los siguientes métodos:

getItem
: Devuelve el objeto en el estado (valores de sus datos) en una revisión.

Nota

Para el registro de cambios de eliminación, el estado devuelto por este método será el estado anterior a la eliminación (los valores antes que tenía el elemento cuando fue eliminado).

getRevisionNumber
: Devuelve el identificador de la revisión.

getRevisionUserName
: Devuelve el nombre del usuario que realizó los cambios registrados.

getRevisionDate
: Devuelve la fecha en el que se registraron los cambios.

isCreate
: Informa si tipo de cambio registrado en este elemento es una *creación*.

isUpdate
: Informa si tipo de cambio registrado en este elemento es una *actualización*.

isDelete
: Informa si tipo de cambio registrado en este elemento es una *eliminación*.

Nota

Para estos casos, el estado devuelto por el método *getItem()* será el estado anterior a la eliminación (los valores antes que tenía el elemento cuando fue eliminado).

getType
: Devuelve una cadena que representa el tipo de cambio del registro: *CREATE*, *UPDATE* o *DELETE*.

Además de lo métodos aquí descritos, cada proveedor puede incluir métodos necesario para dar soporte a su funcionalidad.

16.3.4. Proveedor de registro de cambios Hibernate Envers

Esta implementación de provee la funcionalidad de gestión de revisiones basada en el el módulo de la implementación de JPA Hibernate denominado Envers.

Para seleccionar este proveedor hay que ejecutar el comando: **jpa audit revisionLog --provider H-ENVERS**

Lógicamente, al ser un módulo de *Hibernate*, este proveedor de gestión de revisiones *sólo estará disponible en aquellos proyectos cuyo proveedor de persistencia sea Hibernate*.

Las características de este proveedor son:

- Mantiene el estado de relaciones (siempre que ambas entidades estén gestionadas).
- Los estados se mantienen en tablas adjuntas a las auditadas.
- Soporta búsquedas en el histórico utilizando su propio API. Esto tiene la limitación de únicamente poder filtrar sobre los datos de la entidad principal de la búsqueda (en la implementación actual, aunque en la documentación comentan que en un futuro habrá soporte para filtrar por las relaciones).

En los proyectos en los que *Spring Security* sea el proveedor de seguridad, el proveedor ya genera el código necesario, en la clase *RevisionEntity* para obtener el usuario que está realizando el cambio. Para el resto, será necesario realizar un *push-in* de la clase *RevisionLogEntityListener* y ajustar la implementación del método *newRevision*.

Para acceder a la API de lectura de *Envers* se genera un método estático en las entidad con el soporte establecido llamado *auditReader*. Para ver mas información sobre el uso de *AuditEntityReader* ver la documentación de el JavaDoc de la clase o la documentación del módulo *Envers*.

16.4. Persistencia de entidades con campos de tipo geográfico

Esta funcionalidad permite guardar entidades con campos de tipo geográfico.

16.4.1. Configuración del proyecto para soporte geográfico

Para poder guardar entidades con campos de tipo GEO es necesario configurar el proyecto generado. Para ello se ejecuta el siguiente comando:

jpa geo setup
: Configura el proyecto para poder guardar entidades con campos de tipo geográfico. Este comando solo funcionará si se ha instalado persistencia en el proyecto con proveedor *HIBERNATE* y se ha seleccionado una de las siguientes bases de datos:

- POSTGRES
- MYSQL
- ORACLE
- MSSQL

16.4.2. Añadir campos de tipo geográfico a entidades

Una vez configurado el proyecto para poder guardar entidades con campos de tipo geográfico, ya es posible añadir campos de tipo geográfico a las entidades. Para poder añadir estos nuevos tipos de campo, es necesario ejecutar este comando:

field geo
: Añade un nuevo campo de tipo GEO a la entidad seleccionada. Los nuevos campos añadidos pueden ser de los siguientes tipos:

- POINT (Se guarda un único punto en la base de datos)
- LINESTRING (Se guardan una serie de puntos que forman una linea continua)
- MULTILINESTRING (Se guardan una serie de puntos que forman varias lineas continuas)
- POLYGON (Se guardan una serie de puntos que forman un polígono. Siempre empieza y acaba en el mismo punto.)
- GEOMETRY (Se guarda una serie de puntos que forman una geometría. Acepta cualquiera de las geometrías anteriores.)

A cada campo GEO se le podrá espeficicar un sistema de referencia de cooordenadas (SRS) personalizado, el que utilizará para mostarse sobre el mapa. Registro de sistemas de referencia de coordenadas Si no se especifica ninguno, se utilizará el por defecto de los mapas (3857)

16.4.3. Implementación de buscadores para campos GEO

Para poder realizar búsquedas sobre campos de tipo GEO es necesario generar una serie de métodos. Para ello se ejecutan los siguientes comandos:

finder geo all
Genera los buscadores de todos los campos de tipo GEO de *todas* las entidades registradas en el proyecto.

finder geo add
Genera los buscadores para los campos de tipo GEO seleccionados de la entidad seleccionada.

Los buscadores generados incluyen un parámetro *GeometryFilter*, que podrá ser usado para establecer un filtro específico para una geometría (equals, intersects, touches, etc.). Para modificar estos métodos será necesarios hacer un *push-in* de los mismos.

Capítulo 17. Add-on Monitoring

Uso del componente Java Melody como sistema de monitorización para aplicaciones web.

17.1. Descripción

Este Add-on genera una nueva página desde donde podemos ver lo que consume la generación y ejecución de vistas, consultas, scripts y demás elementos que se generan y componen nuestra aplicación web.

17.2. Instalación de la monitorización

Para instalarlo, hay que utilizar el comando monitoring setup, el cual instalará todo lo necesario para el funcionamiento básico de la monitorización de nuestra aplicación. Se encargará de las dependencias y de modificar los ficheros de configuración necesarios para el funcionamiento de la monitorizacion de JSP, SQL, consultas, etc.

17.3. Monitorizando a través de Spring

Para monitorizar a través de las llamadas de Spring, tendremos que configurar que métodos deseamos monitorizar, para ello utilizaremos alguno de los siguientes comandos, siempre después de haber instalado el add-on: monitoring all, monitoring add class, monitoring add method o monitoring add package. Al ejecutar estos comandos se añadiran anotaciones que nos permitirán acceder a las llamadas que se ejecuten a los metodos monitorizados y ver sus interacciones.

17.4. Accediendo a la monitorización

Una vez todo configurado y con nuestra aplicación en marcha, bastará pulsar en la entrada de menú que se ha generado para su acceso, en caso de haberla desactivado, bastará con acceder a la dirección principal de nuestra aplicación, seguida de "/monitoring".

Capítulo 18. Add-on Web MVC

Add-on de utilidades para la capa Web de tipo Spring MVC.

Este add-on incluye dos funcionalidades:

- Interfaz Web para operaciones de persistencia de entidades en bloque.
- Visualización de vistas usando jQuery como librería de JavaScript.

18.1. Interfaz para operaciones de persistencia en bloque

Esta funcionalidad añade el interfaz web a un controlador para poder recibir peticiones de creación, actualización y eliminación de registros de una entidad en bloque.

Actualmente se utiliza para dar soporte a las operaciones de modificación múltiple que el add-on web mvc datatables.

Los método añadidos por esta funcionalidad se apoyan en las operaciones de persistencia en bloque generadas por el **add-on jpa** proporciona. Por ello, sólo se podrá añadir sobre controladores cuya entidad relacionada tenga generado este servicio.

Para generar estos métodos se pueden utilizar los siguientes comandos:

web mvc batch setup
: Instala las dependencias y ajusta la configuración.

web mvc batch add
: Genera los métodos en un controlador.

web mvc batch all
: Genera los métodos para todos los controladores.

18.1.1. Métodos de creación y actualización

Los métodos de actualización y creación reciben una *Array* de elementos en formato JSON.

Como resultado generan un objeto *JsonResponse* (también en formato JSON) con el resultado de la operación.

El *JsonResponse* se compone de las siguientes propiedades:

status
: Resultado de la operación: *SUCCESS* o *ERROR*

exceptionMessage
: Mensaje de error (si lo hay) encontrado al ejecutar la operación.

bindingResult
: Errores encontrados en la carga y/o validación de los datos recibidos.

  El formato de este objeto en JSON es como el siguiente:

```
{
   OBJECT_INDEX : { FIELD1_NAME : FIELD_ERROR_MSG, FIELD2_NAME : FIELD_ERROR_MSG, ...},
   OBJECT_INDEX2 : { FIELD1_NAME : FIELD_ERROR_MSG,
     FIELD_OBJECT_NAME : { SUBOBJECT_FIELD: FIELD_ERROR_MSG, ... }
     FIELD_LIST_NAME: {
        OBJECT_FIELD_ITEM_INDEX : {ITEM_LIST_FIELD: FIELD_ERROR_MSG, ... },
        OBJECT_FIELD_ITEM_INDEX2 : {ITEM_LIST_FIELD: FIELD_ERROR_MSG, ... },
     },
     ...
   },
   ...
}
```

value
: Objetos persistidos o recibidos.

18.1.2. Método de eliminación

El método de eliminación recibe los siguientes parámetros:

{entidad}
: Valores de las propiedades usado como *filtro base* de la operación de borrado completo.

all
: Si recibe un *true* elimina todos los elementos que cumplan el *filtro base*. Si no se ha recibido *filtro base borrará todos los elementos.* Los parámetros relativos a borrado por identificadores serán ignorados.

idList[]
: Lista de identificadores de elementos a tener en cuenta para el borrado.

deleteIn
: Si recibe un *true* cuyo identificador se encuentra en *idList[]*. De lo contrario elimina los elementos cuyo identificado *no se encuentra en idList[]*.

Como resultado, el método devuelve: el número de elementos eliminados

Si se produce un error en la eliminación el método devuelve un estado *INTERNAL_SERVER_ERROR(500)* y el mensaje de error.

18.1.3. Carga de datos en formato JSON

Debido a que las operaciones de creación y actualización reciben los datos en formato JSON y Spring no tiene soporte todavía para cargar objetos en este formato usando el *conversionService* (ver tarea en el JIRA de Spring). Mientras tanto, esta funcionalidad incluye una librería propia que añade un *MappingHandlerAdapter* que es capaz de hacerlo. Además, este objeto realizar la validación y rellenar un *BindingResult* con los resultados de la carga.

El *Jackson2RequestMappingHandlerAdapter* se configura en el fichero `webmvc-config.xml` y se encarga de gestionar todas las peticiones cuyo `@RequestMapping` esté configurado con

```
consumes = "application/json"
```

.

18.2. Visualización con jQuery

Esta funcionalidad permite modificar la visualización de las vista para que utilice la librería *jQuery* en vez de la estándar *jQuery*.

Las opciones de este add-on *no son compatibles con el add-on Web Screen Patterns* .

Para poder utilizar el *list* generado por el add-on web mvc datatables es necesario que la vista esté convertida a jQuery (ya que el componente *dataTables* que se utiliza está creado en base a esta librería).

Los artefactos que se instalan son:

- Ficheros `.js` de la librería *jQuery* y *jQueryUI* además de otros plugins, como *jQuery.validate*, necesarios para reemplazar todas las funcionalidades disponibles de base en las vistas.
- Ficheros `.tagx` con la nueva implementación de pintado de los componentes.
- Ficheros de estilos e imágenes.

Para utilizar esta funcionalidad dispone de los siguientes comandos:

web mvc jquery setup
: Instala los artefactos necesarios.

web mvc jquery add
: Convierte las vista de un controlador a jQuery.

web mvc jquery all
: Convierte todas las vistas a jQuery.

web mvc jquery update tags
: Actualiza los artefactos requeridos por jQuery. Util para actualizar un proyecto a una nueva versión de gvNIX.

18.2.1. Conversión de las vistas a jQuery

En el proceso de conversión consta de la correspondiente anotación en el controlador `@GvNIXWebJQuery` y la actualización de las rutas a los `.tagx` en sus ficheros `.jspx`.

Los `.tagx` se han creado para que sean compatibles con los originales, por lo que, en principio, no será necesario modificar las páginas `.jspx` generadas de forma estándar por los comandos de ROO.

Si se han añadido nuevos `.tagx` personalizados al proyecto basados en *DOJO*, es posible que tengan que ser replicados usando la nueva librería.

Capítulo 19. Add-on Bootstrap

Implementación de Bootstrap 3 en el proyecto gvNIX.

19.1. Descripción

Este add-on permite configurar el proyecto gvNIX con la estructura HTML necesaria para implementar Bootstrap 3. Además, incluye todos los recursos necesarios para modificar la apariencia de la aplicación por una por defecto basada en Bootstrap 3

19.2. Instalación de Bootstrap 3

Para instalarlo, hay que utilizar el comando web mvc bootstrap setup, el cual solo estará disponible después de instalar el add-on JQuery en nuestro proyecto. Este comando detectará de forma automática, aquellos addons que ya hayan sido instalados y modificará sus componentes para una correcta visualización utilizando Bootstrap. Además, los addons que se instalen después de instalar Bootstrap tendrán apariencia Bootstrap por defecto.

19.3. Actualización de componentes

Para actualizar los componentes instalados de Bootstrap basta con ejecutar web mvc bootstrap update, el cual solo estará disponible después de haber instalado Bootstrap.

19.4. Apendice de comandos

Para ver con mas detalle el comando que proporciona el add-on consultar la sección de comandos del add-on Bootstrap.

Capítulo 20. Add-on Web MVC Datatables

Uso del componente dataTables en las vistas *list* de los controladores.

20.1. Descripción

Este Add-on reemplaza el widget de lista de las vistas de las entidades para usar el plugin de jQuery dataTables. Para ello, se usa una adaptación de la librería de tag JSP Dandelion-DataTables que se integra y adapta a la generación de vistas de Spring Roo.

Las opciones de este add-on *no son compatibles con el add-on Web Screen Patterns* .

Las principales características de este widget son:

Ordenación de datos por uno o más campos.
: La tabla permite al usuario final ordenar los datos, por una o más columnas, y en el sentido que el desee. Las columnas disponibles para la ordenación y su sentido son configurables en la página.

Filtrados por columnas y búsquedas globales
: Soporta filtrar los datos mostrados por columna, además de hacer búsquedas de texto sobre los resultados de los filtros. La configuración de que filtros están disponibles para el usuario se pueden configurar en la página. Además, estos filtros por columna disponen de un asistente para realizar búsquedas dependiendo del tipo de dato representado en cada una de ellas.

Paginación de datos ajustable
: Los datos, filtrados o sin filtrar, se paginan sin necesidad de refrescar la página. El tamaño de la página puede ser seleccionado por el usuario final.

Acceso a los datos en la misma página o por petición por AJAX
: Los datos que alimentan la tabla pueden estar integrados en la página o usar peticiones AJAX para ir solicitándolos a medida que el usuario los requiera.

Soporta múltiples plugins y es extensible
: Soporta múltiples plugins como *ColReorder* (permite al usuario reordenar las columnas), *FixedHeader* (que mantiene las cabeceras de las columnas visibles cuando se desciende por páginas largas), etc.

Registro de Callbacks en distintos eventos
: Soporta registrar funciones JavaScript a ejecutar por el widged cuando: termina la inicialización, al crear una línea, al pintar el píe de tabla, etc...

Visualización en modo Registro
: Permite realizar la visualización en modo Registro, es decir, mostrando un registro por página con sus valores completos.

Visualización de Detalles
: Se puede añadir la visualización de un o varios detalles (listado de entidades relacionadas), con varios niveles, al marcar o visualizar (dependiendo del modo de visualización) un elemento de la tabla.

Actualización en línea
: Modo que permite actualizar los elementos de la tabla desde la misma línea que los está visualizando. Es posible modificar multiples registro a la vez.

20.2. Instalación del soporte para Datatables

Esta funcionalidad depende de las librerías de *jQuery* por lo que, el primer paso, será instalar las librerías usando el comando **web mvc jquery setup** si no estuviese ya instalado.

Para instalar esta funcionalidad hay que ejecutar el comando **web mvc datatables setup**.

Este comando añadirá los siguientes elementos:

1. Imágenes y hojas de estilo utilizadas por el widget. Estos ficheros se instalan en `webapp/images/datatables` y `webapp/styles/datatables` respectivamente.

2. Los archivos javaScript de jQuery.datatables (tanto en su versión optimizada como estándar) y algunos plugins para dataTables, creados por el equipo de gvNIX, para utilizados en las funcionalidades disponibles (como selección, edición, etc...), en el directorio `webapp/scripts/datatables`. Para obtener información de las versiones de estos ficheros, consultar el fichero `README.txt` que se instala en el directorio.

3. `Tagx` necesarios utilizados en las páginas para el pintado del widged, almacenados en `WEB-INF/tags/datatables`.

4. Unos fichero de propiedades para diversos aspectos de configuración en `src/main/resources/datatables*.properties` y las cadenas de internacionalización.

5. Actualiza el fichero `WEB-INF/tags/util/load-scripts.tagx` para que las páginas puedan localizar los recursos de hojas de estilo y javaScript requeridos.

6. Algunos ajustes en el fichero `WEB-INF/web.xml` y `WEB-INF/spring/webmvc-config.xml` requeridos para el correcto funcionamiento de widget.

7. La dependencia a este add-on, librerías de utilidades utilizadas para la gestión de peticiones de datos y a la librería de tag adaptada. Esta última tiene como dependencias las dependencias a las librerías de Dandelion-DataTables.

Para más información sobre el comando ver **web mvc datatables setup** en el apéndice de comandos de este add-on.

20.3. Usar datatables en la vista "list" de un controlador.

Para poder utilizar el widged de datatables en el listado es necesario que esté usando los componentes *jQuery*. Para ello usar el comando **web mvc jquery add** o **web mvc jquery all** antes de utilizar este add-on con un controlador.

Para utilizar el widged de datatables en el listado de un controlador de entidad es necesario ejecutar el comando **web mvc datatables add**.

Este comando añade la anotación `@GvNIXDatatables` al controller para generar los métodos necesario para gestionar las peticiones que realiza el widget.

Además, actualiza la página `list.jspx` para cambiar las rutas a los tagx utilizados para pintar la tabla a los instalados por el add-on en el directorio `WEB-INF/tags/datatables`.

Este add-on incluye, además, el comando **web mvc datatables all** que aplica los cambios a todos los controllers de la aplicación.

Para mas información sobre los comandos ver el apéndice de comandos de este add-on.

20.4. Ajustar la configuración del datatables de una vista.

Los tags creados para este add-on, que se pueden encontrar en `WEB-INF/tags/datatables`, son compatibles con los tags estándar que se incluyen por defecto, incorporando además las opciones que ofrece la librería de Dandelion-DataTables.

Las opciones disponibles para los tags se pueden consultar en la declaración de los propios ficheros de tags, o en la página de referencia de Dandelion-DataTables. Algunos de los atributos no están disponibles debido a la integración con Spring Roo o los algunas opciones necesarias para la integración de los distintos modos de visualización preparados.

Para personalizar los ajustes de alguna vista, modificar el fichero `list.jspx` correspondiente tal y como se realiza con los tags estándar.

20.5. Cambiar el modo de datos de Datatables.

Para proveer de datos al widget, se dispone de dos métodos.

1. *DOM* o la carga de datos incrustados en la misma página. Una vez cargados los datos, el widged se encarga de paginarlos, ordenarlos y filtrarlos a través de javaScript en la propia página.

2. *AJAX* el widged hace peticiones AJAX al servidor para obtener los datos en base a la paginación, orden y filtro. Luego los muestra repintando la tabla a través de javaScript en la propia página.

La opción por defecto que instala el add-on es la de AJAX, ya que es más escalable para volúmenes de datos grandes.

El modo de datos del controlador se establece con la opción `--ajax` al ejecutar el comando **web mvc datatables add** o modificar el atributo *ajax* de la anotación `GvNIXDatatables`. gvNIX actualizará el controlador para que el modo sea accesible tanto en el mismo (para los métodos de peticiones) como en la página.

20.6. El control de búsqueda y filtros por columnas.

Por defecto, la tabla muestra un control de búsqueda que permite filtrar el contenido visualizado en la tabla. Esta búsqueda se realiza *en formato texto*. Es decir, *compara el texto de cada valor* sea cual sea su tipo.

Dependiendo del modo de acceso a datos y del tipo de dato el resultado de la búsqueda puede ser distinto: Para tablas en modo DOM, la búsqueda ser realiza sobre el texto que se visualiza en cada celda, mientras que, en modo AJAX se ejecuta una consulta contra la base de datos.

Para las búsquedas en las tablas en modo AJAX hay que tener en cuenta lo siguiente:

- Para los campos de tipo entidad hay que anotar la propiedad de la relación con la anotación `@GvNIXJpaQuery` del add-on jpa para que se pueda construir la consulta con dichos campos.

- Es posible buscar por los campos de tipo fecha, pero hay que tener en cuenta que, *la transformación a texto la realiza la base de datos con su formato predeterminado* no por el formato en el que se visualiza.

Los filtros por columna funcionan de la misma forma que la búsqueda, aplicando las misma reglas explicadas anteriormente, pero sólo con los datos de dicha columna. Ambas opciones se acumulan para mostrar el resultado en la tabla.

Para activar el filtrado en una columna hay que añadir el parámetro `filterable="true"` en la etiqueta `table:column` correspondiente a la columna en el fichero `list.jspx`. Al añadir este filtrado por columna se añadirán de forma automática un asistente de búsqueda al lado de cada filtro que permitirá realizar búsquedas avanzadas dependiendo del tipo de dato representado en cada columna.

20.7. Filtros Simples

Los filtros por columna llevan predefinidas unas operaciones dependiendo del tipo de campo de la columna. Los nombres de estas operaciones son multidioma y pueden ser configuradas por el desarrollador. Estas operaciones pueden ser introducidas manualmente o utilizando el asistente de búsqueda comentado anteriormente. Por defecto se pueden utilizar las siguientes operaciones:

Campos de Tipo Texto

- *cadena*: Buscará la cadena de texto introducida
- *=cadena*: Buscará la cadena de texto introducida después del simbolo igual
- *CONTIENE(cadena)*: Buscará todos los registros que para esa columna contengan la cadena de texto introducida
- *EMPIEZA(cadena)*: Buscará todos los registros que para esa columna empiecen por la cadena de texto introducida
- *TERMINA(cadena)*: Buscará todos los registros que para esa columna terminen por la cadena de texto introducida
- *ESVACIO*: Buscará todos los registros que para esa columna estén vacios o sean nulos
- *NOESVACIO*: Buscará todos los registros que para esa columna no estén vacios y no sean nulos
- *ESNULO*: Buscará todos los registros que para esa columna sean nulos
- *NONULO*: Buscará todos los registros que para esa columna no sean nulos

Campos de Tipo Numérico

- *número*: Buscará el valor numérico exacto en la columna actual
- *=número*: Buscará el valor numérico exacto en la columna actual
- *>número*: Buscará todos los registros cuyo valor sea mayor que el indicado
- *>=número*: Buscará todos los registros cuyo valor sea mayor o igual que el indicado
- *<número*: Buscará todos los registros cuyo valor sea menor que el indicado

- *<=número*: Buscará todos los registros cuyo valor sea menor o igual que el indicado
- *<>número*: Buscará todos los registros cuyo valor sea distinto que el indicado
- *!=número*: Buscará todos los registros cuyo valor sea distinto que el indicado
- *ENTRENUMERO(n1;n2)*: Buscará todos los registros cuyo valor se encuentre entre el primer número indicado y el segundo
- *ESNULO*: Buscará todos los registros que para esa columna sean nulos
- *NONULO*: Buscará todos los registros que para esa columna no sean nulos

Campos de Tipo Fecha

- *FECHA(fecha)*: Buscará todos los registros con la fecha introducida para la columna del filtro
- *ESANYO(año)*: Buscará todos los registros que contengan el año en la fecha de la columna del filtro
- *ESMES(mes)*: Buscará todos los registros que contengan el mes en la fecha de la columna del filtro
- *ESDIA(dia)*: Buscará todos los registros que contengan el día en la fecha de la columna del filtro
- *ENTREFECHA(fecha1;fecha2)*: Buscará todos los registros cuyas fechas se encuentren entre la primera fecha y la segunda
- *ESNULO*: Buscará todos los registros que para esa columna sean nulos
- *NONULO*: Buscará todos los registros que para esa columna no sean nulos

Campos de Tipo Boolean

- *VERDADERO*: Buscará todos los registros cuyo valor para la columna sea true
- *FALSO*: Buscará todos los registros cuyo valor para la columna sea false
- *ESNULO*: Buscará todos los registros que para esa columna sean nulos
- *NONULO*: Buscará todos los registros que para esa columna no sean nulos

20.8. Añadir Columnas Personalizadas

Por defecto, la tabla muestra una serie de controles relacionados con la creación, actualización, visualización y eliminación de elementos de la misma. En el caso de que se desee añadir alguna acción diferente a las mencionadas, se dispone del tagx *action-column*, el cual, utilizado en la página `list.jspx` y mediante una serie de parámetros, permite configurar tanto la visualización del contenido de la columna, como una llamada a una función javascript que se defina en dicha página y en la que se establezca la acción a realizar.

20.9. Modo visualización de registro.

Este modo de visualización muestra un registro por cada página para mostrar mas información sobre cada uno.

Para ello, se pinta la vista *show* del propio registro y se incrusta dentro de la celda de la tabla.

Este modo de visualización sólo permite el modo de acceso a datos *AJAX* y no están soportadas la herramientas de búsqueda, ordenación, filtros y edición.

Para establece este modo de visualización hay que usar el parámetro `--mode show` al ejecutar el comando **web mvc datatables add**.

Para vistas que ya transformadas, se pude establecer este modo modificando los valores de la anotación `@GvNIXDatatables` y añadirle el atributo `mode = "show"`. Comprobar que también esta establecido el valor `ajax = true`.

20.10. Visualización de detalles.

Este opción permite la visualización de datos relacionados con un registro al pie de la página. Los datos mostrados se obtendrán a parir de una propiedad de relación 1:N de la entidad actual. La propiedad debe tener configurada el valor `mappedBy` en la anotación `@OneToMany`

Para mostrar los datos del detalle se utilizará la vista *list* de la entidad hija, la cual *debe de utilizar también una vista dataTable*, usando exactamente la misma configuración de visualización y modo de datos (pero con sus datos filtrados para mostrar los datos relacionados con el padre). En la vista de detalle estarán disponibles todas las opciones disponibles originalmente, incluido si tiene activado la visualización de detalles.

Los registros mostrados en el detalle dependerá del registro actual. Este dependiendo del modo de visualización se selecciona:

modo tabular (estándar)
: Haciendo *doble click* sobre la linea deseada. La linea actual se marcará de un color distinto (por defecto verde) y en caso de disponer detalles asociados, se desplazará hasta la posición de los mismos para facilitar su localización.

modo registro
: El detalle muestra los datos para el registro actual (sólo se ve un registro por página)

Para añadir las visualización del detalle en una vista hay que usar comando **web mvc datatables details add**.

20.11. Eliminación múltiple.

Las listas soportan eliminación múltiple de líneas. Esta operación se activa de forma automática cuando el controlador tiene activado las operaciones de actualización en bloque.

Cuando esta opción está disponible, en la tabla aparecerá una columna de controles de marcado para permitir la selección de los elemento sobre los que se quiere actuar. En la cabecera de esta columna se mostrará un icono que permite la selección de todos los elementos o limpiar la selección.

La opción de *seleccionar todo* selecciona todos los registros del listado *incluidos los que no se están visualizando en la página actual*. La selección se mantiene con las operaciones de búsqueda, filtrado y paginación.

En la linea de estado se informa del total de registros seleccionados y cuantos de ellos están en la página actual.

Al pulsar sobre el botón elimina de la cabecera de la tabla, se pedirá confirmación, mostrando el total de registros a eliminar. El botón de eliminación de la línea sigue funcionando de la forma habitual.

20.12. Edición en línea.

Las listas soportan la edición sobre la misma de líneas de los datos de la entidad sin necesidad de cambiar de página o recargarla.

Esta opción *requiere que el controlador tenga activado las operaciones de actualización en bloque* y *no está soportado para el modo de visualización registro*

Entrar en modo edición hay que pulsar sobre el botón de editar de dicho registro o seleccionar los registros deseados en el control de marcado y pulsar el botón de edición de la cabecera de la tabla.

Al activar la edición de un registro, el control solicita al servidor el contenido de la vista *update* de la entidad y cambia el contenido de las columnas de dicha fila por los controles de entrada del formulario original. Por tanto los ajuste de los controles de actualización se realizan en un único lugar.

Después de entrar en modo edición aparecerán dos botones en la cabecera de la tabla para guardar los cambios o cancelar la edición. Estas acciones *afectan a todos los registros en edición, se estén visualizando o no*.

Durante la edición, las funciones de paginación, filtrado, búsqueda y ordenación estarán disponibles de la forma habitual, *manteniendo los cambios realizados* en los campos de las filas en edición.

Para que la actualización funcione correctamente *la tabla debe contener todas las columnas requeridas para la edición del elemento*. De no ser así la actualización de los registros fallará.

Al pulsar sobre el botón de guardar, el control recogerá los datos de las filas en edición y realizará una petición de actualización *en bloque*, de forma que *si hay algún problema con algún registro ningún cambio se persistirá*

Si se encuentra algún problema de validación en algún registro, el control mantendrá el estado de edición de las lineas, marcando las lineas afectadas por errores de un color rojizo. Además, mostrará el mensaje de error generado en el servidor debajo de cada campo afectado.

Para activar esta opción hay que usar el parámetro `--inline true` al ejecutar el comando **web mvc datatables add**.

Para vistas que ya transformadas, se pude activar la opción modificando los valores de la anotación `@GvNIXDatatables` y añadirle el atributo `inlineEditing = true`.

20.13. Registro creado en primera posición

Al crear un nuevo registro, se colocará automáticamente en la primera posición del listado sin tener en cuenta los filtros u ordenación del Datatable.

Gracias a esto tendremos siempre visible el registro que se ha editado.

Al recargar la página, se ordenará el registro de forma correcta siguiendo la ordenación y los filtros establecidos en el Datatable.

Esta funcionalidad se aplica tanto a Datatables maestros como para detalles asociados

20.14. Registro editado en primera posición

Al actualizar un registro, se colocará automáticamente en la primera posición sin tener en cuenta los filtros u ordenación del Datatable.

Gracias a esto tendremos siempre visible el registro que se ha editado.

Al recargar la página, se ordenará el registro de forma correcta siguiendo la ordenación y los filtros establecidos en el Datatable.

Esta funcionalidad se aplica tanto a Datatables maestros como para detalles asociados

20.15. Registro seleccionado siempre visible

Al acceder a una página del listado Datatable que contenga un registro seleccionado por el usuario, se navegará de forma automática hasta este registro.

Gracias a esto tendremos siempre posicionado en pantalla el registro que se ha seleccionado.

Capítulo 21. Add-on Web MVC GEO

Uso del componente Mapa para representar y posicionar entidades que dispongan de campos de tipo GEO.

21.1. Descripción

Este Add-on permite representar en vistas de tipo Mapa, aquellas entidades que dispongan de campos de tipo GEO.

Las principales características de este widget son:

Representar entidades en Mapa
: Permite representar aquellas entidades que dispongan de campos de tipo GEO en vistas de tipo Mapa. Se podrán mostrar/ocultar aquellos campos GEO que se deseen.

Filtrado de entidades
: Soporta filtrar los registros de una entidad representados en el mapa utilizando el componente Datatable Por defecto, esta característica se abre en una ventana diferente, el desarrollador puede configurarla para mostrarla como diálogo sobre la vista del mapa, o de forma distinta mediante una función personalizada.

Selección de entidades
: Permite seleccionar uno o varios datos representados en la vista de Mapa utilizando el componente Datatable

Representar Capas Base en Mapa
: Además de representar entidades en la vista de Mapa, también es posible representar capas obtenidas desde un servidor de mapas. Es posible representar capas de tipo "tile" y capas de tipo "WMS".

Personalizar orden de capas
: El orden en el que se muestran las capas en el mapa se puede personalizar mediante la característica "Arrastrar y Soltar", que permite reordenar los elementos de cada grupo de capas. De este modo, las capas se mostrarán en función de su ordenación en el TOC, siendo la primera la que se muestre por encima de las demás.

Barra de Herramientas
: Este componente dispone de una serie de herramientas que pueden ser añadidas a la vista de mapa. También se pueden añadir herramientas estándar que el desarrollador puede personalizar para que realicen las acciones que él desee.

Personalización de componentes (Markers, Colores, etc...)
: Por defecto, la apariencia de las entidades que se representan en el mapa se generan de forma aleatoria. El desarrollador puede personalizar el icono, el color del icono, el color del marker o linea, etc... Si la entidad tiene la opción de ser seleccionada desde el componente Datatable podrá configurar también, la apariencia con la que se representará esta entidad al ser seleccionada. De este mismo modo, se podrá configurar la apariencia de las herramientas genéricas.

21.2. Instalación del soporte para vista de Mapa

Esta funcionalidad depende de las librerías de *jQuery* y el componente *Fancytree* por lo que, el primer paso, será instalar estos recursos usando primero el comando **web mvc jquery setup** y a continuación **web mvc fancytree setup** si no estuviesen ya instalados.

Una vez instalados los requisitos necesarios, para instalar esta funcionalidad hay que ejecutar el comando **web mvc geo setup**.

Este comando añadirá los siguientes elementos:

1. Imágenes y hojas de estilo utilizadas por el widget. Estos ficheros se instalan en `webapp/images` y `webapp/styles/leaflet` respectivamente.
2. Los archivos javaScript de leaflet (tanto en su versión optimizada como estándar) y algunos plugins para leaflet, creados por el equipo de gvNIX, para utilizados en las funcionalidades disponibles (como selección, edición, etc...), en el directorio `webapp/scripts/leaflet`.
3. `Tagx` necesarios utilizados en las páginas para el pintado del widged, almacenados en `WEB-INF/tags/geo`.
4. Actualiza el fichero `WEB-INF/tags/util/load-scripts.tagx` para que las páginas puedan localizar los recursos de hojas de estilo y javaScript requeridos.
5. La dependencia a este add-on y librerías de utilidades utilizadas para la gestión de peticiones de datos.

Para más información sobre el comando ver **web mvc geo setup** en el apéndice de comandos de este add-on.

21.3. Generar vista de Mapa

Para poder visualizar una vista de Mapa en nuestro proyecto, es necesario ejecutar el comando **web mvc geo controller**.

Este comando genera un controlador anotado con `@GvNIXMapViewer` que será el encargado de mostrar la vista del mapa que acabamos de crear. Además se podrá configurar la proyección en la que trabaja el mapa generado.

Además, genera la página `show.jspx` que será la encargada de representar el mapa y todos sus componentes.

Para mas información sobre los comandos ver el apéndice de comandos de este add-on.

21.4. Generar campos de mapa en vistas CRU

Por defecto, a la hora de guardar campos de tipo geográfico se utilizará formato WKT que tendrá que ser introducido de forma manual por el usuario.

Sin embargo, este add-on permite transformar estos campos de texto a componentes de tipo Mapa, gracias a los cuales se facilita la introducción de datos de tipo GEO en una entidad. Para utilizar estos tipos de campo, se ejecutará el comando **web mvc geo field**.

Este comando modifica las vistas de `create.jspx`, `update.jspx` y `show.jspx` de la entidad sobre la que se aplica cambiando los inputs de los campos seleccionados por componentes de tipo mapa.

Para mas información sobre los comandos ver el apéndice de comandos de este add-on.

21.5. Generar agrupaciones de capas sobre un mapa

Al generar capas es posible que se desee que se muestren agrupadas debido a un determinado requerimiento que establezca el usuario.

Para añadir una nueva agrupación a la vista de mapa y que posteriormente se agreguen capas a dicha agrupación, es posible ejecutar el siguiente comando:

web mvc geo group
Añade una nueva agrupación de capas a la vista de mapa que se indique

Este comando modifica la vista `show.jspx` del mapa incluyendo el nuevo grupo a representar.

Para mas información sobre el comando ver el apéndice de comandos de este add-on.

21.6. Añadir entidades a la vista de Mapa

Al generar la vista de mapa se generará una vista vacía con una capa base por defecto.

Para añadir nuevas entidades a la vista de mapa y que sean representadas sus campos de tipo geográfico, es posible ejecutar los siguientes comandos:

web mvc geo entity all
Añade todas las entidades con campos de tipo GEO a la vista de mapa

web mvc geo entity add
Añade la entidad seleccionada a la vista de mapa.

web mvc geo entity simple
Añade el campo de tipo GEO seleccionado de la entidad indicada a la vista de mapa.

web mvc geo base layer field
Añade una capa base para mostrar en el mapa de un campo geográfico.

Estos comandos modifican los controladores de las entidades sobre los que se aplican añadiendo la anotación `@GvNIXWebEntityMapLayer`.

Al añadirse esta anotación, se añade en la vista `show.jspx` del mapa la nueva entidad a representar.

Para mas información sobre los comandos ver el apéndice de comandos de este add-on.

21.7. Añadir Capas Base la vista de Mapa

Al generar la vista de mapa se generará una vista vacía con una capa base por defecto.

Para añadir nuevas capas base, es posible ejecutar los siguientes comandos dependiendo del tipo de Capa Base que queramos crear:

web mvc geo tilelayer
Añade una capa base de tipo Tile a la vista del mapa

web mvc geo wmslayer
Añade una capa base de tipo WMS a la vista del mapa

web mvc geo wmtslayer
Añade una capa base de tipo WMTS a la vista del mapa

Estos comandos añaden en la vista `show.jspx` del mapa la nueva capa base

Para mas información sobre los comandos ver el apéndice de comandos de este add-on.

21.8. Generar nuevas herramientas en la vista del Mapa

Por defecto, la vista de Mapa se genera con una única herramienta que permite desplazarnos por el mapa.

Sin embargo, este add-on permite añadir nuevas herramientas a la vista de mapa ejecutando los siguientes comandos :

web mvc geo tool measure
Añade una herramienta de medición a la vista del mapa

web mvc geo tool custom
Añade una herramienta personalizada a la vista del mapa

Estos comandos añaden en la vista `show.jspx` del mapa la nueva herramienta a utilizar.

Para mas información sobre los comandos ver el apéndice de comandos de este add-on.

21.8.1. Otras herramientas disponibles para añadir al Mapa

Existe la posibilidad de añadir una serie de herramientas al mapa diferentes a las mencionadas anteriormente. Para ello revisar el directorio *WEB-INF/tags/geo/tools* donde se encuentran los *tagx* correspondientes a dichas herramientas

Para añadir una de estas herramientas, situarse en la vista `show.jspx` del mapa donde queremos introducir la herramienta e incluir una llamada al *tagx* correspondiente dentro de *<geo:toolbar>*, introduciendo aquellos parámetros que sean requeridos

Ejemplo de inclusión de la herramienta impresión (*print*):

```
<geo:map id="ps_com_springsource_petclinic_web_Map" projection="EPSG3857" z="user-managed">
    ...
    <geo:toc id="ps_com_springsource_petclinic_web_Map_toc" z="user-managed">
        ...
    </geo:toc>
    ...
    <geo:toolbar id="ps_com_springsource_petclinic_web_Map_toolbar" z="user-managed">
        <tool:print id="ps_com_springsource_petclinic_web_Map_print"/>
    </geo:toolbar>
    ...
</geo:map>
```

Para un correcto funcionamiento de la herramienta *print* se deben de activar las opciones de impresión que permiten imprimir las imágenes de fondo. Dichas opciones se encuentran disponibles en la ventana que se abre al seleccionar la herramienta, en Firefox situadas en la pestaña *Opciones*, mientras que en Chrome se encuentran situadas en el apartado *Configuración*

21.9. Añadir componente Mini Mapa en la vista del Mapa

Mediante el siguiente comando se permite añadir el componente mini mapa a la vista mapa que se seleccione, el cual, contedrá las mismas capas que se incluyan en el mapa general:

web mvc geo component overview
Añade el componente mini mapa a la vista de mapa.

Este comando añade en la vista `show.jspx` del mapa el *tagx* correspondiente al mini mapa. Dicho mini mapa facilita la navegación del mapa al que representa.

Una vez ejecutado el comando se pueden editar las capas que se muestran en el mini mapa yendo a la página `show.jspx` y eliminando o añadiendo capas al elemento *geo:overview*

Para mas información sobre el comando ver el apéndice de comandos de este add-on.

21.10. Añadir Componentes Geográficos

Existe la posibilidad de añadir los siguientes componentes geográficos al mapa:

Coordenadas
Muestra las coordenadas reales en las que se encuentra el puntero de ratón sobre el mapa.

Escala
Muestra la escala en la que se encuentra la vista actual del mapa.

Buscador por callejero
Herramienta que permite la localización de ubicaciones a partir del texto introducido en un campo.

Control de opacidad de capas
Herramienta que permite aumentar o disminuir la opacidad de las capas del mapa.

21.10.1. Coordenadas

Para añadir el componente coordenadas hay que acceder a la vista `show.jspx` del mapa e incluir el *tagx* `coordinates` dentro del *tagx* `components` que sirve como contenedor de componentes geográficos, el cual, se debe de incluir dentro del mapa en caso de que no se encuentre ya establecido dentro del mismo.

Ejemplo:

```
<geo:components id="ps_com_springsource_petclinic_web_Map_components">
    <components:coordinates id="ps_com_springsource_petclinic_web_Map_coordinates"/>
</geo:components>
```

Este componente dispone de diversas opciones. Para más información, revisar los parámetros que contiene el *tagx*.

21.10.2. Escala

Para añadir el componente escala hay que acceder a la vista `show.jspx` del mapa e incluir el *tagx* `scale` dentro del *tagx* `components` que sirve como contenedor de componentes geográficos, el cual, se debe de incluir dentro del mapa en caso de que no se encuentre ya establecido dentro del mismo.

Ejemplo:

```
<geo:components id="ps_com_springsource_petclinic_web_Map_components">
    <components:scale id="ps_com_springsource_petclinic_web_Map_coordinates"/>
</geo:components>
```

Este componente dispone de diversas opciones. Para más información, revisar los parámetros que contiene el *tagx*.

21.10.3. Buscador por callejero

Para añadir la herramienta buscador por callejero hay que acceder a la vista `show.jspx` del mapa e incluir el *tagx* `geosearch` dentro del *tagx* `toolbar` , el cual establece la barra de herramientas disponibles en el mapa y se debe de incluir dentro del mismo en caso de que no se encuentre ya establecido.

Ejemplo:

```
<geo:toolbar id="ps_com_springsource_petclinic_web_Map_toolbar" z="user-managed">
    <tool:geosearch id="ps_com_springsource_petclinic_web_Map_geosearch"/>
</geo:toolbar>
```

Este componente dispone de diversas opciones. Para más información, revisar los parámetros que contiene el *tagx*.

21.10.4. Control de opacidad de capas

Para añadir el deslizador de control de opacidad de las capas, hay que aceder a la vista `show.jspx` del mapa e incluir el *tagx* de la barra de herramientas `toc-toolbar`, donde se añadirá el *tagx* la herramienta `tool-opacity`

Ejemplo:

```
<geo:toc id="ps_com_springsource_petclinic_web_Map_toc" z="YpKTYjQzut/zC96UI52ho3hliRk=">
  <geo:toc-toolbar id="ps_com_springsource_petclinic_web_Map_toc_toolbar" z="user-managed">
    <tool:opacity id="opacity_slider" z="user-managed"/>
  </geo:toc-toolbar>
```

21.11. Desactivar/activar ordenación de capas

En caso de que no se requiere de un orden modificable en las capas de un mapa, es posible desactivar la característica de "Arrastrar y soltar" las capas en un TOC, modificando en valor de la propiedad *allowDragAndDrop* en el *tagx* `map` que se encuentra en el fichero `show.jspx` del mapa.

Ejemplo:

```
<geo:map id="ps_com_springsource_petclinic_web_Mapview" allowDragAndDrop="false"
projection="EPSG3857" z="j4vEuc4wCAAADPl/fMnOqWSG358="
...
```

21.12. Mostrar filtrado de entidades

Es posible configurar la forma en la que se muestra la característica de filtrado de entidades mediante el componente Datatable. En la propiedad *showFilterOn* incluída en *tagx* `entity` se puede seleccionar entre mostrar esta característica en una nueva ventana del navegador o sobre un diálogo en la misma ventana del mapa. Para gestionar como se muestra esta característica en el proyecto mediante una función Javascript personalizada, hay que definir esta propiedad como *custom* y el nombre de la función del usuario a la propiedad *showFilterFunction*. Todos estos parámetros se han de añadir al *tagx* `entity` incluido en el fichero `show.jspx` del mapa.

Ejemplo:

```
<layer:entity id="l_com_springsource_petclinic_domain_Owner" showFilterOn="custom"
showFilterFunction="fnFilterEntities" selection="true" path="/owners"
pk"id" z"nWPaEnqhLlf4IK4LpCT/Sovgspk=">
```

21.13. Personalizar capas generadas

Los comandos mencionados en los puntos anteriores utilizan una serie de *tagx* para incluir los elementos definidos (capas, geometrías, etc) en las respectivas *jspx*. Dichos *tagx* contienen una serie de parámetros no especificados como parámetros de comando, los cuales pueden ser útiles para la personalización de los elementos, por ejemplo *minZoom*, que permite establecer el mínimo zoom a alcanzar en el caso del *tagx* correspondiente a una capa base (*tile*). Para obtener más información sobre estos parámetros consultar los ficheros *tagx* donde estos parámetros incluyen una descripción en su definición. Estos ficheros se encuentran dentro de la carpeta del proyecto *WEB-INF/tags/geo/*

21.13.1. Añadir etiquetas a capas

Para añadir un título a una etiqueta personalizada a una etiqueta del mapa, hay que acceder a la vista `show.jspx` del mapa e incluir dentro del *tagx* de la capa la propiedad *labelingText* con el texto deseado como valor de ésta.

Ejemplo:

```
<layer:entity-field id="l_com_springsource_petclinic_domain_Owner_distance"
labelingText="Distance">
```

21.13.2. Personalizar título de capa

Para personalizar el titulo en formato HTML con el que una capa se mostrará en el TOC, se ha de añadir en el fichero `show.jspx` del mapa, un *tagx* `title` dentro del de la capa a modificar, incluyéndole el texto HTML que se quiera mostrar.

Ejemplo:

```
  <layer-toc-title>
    <span><b> :: Pet Area :: </b></span>
  </layer-toc-title>
```

21.13.3. Herramientas de capa

Existe la posibildad de añadir herramientas a las distintas capas de un mapa

Para añadir herramientas a las capas de un mapa hay que acceder a la vista `show.jspx` del mapa e incluir el *tagx* de la herramienta a añadir dentro del *tagx* `tools` el cual establece las diferentes herramientas de las que dispondrá una capa y se debe de incluir dentro del mismo en caso de que no se encuentre ya establecido.

Ejemplo:

```
<layer-toc:tools>
    <layer-tool:zoom-select id="owner-location-zoom-select"/>
</layer-toc:tools>
```

Capítulo 22. Add-on Campos Lupa

Implementación de campos lupa en el proyecto gvNIX.

22.1. Descripción

Este add-on permite utilizar componentes de tipo lupa en aplicaciones gvNIX. Gracias a estos componentes, podemos buscar registros de forma sencilla de campos relacionados sin tener que visualizar todos los datos en un desplegable.

Al aplicar este add-on sobre un proyecto, en los formularios es posible seleccionar registros de campos relacionados de una entidad. Estos registros son mostrados en una lista dinámica según las coincidencias de carácteres del texto introducido o en la lista completa de los registros de este campo.

El addon permite configurar la representación de registros a mostrar , permitiendo limitar el número máximo de registros, los campos campo por los que se buscará coincidencias, los campos de la entidad relacionada que se mostrarán en cada resultado, filtros personalizados de búsqueda, etc.

22.2. Instalación del componente lupa

Para instalarlo, hay que utilizar el comando web mvc loupe setup, el cual solo estará disponible después de instalar el add-on JQuery y el add-on Datatables en nuestro proyecto.

22.3. Permitiendo a una entidad utilizar el campo lupa

Una vez instalados todos los componentes, es necesario generar una serie de métodos para realizar las búsquedas y la visualización de los resultados. Para que el controlador disponga de estos métodos, utilizaremos el comando web mvc loupe set sobre el controlador seleccionado.

22.4. Utilizando componentes lupa

Ahora que el Controlador ya dispone de los métodos necesarios, ejecutaremos el comando web mvc loupe field sobre el field de la entidad a la que pertenece el controlador con los métodos generados. Una vez ejecutado este comando, transformaremos el campo indicado a un campo de tipo lupa en la capa web de nuestra aplicación.

Importante: El controlador de la entidad a la que referencia el campo también debe tener instalado Datatables con un modo de visualización de listado estándar para la correcta visualización del componente lupa. Si se desea instalar Datatables con un modo no estándar (mode="show", por ejemplo), se puede crear un controlador adicional para la misma entidad con el modo estándar de Datatables y asignar el atributo "listPath" del campo lupa a la vista "list" de este segundo controlador. El atributo "listPath" se explica en la sección Configuración del widget lupa.

22.5. Actualizando componentes lupa

Para mantener actualizados los componentes importados mediante el comando de instalación, podemos ejecutar el comando web mvc loupe update.

22.6. Apendice de comandos

Para ver con mas detalle el comando que proporciona el add-on consultar la sección de comandos del add-on Lupa.

22.7. Configuración del widget lupa

Este addon permite mostrar el componente lupa en las diferentes *jspx* mediante el fichero `loupe.tagx`,el cual es añadido al proyecto al instalar el addon. Este fichero contiene una serie de propiedades para definir las opciones de configuración del componente, como por ejemplo *max*, que permite limitar el número de resultados a mostrar. Para obtener más información sobre estos parámetros consultar el fichero `WEB-INF/tags/loupefield/loupe.tagx` donde estos parámetros incluyen una descripción en su definición.

Ejemplo:

```
<form:create id="fu_com_springsource_petclinic_domain_Pet"
modelAttribute="pet" path=" /pets"
render="${empty dependencies}" z="E4jWxGSZUphUG7qjB1vBKbchssY=">
  <field:input field="name" id="c_com_springsource_petclinic_domain_Pet_name" min="1"
required="true" z="xqcX/Hnb52FNxML/Xqv1yGlbMUM="/>
  <loupefield:loupe returnFields="telephone" searchField="city" hiddeUtilbox="false"
modalWidth="400" required="true" baseFilter="city=Valencia"additionalFields="
lastName,city" max="4" caption="firstName" field="owner" controllerPath="pets"
id="c_com_springsource_petclinic_domain_Pet_owner" listPath="owners/list" mode="update"
pkField="id" z="user-managed"/>
</form:create>
```

- **field** debe ser el campo del objeto expuesto en el formulario que se quiere representar en el campo lupa.
- **required** indica si este campo es obligatorio (el valor por defecto es false).
- **listPath** ruta a la vista de lista del controlador, utilizada para mostrar los registros en el diálogo modal de la lupa.
- **controllerPath** ruta relativa al controller de la entidad a la que pertenece la vista.
- **pkField** es el campo de la clave primaria de la entidad de referencia en el campo lupa. Se utiliza para realizar las búsquedas.
- **caption** campos de la entidad se mostrarán en el campo lupa cuando se elige un resultado. Deben estar separados por comas.
- **additionalFields** campos adicionales de la entidad por los que se buscarán coincidencias. Si no se especifica ninguno, se buscará por id.
- **baseFilter** filtros por defecto por los que se mostrarán resultados. En el ejemplo sólo se mostrarán owners cuya ciudad es valencia.
- **modalWidth** anchura del diálogo modal, en píxeles (el valor por defecto es 800).
- **hiddeUtilbox** especifica si ocultar elementos con la clase utilbox (el valor por defecto es true).

- **searchField** especifica el campo usado para buscar (en el filtro de diálogo) cuando el input pierde el foco (por defecto es el primer campo de caption).
- **returnFields** campos adicionales que devolverá la búsqueda de Ajax, pero por los que no se buscará. Deben estar separados por comas.

Capítulo 23. Add-on OCC (Optimistic Concurrency Control)

Add-on de utilidades enfocadas a la persistencia de las entidades.

Este add-on proporciona un sistema que facilita la detección de modificaciones concurrentes en la persistencia de entidades del modelo de datos.

El sistema consiste en un control de *concurrencia optimista* basada en un *checkSum o suma de control* de los valores de una entidad dada.

23.1. Introducción

Esta utilidad añade a las entidades un sistema de *control de concurrencia* basado en una operación de *checksum o suma de control* sobre todos los campos persistentes que contiene la entidad.

El *control de concurrencia optimista* ya es ofrecido por defecto por el núcleo de Spring Roo. Sin embargo, el sistema que proporciona se basa en la existencia de un campo en cada una de las tabla de base de datos que se desea controlar. Este campo se suele definir como *version* (numérico que identifica la versión del objeto) o *lastupdate* (fecha de la última actualización). Esto es un problema para aquellas bases de datos ya existentes y en las que no se puede o no se quiere alterar su estructura.

Esta funcionalidad nace de la necesidad de controlar la concurrencia sobre modelos de datos heredados en los que no es posible alterar el modelo relacional de base de datos para añadir los campos de control en los que se basa el mecanismo de *control de concurrencia* del núcleo de Spring Roo que sigue la especificación JPA.

El checksum consiste en una operación matemática de los distintos valores que contiene un registro (instancia) de una entidad. El cálculo tiene en cuenta todas las propiedad de tipo simple de la entidad, es decir, no se incluye en el cálculo propiedad que represente una relación con otro u otros objetos. El cálculo se realiza en el momento de la carga del registro de la entidad desde la base de datos y es almacenado en una propiedad no persistente de la misma entidad.

En el momento de *actualizar o eliminar* un registro de dicha entidad se carga el elemento que está persistido en la base de datos y *se compara el valor de checksum* actual con el que se calculó en el momento de la carga del elemento a actualizar o eliminar. *Si no coincide se lanza una Excepción* (javax.persistence.OptimisticLockException) y la operación no se realizará.

Esta operación supone un coste adicional en las tareas de persistencia, pero no es apreciable en condiciones del uso cuotidiano de la aplicación excepto en procesos de actualizaciones masivas de registros.

Para mostrar un mensaje amigable al usuario se puede utilizar el Add-on Web Dialog que, en el conjunto de excepciones gestionadas por defecto se incluye esta. Una vez inicializada la gestión de excepciones es posible personalizar el mensaje usando el comando web mvc dialog exception set language

23.2. Añadir el control en las entidades

Para activar esta característica sobre una entidad se pueden utilizar los siguientes comandos:

occ checksum set
: Aplica el control de concurrencia a una entidad.

occ checksum all
: Aplica el control de concurrencia a todas las entidades de la aplicación.

Estos comandos añaden a uno o a todos los ficheros `.java` de las entidades una anotación `@GvNIXEntityOCCChecksum` y un campo no persistente para almacenar el checksum. En base a esta anotación se generará un *aspecto java* (fichero `*_gvNIX_occChecksum.aj`) con toda la lógica necesaria para calcular el checksum en base a las propiedades de un registro de la entidad y para controlar antes de una actualización o borrado si el registro ha cambiado desde que se cargo inicialmente. Toda esta lógica sera manejada automáticamente por el add-on.

Si se aplica el control de concurrencia sobre una entidad que extiende de otra y en la clase padre existe un campo anotado como campo de control de persistencia con @javax.persistence.Version, el add-on detectará esta circunstancia y aplicará el control de concurrencia sobre la clase padre.

Para ver todos los comandos y sus opciones ver el apéndice de comandos.

Nota

Se ha detectado que al aplicar este control de concurrencia sobre entidades que tienen generados, o sobre los que se van a generar, tests de integración (mediante el comando *test integration* o mediante la opción *--testAutomatically* del comando *entity jpa*) los tests de integración presentarán un error de compilación. Se debe a que el add-on que genera los tests de integración no considera el caso en el que el campo que hace de control de concurrencia sea del tipo cadena. Hay varias opciones para evitar estos errores de compilación:

1. No es recomendable, pero se pueden desactivar los tests de los métodos *flush()* y *merge()* (métodos *testFlush()* y *testMerge()*). Para ello hay que editar los archivos *src/test/java/**/*IntegrationTest.java* y añadir a la anotación *@RooIntegrationTest* los atributos *flush = false* y *merge = false*, de esta manera estos dos tests no se generan en sus aspecto Java correspondiente quedando la batería de pruebas incompleta pero compilando correctamente:

   ```
   @RooIntegrationTest(entity = MiClase.class, flush = false, merge = false)
   ```

2. La más recomendable es corregir los dos test mencionados en el punto anterior. Para ello se debe realizar un *pushin* de estos dos métodos que se encuentran declarados en *src/test/java/**/*IntegrationTest_Roo_IntegrationTest.aj* y también el campo <nombre_entidad>DataOnDemand dod, anotado con @Autowired, llevandolos a la clase Java del test de integración. Una vez los test están en la clase Java, se pueden corregir los errores de compilación reemplazando el operador ">" por el método *equals*. A continuación se muestra un ejemplo extraido del proyecto de ejemplo petclinic que se distribuye junto a gvNIX/Roo en el que se han corregido las líneas marcadas en negrita para los métodos *testFlush()* y *testMerge()*.

   ```
   package com.springsource.petclinic.domain;

   import org.junit.Test;
   import org.springframework.beans.factory.annotation.Autowired;
   import org.springframework.roo.addon.test.RooIntegrationTest;
   ```

```
@RooIntegrationTest(entity = Owner.class)
public class OwnerIntegrationTest {
    @Autowired
    private OwnerDataOnDemand dod;

    @Test
    public void testMarkerMethod() {

    }

    @Test
    public void testFlush() {
        com.springsource.petclinic.domain.Owner obj = dod.getRandomOwner();
        org.junit.Assert.assertNotNull(
          "Data on demand for 'Owner' failed to initialize correctly", obj);
        java.lang.Long id = obj.getId();
        org.junit.Assert.assertNotNull(
          "Data on demand for 'Owner' failed to provide an identifier", id);
        obj = com.springsource.petclinic.domain.Owner.findOwner(id);
        org.junit.Assert.assertNotNull(
          "Find method for 'Owner' illegally returned null for id '"
          + id + "'", obj);
        boolean modified =  dod.modifyOwner(obj);
        java.lang.String currentVersion = obj.getOccChekcsum();
        obj.flush();
        org.junit.Assert.assertTrue(
          "OccCheckSum for 'Owner' failed to modify on flush directive",
          (currentVersion != null &&
          obj.getOccChekcsum().equals(currentVersion)) ||
          !modified);
    }

    @Test
    public void testMerge() {
        com.springsource.petclinic.domain.Owner obj = dod.getRandomOwner();
        org.junit.Assert.assertNotNull(
          "Data on demand for 'Owner' failed to initialize correctly", obj);
        java.lang.Long id = obj.getId();
        org.junit.Assert.assertNotNull(
          "Data on demand for 'Owner' failed to provide an identifier", id);
        obj = com.springsource.petclinic.domain.Owner.findOwner(id);
        boolean modified =  dod.modifyOwner(obj);
        java.lang.String currentVersion = obj.getOccChekcsum();
        com.springsource.petclinic.domain.Owner merged = (
          com.springsource.petclinic.domain.Owner) obj.merge();
        obj.flush();
        org.junit.Assert.assertEquals(
          "Identifier of merged object not the same as identifier of " +
            "original object",
          merged.getId(), id);
        org.junit.Assert.assertTrue(
          "OccCheckSum for 'Owner' failed to modify on merge and flush " +
            "directive",
          (currentVersion != null &&
          obj.getOccChekcsum().equals(currentVersion))
          || !modified);
    }
}
```

Capítulo 24. Add-on Web Dialog

Add-on para la gestión de excepciones no controladas y de mensajes de usuario.

24.1. Descripción

Este add-on permite gestionar las excepciones no controladas en el código de la aplicación y así mostrar mensajes inteligibles al usuario en una ventana modal (pop-up) y en el idioma actual de la aplicación sin necesidad de realizar ningún desarrollo en la capa web.

El add-on, al aplicarse sobre un proyecto, por defecto configura ciertas excepciones que serán visualizadas de forma amigable al usuario en el caso de no ser gestionadas directamente desde la aplicación. Así mismo, se pueden incluir nuevas excepciones y modificar los textos amigables que se mostrarán en lugar de la excepción para cada idioma disponible en la aplicación.

Durante su aplicación se instalan algunas cadenas, tanto en ingles como en castellano, para mostrar mensajes para los errores de conversión de datos numéricos y fechas.

El add-on también permite mostrar mensajes de usuario sin realizar ninguna implementación en la capa web mediante la invocación de métodos desde el controlador que son los encargados de mostrar dichos mensajes en la pantalla del usuario.

Los mensajes de excepción y de usuario se muestran en un diálogo o ventana modal. Se permite además la modificación de las ventanas modales de excepción y de mensaje de usuario para que puedan ser personalizadas de acuerdo a los requerimientos que sean necesarios.

24.2. Instalación

Para aplicar esta funcionalidad sobre un proyecto se debe ejecutar en primer lugar el comando web mvc dialog setup. Al hacerlo se incluye el soporte para diálogos modales en la aplicación.

El componente visual *message-box.tagx* que instala el add-on es el encargado de mostrar en pantalla un diálogo modal al usuario.

Al mismo tiempo, la ejecución de este comando configura una serie de excepciones que serán mapeadas con sus respectivos diálogos modales y mensajes amigables en múltiples idiomas. En las siguiente secciones se verá más detalladamente las excepciones controladas y el uso de los diálogos modales.

24.3. Excepciones controladas por gvNIX

Conjunto de excepciones inicialmente definidas y controladas por gvNIX al ejecutar el comando *web mvc dialog setup*.

- java.sql.SQLException.

  Se ha producido un error en el acceso a la base de datos.

- java.io.IOException.

  Existen problemas para enviar o recibir datos.

- org.springframework.transaction.TransactionException.

Se ha producido un error en la transacción. No se han guardado los datos correctamente.

- java.lang.UnsupportedOperationException.

  Se ha producido un error no controlado.

- javax.persistence.OptimisticLockException.

  No se puede actualizar el registro debido a que ha sido actualizado previamente.

Las excepciones se muestran mediante un dialogo modal. Para ello, se modifica la definición, en el archivo webmvc-config.xml, del bean *SimpleMappingExceptionResolver* reemplazando la clase que se define por una propia. La clase propia se encontrará instalada en el proyecto en la clase *MessageMappingExceptionResolver.java* del subpaquete *web.servlet.handler*. En el subpaquete *web.dialog* se instala también la clase *Dialog.java*, la cual es un bean que representa la información necesaria para visualizar una excepción en la ventana modal renderizada por el componente *message-box.tagx*.

24.3.1. Añadir nuevas excepciones a la gestión

Pasos a seguir para incluir en la gestión automática de excepciones de la aplicación una nueva excepción.

El Add-on proporciona ciertos comandos que permiten realizar de forma automática las siguientes operaciones para la gestión de excepciones:

1. Añadir el control de una excepción a la aplicación:

   - Define la excepción a capturar en el bean *messageMappingExceptionResolverBean* del archivo *webmvc-config.xml*.
   - Asocia una dirección a la excepcion en un nuevo bean *<mvc:view-controller>* en el archivo *webmvc-config.xml*.
   - Crea la nueva jspx asociada a la excepción.
   - Instancia la jspx creada en el archivo *views.xml* en el directorio *WEB-INF* la aplicación.
   - Crear las etiquetas multi idioma en los ficheros de propiedades de todos los idiomas que estén instalados en la aplicación.

2. Mostrar las excepciones gestionadas por la aplicación:

   - Muestra la lista de excepciones definidas en el bean *messageMappingExceptionResolver* del archivo *webmvc-config.xml*.

3. Eliminar excepciones controladas por la aplicación:

   - Elimina del fichero *webmvc-config.xml* las referencias a la excepción y por lo tanto la quita de las excepciones gestionadas por la aplicación.
   - Elimina la página jspx asociada.
   - Elimina las referencias a la excepción en el archivo views.xml.

Para obtener más información sobre estos comandos de gestión de excepciones consultar el apendice Comandos del Add-on Web Dialog.

24.4. Nuevos diálogos modales

Estos diálogos son útiles para mostrar mensajes al usuario informando de cualquier circunstancía tras una operación determinada.

Para aplicar esta funcionalidad sobre el proyecto primero se debe haber ejecutado el comando web mvc dialog setup.

El comando web mvc dialog add añade la anotación *GvNIXModalDialogs* en la clase controladora que se le indica.

```
roo-gvNIX> web mvc dialog add --class ~.web.PetController --name petsdialog
```

La anotación generará dos métodos asociados con el controlador. Un método con el nombre *modalDialog* y otro que tomará como nombre el valor del parámetro *name* del comando y una página jspx en *WEB-INF/dialogs* con también el mismo nombre.

El método *modalDialog* permite al desarrollador mostrar diálogos modales genéricos en la parte visual directamente desde el código Java del controlador. Para ello, el desarrollador proporcionará un título, una descripción y un tipo de mensaje que será mostrado en la página al usuario. La signatura del método es:

```
modalDialog(DialogType dialogType, String title, String description,
      HttpServletRequest httpServletRequest)
```

El segundo método servirá para mostrar un mensaje específico. La diferencia principal es que este nuevo método tiene su propia página jspx asociada y además permite proporcionar parámetros adicionales a dicha página, cosa que no permite el método *modalDialog*.

```
petsdialog(DialogType dialogType, String title, String description,
      HashMap<String, Object> params, HttpServletRequest httpServletRequest)
```

Asociado a este método también se habrá creado en *WEB-INF/dialogs* una página jspx con el nombre *petsdialog.jspx* que podrá personalizarse según las necesidades. El Map *params* será proporcionado a la página y por lo tanto el desarrollador puede insertar valores desde el controlador para utilizarlos en el diálogo.

Los parámetros de cada uno de los dos métodos anteriores definen la siguiente información:

dialogType
: es un tipo enumerado que puede tomar como valores: Error, Info, Alert, Suggest. Cada uno define un nivel de severidad en el diálogo y producirá en el aspecto visual del diálogo que aparezca con distinto color e icono.

title
: es el código del recurso i18n que se usará como título del diálogo modal.

description
: es el código del recurso i18n que se usará como descripción del diálogo modal.

params
: es un Map que se puede utilizar para proporcionar tantos parámetros como sea necesario al componente visual del diálogo modal para confeccionarlo y mostrar en él cualquier información necesaria.

httpServletRequest
: este parámetro se usa internamente en el método para obtener la sesión del usuario y establecer un atributo que será leido por el componente message-box.tagx para mostrar el diálogo. Si al invocar el método no se dispone de este parámetro, se puede obtener añadiendo el parámetro *HttpServletRequest httpServletRequest* a los parámetros del método que contiene la invocación (esta es una característica especial de los métodos que gestionan las URLs en Spring MVC y que permite declarar distintos parámetros de entre algunos dados).

24.4.1. Ejemplos de dialogos personalizados

Con este nuevo soporte para crear diálogos modales gvNIX ofrece una gran libertad para maquetar multitud de mensajes de usuario o diálogos de la aplicación (avisos, errores, mensajes de confirmación, formularios, etc.).

A continuación se muestra una pequeña demostración de implementación de algunos de estos mensajes personalizados.

24.4.1.1. Mensaje de aviso de aplicación

En ocasiones puede ser necesario que tras una acción (una petición al servidor) se muestre en la respuesta algún mensaje acerca del resultado de la operación solicitada o, en el caso de pantallas de búsqueda, indicar que no se han encontrado resultados.

Con el soporte de este add-on se puede incluir la siguiente línea de código en cualquier punto de un método de un controlador para definir un mensaje informativo.

```
modalDialog(DialogType.Info, "message_info_title", "message_description_key",
          httpServletRequest);
```

24.4.1.2. Mensaje de confirmación

Puede ser necesario que tras una acción, la aplicación deba preguntar si se desea ir a una página en concreto. A continuación se verán los pasos y cambios en el código de un controlador para definir este mensaje de confirmación.

En el ejemplo, se va a añadir a la aplicación de ejemplo petclinic un mensaje de confirmación que consultará al usuario si quiere ir al formulario de mascotas (Pets) tras actualizar la información de un propietario (Owner).

1. Añadir un diálogo modal a la aplicación mediante "*web mvc dialog message add*"

   ```
   web mvc dialog add --class ~.web.OwnerController --name confirmgopets
   ```

   Esto generará un método *confirmgopets* disponible en *OwnerController* y una jspx base llamada *confirmgopets* en *WEB-INF/dialogs* que será personalizada.

2. Modificar el método *update* de *OwnerController* para añadir la llamada al método *confirmgopets* justo antes de la línea de *return*. Para ello se debe llevar el método *update* desde

OwnerController_Roo_Controller.aj hasta *OwnerController.java*, podemos usar la opción push-in que ofrece Eclipse. Una vez movido el método, será modificado quedando como sigue:

```
@RequestMapping(method = RequestMethod.PUT, produces = "text/html")
public String update(@Valid Owner owner, BindingResult bindingResult, Model uiModel,
  HttpServletRequest httpServletRequest) {
    if (bindingResult.hasErrors()) {
        populateEditForm(uiModel, owner);
        return "owners/update";
    }
    uiModel.asMap().clear();
    owner.merge();

    HashMap<String, Object> dialogParams = new HashMap<String, Object>();
    dialogParams.put("petsFormLink", "/pets?form");
    confirmgopets(DialogType.Info, "message_confirm_action_title",
      "message_confirm_action_desc", dialogParams, httpServletRequest);

    return "redirect:/owners/" + encodeUrlPathSegment(owner.getId().toString(),
      httpServletRequest);
}
```

Se ha resaltado en negrita la parte específica que ha de modificarse para mostrar el diálogo modal.

El HashMap *dialogParams* permite proporcionar al componente message-box.tagx información adicional para mostrarla o utilizarla como sea necesario. En este caso, es necesario indicar a qué página dirigir al usuario en caso de que responda afirmativamente a la pregunta de confirmación, pero se podría definir cualquier tipo de parámetro, desde un String hasta objetos del Modelo de la aplicación o listas de objetos.

3. Modificar la jspx que ha creado el add-on (confirmgopets.jspx) para adecuarla a las necesidades del proyecto:

```
<div xmlns:c="http://java.sun.com/jsp/jstl/core"
              xmlns:util="urn:jsptagdir:/WEB-INF/tags/util"
              xmlns:fn="http://java.sun.com/jsp/jstl/functions"
              xmlns:spring="http://www.springframework.org/tags"
              xmlns:jsp="http://java.sun.com/JSP/Page" version="2.0">
  <jsp:directive.page contentType="text/html;charset=UTF-8" />
  <jsp:output omit-xml-declaration="yes" />
  <spring:message var="title" code="${titleCode}" text="Message title"
    htmlEscape="false" />
  <util:panel id="title" title="${title}">
    <h2>${fn:escapeXml(title)}</h2>
    <p>
     <spring:message code="${descriptionCode}"
       text="This is the dialog description ..." htmlEscape="false"/>
    </p>
    <div class="closeMessage">
     <spring:message var="closeButtonValue" code="button_close"/>
     <button class="boton" dojoType="dijit.form.Button" type="button"
       onClick="dijit.byId('${dialogId}').hide();">${closeButtonValue}</button>
      <spring:url value="${dialogParams['petsFormLink']}" var="locationUrl"/>
      <spring:message var="confirmButtonValue" code="confirmButtonValue"/>
      <button class="boton" dojoType="dijit.form.Button" type="button"
        onClick="location.href = '${locationUrl}';">${confirmButtonValue}</button>
    </div>
  </util:panel>
</div>
```

En este caso se ha resaltado el accesso al HashMap de parámetros del diálogo (dialogParams) para demostrar el modo en que se debe extraer los datos que le llegan al diálogo desde el controlador.

24.4.1.3. Diálogo modal con formulario

En alguna ocasión puede ser necesario mostrar un formulario en un diálogo modal para solicitar información al usuario, siguiendo el ejemplo anterior, se debería modificar la jspx que crea el add-on para incluir un formulario. En este ejemplo, tras crear un nuevo Owner en la aplicación, se mostrará un formulario para que cree su primera mascota.

1. Añadir un diálogo modal a la apliacaión mediante "*web mvc dialog message add*"

```
web mvc dialog add --class ~.web.OwnerController --name createPetInModal
```

   Esto genera un método *createPetInModal* disponible en *OwnerController* y una jspx base llamada *createPetInModal* en *WEB-INF/dialogs* que será personalizada.

2. Modifica el método create de OwnerController. Al igual que en el ejemplo anterior debemos hacer el Push-in del método. Lo modificamos para que quede como se muestra:

```
@RequestMapping(method = RequestMethod.POST, produces = "text/html")
public String create(@Valid Owner owner, BindingResult bindingResult, Model uiModel,
  HttpServletRequest httpServletRequest) {
    if (bindingResult.hasErrors()) {
        uiModel.addAttribute("owner", owner);
        addDateTimeFormatPatterns(uiModel);
        return "owners/create";
    }
    uiModel.asMap().clear();
    owner.persist();

    HashMap<String, Object> dialogParams = new HashMap<String, Object>();
    Pet firstPetOfOwner = new Pet();
    dialogParams.put("pet", firstPetOfOwner);
    dialogParams.put("pettypes",  Arrays.asList(PetType.class.getEnumConstants()));
    createPetInModal(DialogType.Info, "message_create_pet_title",
      "message_create_pet_desc", dialogParams, httpServletRequest);

    return "redirect:/owners/" + encodeUrlPathSegment(owner.getId().toString(),
      httpServletRequest);
}
```

3. Modificar la jspx incluyendo el formulario de creación de mascotas (Pets):

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<div xmlns:c="http://java.sun.com/jsp/jstl/core"
  xmlns:field="urn:jsptagdir:/WEB-INF/tags/form/fields"
  xmlns:form="urn:jsptagdir:/WEB-INF/tags/form"
  xmlns:jsp="http://java.sun.com/JSP/Page"
  xmlns:spring="http://www.springframework.org/tags"
  version="2.0">
    <jsp:directive.page contentType="text/html;charset=UTF-8"/>
    <jsp:output omit-xml-declaration="yes"/>

    <c:set scope="request" var="pet" value="${dialogParams['pet']}" />
    <c:set scope="request" var="pettypes" value="${dialogParams['pettypes']}" />

    <form:create id="fc_com_springsource_petclinic_domain_Pet" modelAttribute="pet"
      path="/pets" render="${empty dependencies}" z="lgvEyAlAYOudDmaPjwU0ABseTIk=">
        <field:checkbox field="sendReminders"
          id="c_com_springsource_petclinic_domain_Pet_sendReminders"
          z="uPpMX+IWb0KONpvd11fpG8x4/4Q="/>
        <field:input field="name" id="c_com_springsource_petclinic_domain_Pet_name"
          min="1" required="true" z="ZY+k75JeSo9RmejYZRFNIvs2aBg="/>
        <field:input field="weight" id="c_com_springsource_petclinic_domain_Pet_weight"
          min="0" required="true" validationMessageCode="field_invalid_number"
```

```
          z="cOD5zE/z7gy+RZu5kVSPuxCa+/I="/>
        <input type="hidden" id="c_com_springsource_petclinic_domain_Pet_owner"
          name="owner" value="${owner.id}" />
        <field:select field="owner" id="c_com_springsource_petclinic_domain_Pet_owner"
          itemValue="id" items="${owners}" path="/owners" render="false"
          z="fGzswAP4XXvhPhowJKsRVve929c="/>
        <field:select field="type" id="c_com_springsource_petclinic_domain_Pet_type"
          items="${pettypes}" path="pettypes" required="true"
          z="+hDCnUp+Y+A1RlT+AjH07sgipOo="/>
    </form:create>
    <form:dependency dependencies="${dependencies}"
      id="d_com_springsource_petclinic_domain_Pet" render="${not empty dependencies}"
      z="kThDNIW+69h9nI/69ynY1WyUieo="/>
</div>
```

24.5. Futuras versiones

Mejoras a incluir en futuras versiones del add-on.

- Obtención de las excepciones y los mensajes multi idioma de las excepciones no controladas que deben visualizarse de forma amigable desde una base de datos en la que se encuentra almacenada dicha información. Así podría definirse un repositorio central de excepciones para múltiples aplicaciones y sus mensajes amigables.
- Envío de un informe por email al responsable de la aplicación con información detallada cuando se ha producido una excepción.

Capítulo 25. Add-on GVA Security

25.1. Descripción

El add-on permite instalar clientes de autenticación y autorización y lo integra con el sistema de seguridad Spring Security. Por defecto están configurados los proveedores SAFE (acceso mediante usuario y contraseña y acceso mediante certificado digital) y APLUSU (antiguo CIT security addon) pero se pueden añadir nuevos proveedores implementando la interfaz SecurityProvider

La documentación de este add-on se encuentra publicada en el portal de información interno de la GVA *confluence*

Capítulo 26. Add-on Web Report

El add-on añade soporte para la generación de informes con *Jasper Reports* y genera informes totalmente funcionales para una entidad que posteriormente pueden ser personalizados. Los informes son accesibles desde la interfaz web mediante un formulario que se crea para tal efecto.

En la sección de recetas se puede encontrar información adicional para realizar el diseño de informes con sub informes.

26.1. Descripción

El add-on añade por un lado el soporte necesario para trabajar con Jasper Reports y por otro es capaz de generar informes asociados con una entidad cualquiera de la aplicación.

Un informe de *Jasper Reports* se se define principalmente mediante un archivo de diseño de informe (jrxml). El diseño de informe al procesarse junto con una fuente de datos, genera un fichero de salida en un formato determinado incluyendo en él la información existente en la fuente de datos. La fuente de datos puede provenir de distintos orígenes, aunque en este caso siempre será una lista de entidades proporcionada directamente desde la aplicación.

El archivo de diseño es un documento XML con un formato determinado que se puede editar de manera sencilla y gráfica mediante una herramienta como iReport.

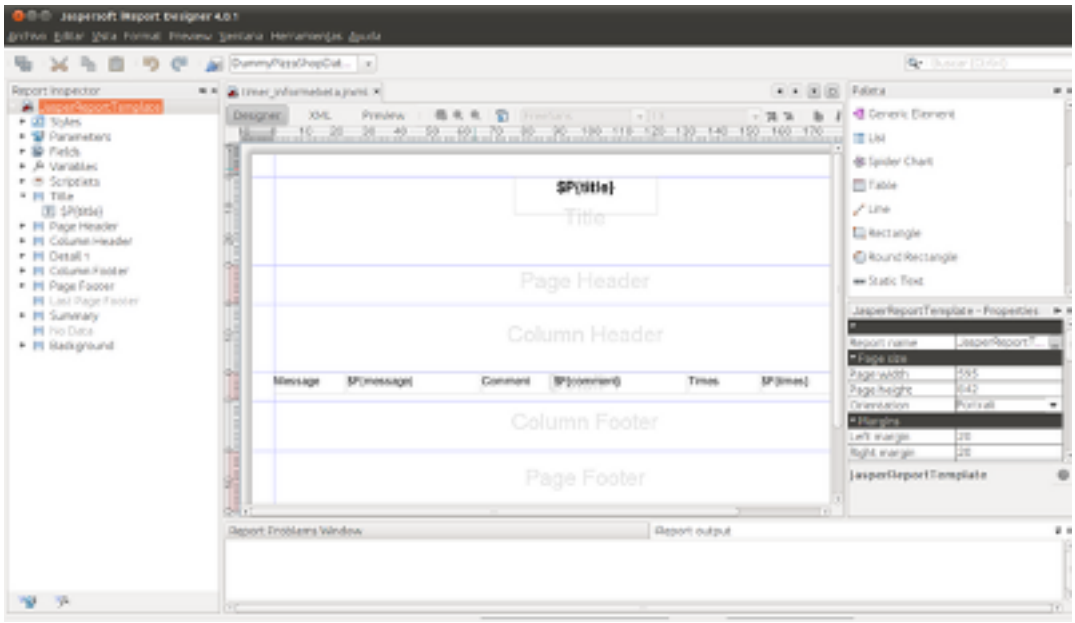

Figura 26.1. Editor de informes iReport

26.2. Instalación

Jasper Reports requiere varias librerías y configuraciones para funcionar, el add-on incluye en el proyecto las dependencias de la librería modificando el pom.xml del mismo e incluye varios ficheros de configuración que se verán a continuación con mas detalle.

El comando sólo estará disponible si el proyecto utiliza Spring MVC en la capa web. Esto es, el archivo de configuración *webmvc-config.xml* debe existir en */WEB-INF/spring*.

Estas operaciones de configuración las realiza el comando del add-on web report setup:

```
roo-gvNIX> web report setup
Updated SRC_MAIN_WEBAPP/WEB-INF/spring/webmvc-config.xml
Created SRC_MAIN_WEBAPP/WEB-INF/spring/jasper-views.xml
Created SRC_MAIN_WEBAPP/WEB-INF/classes/jasperfonts
Created SRC_MAIN_WEBAPP/WEB-INF/classes/jasperfonts/FreeSans.ttf
Created SRC_MAIN_WEBAPP/WEB-INF/classes/jasperfonts/FreeSansBold.ttf
Created SRC_MAIN_WEBAPP/WEB-INF/classes/jasperfonts/FreeSansBoldOblique.ttf
Created SRC_MAIN_WEBAPP/WEB-INF/classes/jasperfonts/FreeSansOblique.ttf
Created SRC_MAIN_WEBAPP/WEB-INF/classes/jasperfonts/gvnix_reportfonts.xml
Updated ROOT/pom.xml [added dependency org.gvnix:org.gvnix.web.report.roo.addon:X.Y.Z;
  added property 'jasperreports.version' = '4.0.1';
  added property 'jasperreportsfonts.version' = '4.0.0';
  added property 'apachepoi.version' = '3.7';
  added dependencies net.sf.jasperreports:jasperreports:${jasperreports.version},
  org.apache.poi:poi:${apachepoi.version},
  net.sf.jasperreports:jasperreports-fonts:${jasperreportsfonts.version}]
Created SRC_MAIN_WEBAPP/WEB-INF/classes/jasperreports_extension.properties
```

El comando configura los siguientes archivos:

- *pom.xml*: añade la dependencia JasperReports, la dependencia Apache POI para informes en formato Excel y una dependencia con el propio add-on para así reconocer la anotación *@GvNIXReports* que se utilizará en el proyecto.
- *webmvc-config.xml*: añade un bean para configurar el sitema de vistas de informes.
- *jasper-views.xml*: añade una definición de vistas por informe.
- *jasperfonts*: directorio que contiene fuentes True Type que usará JasperReports para confeccionar el informe.
- *jasperreports_extension.properties*: define el lugar donde se encuentran las fuentes a utilizar.

Cada uno de los informes se comporta como una vista más de la aplicación y por tanto el add-on habrá incluido en el fichero de configuración de la capa web *webmvc-config.xml* un componente que para cada petición de este tipo devolverá el nuevo tipo de salida (un informe). Esto se logra añadiendo un nuevo *bean*:

```
<bean id="jasperReportsXmlViewResolver"
      class="org.springframework.web.servlet.view.XmlViewResolver"
      p:location="/WEB-INF/spring/jasper-views.xml" p:order="0" />
```

Este *bean* referencia al fichero *jasper-views.xml* como fichero de configuración de las nuevas vistas, por tanto el add-on habrá creado este archivo en *src/main/webapp/WEB-INF/spring*.

Los informes generados pueden contener textos que se deban visualizar con características especiales tales como negrita y cursiva. Para ello, Jasper Reports requiere que las fuentes utilizadas para generar el informe se encuentren en el classpath de la aplicación. Así pues, el add-on se ocupa de copiar en *src/main/webapp/WEB-INF/classes/jasperfonts* varios archivos de fuentes True Type y su definición para poder hacer uso de ellas en el archivo *gvnix_reportfonts.xml*. En el archivo *src/main/webapp/WEB-INF/classes/jasperreports_extension.properties* se indica a Jasper Reports que utilice estas fuentes a la hora de generar el informe.

Si se desean utilizar fuentes de letra adicionales a parte de las incluidas por defecto por el add-on, deberán ser instaladas de forma manual en el proyecto. Para ello se puede consultar la sección Instalar fuentes de letra para los informes de la sección de recetas.

26.3. Generación de un informe

El add-on realiza la generación automática de informes sobre cualquier entidad de la applicación en la que se está utilizando. Posteriormente, los informes podrán ser personalizados por parte del desarrollador de la aplicación para adecuarlos a los requerimientos modificando o bien lo lógica de negocio o bien el estilo visual como se desee.

La petición de un informe, al tratarse de una vista más de la aplicación, ha de ser atendida por un controlador. No obstante, los informes muestran información referente a una entidad del modelo de datos de la aplicación, por tanto el controlador debe atender peticiones referentes a una determinada entidad. Esto, como se comentó en CRUDS automático, se configura mediante la anotación *@RooWebScaffold* y su atributo *formBackingObject.* Así pues, no se puede definir un informe sobre un controlador que no disponga de la anotación *@RooWebScaffold*.

La generación de un informe se realiza mediante el comando web report add o incluyendo la anotación @GvNIXReports en la clase del controlador. Los formatos de fichero en los que se puede generar el informe son:

- *PDF*: Portable Document Format.
- *XLS*: Excel.
- *CSV*: Comma Separated Values.
- *HTML*: Hyper Text Markup Language.

```
roo-gvNIX> web report add --controller ~.web.EntityController --reportName informe
 Updated SRC_MAIN_JAVA/org/gvnix/test/web/EntityController.java
 Created SRC_MAIN_WEBAPP/WEB-INF/reports
 Created SRC_MAIN_WEBAPP/WEB-INF/reports/entity_informe.jrxml
 Created SRC_MAIN_JAVA/org/gvnix/test/web/servlet/view/jasperreports
 Created SRC_MAIN_JAVA/org/gvnix/test/web/servlet/view/jasperreports/
   CustomJasperReportsMultiFormatView.java
 Updated SRC_MAIN_WEBAPP/WEB-INF/spring/jasper-views.xml
 Updated SRC_MAIN_WEBAPP/WEB-INF/i18n/application.properties
 Created SRC_MAIN_WEBAPP/WEB-INF/views/pets/informe.jspx
 Updated SRC_MAIN_WEBAPP/WEB-INF/views/pets/views.xml
 Updated SRC_MAIN_WEBAPP/WEB-INF/i18n/application.properties
 Created SRC_MAIN_JAVA/org/gvnix/test/web/EntityController_Roo_GvNIXReport.aj
 Updated SRC_MAIN_WEBAPP/WEB-INF/views/menu.jspx
```

El comando incluirá la anotación *@GvNIXReports* en la clase controladora e incluira como atributo un array de cadenas en la que cada elemento define un informe y los formatos en los que se podrá generar dicho informe. Así pues, la anotación:

```
@GvNIXReports({ "informe|pdf,xls,csv" })
```

Define un informe cuyo nombre es *informe* y estará disponible en los formatos pdf, xls y csv.

Al lanzar el comando *web report add* o anotar la clase del controlador con *@GvNIXReports* se realizan una serie de modificaciones en el proyecto que darán como resultado la posibilidad de generar un informe desde un formulario de la aplicación. Las modificaciones que realiza el comando sobre el proyecto son:

- Crea la clase Java **CustomJasperReportsMultiFormatView**. La nueva clase se creará en el subpaquete *servlet.view.jasperreports* dentro del paquete donde se encuentre el controlador. Esta

clase se instala solo una vez y en sucesivas ejecuciones se comprueba si existe. Esta clase tiene como principal cometido establecer, de manera dinámica, el nombre del archivo del informe que se genera para su descarga.

- En el archivo `jasper-views.xml` añade la definición de una nueva vista que será el nuevo informe añadido a la aplicación. Se define para ello un bean cuyo id es *<fromBackingObject>_<reportname>* y *CustomJasperReportsMultiFormatView* como clase del bean. Si *formBackingObject* de *@RooWebScaffold* tiene como valor *Entity* y el nombre definido para el informe es *informe*:

```
<bean id="entity_informe"
class="org.gvnix.test.web.servlet.view.jasperreports.CustomJasperReportsMultiFormatView"
p:url="/WEB-INF/reports/entity_informe.jrxml"
p:subReportUrls-ref="subReportUrls"
p:subReportDataKeys-ref="subReportDataKeys" />
```

- Crea un archivo de diseño de informe (jrxml) con carios campos de la clase Entity (siguiendo el ejemplo anterior). El archivo se creará en `src/main/webapp/WEB-INF/reports` con el nombre `entity_informe.jrxml`. Este archivo se referenciará en el *bean* del fichero `jasper-views.xml` desde el atributo *p:url*.

- Crea un formulario web bajo `src/main/webapp/WEB-INF/views/<entity>` con nombre *informe.jspx* siguiendo con el ejemplo anterior. Y define esta nueva vista en el fichero `views.xml` situado en el mismo directorio. Esta página permitirá solicitar la generación del informe y por defecto lo hará incluyendo los 10 primeros registros de la entidad.

- Incluye las etiquetas multidioma que necesita visualizar la generación de informes en *src/main/webapp/WEB-INF/i18n/application.properties* y un nuevo enlace del menú en *src/main/webapp/WEB-INF/views/menu.jspx* para poder acceder a la página de generación del informe.

- Crea el aspecto Java *<controller>_Roo_GvNIXReport.aj* con los métodos que atenderán las peticiones relacionadas con el informe añadido. Se añaden dos métodos por informe:

  - *generate<Reportname>Form(..)*: devuelve la vista del formulario web que permite al usuario seleccionar el formato de salida del informe y solicitar su generación mediante un botón.

  - *generate<Reportname>(..)*: atiende la petición de generación del informe recopilando los datos necesarios que se deben incluir en él. Para ello, por defecto invoca el método *<formBackingObject>.find<formBackingObject>Entries(0, 10)* por lo que se tomarán los 10 primeros registros de la entidad para rellenar el informe.

```
@RequestMapping(value = "/reports/informe", params = "form",
  method = RequestMethod.GET)
public String EntityController.generateInformeForm(Model uiModel) {
    String[] reportFormats =  {"pdf"};
    Collection<String> reportFormatsList = Arrays.asList(reportFormats);
    uiModel.addAttribute("report_formats", reportFormatsList);
    return "users/informe";
}

@RequestMapping(value = "/reports/informe", method = RequestMethod.GET)
public String EntityController.generateInforme(
  @RequestParam(value = "format", required = true) String format, Model uiModel) {
    if ( null == format || format.length() <= 0 ) {
            uiModel.addAttribute("error", "message_format_required");
            return "users/informe";
    }
```

```
    final String REGEX = "(pdf)";
    Pattern pattern = Pattern.compile(REGEX, Pattern.CASE_INSENSITIVE);
    Matcher matcher = pattern.matcher(format);
    if ( !matcher.matches() ) {
            uiModel.addAttribute("error", "message_format_invalid");
            return "users/informe";
    }
    Collection<Entity> dataSource = Entity.findEntityEntries(0, 10);
    if (dataSource.isEmpty()) {
            uiModel.addAttribute("error", "message_emptyresults_noreportgeneration");
            return "users/informe";
    }
    uiModel.addAttribute("format", format);
    uiModel.addAttribute("title", "INFORME");
    uiModel.addAttribute("informeList", dataSource);
    return "entity_informe";
}
```

Nota

El comando web report add puede ejecutarse tantas veces como se desee sobre el mismo controlador. Si el nombre del informe a añadir ya existe previamente, se añadirán los formatos especificados a los ya definidos. Si el nombre dado al nuevo informe definido no existía, se añadirá su definición en la anotación *@GvNIXReports*.

```
@GvNIXReports({ "informe|pdf,xls,csv", "otroinforme|pdf" })
```

Nota

Los valores de la anotación *@GvNIXReports* **no son sensibles** a mayúsculas por tanto si manualmente se establecen los valores de la anotación como:

```
@GvNIXReports({ "informe|pdf", "INFORME|xls,csv" })
```

es equivalente a:

```
@GvNIXReports({ "informe|pdf,xls,csv" })
```

26.4. Futuras versiones

>

- Incrementar la funcionalidad del informe generado, incluyendo la visualización de las relaciones que pueda tener la entidad sobre la que se declara el informe. En la sección de recetas se muestra una forma de mostrar las relaciones de una entidad en el informe mediante el uso de subinformes.

Capítulo 27. Add-on Service

Permite crear servicios locales, servidores de servicios web y clientes de servicios web de forma automática.

27.1. Descripción

Permite crear de forma automática servicios locales, servidores de servicios web a partir de un método del código fuente, crear servidores de servicios web basándose en un archivo de descripción de servicios *WSDL* y crear clientes de servicios web a partir de un archivo de descripción de servicios *WSDL*.

La creación de servicios, del mismo modo que todas las funcionalidades que proporciona el framework, se realiza mediante anotaciones. Para simplificar esta tarea se puede hacer uso de los comandos que ofrece. A medida que se van ejecutando comandos del add-on, se van modificando anotaciones y métodos de la clase Java de forma que acabará conformando el servicio web que se desea publicar.

27.2. Creación de servicios locales

Un servicio local es una clase Java con una serie de métodos que se desea que estén disponibles de forma interna para toda la aplicación. La clase se puede crear con el comando remote service class.

```
roo-gvNIX> remote service class --class ~.service.Service
Created SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
```

La ejecución del comando creará una clase en la ruta especificada con la anotación *@Service* que le proporciona unas características especiales y facilita su uso como servicio local dentro de la aplicación.

```
...
@Service
public class Service {
}
```

De esta forma, el servicio puede ser utilizar desde cualquier otra clase Java definiendo una propiedad del tipo de la clase de servicio con la anotación *@Autowired*.

```
@Autowired
Service service;
```

Una vez creada la clase se pueden crear los métodos que se desea ofrecer en el servicio. Los métodos se pueden crear mediante el comando remote service operation que permite añadir métodos en una clase.

```
roo-gvNIX> remote service operation --name myMethod --service ~.service.Service
  --return java.lang.Long --paramNames param1,param2
  --paramTypes "java.lang.String,java.util.List"
Updated SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
```

El comando anterior creará el método con las características solicitadas en la clase que se le indica como parámetro.

```
public Long myMethod(String param1, List param2) {
  return null;
}
```

Como se puede observar, la implementación del método está vacía y como es obvio deberá ser definida por el desarrollador.

27.3. Creación de servidores desde Java

Permite que los métodos de una clase Java puedan ser accedidos desde el exterior mediante operaciones de un servicio web.

En primer lugar se debe elegir una clase Java que contendrá los métodos que se desea ofrecer al exterior mediante el servicio web. La clase puede ser cualquiera existente previamente o se puede crear una nueva mediante el comando remote service class.

A continuación, se debe elegir los métodos a ofrecer a través del servicio web. Si todavía no existen, se pueden crear mediante el comando remote service operation.

De momento la clase no es accesible mediante un servicio web. Para ello se debe ejecutar el comando remote service define ws.

```
roo-gvNIX> remote service define ws --class ~.service.Service
Updated ROOT/pom.xml [
  added repository http://repository.gvnix.org;
  added dependency org.gvnix:org.gvnix.service.roo.addon:X.Y.Z;
  added plugin org.apache.cxf:cxf-java2ws-plugin:${cxf.version}]
Created SRC_MAIN_WEBAPP/WEB-INF/cxf-appname.xml
Updated SRC_MAIN_WEBAPP/WEB-INF/web.xml
Creating a new class 'Service' to export web service.
Created SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Updated SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Updated ROOT/pom.xml [
  added dependency org.apache.cxf:cxf-rt-core:${cxf.version};
  added dependency org.apache.cxf:cxf-rt-bindings-soap:${cxf.version};
  added dependency org.apache.cxf:cxf-rt-databinding-jaxb:${cxf.version};
  added dependency org.apache.cxf:cxf-rt-frontend-jaxws:${cxf.version};
  added dependency org.apache.cxf:cxf-rt-transports-http:${cxf.version};
  added dependency javax.xml.bind:jaxb-api:2.1;
  added dependency javax.xml.ws:jaxws-api:2.1;
  added property 'cxf.version' = '2.4.2']
Updated SRC_MAIN_WEBAPP/WEB-INF/cxf-petclinic.xml
Updated ROOT/pom.xml
Created SRC_MAIN_JAVA/org/gvnix/test/service/Service_Roo_GvNix_WebService.aj
* New service defined without operations, use 'service export operation' command to add it
* New service can be shown adding '/services/' suffix to your base application URL
```

El comando incluirá la anotación *@GvNIXWebService* en la clase Java indicada. También generará los componentes necesarios para exponer la clase como un servicio web a partir de los atributos de la anotación.

```
...
@GvNIXWebService(name = "ServicePortType",
  targetNamespace = "http://service.test.gvnix.org/",
  serviceName = "Service", address = "Service",
  fullyQualifiedTypeName = "org.gvnix.test.service.Service", exported = false)
public class Service {
}
```

La clase Java será enlazada en la definición del servicio con un *PortType* lo que permitirá la comunicación con la clase desde el exterior mediante los protocolos y tecnologías de servicios web.

```
privileged aspect Service_Roo_GvNix_WebService {
declare @type: Service: @WebService(name = "ServicePortType",
```

```
  targetNamespace = "http://service.test.gvnix.org/",
  serviceName = "Service", portName = "ServicePortType");
declare @type: Service: @SOAPBinding(style = Style.DOCUMENT, use = Use.LITERAL,
  parameterStyle = ParameterStyle.WRAPPED);
}
```

En este momento el servicio web ya está publicado y es accesible, sin embargo no dispondrá de ninguna operación ya que por defecto todos los métodos de la clase publicada son configurados inicialmente como no accesibles por motivos de seguridad. Se podrá hacer accesible cada uno de los métodos de la clase por separado mediante el comando remote service export operation

```
roo-gvNIX> remote service export operation --class ~.service.Service --method method
Updated SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Updated SRC_MAIN_WEBAPP/WEB-INF/cxf-appname.xml
Updated ROOT/pom.xml
Deleted SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Created SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Created SRC_MAIN_JAVA/org/gvnix/test/service/Service_Roo_GvNix_WebService.aj
Updated SRC_MAIN_JAVA/org/gvnix/test/service/Service_Roo_GvNix_WebService.aj
```

El comando anterior añade la anotación *@GvNIXWebMethod* al método, de forma que que se generará el código necesario para que la operación esté disponible en el servicio web.

```
@GvNIXWebMethod(operationName = "method", requestWrapperName = "method",
        requestWrapperNamespace = "http://service.test.gvnix.org/",
        requestWrapperClassName = "org.gvnix.test.service.MethodRequestWrapper",
        resultName = "return", resultNamespace = "http://service.test.gvnix.org/",
        webResultType = Long.class, responseWrapperName = "methodResponse",
        responseWrapperNamespace = "http://service.test.gvnix.org/",
        responseWrapperClassName = "org.gvnix.test.service.MethodResponse")
public Long method(@GvNIXWebParam(name = "param1", type = String.class)
  @WebParam(name = "param1", partName = "parameters", mode = Mode.IN, header = false)
  String param1,
  @GvNIXWebParam(name = "param2", type = List.class)
  @WebParam(name = "param2", partName = "parameters", mode = Mode.IN, header = false)
  List param2) {
    return null;
}
```

Como es obvio, el método no dispone de ningún código en su cuerpo y será responsabilidad del desarrollador el implementar la lógica de negocio que sea necesaria.

27.4. Creación de servidores desde WSDL

Existe otro modo para crear servidores de servicios web. Consiste en ofrecer al exterior un servicio web partiendo de un archivo de contrato de servicio conocido como *WSDL*. Con el comando remote service export ws se proporciona la ruta a un archivo *WSDL* y se generará una réplica del servicio en la que las operaciones son métodos vacios que posteriormente deberán ser personalizadas con la lógica de negocio adecuada.

```
roo-gvNIX> remote service export ws --wsdl ruta
```

La ruta al *WSDL* puede ser un archivo local mediante *file://ruta*, una dirección web mediante *http://ruta* o una dirección web segura mediante *https://ruta*.

Este modo es especialmente cómodo para realizar la migración de servicios que están implementados en otras aplicaciones o tecnologías o que se desean integrar en otra aplicación.

27.5. Creación de clientes

Es posible generar un cliente que permita realizar peticiones a un servicio web remoto existente utilizando para ello el comando remote service import ws. Facilitando la ruta a un *WSDL*, se generará una clase en la ruta que se especifique y que contendrá los métodos del cliente que darán acceso al servicio web remoto.

```
remote service import ws --class org.gvnix.test.service.Service
        --wsdl ruta
Created SRC_MAIN_JAVA/org/gvnix/test/service/
Created SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Updated SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Updated ROOT/pom.xml [
  added dependency org.hibernate.javax.persistence:hibernate-jpa-2.0-api:1.0.0.Final;
  removed dependency org.hibernate.javax.persistence:hibernate-jpa-2.0-api:1.0.1.Final;
  added plugin org.apache.cxf:cxf-codegen-plugin:${cxf.version}]
Updated ROOT/pom.xml
```

El comando dará lugar a la creación, si no existía, de la clase Java y a la inclusión de la anotación *@GvNIXWebServiceProxy* que define el servicio web al que da acceso la clase.

```
...
@Service
@GvNIXWebServiceProxy(wsdlLocation = "ruta")
public class Service {
}
```

La anotación desencadenará la creación en el aspecto Java correspondiente de tantos métodos como operaciones ofrece el servicio web y que enmascaran la comunicación con el sistema remoto.

```
privileged aspect Service_Roo_GvNix_WebServiceProxy {

 public String Service.method(String param1) {
   org.web.service.RemoteService s = new org.web.service.RemoteService();
  org.web.service.RemoteServiceSoap p = s.getRemoteServiceSoap12();
  return p.method(param1);
 }

}
```

Esta clase se podrá utilizar como si de un servicio local se tratase. Para usar esta clase simplemente habrá que crear una propiedad en la clase donde se desea hacer uso de ella y añadirle la anotación *@Autowired*.

```
public class MyClass {
  ...
  @Autowired
  private Service service;
  ...
  public void myMethod() {
    ...
    service.method(..);
    ...
  }
}
```

La ruta al *WSDL* puede ser un archivo local mediante *file://ruta*, una dirección web mediante *http://ruta* o una dirección web segura mediante *https://ruta*.

27.6. Acceso a un WSDL en un servidor seguro

Cuando el *WSDL* se encuentra en un archivo local o en un servidor no seguro como HTTP, se puede acceder directamente sin ningún problema.

En el caso que un *WSDL* se encuentra en un servidor seguro al que se accede mediante HTTPS pueden darse dos posibles escenarios:

1. El certificado del servidor ha sido creado por una **Autoridad de certificación (CA) confiable por la JVM**.

   En este caso, el procesado del *WSDL* funciona de la misma manera que en el caso de acceso por HTTP, transparente para el usuario.

2. El certificado de servidor ha sido creado por un **CA no confiable** (caso de certificados autofirmados).

   Aquí, es responsabilidad del usuario el aceptar las credenciales del servidor para poder acceder al *WSDL*. Recordando como trabajan los navegadores web, cuando se intenta acceder a un recurso seguro, si el certificado del servidor no ha sido creado por un CA que se encuentre entre la lista de CAs conocida (Verisign, Thwate, Equfax, ...), el navegador muestra un mensaje de advertencia y pregunta si se desea confiar en la identidad del servidor. Cada usuario decide aceptar o no el certificado.

   Para simplificar el proceso de importación de servicios web, se intenta hacer esta tarea de manera transparente para el usuario.

   Para ello, se manipula el almacén de certificados de la máquina virtual Java siempre que sea posible. Esto es:

   a. Existe el archivo de keystore en el directorio donde está instalado Java, por ejemplo en sistemas linux *$JAVA_HOME/jre/lib/security/cacerts* y es modificable por el usuario del sistema que está ejecutando gvNIX.

   b. La contraseña del *keystore* es changeit (la contraseña por defecto del keystore).

   Si se puede modificar el keystore, se importarán los certificados implicados en la autenticación del servidor y entonces se podrá acceder al *WSDL* requerido para generar el cliente del WS.

   Al mismo tiempo, se crea en *src/main/resources* una copia del almacén de certificados en el archivo gvnix-cacerts y una copia local de los certificados importados con el nombre <*servidor*>-<*indice_certificado*>.*cer*. De esta forma, los certificados pueden ser distribuidos con el resto de archivos del proyecto para que puedan ser instalados en otros entornos.

   Si no puede modificar el keystore de la JVM, porque no se cumplen alguna de las 2 condiciones enumeradas anteriormente, de todas formas realizará la copia de los certificados necesarios para que sea el usuario quien instale los mismos en el almacén de certificados de su JVM. Para ello se puede hacer uso de la herramienta keytool (distribuida también con el JDK). La operación de importar los certificados equivale a la aceptación que se hace con el navegador.

27.6.1. Creación de clientes con firma

El add-on permite añadir una firma digital a las peticiones realizadas a un servicio web externo que se encuentra importado en la aplicación.

Actualmente esta opción sólo está disponible para los servicio web que utilicen la librería Axis (RPC/Encoded). En futuras versiones se añadirá esta misma opción para servicios que utilicen la librería CXF (Document/Literal) y otra operaciones relacionadas con la seguridad en servicios web.

Para ello se debe disponer de:

1. Servicio web importado en la aplicación.
2. Fichero *pkc12* con el certificado a usar para firmar la petición.
3. Contraseña del certificado.
4. Alias a usar con el certificado

Disponiendo de esto elementos, se puede hacer uso del comando remote service security ws para añadir la firma en las peticiones del cliente.

```
roo-gvNIX> remote service security ws --class org.gvnix.test.service.Service
          --alias alias --certificate /tmp/certificate.p12 --password clave
Created ROOT/src/main/resources/org/gvnix/test/service
Created ROOT/src/main/resources/org/gvnix/test/service/certificate.p12
Updated SRC_MAIN_JAVA/org/gvnix/test/service/Service.java
Created ROOT/src/main/resources/client-config.wsdd
Created ROOT/src/main/resources/org/gvnix/test/service/ServiceSecurity.properties
Updated ROOT/pom.xml [added dependency org.apache.ws.security:wss4j:1.5.11]
Updated ROOT/src/main/resources/client-config.wsdd
Created SRC_MAIN_JAVA/org/gvnix/test/service/Service_Roo_GvNIX_WebSecurity.aj
```

Se crearán dos ficheros con distintos parámetros de configuración en `src/main/resources/org/gvnix/test/service/ServiceSecurity.properties` y `src/main/resources/client-config.wsdd`. El primero contiene los parámetros introducidos para la configuración de la seguridad y el segundo la configuración para axis.

También dará lugar a la inclusión de la anotación *@GvNIXWebServiceSecurity*.

```
...
@GvNIXWebServiceSecurity
public class Service {
}
```

La anotación anterior dará lugar a la generación del aspecto Java correspondiente que establece la clave de acceso al certificado que se encuentra en el fichero `ServiceSecurity.properties`.

```
privileged aspect Service_Roo_GvNIX_WebSecurity {

    declare parents: Service implements CallbackHandler;

    public Service.new() {
        super();
    }

    public void Service.handle(Callback[] callbacks)
      throws IOException, UnsupportedCallbackException {
        final String propPath = "org/gvnix/test/service/ServiceSecurity.properties";
```

```
        final String propKey = "org.apache.ws.security.crypto.merlin.keystore.password";
        try {
// Get class loader to get file from project
            ClassLoader classLoader = Thread.currentThread().getContextClassLoader();
            java.io.File file = new java.io.File(classLoader.getResource(propPath).toURI());
            if (file != null && file.exists()) {
// Load properties
                java.util.Properties properties = new java.util.Properties();
                java.io.FileInputStream ins = null;
                try {
                    ins = new java.io.FileInputStream(file);
                    properties.load(ins);
                } finally {
                    if (ins != null) {
                        ins.close();
                    }
                }
                String value = properties.getProperty(propKey);
                if (value != null) {
                    ((org.apache.ws.security.WSPasswordCallback) callbacks[0]).setPassword(value);
                } else {
                    throw new IOException("Property ".concat(propKey).concat(" not exists"));
                }
            } else {
                throw new IOException("File ".concat(propPath).concat(" not exists"));
            }
        } catch (java.net.URISyntaxException e) {
            throw new IOException("Problem getting ".concat(propPath).concat(" file"),e);
        }
    }
}
```

Para realizar cambios en los parámetros de firma del servicio es posible ejecutar de nuevo el comando remote service security ws con los nuevos datos sobre la misma clase (opción recomendada) o modificar los ficheros antes mencionados a mano.

27.7. Listar los servicios

El comando remote service list operation muestra los métodos existentes en una clase que están disponibles para ser ofrecidos al exterior siempre y cuando la clase esté definida como un servicio web.

```
roo-gvNIX> remote service list operation --class ~.service.Service
Method list to export as web service operation in '~.service.Service':
    * myMethod2
```

El comando remote service ws list permite obtener un listado de los servicios ofrecidos al exterior o aquellos servicios externos que son utilizados en la aplicación.

El resultado es una salida como esta:

```
roo-gvNIX> remote service ws list
Services                         exported    imported
-------------------------------  ---------   ----------
org.gvnix.test.service.Service1      X
org.gvnix.test.service.Service2                 X
org.gvnix.test.service.Service3                 X
```

Capítulo 28. Add-on Web MVC i18n

Permite añadir soporte para nuevos idiomas en el proyecto.

28.1. Descripción

Este add-on permite añadir en el proyecto soporte para nuevas lenguas utilizando para ello una infraestructura común para la inclusión de nuevos idiomas. Al incluir un nuevo idioma, se añaden en la aplicación de forma automática los textos traducidos a dicho idioma.

Los idiomas soportados son: de (Alemán), en (Inglés), es (Español), it (Italiano), nl (Holandés), sv (Finlandés), **ca (Valenciano/Catalán)**. Este último es añadido por gvNIX utilizando la base del add-on de idiomas de su núcleo Spring Roo.

28.2. Instalación de un idioma

Para instalar uno de los idiomas soportados hay que ejecutar el comando web mvc install language y proporcionar como parámetro *code* el código del idioma deseado.

```
roo-gvNIX> web mvc install language --code ca
 Created SRC_MAIN_WEBAPP/WEB-INF/i18n/messages_ca.properties
 Created SRC_MAIN_WEBAPP/images/ca.png
 Updated SRC_MAIN_WEBAPP/WEB-INF/views/footer.jspx
```

Este comando instala un fichero de propiedades en el proyecto con la traducción de las etiquetas necesarias para el idioma que se requiera. Por ejemplo, para el idioma valenciano/catalán se creará el fichero *src/main/webapp/WEB-INF/i18n/messages_ca.properties*.

```
#menu
 global_menu_new=Crear nou {0}
 global_menu_list=Llistar {0}
 global_menu_find=Cercar per {0}
 ...
```

También realiza las modificaciones necesarias para poder solicitar el cambio de idioma desde la interfaz visual, lo que implica la inclusión de una imagen con la bandera del idioma solicitado y la modificación del pie de página para mostrar el cambio a dicho idioma. Esto se realiza en el fichero *src/main/webapp/WEB-INF/views/footer.jspx* mediante el tag *<util:language>* que será el encargado de incluir la imagen y generar el enlace para el cambio de idioma.

```
<util:language label="Valencian_Catalan" locale="ca"/>
```

Para mas información sobre este comando ver el apéndice de comandos del add-on.

28.3. Futuras versiones

Mejoras a incluir en futuras versiones del add-on.

- Incluir el soporte para el resto de lenguas cooficiales del estado español, es decir, Gallego y Euskera. Se deja abierta la posibilidad de añadir idiomas de otros estados que pudiesen ser de utilidad.

Chapter 29. Add-on Dynamic Configuration

29.1. Descripción

Este add-on pretende simplificar la gestión de distintos valores de configuración por entorno. Cuando se utiliza Maven como gestor del ciclo de desarrollo del proyecto existe la posibilidad de utilizar perfiles para definir los distintos entornos de ejecución que tendrá la aplicación (desarrollo, pruebas, integración, preproducción, producción, ...).

A continuación se muestra un ejemplo de parte de la configuración necesaria para la definición de perfiles en el fichero *pom.xml*.

```
<profiles>
  <profile>
    <id>nombre-entorno</id>
    <properties>
      <nombre.propiedad>valor</nombre.propiedad>
      ...
    </properties>
  </profile>
</profiles>
```

Esta funcionalidad permite configurar el comportamiento de la aplicación de manera distinta según donde vaya a ser ejecutada. Por ejemplo, la configuracíón de la conexión con la base de datos de la aplicación posiblemente será una URL distinta según se esté trabajando en un entorno de desarrollo o un entorno de producción.

Para esto, en la sección <*properties*> de cada sección <*profile*> se pueden declarar tantas propiedades como se desee de forma que al empaquetar la aplicación (habitualmente en formato WAR), se reemplace el valor de estas propiedades en los ficheros adecuados. Esto implica que en distintos archivos del proyecto, como puede ser *persistence.xml*, empiecen a aparecer definidas variables con el formato *${nombre.propiedad}*. Esta circunstancia, a la larga, puede complicar la comprensión de la configuración del proyecto o dificultar su gestión. Por ejemplo, podría aparecer la siguiente modificación en el fichero *src/main/resources/META-INF/persistence.xml*:

```
<property name="hibernate.hbm2ddl.auto" value="${hibernate.hbm2ddl.auto}"/>
```

Esta circunstancia, a la larga, puede complicar la comprensión de la configuración del proyecto o dificultar su gestión.

29.2. Funcionalidad

Vista la problemática descrita en el punto anterior, el add-on dynamic configuration pretende simplificar esta gestión. Permite definir distintas configuraciones de manera rápida y simple. Además, es independiente del sistema de gestión del ciclo de vida del proyecto, ya que aunque actualmente las configuraciones solo se pueden exportar a perfiles Maven, el add-on es lo suficientemente general como para poder exportar a formatos Ant, etc.

A medida que se van ejecutando comandos del add-on se irá modificando el archivo *src/main/resources/dynamic-configuration.xml* que almacena la información sobre las distintas configuraciones definidas hasta el momento.

La creación de una nueva configuración se realiza con el comando configuration create que tiene un único parámetro obligatorio (*name*) que define el nombre para la nueva configuración.

```
roo-gvNIX> script --file clinic.roo
...
roo-gvNIX> configuration create --name dev
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Configuration created with currently available properties
First created configuration set as default
(use 'configuration property add' to make a property available for all configurations)
roo-gvNIX> configuration create --name pro
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Configuration created with currently available properties
(use 'configuration property add' to make a property available for all configurations)
```

Resaltar que la creación de las configuraciones no es obligatorio realizarlas al inicio y podrán realizarse en cualquier momento.

A partir de este momento es posible añadir nuevas propiedades para que estén disponibles para su gestión desde todas las configuraciones con configuration property add. Las propiedades inicialmente tomarán el valor que tengan definido en el proyecto.

```
roo-gvNIX> configuration property add --name database.url
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property available for all configurations
(use 'configuration property value' to set property new values)
(use 'configuration property undefined' to set property with no values)
roo-gvNIX> configuration property add --name hibernate.hbm2ddl.auto
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property available for all configurations
(use 'configuration property value' to set property new values)
(use 'configuration property undefined' to set property with no values)
roo-gvNIX> configuration property add --name log4j.rootLogger
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property available for all configurations
(use 'configuration property value' to set property new values)
(use 'configuration property undefined' to set property with no values)
roo-gvNIX> configuration property add --name database.password
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property available for all configurations
(use 'configuration property value' to set property new values)
(use 'configuration property undefined' to set property with no values)
```

A continuación, se podrán modificar los valores de cada propiedad con configuration property value.

```
roo-gvNIX> configuration property value --configuration dev --property database.url
  --value jdbc:hsqldb:mem:mydevdb
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property value seted
(use 'configuration list' to show configurations and their properties)
roo-gvNIX> configuration property value --configuration pro
  --property database.url --value jdbc:hsqldb:file:myprodb
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property value seted
(use 'configuration list' to show configurations and their properties)
roo-gvNIX> configuration property value --configuration pro
  --property hibernate.hbm2ddl.auto --value update
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property value seted
(use 'configuration list' to show configurations and their properties)
roo-gvNIX> configuration property value --configuration pro
  --property log4j.rootLogger --value "ERROR, stdout"
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
```

```
Property value seted
(use 'configuration list' to show configurations and their properties)
```

En algunos casos, por seguridad, puede ser necesario que el valor de una propiedad no esté almacenado en el proyecto, para ello puede utilizarse el comando configuration property undefined de forma que el valor se tendrá que proporcionar al empaquetar la aplicación con Maven como un parámetro mediante el modificador *-D nombre=valor*.

```
roo-gvNIX> configuration property undefined --configuration pro
  --property database.password
Updated SRC_MAIN_RESOURCES/dynamic-configuration.xml
Property value undefined
(use '-D propname=propvalue' on maven commands to set the property value)
(use 'configuration list' to show configurations and their properties)
```

Si se desea ver las distintas configuraciones que creadas, sus propiedades y los valores definidos para cada una de ellas utilizar el comando configuration list.

```
roo-gvNIX> configuration list
      (Active)      dev
----------------------------------------
 * Database Connection Properties
   - database.url = "jdbc:hsqldb:mem:mydevdb"
   - database.password = ""
 * Persistence Property Attributes XML
   - hibernate.hbm2ddl.auto = "create"
 * Logging Service Properties
   - log4j.rootLogger = "INFO, stdout"
                   pro
----------------------------------------
 * Database Connection Properties
   - database.url = "jdbc:hsqldb:file:myprodb"
   - database.password = (UNDEFINED)
 * Persistence Property Attributes XML
   - hibernate.hbm2ddl.auto = "update"
 * Logging Service Properties
   - log4j.rootLogger = "ERROR, stdout"
(use 'configuration export' to write configurations into the project)
```

Para escribir las configuraciones actuales en el proyecto se debe utilizar el comando configuration export. Es muy importante destacar que hasta que no se haya ejecutado este comando, las configuraciones no serán escritas en los ficheros destino y por lo tanto hasta ese momento no se podrán utilizar las configuraciones.

```
roo-gvNIX> configuration export
Updated ROOT/pom.xml
Updated ROOT/pom.xml
Updated SRC_MAIN_RESOURCES/META-INF/spring/database.properties
Updated SRC_MAIN_RESOURCES/log4j.properties
Updated SRC_MAIN_RESOURCES/META-INF/persistence.xml
Updated SRC_MAIN_RESOURCES/META-INF/spring/database.properties
Updated SRC_MAIN_RESOURCES/log4j.properties
Updated SRC_MAIN_RESOURCES/META-INF/persistence.xml
Configurations exported into project
(use '-P name' on maven commands to use a configuration)
(use 'configuration create' to define a new configuration)
```

Una vez exportadas las configuraciones, pueden utilizarse como perfiles desde Maven mediante el modificador *-P nombre* utilizando como nombre el valor que se definió para la configuración con el comando configuration create. Por ejemplo, al empaquetar la aplicación para desplegarla en uno u otro

entorno, debe especificarse el nombre de la configuración y opcionalmente el valor de los parámetros que se crearon como indefinidos en la configuración.

```
shell> mvn clean package -P dev
... (Empaquetando la aplicación con la configuración llamada dev) ...
shell> mvn clean package -P pro -D database.password=mypassword
... (Empaquetando la aplicación con la configuración llamada pro) ...
... (Se utilizará como clave de acceso a la base de datos "mypassword") ...
```

También es posible elegir la configuración (perfil) a utilizar desde Eclipse/STS desde las propiedades del proyecto accediendo a la opción *Maven* del menú.

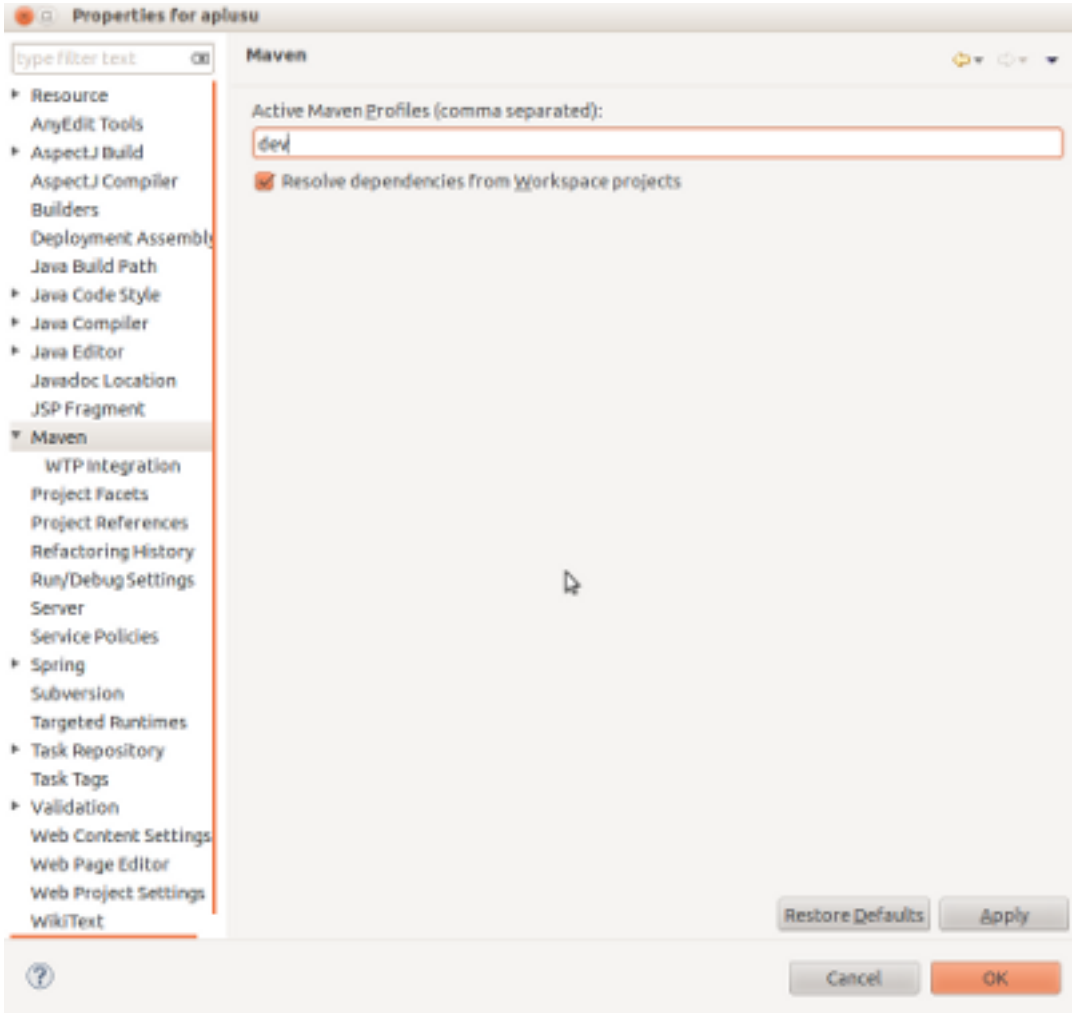

Figure 29.1. Selección del perfil Maven desde Eclipse

29.3. Mejoras de renimiento

Se pueden definir algunos valores óptimos de rendimiento definidos en la sección Patrones de conversión óptimos en el log.

29.4. Futuras versiones

- Incluir nuevas propiedades a la lista de propiedades disponibles para las configuraciones. O por contra, permitir cierta libertad para añadir cualquier propiedad del proyecto como configurable.
- Posibilidad de incluir otro tipo de elementos en las configuraciones como, por ejemplo, distintas dependencias.

- Definir plantillas de configuración de modo que establezcan unos valores estándar que se consideren adecuados para una determinada configuración. Por ejemplo, establecer el formato de los logs en producción para reducir el consumo de recursos.

Capítulo 30. Add-on Web MVC Binding

30.1. Descripción

El add-on permite registrar ciertos editores de propiedades en Spring MVC para uno o todos los controladores del proyecto.

Los editores son una pieza más de la arquitectura de Spring MVC. Se encargan de controlar el modo en que las cadenas enviadas por el navegador cliente, por ejemplo desde un formulario, se convierten en objetos al llegar al servidor.

Para obtener más información sobre los editores de propiedades ver la sección de conversión de beans de la documentación de Spring.

El comando web mvc binding stringTrimmer permite registrar un editor especialmente útil. Este editor elimina los espacios en blanco al principio y al final de las cadenas y además permite transformar las cadenas vacías en valores nulos.

Este editor es especialmente interesante al realizar mantenimientos de valores almacenados en bases de datos. Por defecto, cuando se deja vacío un valor de un formulario y está relacionado con una columna de base de datos del tipo cadena, al almacenar se escribe una cadena vacía. En estos casos puede suele ser más interesante almacenar un valor nulo.

```
roo-gvNIX> web mvc binding stringTrimmer --class ~.web.EntityController
Updated SRC_MAIN_JAVA/org/gvnix/test/web/EntityController.java
Updated ROOT/pom.xml added dependency org.gvnix:org.gvnix.web.mvc.binding.roo.addon:X.Y.Z
Created SRC_MAIN_JAVA/org/gvnix/test/web/EntityController_Roo_GvNIXStringTrimmerBinder.aj
```

Al ejecutar el comando, el add-on incluirá la anotación @GvNIXStringTrimmerBinder en la clase controladora especificada en el parámetro *--class* del comando o en todas las clases controladoras del proyecto si no se especificó dicho parámetro. El parámetro *--emptyAsNull* por defecto toma el valor *true* indicando que las cadenas vacías deben convertirse en valores nulos.

```
...
@GvNIXStringTrimmerBinder(emptyAsNull = true)
public class EntityController {
}
```

Esto generará en un aspecto Java propio todo el código necesario para registrar el editor, representado en Spring MVC mediante la clase StringTrimmerEditor.

```
privileged aspect EntityController_Roo_GvNIXStringTrimmerBinder {

    @InitBinder
    public void EntityController.initStringTrimmerBinder(WebDataBinder binder) {
        binder.registerCustomEditor(String.class, new StringTrimmerEditor(true));
    }

}
```

Es interesante resaltar que este método puede ser reutilizado haciendo push-in para registrar cualquier editor asociado a un tipo de datos adecuado, como se hace en el ejemplo anterior asociando *String* con *StringTrimmerEditor*.

Para obtener mas detalles sobre el comando ver el apéndice de comandos de este add-on.

30.2. Futuras versiones

- Para futuras versiones este add-on permitirá registrar más editores de entre los enumerados en la Tabla 5.2 de la documentación de Spring.

Capítulo 31. Add-on Web MVC Fancytree

Implementación de vistas tipo árbol en el proyecto gvNIX.

31.1. Descripción

Este add-on permite añadir componentes visuales de tipo "árbol" en las aplicaciones gvNIX. Con este componente podremos disponer de representaciones interactivas e intuitivas, fácilmente extensibles, personalizables y editables.

Las principales características de este widget son:

Representación de datos
: Permite mostrar una estructura en forma de árbol con los datos introducidos o seleccionados por el usuario. Esta estructura podrá contar con tantos niveles como se desee.

Edición de datos
: Se permite editar los datos mostrados en la representación a través del widget: Renombrar, eliminar, crear nuevos hijos, etc.

Arrastrar y soltar (Drag and Drop)
: Además de editar los datos a través del árbol, se permite desplazarlos a un nuevo nivel o posición arrastrándolo con el ratón dentro de la estructura del árbol. Esta característica es personalizable, permitiendo añadirle opciones y restricciones.

Cambiar apariencia
: Se permite cambiar la apariencia (skin) de la visualización, entre las siete diferentes que se instalarán en el proyecto o utilizar una apariencia personalizada.

Personalización de componentes (botones, checkbox, menú contextual)
: El desarrollador puede añadir objetos como casillas de selección y botones HTML al componente árbol, e implementar las funciones de estos componentes como se desee. Además se permite añadir o modificar los elementos y sus funciones en el menú contextual para la edición del árbol. Estas y otras características de personalización del árbol se encuentran en el fichero WEB-INF/tags/fancytree/tree.tagx que se añade al proyecto al instalar el addon Fancytree.

31.2. Instalación del soporte para Fancytree

Para instalar esta funcionalidad hay que utilizar el comando web mvc fancytree setup,el cual solo estará disponible después de instalar el add-on JQuery en nuestro proyecto.

Este comando añadirá los siguientes elementos:

1. Las hojas de estilo necesarias para mostrar el componente visual, con siete skins diferentes. Estos ficheros se instalarán en `webapp/styles/fancytree`

2. Los archivos JavaScript de Fancytree y el plugin creado por el equipo de gvNIX para gestionar los datos que se muestran a través del Controller y utilizar las funcionalidades disponibles (Personalización de componentes, Drag and Drop, etc.)

3. `Tagx` necesarios utilizados en las páginas para el pintado y personalización del widgedt almacenados en `WEB-INF/tags/fancytree`.

4. Actualiza el fichero `WEB-INF/tags/util/load-scripts.tagx` para que las páginas puedan localizar los recursos de hojas de estilo y javaScript requeridos.

5. La dependencia a este add-on y librerías de utilidades utilizadas para el manejo y gestión de los datos.

31.3. Añadiendo plantilla para inserción de datos de un árbol

Para crear un menú de tipo árbol en un proyecto gvNIX, ejecutaremos el comando web mvc fancytree add show, el cual añadirá al controller seleccionado la plantilla del método en el que se añadirán y servirán por petición Ajax. los datos y las propiedades del componente visual tipo árbol.

Ejemplo:

```
(treeUtils.isRootNode(id)){
  List<TreeNode>  myTree  = new ArrayList<TreeNode>();
  TreeNode node1  = new TreeNode(“Node1”, true);
  node1.addChild(new TreeNode(“subnode1”));
  node1.addChild(new TreeNode(“subnode2”));
  node1.addChild(new TreeNode(“subnode3”));
  myTree.add(node1);
}else{
  Node node2  = new TreeNode(“Node2”);
  myTree.add(node2);
}
```

31.4. Añadiendo plantilla para inserción y edición de datos de un árbol

Para crear un menú de tipo árbol editable, ejecutaremos el comando web mvc fancytree add edit, el cual añadirña al controller seleccionado las plantillas de los métodos necesarios para añadir y mostrar datos, editarlos, eliminarlos o crear nuevos elementos.

Si se ha especificado el parámetro “page” en el comando, el desarrollador puede añadir propiedades a la página .jspx generada para añadir añadir botones y casillas de selección, activar o desactivar características, utilizar funciones o componentes personalizados, etc.

Ejemplo:

```
 <fancytree:tree id="ps_com_springsource_petclinic_web_mytree" checkbox="true"
editable="true" path="/owners/mytree" allowDragAndDrop="true"
 contextMenu="true" z="user-managed" />
```

31.5. Actualizando componentes fancytree

Para mantener actualizados los componentes importados mediante el comando de instalación, podemos ejecutar el comando web mvc fancytree update tags.

31.6. Apendice de comandos

Para ver con mas detalle el comando que proporciona el add-on consultar la sección de comandos del add-on Fancytree.

Parte IV. Recetas de desarrollo

En esta parte se verán distintas guías para realizar ciertas modificaciones sobre el código generado para adaptarlo a distintas necesidades.

Capítulo 32. Recetas

En este capítulo se verá como realizar algunas modificaciones sobre el código generado para adaptar la aplicación a distintos requerimientos habituales dada la experiencia de uso del framework.

32.1. Repositorios Maven

Cuando una funcionalidad generada por el framework requiere de librerías externas, el add-on correspondiente registrará el repositorio de artefactos Maven necesario en el fichero *pom.xml*, si todavía no lo estaba.

No es habitual, pero en el caso de ser necesario añadir algún repositorio adicional al proyecto puede realizarse mediante el comando de la consola maven repository add . El siguiente ejemplo define el repositorio Maven interno de la Consejería de Infraestructuras, Transporte y Medio Ambiente. Este repositorio es accesible únicamente desde la red interna de la Consejería.

```
roo-gvNIX> maven repository add --id citma-maven --name "CITMA Maven Repository"
  --url http://benigno.coput.gva.es:8081/nexus/content/groups/public
Updated ROOT/pom.xml [
  added repository http://benigno.coput.gva.es:8081/nexus/content/groups/public]
```

El comando modificará el fichero pom.xml y en la sección <*repositories*> añadirá la definición del repositorio indicado.

```
<repositories>
    ...
    <repository>
        <id>citma-maven</id>
        <url>http://benigno.coput.gva.es:8081/nexus/content/groups/public</url>
        <name>CITMA Maven Repository</name>
    </repository>
    ...
</repositories>
```

En el lado opuesto, también es posible eliminar repositorios Maven del proyecto mediante el comando maven repository remove .

32.2. Desarrollo de buscadores con gran cantidad de campos

El add-on de los buscadores (finder) tiene una limitación impuesta por el sistema operativo Windows en el tamaño máximo del nombre de archivos. La limitación del tamaño máximo del nombre de archivo limita al comando *finder add* a generar nombres de buscadores que generen paths a archivos jspx de tamaño menor de 244 caracteres.

Existe una forma para generar buscadores con un tamaño superior al de la limitación impuesta. Consiste en dividir el buscador necesario en varios más pequeños, parciales de unos pocos campos cada uno, y al final unir el código que se genera para cada uno de ellos en un unico método. Esto implica tener que modificar el código que se ha generado para el buscador en la entidad, el controlador y la jspx de la siguiente forma:

- En el archivo Java de la entidad para la que se está generando el buscador crear un método con el nombre, por ejemplo, *findByTodo* y cuyos parámetros de entrada serán todos los parámetros que reciben los métodos de los buscadores parciales que se hayan generado. Del mismo modo, el cuerpo de este nuevo método es la unión de la implementación de todos los buscadores parciales. Es posible

modificar el código, según se explica en el siguiente apartado, de forma que los parámetros sean opcionales.

- De manera similar, en el fichero Java del controlador de la entidad crear dos métodos: uno *findByTodoForm* que devolverá la vista del formulario del buscador y otro *findByTodo* que responderá a las peticiones que lleguen desde el formulario del buscador. Este segundo método, al igual que en el método creado en la entidad, deberá tener como parámetros la unión de todos parámetros de los métodos de los buscadores parciales en el controlador. El cuerpo de este método deberá invocar al método *findByTodo* creado en la entidad. Para definir la anotación @RequestMapping de estos dos métodos se recomienda utilizar como guía los buscadores parciales ya existentes.

- Finalmente, crear una página jspx (por ejemplo, con nombre findByTodo.jspx) en el directorio WEB-INF/views/<entidad> que contendrá el formulario del buscador. Esta página debe incluir todos los campos que hayan en el resto de páginas jspx de los buscadores parciales. Recordar, que esta vista se ha de definir también en el fichero views.xml que hay en la misma carpeta y que esta vista es la que devuelve el método findByTodoForm creada anteriormente en el controlador.

Con todo lo anterior se ha descrito un proceso manual que permite montar buscadores con cualquier cantidad de campos.

32.3. Campos opcionales en los buscadores

El comportamiento que implementa automáticamente la generación de buscadores es hacer todos los campos de busqueda obligatorios. Por lo tanto, si no se proporciona alguno de los parámetros se generará una excepción indicando la falta de un parámetro obligatorio. En ocasiones no es lo deseado, es decir, puede necesario que los parámetros no informados simplemente no se incluyan en la consulta si no se especifican. Para ello se debe modificar el código del método *finderByXxxx* declarado en el aspecto Java **_Roo_Finder.aj*. A continuación se ejemplifica como hacer este cambio.

Hacer *push-in* del método tal y como se comenta en la sección sobre modificación del código generado y llevarlo al fichero Java de la entidad. Una vez allí modificarlo para que no requiera los campos y así construir la consulta considerando tan solo los parámetros que lleguen informados:

```
public static TypedQuery<Owner> findOwnersByLastNameLikeAndCityLike(
      String lastName, String city) {
  StringBuilder query = new StringBuilder("SELECT o FROM Owner AS o");
  StringBuilder whereClause = new StringBuilder();

  // if (lastName == null || lastName.length() == 0)
  // throw new IllegalArgumentException("The lastName argument is required");
  if (lastName != null && lastName.length() > 0) {
    lastName = lastName.replace('*', '%');
    if (lastName.charAt(0) != '%') {
        lastName = "%" + lastName;
    }
    if (lastName.charAt(lastName.length() - 1) != '%') {
        lastName = lastName + "%";
    }
    whereClause.append(" WHERE LOWER(o.lastName) LIKE LOWER(:lastName)");
  }

  // if (city == null || city.length() == 0)
  // throw new IllegalArgumentException("The city argument is required");
  if (city != null && city.length() > 0) {
    city = city.replace('*', '%');
    if (city.charAt(0) != '%') {
```

```
        city = "%" + city;
    }
    if (city.charAt(city.length() - 1) != '%') {
        city = city + "%";
    }
    if (whereClause.length() > 0) {
      whereClause.append(" AND LOWER(o.city) LIKE LOWER(:city)");
    } else {
      whereClause.append(" WHERE LOWER(o.city) LIKE LOWER(:city)");
    }
  }
  query.append(whereClause);

  EntityManager em = Owner.entityManager();
  TypedQuery<Owner> q = em.createQuery(query.toString(), Owner.class);
  if (lastName != null && lastName.length() > 0) {
    q.setParameter("lastName", lastName);
  }
  if (city != null && city.length() > 0) {
    q.setParameter("city", city);
  }
  return q;
}
```

Se han marcado en negrita las líneas de código modificadas. Como puede observarse, se construye la consulta de forma dinámica de forma que se añaden los distintos filtros al *where* de la consulta solo si el campo asociado al filtro tiene algún valor y se elimina la generación de excepciones al detectar alguno de dichos campos como vacío.

En este ejemplo se ha construido la consulta usando un StringBuilder en el que se genera la consulta deseada, pero se podría utilizar cualquier otra tecnología como por ejemplo Hibernate Criteria Builder.

Como ya se ha comentado, el código generado fuerza que todos los parámetros de búsqueda sean obligatorios, esto implica también al formulario situadao en la página jspx. Tal como se genera, si alguno de los campos no se rellena, no dejará enviar la petición. Se debe editar la jspx y modificar el atributo *required* de los campos *field* y establecer su valor a *false*, no olvidar cambiar el valor del atributo *z* a "*user-managed*" para indicar que este campo ha sido personalizado.

```
<field:input disableFormBinding="true" field="city"
      id="f_com_springsource_petclinic_domain_Owner_city" max="30"
          required="false" z="user-managed"/>
```

En los buscadores en los que uno de los parámetros sea una entidad del modelo, es decir una relación con otra entidad del modelo, será necesario realizar un cambio adicional. En estos casos se visualiza un selector desplegable, mediante el componente *select.tagx*. Sin embargo, este selector tiene el inconveniente de que no incluye una opción vacía entre las disponibles, con lo que obliga a elegir siempre algún valor. Un ejemplo de este caso se encuentra en el buscador findPetsByOwner generado para la entidad Pet en el ejemplo de *clinic.roo* .

La forma de tratar estas peticiones en Spring MVC es intentar cargar el registro asociado con el valor del selector que llega como parámetro. Por tanto si llega como parámetro un dato que no concuerda con ningún registro de la entidad parámetro, usará null como valor. Como se ha comentado, el componente *select.tagx* no ofrece una opción vacía. gvNIX en su Add-on Web MVC Pattern incluye un componente *select-withempty.tagx* que sí lo ofrece. Mediante el comando *web mvc pattern setup* se instalan los componentes de MVC Pattern entre los que se incluye el selector con opción vacía. Entonces es posible modificar la jspx del buscador findPetsByOwner para que utilice *select-withempty* en lugar del original de la siguiente forma:

```
<div xmlns:field="urn:jsptagdir:/WEB-INF/tags/form/fields" ...
      xmlns:pattern-field="urn:jsptagdir:/WEB-INF/tags/pattern/form/fields"
      version="2.0">
  <jsp:directive.page contentType="text/html;charset=UTF-8"/>
  <jsp:output omit-xml-declaration="yes"/>
  <form:find finderName="ByOwner" id="ff_com_springsource_petclinic_domain_Pet"
    path="/pets" z="ThIGSmua6R7WM6q2P1BpNC3zMi4=">
      <pattern-field:select-withempty disableFormBinding="true" field="owner"
        emptyValueEnable="true" emptyValue="0"
        id="f_com_springsource_petclinic_domain_Pet_owner" itemValue="id"
        items="${owners}" path="/owners" required="false" z="user-managed"/>
  </form:find>
</div>
```

El componente select-withempty.tagx tiene los mismos atributos que el select.tagx y otros tres adicionales:

- *emptyValueEnable*: Valor booleano que habilita la opción vacía en el selector.
- *emtpyValue*: Valor a dar para la opción vacía del selector. Si se corresponde con una entidad, este valor no debería corresponder con ningún identificador existente para esa entidad, de esta forma el valor que llega al buscador es nulo.
- *emptyLabel*: Descripción a mostrar para la opción vacía del selector.

Con este cambio y con el método findPetsByOwner modificado para que no sea requerido el parámetro Owner, de la misma forma que se explicado antes, se consigue realizar búsquedas por parámetros opcionales sea cual sea el tipo del parámetro.

32.4. Instalar fuentes de letra para los informes

Las fuentes instaladas automáticamente proporcionan soporte para la inclusión de texto en negrita y cursiva. El desarrollador debe instalar cualquier otro tipo de fuente que se utilice en los informes que diseñe. Para ello, basta con ampliar las definiciones *fontFamily* realizadas en el archivo *gvnix_reportfonts.xml* y copiar los archivos de fuente en la carpeta *jasperfonts*. La inclusión de nuevas fuentes se puede realizar utilizando una utilidad que ofrece iReport. Desde el menú *Herramientas* > *Opciones* > *Pestaña Fonts* se accederá a la siguiente pantalla:

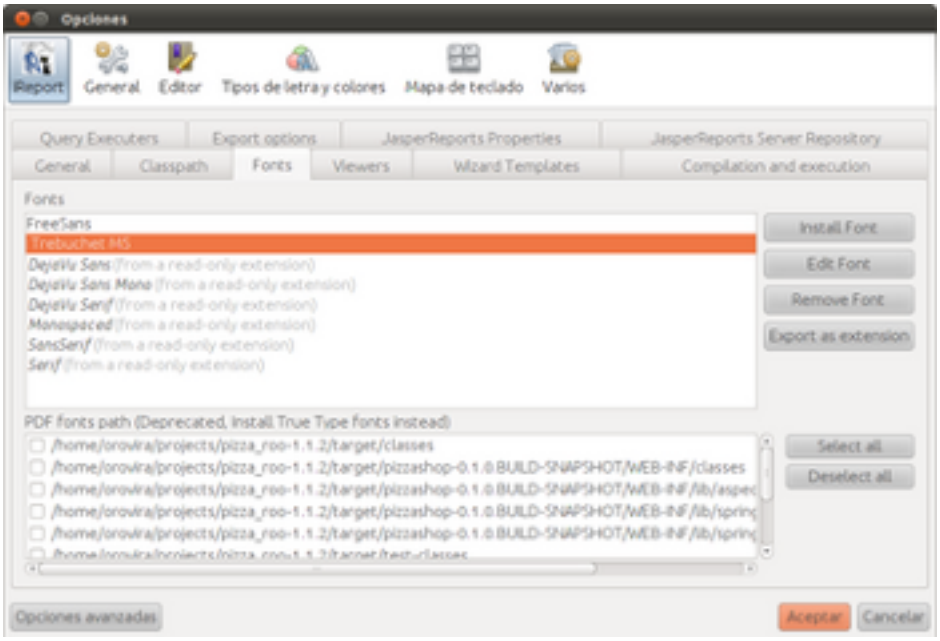

Figura 32.1. Herramienta de gestión de fuentes de IReport

Desde esta pantalla, seleccionando una fuente de la lista de fuentes instaladas en iReports y con el botón *"Export as extension"* se creará un archivo JAR que al descomprimirlo contiene:

- Directorio fonts
  - Archivos de fuentes en formato TTF.
  - Archivo *fontsfamily<IdentificadorAleatorio>.xml*
- Archivo *jasperreports_extension.properties*

Para instalar esta fuente en la aplicación se deben copiar los ficheros con extensión TTF a */WEB-INF/classes/jasperfonts* y del archivo *fontsfamily<IdentificadorAleatorio>.xml* adaptar el elemento <fontFamily/> para copiarlo en el archivo del proyecto *gvnix_reportfonts.xml* (modificando las rutas para que apunten a *jasperfonts*). El archivo *jasperreports_extension.properties* se puede ignorar porque ya se encuentra definido en el proyecto en */WEB-INF/classes.*

32.5. Diseño de informes con sub informes

gvNIX mediante el Add-on Web Report facilita la creación de informes sobre una entidad. El add-on genera el diseño de un informe Jasper Reports básico el cual se puede modificar utilizando un diseñador gráfico como iReport.

Puede ser necesario mostrar en un informe una lista de entidades relacionadas con la entidad sobre la que se ha generado el informe. Por ejemplo, en la aplicación del script de ejemplo clinic.roo, se podría definir un informe sobre la entidad Owner y desear mostrar para cada Owner la lista de mascotas que le pertenecen, es decir, listar la relación definida por el campo *Set<Pet> pets* de la entidad Owner.

Nota

Para este ejemplo se utilizará la versión 4.0.1 de iReport.

Partiendo del informe definido por el siguiente comando de gvNIX:

```
roo-gvNIX> web report add --controller ~.web.OwnerController --reportName ownerpets
```

En src/main/webapp/WEB-INF/reports se creará el archivo con el diseño del informe *owner_ownerpets.jrxml*. Abriendo este archivo usando iReport (*Archivo* > *Open*) aparecerá una pantalla como la que se muestra a continuación.

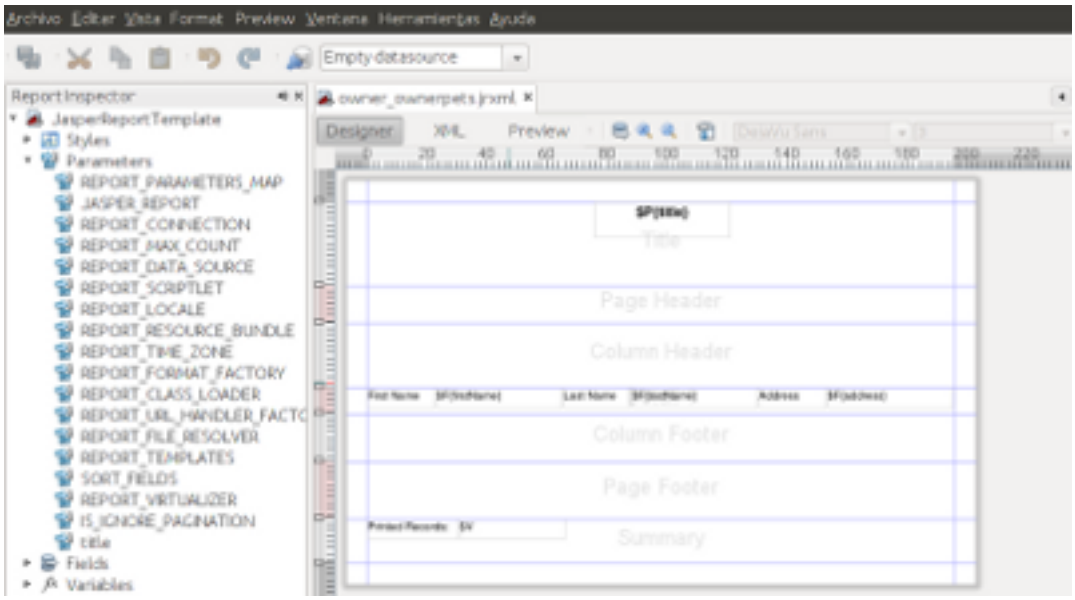

Figura 32.2. iReport - Diseñador

En la parte izquierda, en la pestaña con título *Report Inspector* se pueden observar los elementos principales del diseño de informes.

Figura 32.3. iReport - Report Inspector

Lo primero que se debe hacer es declarar un nuevo parámetro con, por ejemplo, el nombre *SubReportPetsLocation*. Para ello hacer click con el botón derecho sobre el elemento *Parameters* y seleccionar la opción *Agregar Parameter*. Justo debajo del último parámetro aparecerá uno nuevo con nombre *parameter1*. Al seleccionarlo, en la parte derecha de la ventana de iReport se podrán ver sus propiedades. Modificar sus propiedades estableciendo en la propiedad*Name* el valor *SubReportPetsLocation* y en la propiedad *Parameter Class* el valor *net.sf.jasperreports.engine.JasperReport*.

A continuación, añadir un elemento *Subreport* al diseño. Para que aparezca el *Subreport* para cada elemento de la lista de Owners, se debe añadir en la banda de detalle *Detail1* del informe. Será necesario aumentar la altura de la banda *Detail1*. Para ello, seleccionarla y en sus propiedad *Band height* especificar el valor *55*. De la paleta de elementos seleccionar el *Subreport* y arrastrarlo a la banda de detalle. Se abrirá un asistente, seleccionar la opción *Just create the subreport element* y cerrar con *Terminar*. Es posible modificar las dimensiones del elemento *Subreport* arrastrando las esquinas de la caja del elemento.

Figura 32.4. iReport - Subreport

Antes de seguir con el establecimiento de las propiedades del Subreport, es necesario declarar un campo del informe que permitirá acceder al campo *pets* de la Entidad *Owner*. En *Report Inspector*, seleccionar *Fields* y con el botón derecho *Agregar Field*. Modificar también este elemento estableciendo en la propiedad *Name* el valor *pets* y en la propiedad *Field Class* el valor *java.util.Collection*.

Se debe indicar cuales son las propiedades del *Subreport*. Para ello, en la ventana de propiedades de la derecha buscar las siguientes propiedades y establecer los valores que se especifican a continuación:

1. Subreport Expression = $P{SubReportPetsLocation}

2. Expression Class = net.sf.jasperreports.engine.JasperReport

3. Connection type = Use a datasource expression

4. Data Source Expression = new net.sf.jasperreports.engine.data.JRBeanCollectionDataSource($F{pets})

En el flujo de ejecución de un informe en Jasper Reports, los parámetros del mismo han de ser informados antes de su visualización o, de lo contrario, se tomarán los valores por defecto que se hubiesen definido o *null* en su defecto. Observar que se ha definido el parámetro *SubReportPetsLocation* y el valor que toma este parámetro en tiempo de ejecución ha de ser una ruta absoluta al sistema de archivos o al classpath de la aplicación. En la sección del Add-on Web Report se explicó que al añadir un informe a la aplicación, utilizando el comando *web report add*, en el archivo *src/main/webapp/WEB-INF/spring/jasper-views.xml* se declara un bean que establece la ruta hasta el fichero jrxml del informe añadido. Este bean tiene un atributo que indica el lugar donde se declaran los subreports que se usan en los distintos informes mediante el atributo *p:subReportsUrls-ref*.

```
<bean id="owner_ownerpets"
class=
 "com.springsource.petclinic.web.servlet.view.jasperreports.CustomJasperReportsMultiFormatView"
p:reportDataKey="ownerpetsList" p:url="/WEB-INF/reports/owner_ownerpets.jrxml"
p:subReportDataKeys-ref="subReportDataKeys" p:subReportUrls-ref="subReportUrls" />
```

El elemento anterior marcado en negrita apunta al elemento *<util:map/>*, declarado también en el fichero jasper-views.xml. Aquí se ha de especificar el valor que debe tomar el parámetro *SubReportPetsLocation.* Modificar este elemento añadiendo una entrada, quedando como se muestra a continuación.

```
<util:map id="subReportUrls">
  <!-- This entry key must be declared exactly as declared in the master JRXML file -->
  <!-- Add one <entry /> node for each sub-report you need to define -->
  <entry key="SubReportPetsLocation"
    value="/WEB-INF/reports/owner_ownerpets_sub_pets.jrxml"/>
```

```
</util:map>
```

En el elemento <*util:map/*> se pueden definir tantos elementos <*entry/*> como sea necesario para definir los parámetros que indican el lugar donde se localizan los archivos jrxml de los distintos subreports.

Ahora se debe diseñar el informe que mostrará el listado de mascotas (pets) que conformará el subreport.

Crear un nuevo diseño de informe (*Archivo > Nuevo*), y en el asistente seleccionar *Blank A4* y *Open This Template*. Se solicitará un nombre para el diseño y el lugar donde guardarlo. Indicar, por ejemplo *owner_ownerpets_sub_pets.jrxml* y guardarlo en el mismo directorio donde se encuentre el report padre (*owner_ownerpets.jrxml*).

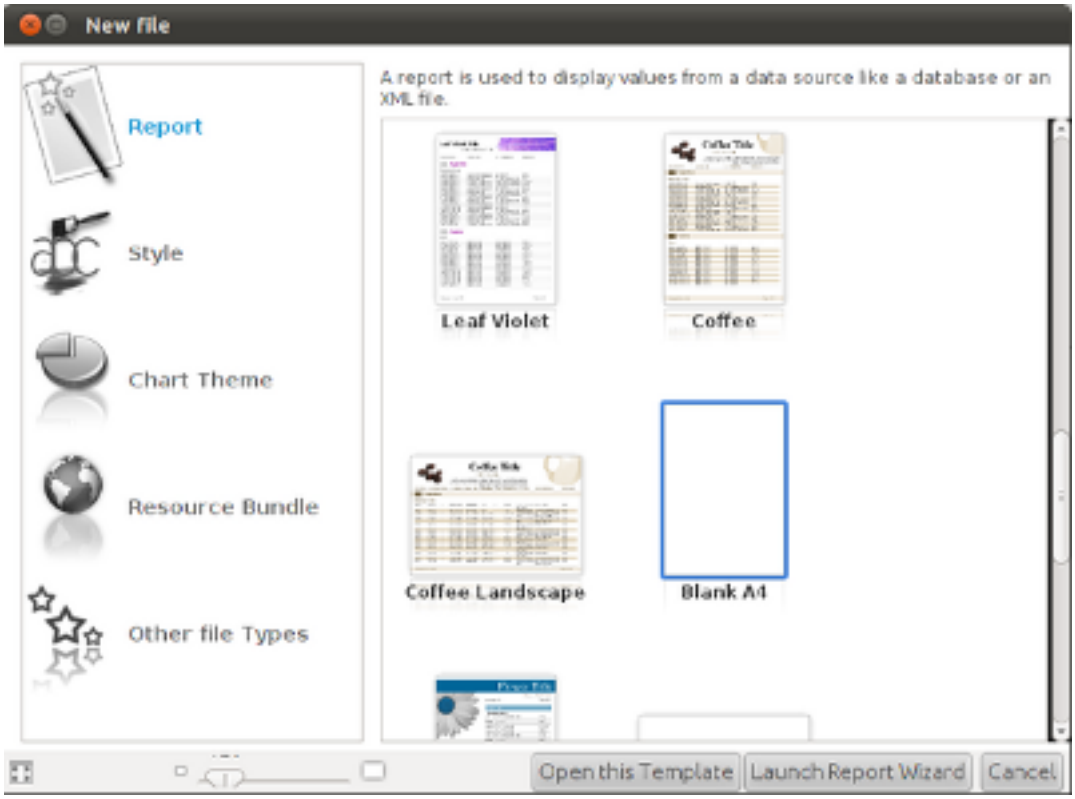

Figura 32.5. iReport - Nuevo diseño

Importante

Se debe validar que el nuevo informe se crea con el lenguage Java configurado para las expresiones. En *Report Inspector*, al seleccionar el primer elemento (el nombre del informe) y mirar en sus propiedades (parte derecha de la pantalla de iReport) se debe buscar la propiedad *Language*, si no tiene el valor Java, cambiarla usando el desplegable.

Este nuevo informe, que constituirá el subreport, es un informe a todos los efectos, por tanto se debe definir los campos (elemento Fields del diseño), parámetros (elemento Parameters), etc.

Al igual que se ha hecho anteriormente con el campo pets del informe padre, definir los campos que permiten acceder a los valores de la entidad Pet que se visualizarán en el subreport. Click con el botón derecho sobre Fields y Agregar Field:

1. Crear field con nombre *name* y clase *java.lang.String*

2. Crear field con nombre *weight* y clase *java.lang.Float*

3. Crear field con nombre *type* y clase *com.springsource.petclinic.reference.PetType* (enumerado declarado en el proyecto clinic.roo)

El diseño del informe se compone de distitnas bandas, donde se colocan los elementos a mostrar (*Title, Page Header, Column Header, Detail1, Column Footer, ...*). Para un subreport como el de este ejemplo, algunas bandas son innecesarias y pueden ser eliminadas. Esto permitirá ajustar mejor el diseño completo del informe (informe padre ownerpets + subreport pets). Si se hace click con el botón derecho sobre *Title* y se selecciona *Delete Band*, es eliminada. Hacerlo con todas las bandas excepto con *Column Header, Detail1 y Summary*.

Figura 32.6. iReport - Bandas subreport

Column Header es una banda que se dibujará solo al inicio de la columna del informe (en este caso solo existirá una). Añadir textos estáticos que harán las veces de cabecera de tabla. Para ello, de la paleta de componentes, seleccionar *Static Text* y arrastrarlo hasta la banda *Column Header*.

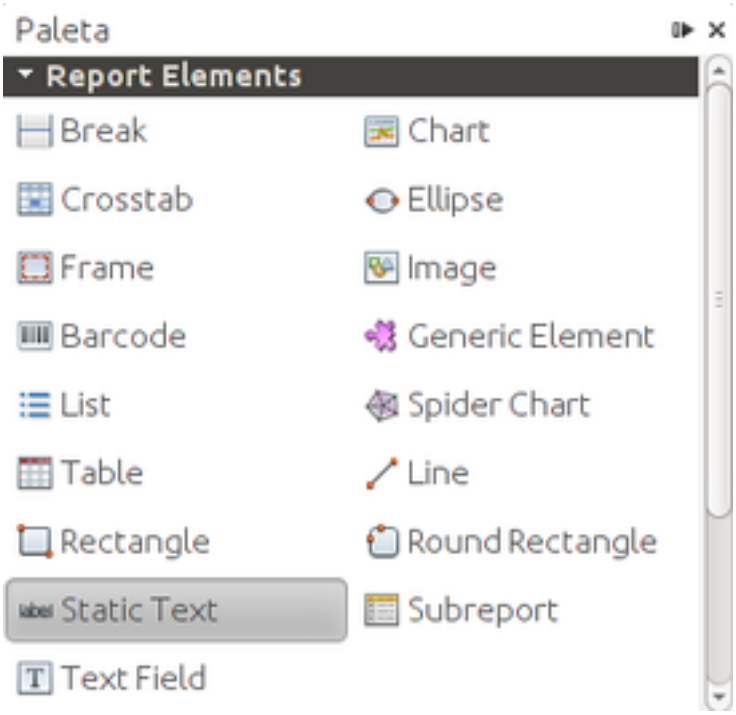

Figura 32.7. iReport - Paleta

En el primer *Static Text* establecer el valor *Nombre* y repetir la misma operación para definir un texto para el campo *Peso* y para el campo *Tipo*. Es posible distribuir las etiquetas para que quede tal y como se muestra en la captura anterior.

A continuación, hacer lo mismo pero usando el elemento *Text Field*. Este elemento sirve para mostrar el valor de un campo de la entidad *Pet* y se ha de corresponder con los field definidos anteriormente (*name, weight, type*). Estos elementos *Text Field* se definirán en la banda *Detail1*, puesto que se repetirán para cada uno de los elementos de la fuente de datos del subreport. Recordar que la fuente de

datos de este subreport se había definido anteriormente como *JRBeanCollectionDataSource* usando la colección de *Pets* que pertenecen a un *Owner*.

En los Text Field, es importante establecer correctamente la propiedad Expression Class, esta se ha de corresponder con el tipo Java del field definido y que se va a usar para mostrarlo. Por ejemplo, el Text Field que mostará el peso de la mascota tendría las propiedades como sigue:

- Text Field Expression = $F{weight}
- Expression Class = java.lang.Float

Para establecer el Text Field Expression se puede utilizar el diálogo que se muestra a continuación Este diálogo permite seleccionar elementos de los campos definidos en el informe, de los parámetros e incluso acceder a alguna de las operaciones que proporcionan estos objetos.

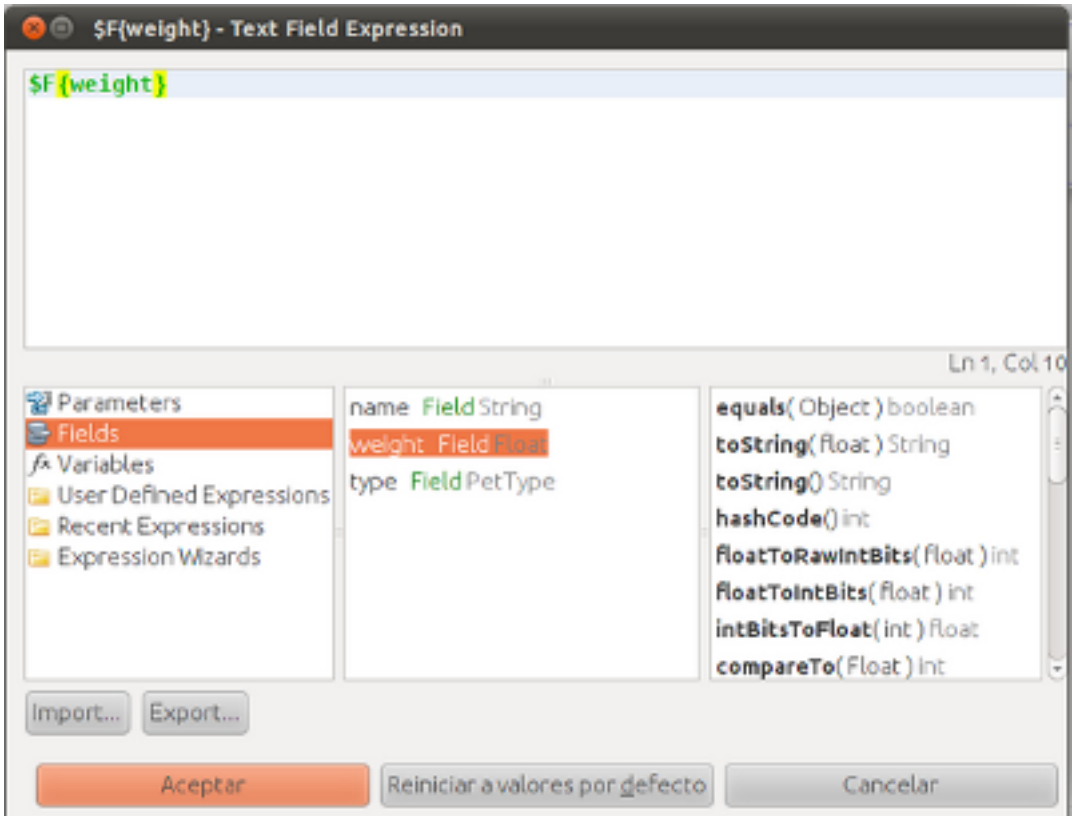

Figura 32.8. iReport - Diálogo Text Field Expression

Obviamente, en un *Text Field* es posible establecer expresiones más elaboradas, como añadir a la cadena las unidades de peso. Se debe tener en cuenta que la propiedad *Expression Class* deberá establecerse al tipo Java resultante de la expresión (por ejemplo, $F{weight} + "kg" da como resultado un java.lang.String). Establecer en el *Text Field* asociado con el tipo de mascota la expresión *$F{type}.toString()* y la clase *java.lang.String*.

Ya está diseñada la banda de detalle y para terminar el subreport, opcionalmente, se puede definir un *Text Field* que indique cuantos registros de *Pet* se han impreso cada vez que se ejecuta el subreport. Esto ya se encuentra incluido en el informe padre (owners) y se consigue utilizando la variable del report *REPORT_COUNT*, ya difinida por defecto. Incluirla en la banda *Summary* con un *Text Field* en la que el *Expression* tome el valor *$V{REPORT_COUNT}* y el *Class* sea *java.lang.Integer*.

Solo queda arrancar de nuevo la aplicación, desde el Tomcat de Eclipse o desde la línea de comandos y probar la generación del informe de Owners.

Figura 32.9. Informe Owners con subreport Pets

JasperReports permite gran cantidad de opciones al trabajar con subreports (paso de parámetros desde el informe padre a los subinformes, devolución de parámetros desde los subinformes al informe padre, actualización de variables, ...). Para conocerlas con profundidad consultar la documentación de referencia de Jasper Reports e iReport.

32.6. Operaciones durante el inicio de la aplicación

Se pueden crear clases que se invoquen durante la inicialización de una aplicación.

En alguna aplicación puede ser necesario realizar alguna acción durante el inicio de una aplicación, como por ejemplo. Existen diversas formas para hacer esto en Spring.

Sino se va a interaccionar con datos, se puede usar la anotación @PostConstruct en un método de una clase anotada con *@Component*. Hay que tener en cuenta que la inicialización del soporte de datos y transacciones puede no estar todavía listo en este punto.

Crear un listener del contexto de Spring de la siguiente forma:

```
@Component
  public class DataSeeder implements ApplicationListener<ContextRefreshedEvent> {

    @Override
        @Transactional
    public void onApplicationEvent(ContextRefreshedEvent event) {
        if (Model.countModels() == 0) {
            // Create initial Model values
        }
    }
```

32.7. Obtener el BindStatus de un atributo dentro de un formulario

Obtener el BindStatus de un atributo dentro de un formulario:

```
<spring:bind path="${field}">
        <c:set var="fieldValue" value="${status.value}" />
    </spring:bind>
```

Parte V. Apéndices

Listado de comandos adicionales incluidos en gvNIX detallando cada uno de sus parámetros.

Capítulo 33. Apéndice de comandos de gvNIX

33.1. Comandos del add-on OCC

Para una descripción detallada de las características del add-on consultar la documentación del add-on OCC (Optimistic Concurrency Control).

Table 33.1. Comandos del add-on OCC

Comando	Descripción
occ checksum set	Establece un control de concurrencia optimista mediante una suma de comprobación (checksum) para una determinada entidad del proyecto
occ checksum all	Establece un control de concurrencia optimista mediante una suma de comprobación (checksum) para todas las entidades del proyecto

33.1.1. occ checksum set

Establece un control de concurrencia optimista mediante una suma de comprobación (checksum) en una entidad del modelo. Si la entidad ya tiene configurado este sistema de control, el comando no permitirá sobreescribir su configuración y cualquier modificación deberá realizarse sobre la anotación relacionada *@GvNIXEntityOCCChecksum* que se habrá incluido en la clase Java de la entidad.

Sus parámetros son:

--entity
[Requerido si no está en el contexto] [Clase Java] [Autocompletado]

Entidad sobre la que establecer el sistema de control de concurrencia optimista mediante una suma de comprobación (checksum).

--fieldName
[Opcional] [Cadena]

Nombre del campo de la entidad en donde almacenar la suma de comprobación (checksum). Habitualmente este parámetro no necesita ser establecido. Si no se especifica, tomará por convención el nombre *occChekcsum* para dicha propiedad.

--digestMethod
[Opcional] [Cadena]

Método a utilizar en el cálculo de la suma de comprobación (checksum). Habitualmente este parámetro no necesita ser establecido. Los posibles valores vienen definidos por los algoritmos aceptados por MessageDigest.getInstance(String algorithm). Por defecto, se utilizará el algoritmo *md5*, pero pueden utilizarse otros como por ejemplo *sha*.

33.1.2. occ checksum all

Estable un control de concurrencia optimista mediante una suma de comprobación (checksum) en todas las entidades de la aplicación que no la tuviesen todavía establecida. Cualquier modificación una vez

establecido el control deberá realizarse sobre la anotación relacionada *@GvNIXEntityOCCChecksum* que se habrá incluido en todas las clases Java de entidad.

Sus parámetros son:

--fieldName
[Opcional] [Cadena]

Nombre del campo en cada entidad en donde almacenar la suma de comprobación (checksum). Habitualmente este parámetro no necesita ser establecido. Si no se especifica, tomará por convención el nombre *occChekcsum* para dicha propiedad.

--digestMethod
[Opcional] [Cadena]

Método a utilizar en el cálculo de la suma de comprobación (checksum). Habitualmente este parámetro no necesita ser establecido. Los posibles valores vienen definidos por los algoritmos aceptados por MessageDigest.getInstance(String algorithm). Por defecto, se utilizará el algoritmo *md5*, pero pueden utilizarse otros como por ejemplo *sha*.

33.2. Comandos del add-on JPA

Para una descripción detallada de las características del add-on consultar la documentación del add-on JPA.

Table 33.2. Comandos de add-on jpa

Comando	Descripción
jpa gvnix setup	Instala el soporte para la funcionalidad
jpa batch add	Genera el servicio de persistencia en bloque para una entidad.
jpa batch all	Genera el servicio de persistencia en bloque para todas las entidades.
jpa audit setup	Configura la auditoría de historico creando la clase que proveera del nombre de usuario que realiza el cambio. Sólo se puede ejecutar una vez.
jpa audit add	Añade el soporte de auditoría a una entidad.
jpa audit all	Añade el soporte de auditoría todas las entidades.
jpa audit revisionLog	Establece el proveedor de registro de cambios a usar.
jpa geo setup	Configura el proyecto para poder guardar entidades con campos de tipo geográficos.
field geo	Añade un nuevo campo de tipo geográfico a la entidad seleccionada.
finder geo all	Genera los buscadores de todos los campos de tipo GEO de *todas* las entidades registradas en el proyecto.
finder geo add	Genera los buscadores de todos los campos de tipo GEO para la entidad seleccionada.

33.2.1. jpa gvnix setup

Instala las dependencias necesarias para las funcionalidades del add-on.

33.2.2. jpa batch add

Crea el servicio de persistencia en bloque para una entidad.

Sus parámetros son:

--entity
[Requerido] [Clase entidad] [Autocompletado]

Entidad destino de las operaciones de persistencia.

--type
[Opcional] [Clase] [Autocompletado]

Clase de servicio a generar. Si se omite se generará una clase, con nombre basado en el nombre de la entidad, en el mismo paquete que la entidad.

33.2.3. jpa batch all

Crea el servicio de persistencia en bloque para todas las entidades de la aplicación.

Sus parámetros son:

--package
[Opcional] [Paquete] [Autocompletado]

Paquete donde se generarán las clases de servicio a generar. Si se omite se generarán en el mismo paquete que la entidad a la que afecta. El nombre de la clase del servicio se genera a partir del nombre de la entidad.

33.2.4. jpa audit setup

Configura la auditoría de historico creando la clase que proveera del nombre de usuario que realiza el cambio. Sólo se puede ejecutar una vez.

Sus parámetros son:

--service
[Requerido] [Clase]

Nombre de la clase a crear para hacer de proveedor del objeto usuario.

--userType
[Opcional] [Clase] [Autocompletado]

Clase a usar como usuario. Por defecto ``String``.

Note

Lanzará un warning informando que tiene que ajustar la implementación de la clase en dos circunstancias:

Si no está configurado Spring Security

Si la clase ``userType`` no es String y no implementa ``UserDetails``.

33.2.5. jpa audit add

Añade el soporte de auditoría a una entidad.

Sus parámetros son:

--entity
[Requerido] [Clase entidad] [Autocompletado]

Entidad a la que añadir la auditoría.

--type
[Opcional] [Clase] [Autocompletado]

Clase a crear como *EntityListener* para establecer los datos de auditoría. Si no se establece se genera una clase, basada en el nombre de la entidad, en el mismo paquete que la entidad.

33.2.6. jpa audit all

Añade el soporte de auditoría a todas las entidades de la aplicación.

Sus parámetros son:

--package
[Opcional] [Paquete] [Autocompletado]

Paquete donde se generarán las clases *EntityListener*. Si se omite se generarán en el mismo paquete que la entidad a la que afecta. El nombre de la clase *EntityListener* se genera a partir del nombre de la entidad.

33.2.7. jpa audit revisionLog

Selecciona el proveedor de gestión de revisiones de entidades a usar.

Sus parámetros son:

--provider
[Requerido] [Identificador de proveedor] [Autocompletado]

Identificador del proveedor de revisiones a usar. Sólo estarán visibles aquellos proveedor que puedan utilizarse en el proyecto actual (esto se consulta al la propia instancia de proveedor). Los proveedores se registran como add-ons en el framework (al estilo de los proveedores de JDBC para la ingeniería inversa) y deben de implementar un interfaz definido.

33.2.8. jpa geo setup

Configura el proyecto para poder guardar entidades con campos de tipo geográficos.

Sus parámetros son:

--provider
[Requerido] [Identificador de proveedor] [Autocompletado]

Identificador del proveedor de persistencia GEO a usar. Sólo estarán visibles aquellos proveedor que puedan utilizarse en el proyecto actual (esto se consulta al la propia instancia de proveedor). Los proveedores se registran como add-ons en el framework (al estilo de los proveedores de JDBC para la ingeniería inversa) y deben de implementar un interfaz definido.

33.2.9. field geo

Añade un nuevo campo de tipo geográfico a la entidad seleccionada.

Sus parámetros son:

--class
[Requerido] [Autocompletado]

Entidad sobre la que se quiere añadir el nuevo campo de tipo GEO

--fieldName
[Requerido]

Nombre del nuevo campo que se quiere añadir

--type
[Requerido] [Autocompletado]

Selecciona el tipo de dato GEO que representará el nuevo campo añadido a la entidad. Puede tomar los siguientes valores:

- POINT (Se guarda un único punto en la base de datos)
- LINESTRING (Se guardan una serie de puntos que forman una linea continua)
- MULTILINESTRING (Se guardan una serie de puntos que forman varias lineas continuas)
- POLYGON (Se guardan una serie de puntos que forman un polígono. Siempre empieza y acaba en el mismo punto.)
- GEOMETRY (Se guarda una serie de puntos que forman una geometría. Acepta cualquiera de las geometrías anteriores.)

33.2.10. finder geo all

Genera los buscadores de todos los campos de tipo GEO de *todas* las entidades registradas en el proyecto.

33.2.11. finder geo add

Genera los buscadores de todos los campos de tipo GEO para la entidad seleccionada.

Sus parámetros son:

--class
[Requerido] [Autocompletado]

Entidad sobre la que se quieren generar los métodos de búsqueda.

33.3. Comandos del add-on Web Dialog

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web Dialog.

Table 33.3. Comandos del add-on Web Dialog

Comando	Descripción
web mvc dialog setup	Instala el soporte para diálogos modales y realiza la gestión de ciertas excepciones para que sean visualizadas de forma amigable y en múltiples idiomas
web mvc dialog exception list	Muestra por la consola la lista de excepciones que están siendo actualmente gestionadas
web mvc dialog exception add	Incluye una nueva excepción dentro de la gestión de excepciones y permite definir el mensaje amigable asociado a la excepción en un determinado idioma
web mvc dialog exception set language	Añade o actualiza un mensaje amigable asociado a una excepción en un determinado idioma para lo que la excepción debe haber sido incluida previamente en el sistema de gestión de excepciones con el comando anterior
web mvc dialog exception remove	Quita una excepción de la gestión y por lo tanto no se mostrará un mensaje amigable y en múltiples idiomas en el caso de producirse y no ser controlada desde el proyecto
web mvc dialog add	Añade dos métodos en el código fuente de un controlador que permiten abrir un diálogo modal en una página de la aplicación sin realizar ninguna implementación en la capa web

33.3.1. web mvc dialog setup

Instala en el proyecto el soporte para diálogos de error y de usuario en ventana modal (pop-up). A partir de ese momento los mensajes de error de la aplicación se mostrarán con ese formato. También configura un sistema para visualizar de forma amigable y con soporte multi idioma ciertas excepciones que se pueden producir en esta clase de aplicaciones. Ver Add-on Web Dialog para una información más detallada.

33.3.2. web mvc dialog exception list

Muestra por consola la lista de excepciones que actualmente están siendo gestionadas. Es decir, las excepciones que si no son controladas desde la aplicación y se producen, aparecerán al usuario en una ventana modal con un mensaje amigable y en multiples idiomas. El resultado mostrado por la consola será algo similar a lo siguiente:

```
Handled Exceptions:
.DataAccessException
.NoSuchRequestHandlingMethodException
.TypeMismatchException
```

```
.MissingServletRequestParameterException
java.sql.SQLException
java.io.IOException
org.springframework.transaction.TransactionException
java.lang.UnsupportedOperationException
javax.persistence.OptimisticLockException
org.hibernate.NonUniqueObjectException
org.hibernate.exception.ConstraintViolationException
org.hibernate.id.IdentifierGenerationException
```

33.3.3. web mvc dialog exception add

Incluye una nueva excepción a la gestión de excepciones. De esta forma, si se produce dicha excepción en la aplicación se visualizará de forma amigable mediante un mensaje modal y en múltiples idiomas. Este comando permite también definir el mensaje amigable asociado a la excepción en un determinado idioma. El mensaje amigable se compone de título y descripción.

Sus parámetros son:

--exception
[Requerido] [Cadena]

Nombre de la excepción a incluir en la gestión de excepciones. Si se desea que la definición sea única, se ha de especificar el paquete java completo del que proviene, por ejemplo *java.lang.NullPointerException*, ya que si se especifica únicamente el nombre, por ejemplo *NullPointerException*, aplicaría a todas las excepciones con dicho nombre sea cual sea el paquete en el que se encuentre.

--title
[Requerido] [Cadena]

Título de la ventana modal que se mostrará en pantalla cuando se produzca la excepción.

--description
[Requerido] [Cadena]

Texto con el contenido de la ventana modal que se mostrará en pantalla cuando se produzca la excepción.

--language
[Requerido] [Cadena]

Idioma del título y de la descripción en su formato ISO: *es, de, it, nl, sv, en, etc*.

33.3.4. web mvc dialog exception set language

Añade o actualiza el mensaje amigable asociado a una excepción en un determinado idioma. La excepción debe haber sido incluida anteriormente en el sistema de gestión de excepciones con el comando *web mvc dialog exception add*. El mensaje amigable se compone de título y descripción.

Sus parámetros son:

--exception
[Requerido] [Cadena]

Nombre de la excepción ya incluida en la gestión de excepciones para la que se desea añadir o modificar sus mensajes amigables en un idioma determinado. Se puede obtener el nombre exacto de la excepción mediante el comando *web mvc dialog exception list*.

--title
[Requerido] [Cadena]

Título de la ventana modal que se mostrará en pantalla cuando se produzca la excepción.

--description
[Requerido] [Cadena]

Texto con el contenido de la ventana modal que se mostrará en pantalla cuando se produzca la excepción.

--language
[Requerido] [Cadena]

Idioma del título y de la descripción en su formato ISO: *es, de, it, nl, sv, en, etc*.

33.3.5. web mvc dialog exception remove

Quita una excepción del sistema de gestión de excepciones y por lo tanto no se mostrará un mensaje amigable y en múltiples idiomas en el caso de producirse cuando no es controlada desde el código fuente de la aplicación.

Sus parámetros son:

--exception
[Requerido] [Cadena]

Nombre de la excepción ya incluida en la gestión de excepciones que se desea eliminar. Se puede obtener el nombre exacto de la excepción mediante el comando *web mvc dialog exception list*.

33.3.6. web mvc dialog add

Añade dos métodos en el código fuente de un controlador que permiten abrir un diálogo modal en una página de la aplicación sin realizar ninguna implementación en la capa web. Ver la sección de inclusión de nuevos diálogos modales para una información más detallada.

Sus parámetros son:

--class
[Requerido] [Clase Java] [Autocompletado]

Controlador para el cual generar los métodos de creación de un diálogo modal.

--name
[Requerido] [Cadena]

Nombre que se dará a uno de los métodos en el controlador. Este método y el método por defecto con nombre *modalDialog* permitirán mostrar un diálogo modal cada uno con distintas características.

33.4. Comandos del add-on Web Menu

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web Menu.

Table 33.4. Comandos del add-on Web Menu

Comando	Descripción
menu setup	Instala el sistema de gestión del menú web que permite organizar de forma sencilla su estructura de páginas y permisos
menu entry add	Añade un elemento al menú, sea una página o un grupo
menu entry visibility	Define un elemento del menú como visible o oculto
menu entry roles	Establece una lista de perfiles de usuario para los que se mostrará una entrada del menú
menu entry move	Mueve un elemento del menú y todos los hijos que pueda tener
menu entry update	Actualiza un elemento del menú
menu entry info	Muestra por consola los valores establecidos en un elemento del menú y en todos los hijos que pueda tener
menu tree	Muestra por consola un árbol con un resumen de los elementos del menú.

33.4.1. menu setup

Instala y activa en la aplicación el sistema de gestión del menú web para la organización de su estructura de páginas y la definición de permisos. A partir de este momento se podrán utilizar todos los comandos de gestión del menú para modificar su modelo de datos y definir permisos de visualización sobre las distintas opciones del menú en función del perfil de cada usuario. El modelo de datos se encuentra definido en el fichero `src/main/webapp/WEB-INF/views/menu.xml`.

Para una descripción detallada de las tareas que realiza este comando consultar la documentación de instalación de la gestión del menú.

33.4.2. menu entry add

Añade un elemento al menú, sea una página o un grupo. La diferencia entre ambos es que una página tiene asociada una URL y un grupo no. Este comando crea un elemento del menú, pero no crea una nueva vista en el proyecto. La creación de una vista puede realizarse con el comando web mvc view.

Sus parámetros son:

--category
[Opcional] [Elemento del menú] [Autocompletado]

Identificador del elemento del menú donde incluir la nueva entrada. El elemento del menú puede ser una página o un grupo. Si no se especifica este parámetro la nueva entrada se incluirá en el grupo *Page* que si no existe será creado al final del menú.

--url
[Opcional] [Cadena]

URL con la que enlazará el elemento del menú.

--label
[Requerido] [Cadena]

Título para el elemento del menú. Se creará una etiqueta con este título en el fichero `src/main/webapp/WEB-INF/i18n/application.properties` . Este título será ignorado si se define el parámetro *messageCode*.

--messageCode
[Opcional] [Cadena]

Etiqueta que representará la traducción en múltiples idiomas del título del elemento del menú. Si todavía no existe, se deberá crear dicha etiqueta en cada fichero de idioma que exista configurado en el proyecto en `src/main/webapp/WEB-INF/i18n/messages_xx.properties` .

--roles
[Opcional] [Cadena]

Lista de perfiles con permiso para ver el elemento del menú. Los distintos elementos de la lista de perfiles deben estar separados por comas. Si no se establece este parámetro, el elemento del menú aparecerá para todos los perfiles.

Al finalizar la ejecución de este comando se informará del identificador del nuevo elemento del menú añadido con un texto similar al siguiente.

```
New page 'i_page_xxxx_xxxx' added.
```

33.4.3. menu entry visibility

Cambia la visibilidad de un elemento del menú, es decir, permite definirlo como visible o oculto.

Sus parámetros son:

--id
[Requerido] [Elemento del menú] [Autocompletado]

Identificador del elemento del menú, sea página o grupo.

--hidden
[Requerido] [Booleano] [Autocompletado]

Indica si este elemento del menú debe ocultarse. Si se especifica este parámetro sin ningún valor, tomará el valor *true* y ocultará el elemento del menú. Si no se especifica este parámetro, tomará el valor *false* y mostrará el elemento del menú.

33.4.4. menu entry roles

Establece una lista de perfiles de usuario para los que se mostrará una entrada del menú. No se mostrará a los usuarios que no tengan algún perfil de la lista.

Sus parámetros son:

--id
[Requerido] [Elemento del menú] [Autocompletado]

Identificador del elemento del menú, sea página o grupo.

--roles
[Requerido] [Cadena]

Lista de perfiles con permiso para ver el elemento del menú. Los distintos elementos de la lista de perfiles deben estar separados por comas.

33.4.5. menu entry move

Mueve un elemento del menú y todos los hijos que pueda tener. Se puede cambiar su posición en el orden de elementos del menú o se puede mover dentro de otro elemento del menú.

Sus parámetros son:

--id
[Requerido] [Elemento del menú] [Autocompletado]

Identificador del elemento del menú, sea página o grupo.

--into
[Opcional] [Elemento del menú] [Autocompletado]

Identificador de un elemento del menú donde añadir como hijo el elemento del menú. El elemento se insertará en la última posición de la lista de hijos. Este parámetro no se puede especificar simultaneamente junto al parámetro *before*.

--before
[Opcional] [Elemento del menú] [Autocompletado]

Identificador de un elemento del menú antes del cual insertar el elemento del menú. Este parámetro no se puede especificar simultaneamente junto al parámetro *into*.

33.4.6. menu entry update

Actualiza un elemento del menú, sea una página o un grupo. Los valores no especificados en el comando se mantendrán con su valor original.

Sus parámetros son:

--id
[Requerido] [Elemento del menú] [Autocompletado]

Identificador del elemento del menú, sea página o grupo.

--nid
[Opcional] [Cadena]

Nuevo identificador para el elemento del menú. Por convención, se utiliza el prefijo *c_* para los grupos e *i_* para las páginas.

--label
[Opcional] [Cadena]

Título para el elemento del menú. Se actualizará con el nuevo título la etiqueta correspondiente en el fichero `src/main/webapp/WEB-INF/i18n/application.properties` . Este título será ignorado si se define o ya estaba definido el parámetro *messageCode*.

--messageCode
[Opcional] [Cadena]

Etiqueta que representará la traducción en múltiples idiomas del título del elemento del menú. Si todavía no existe, se deberá crear dicha etiqueta en cada fichero de idioma que exista configurado en el proyecto en `src/main/webapp/WEB-INF/i18n/messages_xx.properties` .

--url
[Opcional] [Cadena]

URL con la que enlazará el elemento del menú.

--roles
[Opcional] [Cadena]

Lista de perfiles con permiso para ver el elemento del menú. Los distintos elementos de la lista de perfiles deben estar separados por comas.

--hidden
[Requerido] [Booleano] [Autocompletado]

Indica si este elemento del menú debe ocultarse. Si se especifica este parámetro sin ningún valor, tomará el valor *true* y ocultará el elemento del menú. Si no se especifica este parámetro, tomará el valor *false* y mostrará el elemento del menú.

33.4.7. menu entry info

Muestra por consola los valores establecidos en un elemento del menú y en todos los hijos que pueda tener, formateados de forma amigable para su fácil lectura.

Sus parámetros son:

--id
[Requerido] [Elemento del menú] [Autocompletado]

Identificador del elemento del menú, sea página o grupo.

--lang
[Opcional] [Idioma]

Idioma en el que mostrar los valores de las distintas etiquetas definidas para cada elemento del menú.

El comando mostrará un resultado similar al siguiente:

```
roo-gvNIX> menu entry info --id c_person
[c_person]
URL          : No
Label Code   : menu_category_person_label
Label        : Person
Message Code :
Message      :
Roles        :
Hidden       : false
Children     :
               [i_person_new]
               URL          : /people?form
               Label Code   : menu_item_person_new_label
               Label        : Person
               Message Code : global_menu_new
               Message      : Create new {0}
               Roles        :
               Hidden       : false

               [i_person_list]
               URL          : /people?page=1&size=${empty param.size ? 10 : param.size}
               Label Code   : menu_item_person_list_label
               Label        : People
               Message Code : global_menu_list
               Message      : List all {0}
               Roles        :
               Hidden       : false
```

33.4.8. menu tree

Muestra por consola un árbol con un resumen de los elementos del menú, opcionalmente a partir de un determinado elemento del menú.

Sus parámetros son:

--id
[Opcional] [Elemento del menú] [Autocompletado]

Identificador del elemento del menú, sea página o grupo, a partir del cual comenzar a mostrar el árbol resumen. Si no se especifica, se muestran todas las página del menú.

El comando mostrará un resultado similar al siguiente:

```
roo-gvNIX> menu tree
[c_pet, visible, no-URL]
          /pets?form  [i_pet_new, visible]
          /pets?page=1&size=${empty param.size ? 10 : param.size}  [i_pet_list, visible]
          /pets?find=ByNameAndWeight&form  [fi_pet_nameandweight, visible]
          /pets?find=ByOwner&form  [fi_pet_owner, visible]
          /pets?find=BySendRemindersAndWeightLessThan&form
            [fi_pet_sendremindersandweightlessthan, visible]
          /pets?find=ByTypeAndNameLike&form  [fi_pet_typeandnamelike, visible]

[c_owner, visible, no-URL]
          /owners?form  [i_owner_new, visible]
          /owners?page=1&size=${empty param.size ? 10 : param.size}
            [i_owner_list, visible]

[c_visit, visible, no-URL]
          /visits?form  [i_visit_new, visible]
          /visits?page=1&size=${empty param.size ? 10 : param.size}
```

```
            [i_visit_list, visible]
          /visits?find=ByDescriptionAndVisitDate&form
            [fi_visit_descriptionandvisitdate, visible]
          /visits?find=ByDescriptionLike&form  [fi_visit_descriptionlike, visible]
          /visits?find=ByVisitDateBetween&form  [fi_visit_visitdatebetween, visible]

[c_vet, visible, no-URL]
          /vets?form  [i_vet_new, visible]
          /vets?page=1&size=${empty param.size ? 10 : param.size}  [i_vet_list, visible]

[c_seleniumtests, visible, no-URL]
          /resources/selenium/test-suite.xhtml  [si_seleniumtests_test, visible]
```

33.5. Comandos del add-on Web Report

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web Report.

Table 33.5. Comandos del add-on Web Report

Comando	Descripción
web report setup	Configura en el proyecto el soporte para la generación de informes en formato PDF, XLS, CSV y HTML
web report add	Crea un nuevo informe y permite su acceso desde el controlador de una entidad que se le proporciona como parámetro

33.5.1. web report setup

Configura en el proyecto el soporte para la generación de informes en formato PDF, XLS, CSV y HTML mediante *Jasper Reports* y su descarga desde la aplicación web.

El comando sólo estará disponible si el proyecto utiliza Spring MVC en la capa web y Tiles como motor de vistas web.

Este comando no tiene ningún parámetro obligatorio ni opcional.

Para una descripción detallada de las configuraciones que realiza este comando en el proyecto, consultar la sección de instalación del add-on Web Report.

33.5.2. web report add

Crea un nuevo informe y permite su acceso desde el controlador de una entidad que se le proporciona como parámetro al comando. El controlador debe obligatoriamente estar gestionando una entidad del proyecto. El informe, por defecto, mostrará los 10 primeros registros de la entidad.

Si el comando se ejecuta varias veces sobre el mismo controlador y con el mismo nombre de informe, se añadirán al informe existente los nuevos formatos de generación que se hayan podido especificar. Para mas información consultar la sección de generación de un informe del add-on Web Report.

El comando sólo estará disponible si el proyecto utiliza Spring MVC en la capa web y Tiles como motor de vistas web.

Sus parámetros son:

--controller
[Requerido] [Clase Java] [Autocompletado]

Clase controladora que gestiona una entidad en la que se desea añadir el informe. Este controlador ha de contener la anotación *@RooWebScaffold* y su atributo *formBackingObject* indicará la entidad para la que se generará el informe.

--reportName
[Requerido] [Cadena]

Nombre que se le quiere dar al informe y que servirá de identificador único. No admite espacios y el valor proporcionado será convertido a minúsculas.

--format
[Opcional] [Cadena]

Lista de formatos en los que se permitirá generar el informe. Los formatos de la lista deberán estar separados por comas y sin espacios. Los formatos soportados son: *pdf, xls, csv y html*. Si el nombre del informe ya existe en un controlador, los formatos nuevos que se hubiesen podido definir se añadirán al informe existente. Los formatos siempre serán convertidos a minúsculas. Si este parámetro no se especifica, tomará por defecto el valor *pdf*.

33.6. Comandos del add-on Service

Para una descripción detallada de las características del add-on consultar la documentación del add-on Service.

Table 33.6. Comandos del add-on Service

Comando	Descripción
remote service class	Crea una clase Java que actuará como servicio local dentro de la aplicación
remote service operation	Añade un nuevo método en una clase Java
remote service define ws	Hace que una clase Java pueda ser accedida desde el exterior a través de un servicio web
remote service export operation	Hace que un método de una clase Java pueda ser accedido desde el exterior a través de una operación del servicio web
remote service list operation	Muestra el listado de métodos de una clase Java que podrían ser ofrecidos al exterior como operaciones del servicio web
remote service export ws	Crea en el proyecto la infraestructura necesaria definida por un *WSDL* para ofrecer ciertas operaciones mediante un servicio web
remote service import ws	Crea en el proyecto la infraestructura necesaria definida por un *WSDL* para acceder a ciertas operaciones situadas en un servicio web
remote service ws list	Lista los servicios web que se encuentran definidos en la aplicación
remote service security ws	Añade o actualiza la configuración necesaria para firmar las peticiones que se hacen a un servicio web remoto

33.6.1. remote service class

Crea una clase Java que actuará como servicio local dentro de la aplicación. El servicio se podrá utilizar desde cualquier otra clase Java definiendo una propiedad del tipo de la clase de servicio con la anotación *@Autowired*.

Sus parámetros son:

--class
[Requerido] [Clase Java] [Autocompletado]

Paquete y nombre donde crear la clase Java de servicio.

33.6.2. remote service operation

Añade un nuevo método en una clase Java. La clase Java puede ser una entidad, controlador, servicio o cualquier otro tipo.

Sus parámetros son:

--name
[Requerido] [Cadena]

Nombre del método a añadir en la clase Java.

--service
[Requerido] [Clase Java] [Autocompletado]

Paquete y nombre de la clase Java donde crear el nuevo método.

--return
[Opcional] [Tipo Java] [Autocompletado]

Tipo de dato Java que devolverá el método.

--paramNames
[Opcional] [Cadena]

Nombre de los parámetros de entrada del método, separados mediante comas y sin espacios.

--paramTypes
[Opcional] [Tipos Java] [Autocompletado]

Tipos Java de los parámetros de entrada del método, separados mediante comas y sin espacios.

--exceptions
[Opcional] [Cadena]

Tipos Java de las excepciones que puede lanzar el método, separados mediante comas y sin espacios.

33.6.3. remote service define ws

Hace que una clase Java pueda ser accedida desde el exterior a través de un servicio web. Para ello, creará toda la infraestructura necesaria para que sistemas externos puedan comunicarse con ella

mediante los protocolos y tecnologías propias de los servicios web. Si la clase indicada como parámetro no existe, será creada automáticamente.

Se recomienda no especificar en este comando ningún parámetro opcional si no es estrictamente necesario y se conoce las implicaciones que tendrá desde el punto de vista de la interoperabilidad de servicios web. Si no se especifica, para cada parámetro se proporcionará un valor adecuado para facilitar la máxima interoperabilidad del servicio web con el exterior.

Tras publicar una clase Java a través de un servicio web, sus métodos todavía no serán accesibles como operaciones del servicio web hasta que no se le indique para cada método.

Sus parámetros son:

--class
[Requerido] [Clase Java] [Autocompletado]

Paquete y nombre de la clase Java a hacer accesible desde el exterior a través de un servicio web. Si la clase Java no existe, será creada y además se le aplicará las características propias de un servicio local. Por ejemplo, *org.gvnix.test.service.Service*.

--serviceName
[Opcional] [Cadena]

Nombre con el que se quiere publicar el servicio web. Por defecto, tomará el mismo nombre que la clase Java. Por ejemplo, *Service*.

--portTypeName
[Opcional] [Cadena]

Nombre para el tipo de puerto o *PortType* del servicio web. Por defecto, tomará el mismo nombre que la clase Java seguido del sufijo *PortType*. Por ejemplo, *ServicePortType* .

--addressName
[Opcional] [Cadena]

Dirección URL relativa a la raíz de la aplicación desde donde se podrá acceder al servicio web. Por defecto, tomará el mismo nombre que la clase Java. Por ejemplo, *Service*.

--targetNamespace
[Opcional] [Cadena]

Espacio de nombres o *Namespace* del servicio web. Debe tener formato de URL. Por defecto, tendrá el mismo valor que el paquete de la clase Java en sentido inverso y con formato de URL. Por ejemplo, *http://service.test.gvnix.org/*.

33.6.4. remote service export operation

Hace que un método de una clase Java pueda ser accedido desde el exterior a través de una operación del servicio web. Para ello, creará toda la infraestructura necesaria para que sistemas externos puedan comunicarse con ella mediante los protocolos y tecnologías propias de los servicios web. Si la clase indicada como parámetro no está configurada todavía para ser accedida a través de un servicio web, se le aplicará esta configuración automáticamente.

Se recomienda no especificar en este comando ningún parámetro opcional si no es estrictamente necesario y se conoce las implicaciones que tendrá desde el punto de vista de la interoperabilidad de servicios web. Si no se especifica, para cada parámetro se proporcionará un valor adecuado para facilitar la máxima interoperabilidad de la operación del servicio web con el exterior.

Sus parámetros son:

--class
[Requerido] [Clase Java] [Autocompletado]

Paquete y nombre de la clase Java que contiene el método que se desea hacer accesible a través de una operación del servicio web. Por ejemplo, *org.gvnix.test.service.Service*.

--method
[Requerido] [Cadena]

Nombre del método que se desea hacer accesible a través de una operación del servicio web. Por ejemplo, *method*.

--operationName
[Opcional] [Cadena]

Nombre que tendrá la operación del servicio web y que dará acceso al método de la clase Java. Por defecto, tomará el mismo nombre que el método. Por ejemplo, *method*.

--resultName
[Opcional] [Cadena]

Nombre que tendrá el resultado de la operación del servicio web y que será de un tipo de datos equivalente al resultado del método. Por defecto, tomará el nombre *result*.

--resultNamespace
[Opcional] [Cadena]

Espacio de nombres o *Namespace* del resultado de la operación del servicio web. Debe tener formato de URL. Por defecto, tendrá el mismo valor que el paquete de la clase Java en sentido inverso y con formato de URL. Por ejemplo, *http://service.test.gvnix.org/*.

--responseWrapperName
[Opcional] [Cadena]

Nombre que tendrá el objeto que contendrá la respuesta de la operación del servicio web. Por defecto, tomará el mismo nombre que el método seguido del sufijo *Response* . Por ejemplo, *methodResponse*.

--responseWrapperNamespace
[Opcional] [Cadena]

Espacio de nombres o *Namespace* del objeto que contendrá el resultado de la operación del servicio web. Debe tener formato de URL. Por defecto, tendrá el mismo valor que el paquete de la clase Java en sentido inverso y con formato de URL. Por ejemplo, *http://service.test.gvnix.org/*.

--requestWrapperName
[Opcional] [Cadena]

Nombre que deberá tener el objeto que contendrá la petición a la operación del servicio web. Por defecto, tomará el mismo nombre que el método. Por ejemplo, *method*.

--requestWrapperNamespace
[Opcional] [Cadena]

Espacio de nombres o *Namespace* del objeto que contendrá el la petición a la operación del servicio web. Debe tener formato de URL. Por defecto, tendrá el mismo valor que el paquete de la clase Java en sentido inverso y con formato de URL. Por ejemplo, *http://service.test.gvnix.org/*.

33.6.5. remote service list operation

Muestra el listado de métodos de una clase Java que podrían ser ofrecidos como operaciones del servicio web. La clase Java debe haber sido previamente definida como accesible a través de un servicio web.

Sus parámetros son:

--class
[Requerido] [Clase Java] [Autocompletado]

Paquete y nombre de la clase Java de la que obtener el listado de métodos que podrían ser ofrecidos como operaciones del servicio web. Por ejemplo, *org.gvnix.test.service.Service* .

33.6.6. remote service export ws

Crea en el proyecto toda la infraestructura necesaria para ofrecer ciertas operaciones mediante un servicio web. El servicio web y sus operaciones vendrán definidas mediante un contrato de servicio o *WSDL*.

El comando creará una clase Java que representará en el proyecto al servicio web y creará dentro tantos métodos como operaciones hubiesen definidas en el contrato de servicio. Con esto, el servicio web y sus operaciones ya son accesibles desde el exterior. Será responsabilidad del desarrollador el implementar la lógica de negocio de cada método.

Sus parámetros son:

--wsdl
[Requerido] [Cadena]

Ruta al archivo que define el contrato de servicio o *WSDL* cuya infraestructura Java se desea crear en el proyecto. La ruta puede ser a un archivo local mediante *file://ruta*, a una dirección web mediante *http://ruta* o a una dirección web segura mediante *https://ruta*.

33.6.7. remote service import ws

Crea en el proyecto toda la infraestructura necesaria para acceder a ciertas operaciones situadas en un servicio web. El servicio web y sus operaciones vendrán definidas mediante un contrato de servicio o *WSDL*.

El comando creará una clase Java que representará en el proyecto al servicio web y creará dentro tantos métodos como operaciones hubiesen definidas en el contrato de servicio. Invocando a alguno de estos métodos se estará accediendo a la operación correspondiente del sistema externo.

Sus parámetros son:

--wsld
[Requerido] [Cadena]

Ruta al archivo que define el contrato de servicio o *WSDL* cuya infraestructura Java se desea crear en el proyecto. La ruta puede ser a un archivo local mediante *file://ruta*, a una dirección web mediante *http://ruta* o a una dirección web segura mediante *https://ruta*.

--class
[Requerido] [Clase Java] [Autocompletado]

Paquete y nombre de la clase Java que permitirá acceder al servicio web externo creando un método por cada una de las operaciones del servicio web. Si la clase Java no existe, será creada y además se le aplicará las características propias de un servicio local. Por ejemplo, *org.gvnix.test.service.Service*.

33.6.8. remote service ws list

Lista los servicios web que se encuentran definidos en la aplicación, tanto los que ofrecen servicios web al exterior como los que acceden a servicios web externos.

Este comando no tiene ningún parámetro obligatorio ni opcional.

33.6.9. remote service security ws

Añade o actualiza la configuración necesaria para firmar las peticiones que se hacen a un servicio web remoto.

Sus parámetros son:

--class
[Requerido] [Clase Java] [Autocompletado]

Paquete y nombre de la clase Java que define realiza el acceso al servicio web externo. Por ejemplo, *org.gvnix.test.service.Service*.

--certificate
[Requerido] [Cadena]

Ruta y nombre del fichero donde se encuentra el certificado a utilizar para firmar la petición. El certificado debe de tener un formato *pkcs12* y una extensión *p12*. El certificado será copiado al directorio de recursos del proyecto. Por ejemplo, */tmp/certificado.p12*.

--password
[Requerido] [Cadena]

Contraseña para acceder al certificado de firma.

--alias
[Requerido] [Cadena]

Alias a utilizar para la firma.

33.7. Comandos del add-on Web MVC i18n

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web MVC i18n.

Table 33.7. Comandos de add-on Web MVC i18n

Comando	Descripción
web mvc install language	Instala el soporte para un nuevo idioma en el proyecto

33.7.1. web mvc install language

Crea un fichero de propiedades con las traducciones de las etiquetas multi idioma del proyecto en el lenguage solicitado y configura en la capa web un acceso para cambiar a dicho idioma.

Sus parámetros son:

--code
[Requerido] [Idioma] [Autocompletado]

Código del idioma a instalar. Los idiomas disponibles son *de, en, es, it, nl y sv*.

33.8. Comandos del add-on Dynamic Configuration

Para una descripción detallada de las características del add-on consultar la documentación del add-on Dynamic Configuration.

Table 33.8. Comandos del add-on Dynamic Configuration

Comando	Descripción
configuration create	Crea una nueva configuración con un determinado nombre
configuration property add	Añade una propiedad a la gestión de configuraciones
configuration property value	Cambia el valor que tiene una propiedad en una determinada configuración
configuration property undefined	Deja sin valor una propiedad en una determinada configuración
configuration list	Lista todas las configuraciones definidas
configuration export	Escribe en el proyecto los cambios definidos hasta el momento en las configuraciones

33.8.1. configuration create

Crea una nueva configuración con un determinado nombre.

La primera configuración creada quedará marcada como la configuración por defecto de modo que en el caso de empaquetar la aplicación sin elegir ninguna configuración, será esta la que se aplique.

Sus parámetros son:

--name
[Requerido] [Cadena]

Nombre para la nueva configuración que la referenciará de forma unívoca.

33.8.2. configuration property add

Añade una propiedad a la gestión de configuraciones de modo que dicha propiedad podrá tener distinto valor para cada configuración.

Al añadir una nueva propiedad, quedará almacenado internamente el valor que tubiese en dicho momento. En todas las configuraciones existentes en ese momento y en aquellas que se creen en un futuro la propiedad tendrá por defecto dicho valor.

Sus parámetros son:

--name
[Requerido] [Propiedad] [Autocompletado]

Nombre de la propiedad cuyo valor puede tomar distinto valor en cada configuración.

33.8.3. configuration property value

Cambia el valor que tiene una propiedad en una determinada configuración.

Sus parámetros son:

--configuration
[Requerido] [Configuración] [Autocompletado]

Nombre de la configuración cuyo valor para una propiedad se quiere modificar.

--property
[Requerido] [Propiedad] [Autocompletado]

Nombre de la propiedad cuyo valor se desea modificar.

--value
[Requerido] [Cadena]

Nuevo valor para la propiedad en la configuración.

33.8.4. configuration property undefined

Deja sin valor una propiedad en una determinada configuración.

Esto es útil para evitar que quede almacenado en el código fuente del proyecto algunos valores críticos como puede ser la clave de conexión con la base de datos de producción. El valor se tendrá que proporcionar al empaquetar la aplicación con Maven como un parámetro mediante el modificador *-Dnombre.propiedad=valor*.

Sus parámetros son:

--configuration
[Requerido] [Configuración] [Autocompletado]

Nombre de la configuración cuyo valor para una propiedad se desea dejar sin valor.

--property
[Requerido] [Propiedad] [Autocompletado]

Nombre de la propiedad cuyo valor se desea dejar sin valor.

33.8.5. configuration list

Lista todas las configuraciones definidas junto con las propiedades incluidas y los valores definidos para cada una de ellas.

Este comando no tiene ningún parámetro obligatorio ni opcional.

33.8.6. configuration export

Escribe en el proyecto los cambios definidos hasta el momento en las configuraciones mediante todos los comandos anteriores utilizando para ello el sistema de perfiles de Maven. Mientras no se ejecute este comando, las configuraciones no serán aplicadas en el proyecto y por lo tanto no podrán ser utilizadas.

Este comando no tiene ningún parámetro obligatorio ni opcional.

33.9. Comandos del add-on Web MVC Binding

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web MVC Binding.

Table 33.9. Comandos del add-on Web MVC Binding

Comando	Descripción
web mvc binding stringTrimmer	Registra el StringTrimmerEditor.

33.9.1. web mvc binding stringTrimmer

Configura en un controlador concreto o en todos los controladores del proyecto un editor de propiedades para cadenas que elimina espacios al inicio y al fin y opcionalmente permite transformar cadenas vacías a valores nulos.

Sus parámetros son:

--class
[Opcional] [Clase Java] [Autocompletado]

Nombre de la clase Java controladora en la que registrar el editor. Si no se indica este parámetro, se registrará el editor en todos los controladores del proyecto.

--emptyAsNull
[Opcional] [Booleano] [Autocompletado]

Indica si las cadenas vacias deben convertirse en valores nulos.

33.10. Comandos del add-on Web MVC

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web MVC.

Table 33.10. Comandos de add-on Web MVC

Comando	Descripción
web mvc batch setup	Instala el soporte para la funcionalidad Web Batch
web mvc batch add	Genera los métodos de de persistencia en bloque para un controlador.
web mvc batch all	Genera los métodos de de persistencia en bloque para todos los controladores.
web mvc jquery setup	Instala los artefacto necesario para usar vistas usando la librería JavaScript jQuery.
web mvc jquery update tags	Actualiza los artefactos, ya instalados en el add-on, por los actuales.
web mvc jquery add	Convierte las vistas de un controlador a jQuery.
web mvc jquery all	Convierte todas las vistas a jQuery.

33.10.1. web mvc batch setup

Instala las dependencias necesarias y configura el proyecto para dar soporte a las peticiones de persistencia en bloque.

33.10.2. web mvc batch add

Añade los métodos al controlador para peticiones de persistencia en bloque.

Sus parámetros son:

--controller
[Requerido] [Clase controlador] [Autocompletado]

Controlador destino.

--service
[Opcional] [Clase de servicios de persistencia] [Autocompletado]

Añade los métodos al todos los controladores para peticiones de persistencia en bloque.

33.10.3. web mvc batch all

Crea el servicio de persistencia en bloque para todas las entidades de la aplicación.

33.10.4. web mvc jquery setup

Instala los artefactos necesario para usar vistas que use la librería jQuery.

33.10.5. web mvc jquery update tags

Actualiza todos los artefactos necesario para que el add-on funcione. Este comando es útil cuando se actualiza la versión del add-on y se desea utilizar los cambios en los tagx u otros artefactos de la nueva versión.

Advertencia: Al ejecutar este comando, *los artefactos del proyecto serán reescritos*. Cualquier cambio sobre los originales se perderá. Por lo tanto, es conveniente disponer de una *copia de seguridad o un sistema de control de versiones* para revisar las diferencias entre los artefactos anteriores y la actualización.

33.10.6. web mvc jquery add

Convierte las vistas de un controlador a jQuery.

Sus parámetros son:

--type
[Requerido] [Clase controlador] [Autocompletado]

Controlador cuyas vistas se desea convertir a jQuery.

33.10.7. web mvc jquery all

Convierte las vistas de todos los controladores a jQuery.

33.11. Comandos del add-on WEB MVC Bootstrap

Para una descripción detallada de las características del add-on consultar la documentación del add-on Bootstrap.

Table 33.11. Comandos de Addo-on Bootstrap

Comando	Descripción
web mvc bootstrap setup	Este comando importará e instalará todos los recursos necesarios para utilizar bootstrap. Además, modificará la estructura de algunos tagx existentes para que funcionen correctamente con Bootstrap y modificará todas las vistas de la aplicación para que utilicen los tags de JQuery.
web mvc bootstrap update	Este comando permite regenerar de forma correcta todos los componentes importados por Bootstrap. En caso de una modificación incorrecta de estos componentes, podremos regenerarlos para que vuelvan a su estado inicial.

33.11.1. web mvc bootstrap setup

Este comando importará e instalará todos los recursos necesarios para utilizar bootstrap. Además, modificará la estructura de algunos tagx existentes para que funcionen correctamente con Bootstrap y modificará todas las vistas de la aplicación para que utilicen los tags de JQuery.

33.11.2. web mvc bootstrap update

Este comando permite regenerar de forma correcta todos los componentes importados por Bootstrap. En caso de una modificación incorrecta de estos componentes, podremos regenerarlos para que vuelvan a su estado inicial.

33.12. Comandos del add-on Web MVC Datatables

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web MVC Datatables.

Table 33.12. Comandos del add-on Web MVC Datatables

Comando	Descripción
web mvc datatables setup	Instala los artefactos y dependencias requeridas para el funcionamiento del add-on.
web mvc datatables update tags	Actualiza los artefactos, ya instalados en el add-on, por los actuales.
web mvc datatables add	Añade el uso de datatables al controlador especificado.
web mvc datatables all	Añade el uso de Datatables a todos los controladores de la aplicación.
web web mvc datatables detail add	Añade un detalle a la vistas list de un controlador.

33.12.1. web mvc datatables setup

Instala todos los artefactos necesario para que el add-on funcione.

Los cambios en el proyecto son:

webapp/styles/datatables
: Css usadas por el widget.

webapp/images/datatables/
: Imágenes usadas por el widget.

webapp/scripts/datatables
: JavaScript usado por el widget.

webapp/WEB-INF/tags/datatables
: Tagx que adaptan los parámetros de los tags estándar a los de Dandelion-DataTables.

src/main/resources/datatables*.properties
: Fichero con opciones de configuración de Dandelion-DataTables y cadenas de internacionalización.

webapp/WEB-INF/tags/util/load-scripts.tagx
: Añade la carga de ficheros js y css a la páginas.

webapp/WEB-INF/web.xml
: Registra configuración necesaria para el uso de Dandelion-DataTables.

webapp/WEB-INF/spring/webmvc-config.xml
Registra configuración necesaria para el uso de Dandelion-DataTables.

33.12.2. web mvc datatables update tags

Actualiza todos los artefactos necesario para que el add-on funcione. Este comando es útil cuando se actualiza la versión del add-on y se desea utilizar los cambios en los tagx u otros artefactos de la nueva versión.

Advertencia: Al ejecutar este comando, *los artefactos del proyecto serán reescritos*. Cualquier cambio sobre los originales se perderá. Por lo tanto, es conveniente disponer de una *copia de seguridad o un sistema de control de versiones* para revisar las diferencias entre los artefactos anteriores y la actualización.

33.12.3. web mvc datatables add

Añade el uso de Datatables al controlador especificado.

Sus parámetros son:

--type
[Requerido] [Clase Java] [Autocompletado]

Nombre de la clase Java controladora a la que se aplicará los cambios.

--ajax
[Opcional] [Booleano] [Autocompletado]

Establece el modo de datos a AJAX (por defecto) o a DOM (cuando es *false*).

--mode
[Opcional] [Cadena]

Muestra la vista de la entidad que elijamos. En caso de utilizar la vista de "show" se establece el modo de visualización a modo registro. Si se utiliza "--mode list" se estará mostrando la página list.jspx asociada a la entidad.

--inline
[Opcional] [Boolean] [Autocompletado]

Habilita la funcionalidad edición en línea.

33.12.4. web mvc datatables all

Añade el uso de Datatables a todos los controladores de la aplicación, usando los valores por defecto del comando web mvc jquery add.

33.12.5. web mvc datatables details add

Un detalle a la vista list para en base a una propiedad *1 a N* de la entidad relacionada (ver descripción en Visualización de detalles).

Sus parámetros son:

--type
[Requerido] [Clase Java] [Autocompletado]

Nombre de la clase Java controladora a la que se aplicará los cambios. Este controlador ya debe estar usando la funcionalidad *dataTables*

--property
[Requerido] [Nombre de propiedad]

Nombre de la propiedad *1 a N* de la entidad relación con el controlador a usar para el detalle.

33.13. Comandos del add-on Web MVC GEO

Para una descripción detallada de las características del add-on consultar la documentación del add-on Web MVC GEO.

Table 33.13. Comandos del add-on Web MVC GEO

Comando	Descripción
web mvc geo setup	Instala los artefactos y dependencias requeridas para el funcionamiento del add-on.
web mvc geo controller	Genera una nueva vista de Mapa en nuestro proyecto
web mvc geo field	Transforma los campos de texto generados por defecto, a campos de tipo mapa para facilitar el guardado de campos geográficos.
web mvc geo base layer field	Añade una capa base para mostrar en el mapa de un campo geográfico.
web mvc geo group	Añade una nueva agrupación de capas a la vista de mapa que se indique.
web mvc geo entity all	Añade todas las entidades con campos de tipo GEO a la vista de mapa.
web mvc geo entity add	Añade la entidad seleccionada a la vista de mapa.
web mvc geo entity simple	Añade el campo de tipo GEO seleccionado de la entidad indicada a la vista de mapa.
web mvc geo tilelayer	Añade una capa base de tipo Tile a la vista del mapa
web mvc geo wmslayer	Añade una capa base de tipo WMS a la vista del mapa
web mvc geo wmtslayer	Añade una capa base de tipo WMTS a la vista del mapa
web mvc geo tool measure	Añade una nueva herramienta de medición a la vista del mapa
web mvc geo tool custom	Añade una nueva herramienta personalizada a la vista del mapa
web mvc geo component overview	Añade el componente mini mapa a la vista del mapa

33.13.1. web mvc geo setup

Instala todos los artefactos necesario para que el add-on funcione.

33.13.2. web mvc geo controller

Genera una vista de mapa en nuestro proyecto. Es posible generar tantas vistas de mapa como se necesiten. Cada una de ellas será independiente y podrán personalizarse de manera individual

Sus parámetros son:

--class
[Requerido] [Clase Java]

Nombre de la clase Java controladora que se desea crear.

--preferredMapping
[Requerido] [String]

Path que se quiere utilizar en el controller

--projection
[Opcional] [CRS] [Autocompletado]

Proyección con la que trabajará el mapa generado

33.13.3. web mvc geo field

Por defecto, a la hora de guardar campos de tipo geográfico se utilizará formato WKT que tendrá que ser introducido de forma manual por el usuario.

Sin embargo, este add-on permite transformar estos campos de texto a componentes de tipo Mapa, gracias a los cuales se facilita la introducción de datos de tipo GEO en una entidad.

Sus parámetros son:

--controller
[Requerido] [Clase Java] [Autocompletado]

Controlador Java asociado a la entidad sobre la cual queremos actualizar campo de tipo input a campo de tipo mapa.

--field
[Requerido] [String]

Campo que queremos modificar

--center
[Opcional] [String]

Configura el centro del mapa que se va a generar. Formato 'lat, lng'

--color
[Opcional] [String]

Color con el que se pintarán los datos en el mapa

--maxZoom
[Opcional] [String]

Nivel de Zoom máximo que se aplicará al mapa generado

--weight
[Opcional] [String]

Grosor de la linea que se utilizará para pintar sobre el mapa

--zoom
[Opcional] [String]

Nivel de Zoom inicial con el que aparecerá el mapa

33.13.4. web mvc geo base layer field

Añade una capa base para mostrar en el mapa de un campo geográfico.

Sus parámetros son:

--controller
[Requerido] [Clase Java]

Selecciona el controlador asociado a la entidad en la que se encuentra el campo geográfico al que se le quiere añadir la capa base.

--field
[Requerido] [String]

Selecciona el campo geográfico al que añadir una capa base.

--url
[Requerido] [String]

URL del servidor de mapas que devolverá la capa base.

--type
[Requerido] [String]

Indica el tipo de la capa base que se va a añadir ('Tile', 'WMS' o 'WMTS').

--labelCode
[Opcional] [String]

Código i18n para identificar la etiqueta que se establece sobre el grupo en el fichero de propiedades del lenguaje.

--label
[Opcional] [String]

Texto utilizado como etiqueta para identificar el grupo.

33.13.5. web mvc geo group

Añade una nueva agrupación de capas a la vista de mapa que se indique.

Sus parámetros son:

--name
[Requerido] [String]

Nombre con el que se identifica el grupo generado.

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se asocia el grupo. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

--group
[Opcional] [String]

Nombre que identifica al grupo que se establece como padre del que se está definiendo.

--index
[Opcional] [String]

Indica la posición en la que el grupo debe ser mostrado dentro del mapa.

--label
[Opcional] [String]

Texto utilizado como etiqueta para identificar el grupo.

--labelCode
[Opcional] [String]

Código i18n para identificar la etiqueta que se establece sobre el grupo en el fichero de propiedades del lenguaje.

33.13.6. web mvc geo entity all

Añade todas las entidades con campos de tipo GEO a la vista de mapa

Sus parámetros son:

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorporan las entidades con campos GEO. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

33.13.7. web mvc geo entity add

Añade la entidad seleccionada a la vista de mapa.

Sus parámetros son:

--controller
[Requerido] [Clase Java] [Autocompletado]

Selecciona el controlador asociado a la entidad que quieres añadir a la vista de Mapa.

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora la entidad seleccionada. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

33.13.8. web mvc geo entity simple

Añade el campo de tipo GEO seleccionado de la entidad indicada a la vista de mapa.

Sus parámetros son:

--controller
[Requerido] [Clase Java] [Autocompletado]

Selecciona el controlador asociado a la entidad que quieres añadir a la vista de Mapa.

--field
[Requerido] [String]

Indica el campo asociado de la entidad definida mediante el parámetro *controller* que quieres añadir a la vista de Mapa.

--pk
[Requerido] [String]

Indica la clave primaria perteneciente a la entidad definida mediante el parámetro *controller*

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora la entidad seleccionada. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

--index
[Opcional] [String]

Posición en la que aparecerá la nueva capa definida. Si no se introduce ningun indice, se generará uno automáticamente para evitar erorres en la visualización de capas.

--group
[Opcional] [String]

Indica el grupo en el cual se incluye la nueva capa definida.

--label
[Opcional] [String]

Texto utilizado como etiqueta para identificar la nueva capa definida.

--labelCode
[Opcional] [String]

Código i18n para identificar la etiqueta que se establece sobre la nueva capa definida en el fichero de propiedades del lenguaje.

33.13.9. web mvc geo tilelayer

Añade una capa base de tipo Tile a la vista del mapa

Sus parámetros son:

--name
[Requerido] [String]

Indica el nombre de la nueva Capa Base. Este nombre debe ser único para cada una de las capas añadidas.

--url
[Requerido] [String]

URL del servidor de mapas que devolverá la capa base

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora la nueva capa base. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

--index
[Opcional] [String]

Posición en la que aparecerá la capa base. Si no se introduce ningun indice, se generará uno automáticamente para evitar erorres en la visualización de capas.

--opacity
[Opcional] [Number]

Indica la opacidad que tendrá la capa. Podemos indicar valores entre el 0 y el 1. 1 significa que no es transparente y 0 que es totalmente transparente.

--group
[Opcional] [String]

Indica el grupo en el cual se incluye la nueva capa base definida.

33.13.10. web mvc geo wmslayer

Añade una capa base de tipo WMS a la vista del mapa

Sus parámetros son:

--name
[Requerido] [String]

Indica el nombre de la nueva Capa WMS. Este nombre debe ser único para cada una de las capas añadidas.

--url
[Requerido] [String]

URL del servidor de mapas que devolverá la capa WMS

--layers
[Requerido] [String]

Indica qué capas quieres obtener del servicio WMS

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora la nueva capa WMS. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

--index
[Opcional] [String]

Posición en la que aparecerá la capa base. Si no se introduce ningun indice, se generará uno automáticamente para evitar erorres en la visualización de capas.

--opacity
[Opcional] [Number]

Indica la opacidad que tendrá la capa. Podemos indicar valores entre el 0 y el 1. 1 significa que no es transparente y 0 que es totalmente transparente.

--format
[Opcional] [String]

Indica en qué formato se obtienen las imágenes del servicio WMS. Formato 'image/png'

--transparent
[Opcional] [Boolean]

Indica si la imágen que devuelve tiene transparencia o no

--styles
[Opcional] [String]

Indica qué estilos quieres obtener para cada una de las capas

--version
[Opcional] [String]

Versión del servicio WMS

--crs
[Opcional] [String]

Indica en qué proyección se obtiene la capa WMS. Por defecto EPSG3857

--label
[Opcional] [String]

Texto utilizado como etiqueta para identificar la capa WMS.

--labelCode
[Opcional] [String]

Código i18n para identificar la etiqueta que se establece sobre la capa WMS en el fichero de propiedades del lenguaje.

--group
[Opcional] [String]

Indica el grupo en el cual se incluye la nueva capa WMS definida.

33.13.11. web mvc geo wmtslayer

Añade una capa base de tipo WTMS a la vista del mapa

Sus parámetros son:

--name
[Requerido] [String]

Indica el nombre de la nueva Capa WTMS. Este nombre debe ser único para cada una de las capas añadidas.

--url
[Requerido] [String]

URL del servidor de mapas que devolverá la capa WMTS

--layer
[Requerido] [String]

Indica qué capa quieres obtener del servicio WMTS

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora la nueva capa WMTS. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

--index
[Opcional] [String]

Posición en la que aparecerá la capa base. Si no se introduce ningun indice, se generará uno automáticamente para evitar erorres en la visualización de capas.

--opacity
[Opcional] [Number]

Indica la opacidad que tendrá la capa. Podemos indicar valores entre el 0 y el 1. 1 significa que no es transparente y 0 que es totalmente transparente.

--label
[Opcional] [String]

Texto utilizado como etiqueta para identificar la capa WMTS.

--labelCode
[Opcional] [String]

Código i18n para identificar la etiqueta que se establece sobre la capa WMTS en el fichero de propiedades del lenguaje.

--group
[Opcional] [String]

Indica el grupo en el cual se incluye la nueva capa WMTS definida.

--addCheckBox
[Opcional] [Boolean] [Autocompletado]

Indica si se muestra un check en el cuadro de opciones del mapa que permita seleccionar mostrar o no la capa WMTS.

33.13.12. web mvc geo tool measure

Añade una nueva herramienta de medición a la vista del mapa

Sus parámetros son:

--name
[Requerido] [String]

Indica el nombre de la herramienta de medición. Este nombre es único y cada una de las herramientas deben tener el suyo propio.

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora la herramienta medición. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

--preventExitMessageCode
[Opcional] [String]

Código del mensaje multi-idioma que aparecerá al cambiar de herramienta para evitar que se pierdan los cambios aplicados sobre el mapa.

33.13.13. web mvc geo tool custom

Añade una nueva herramienta personalizada a la vista del mapa

Sus parámetros son:

--name
[Requerido] [String]

Indica el nombre de la herramienta de medición. Este nombre es único y cada una de las herramientas deben tener el suyo propio.

--icon
[Requerido] [String]

Indica el icono que mostrará la herramienta personalizada

--activateFunction
[Requerido] [String]

Nombre de la función Javascript que se ejecutará al seleccionar la herramienta personalizada. Si la función devuelve *false*, la herramienta no será seleccionada

--deactivateFunction
[Requerido] [String]

Nombre de la función Javascript que se ejecutará al cambiar de herramienta si esta se encuentra seleccionada. Si la función devuelve *false*, la herramienta se mantendrá seleccionada

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora la herramienta personalizada. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

--iconLibrary
[Opcional] [String]

Indica qué librería se utilizará para representar el icono.

--actionTool
[Opcional] [Boolean]

Indica si la herramienta es únicamente clicable (*true*) o si puede mantenerse seleccionada (*false*)

--cursorIcon
[Opcional] [String]

Indica el icono que tendrá el cursor sobre el mapa al activar esta herramienta.

--preventExitMessageCode
[Opcional] [String]

Código del mensaje multi-idioma que aparecerá al cambiar de herramienta para evitar que se pierdan los cambios aplicados sobre el mapa.

33.13.14. web mvc geo component overview

Añade el componente mini mapa a la vista del mapa

Sus parámetros son:

--class
[Opcional] [Clase Java] [Autocompletado]

Selecciona el controlador que incluye el mapa al que se incorpora el mini mapa. En caso de no indicar el parámetro, se esteblece sobre el controlador que tiene el foco.

33.14. Comandos del add-on WEB MVC Lupa

Para una descripción detallada de las características del add-on consultar la documentación del add-on Lupa.

Table 33.14. Comandos de Addo-on Componentes Lupa

Comando	Descripción
web mvc loupe setup	Este comando importará a nuestro proyecto todos los recursos necesarios para utilizar componentes lupa.
web mvc loupe set	Este comando nos permite generar los métodos necesarios para la búsqueda y visualización de registros del componente lupa en el controlador seleccionado.
web mvc loupe field	Este comando nos permite transformar a componente lupa cualquier campo válido de la entidad a la que está asociado el Controlador en el que se han generado los métodos de lupa.
web mvc loupe update	Este comando nos permite mantener actualizados todos los componentes importados mediante el comando de instalación.

33.14.1. web mvc loupe setup

Este comando importará a nuestro proyecto todos los recursos necesarios para utilizar componentes lupa.

33.14.2. web mvc loupe set

Este comando nos permite generar los métodos necesarios para la búsqueda y visualización de registros del componente lupa en el controlador seleccionado.

Sus parámetros son:

--controller
[Requerido] [Cadena]

Nombre y ruta del Controlador sobre el cual queremos generar los nuevo métodos de búsqueda para el componente lupa.

33.14.3. web mvc loupe field

Este comando nos permite transformar a componente lupa cualquier campo válido de la entidad a la que está asociado el Controlador en el que se han generado los métodos de lupa.

Sus parámetros son:

--controller
[Requerido] [Cadena]

Nombre y ruta del Controlador que ya dispone de los métodos generados.

--field
[Requerido] [Cadena]

Nombre del campo que queremos transformar a componente lupa en la vista de creación y actualización de la capa web.

--additionalFields
[Cadena]

Campos adicionales de la entidad sobre los que se realizará la búsqueda cuando el usuario escriba en el componente lupa. Si no se especifica ningún valor solo será posible buscar por id.

--baseFilter
[Cadena]

Filtro base que se aplica a todas las búsquedas realizadas en el componente lupa.

--caption
[Cadena]

Campo de la entidad relacionada que se mostrará en el desplegable del componente lupa cuando se encuentré un resultado.

--listPath
[Cadena]

Opcionalmente, añadiendo la ruta del fichero a este comando, se puede mostrar un listado distinto al generado por defecto para cada controlador.

--max
[Cadena]

Se especifica el número máximo de resultados que se quiere mostrar en el desplegable del componente lupa. Por defecto se muestran los 3 primeros resultados obtenidos.

33.14.4. web mvc loupe update

Este comando mantiene actualizados todos los componentes que se importan al proyecto gvNIX al realizar la instalación del add-on.

33.15. Comandos del add-on Monitoring

Para una descripción detallada de las características del add-on consultar la documentación del add-on Monitoring.

Table 33.15. Comandos del add-on Monitoring

Comando	Descripción
monitoring setup	Instala las dependencias de JavaMelody además de configurar el proyecto con lo necesario para su funcionamiento.
monitoring all	Establece todas las clases del proyecto para ser monitorizadas a través de Spring.
monitoring add class	Añade la monitorización a una clase determinada.
monitoring add method	Añade la monitorización de un método de una clase. Es posible que si se trata de una clase controlador y el método que deseamos monitorizar está incluido en el .aj no funcione, para solucionarlo bastará con hacer push-in del método previo a ejecutar el comando.
monitoring add package	Añade la monitorización a un paquete determinado.

33.15.1. monitoring setup

Instala las dependencias de JavaMelody además de configurar el proyecto con lo necesario para su funcionamiento.

Sus parámetros son:

--path
[Cadena]

Ruta de la carpeta donde se guardaran los datos de la monitorización dentro del servidor.

33.15.2. monitoring all

Establece todas las clases del proyecto para ser monitorizadas a través de Spring.

33.15.3. monitoring add class

Añade la monitorización a una clase determinada.

Sus parámetros son:

--name
[Requerido] [Cadena]

Ruta de la clase sobre la que se va a establecer la monitorización.

33.15.4. monitoring add method

Añade la monitorización de un método de una clase. Es posible que si se trata de una clase controlador y el método que deseamos monitorizar está incluido en el .aj no funcione, para solucionarlo bastará con hacer push-in del método previo a ejecutar el comando.

Sus parámetros son:

--name
[Requerido] [Cadena]

Nombre del método de la clase que se va a establecer la monitorización.

--class
[Requerido] [Cadena]

Ruta de la clase sobre cuyo método se va a establecer la monitorización.

33.15.5. monitoring add package

Añade la monitorización a un paquete determinado.

Sus parámetros son:

--path
[Requerido] [Cadena]

Ruta del paquete sobre cuyas clases se va a establecer la monitorización.

33.16. Comandos del add-on WEB MVC fancytree View

Para una descripción detallada de las características del add-on consultar la documentación del add-on fancytree.

Table 33.16. Comandos de Add-on componentes visuales tipo Árbol

Comando	Descripción
web mvc fancytree setup	Este comando importará a nuestro proyecto todos los recursos necesarios para utilizar el addon Fancytree.
web mvc fancytree update tags	Este comando mantiene actualizados todos los componentes que se importan al proyecto gvNIX al realizar la instalación del add-on
web mvc fancytree add show	Este comando nos permite añadir a una clase controller la plantilla de un método con la que añadir los datos y las propiedades de un árbol, y las servirá por petición Ajax para su carga y visualización.
web mvc fancytree add edit	Este comando nos permite añadir a una clase controller las plantillas de métodos para gestionar la edición de un árbol mediante peticiones Ajax.

33.16.1. web mvc fancytree setup

Este comando importará a nuestro proyecto todos los recursos necesarios para utilizar el addon Fancytree.

33.16.2. web mvc fancytree add show

Este comando nos permite añadir a una clase controller la plantilla de un método con la que añadir los datos y las propiedades de un árbol, y las servirá por petición Ajax para su carga y visualización.

Sus parámetros son:

--controller
[Requerido] [Cadena]

Nombre y ruta del Controlador sobre el cual queremos añadir el método para crear componente a#bol. Si no se especifica se asumirá que es la clase actual (que tiene actualmente el foco).

--page
[Cadena]

Página adicional donde se mostrará el componente de árbol.

--mapping
[Cadena]

Ruta (URL) relativa al controller donde responderá el método. Si se omite será el valor de page (si ha sido especificado) o la ruta raíz del controller..

33.16.3. web mvc fancytree add edit

Este comando nos permite añadir a una clase controller las plantillas de métodos para gestionar la edición de un árbol mediante peticiones Ajax.

Sus parámetros son:

--controller
[Requerido] [Cadena]

Nombre y ruta del Controlador sobre el cual queremos añadir el método para crear componente a#bol. Si no se especifica se asumirá que es la clase actual (que tiene actualmente el foco).

--page
[Cadena]

Página adicional donde se mostrará el componente de árbol.

--mapping
[Cadena]

Ruta (URL) relativa al controller donde responderá el método. Si se omite será el valor de page (si ha sido especificado) o la ruta raíz del controller..

33.16.4. web mvc fancytree update tags

Este comando mantiene actualizados todos los componentes que se importan al proyecto gvNIX al realizar la instalación del add-on.

Parte VI. Recursos

Chapter 34. Recursos

En este apartado pondremos una lista de recursos de documentación, tutoriales, etc que pueden ser útiles.

34.1. Proyectos relacionados con gvNIX

Proyectos relacionados con gvNIX y con la Consejería de Infraestructuras, Transporte y Medio Ambiente de la Generalidad Valenciana.

- gvPONTIS
- gvSIG
- gvHIDRA
- Moskitt

34.2. Recursos de Spring Roo

Esta es una lista de recursos relacionados con Spring Roo:

- Documentación de referencia Spring Roo (*on-line*)
- Página del proyecto
- Foro de Spring Roo
- JIRA de Spring Roo
- Spring STS (Spring Tools Suite)
- Página con recursos para Roo

34.3. Recursos de librerías relacionadas

Enlaces a recursos de librerías relacionadas con los proyectos generados.

- Spring
- Documentación de Spring Framework
- Documentación de Spring MVC
- Spring Security
- AspectJ (Aspectos Java)
- AJDT
- jQuery
- jQueryUI

- DatatTables
- Dandelion-DatatTables
- jQuery Validation Plugin