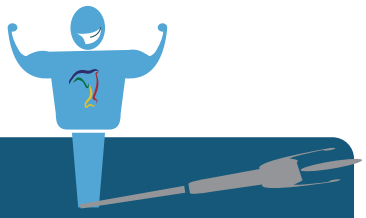
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

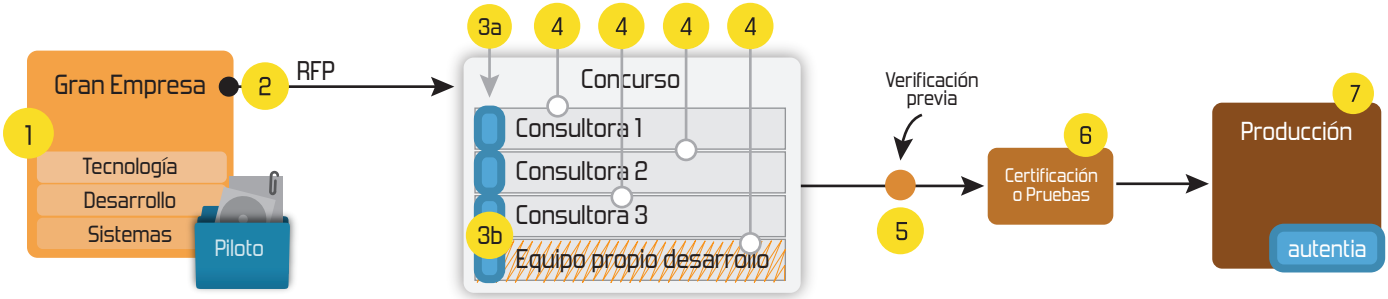
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Contacte con Cristóbal González
criskerberos-tutoriales@yahoo.com

Cristóbal González AlmirónConsultor de desarrollo de proyectos informáticos. Su experiencia profesional se ha desarrollado en empresas como Compaq, HP, Mapfre, Endesa, Repsol, Universidad Autónoma de Madrid, en las áreas de Desarrollo de Software (Orientado a Objetos), tecnologías de Internet, Técnica de Sistemas de alta disponibilidad y formación a usuarios.

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2014-07-02 Tutorial visitado 1 veces Descargar en PDF

Introducción a Groovy y Grails con Maven: el patrón CRUD

Índice de contenidos

Resumen

En este artículo echaremos un primer vistazo a Grails, el servidor de aplicaciones para el lenguaje Groovy. Además haremos un pequeño ejemplo implementando el patón CRUD (consultas, altas, bajas y modificaciones), que es la base de cualquier aplicación de gestión.

Para hacer esto más sencillo vamos a utilizar como plataforma de desarrollo Maven, ya que nos permite poner en marcha el proyecto de ejemplo en poco tiempo.

1. Introducción a Grails y Groovy

Groovy es un lenguaje interpretado basado en Java al estilo de Ruby y Grails es un servidor de aplicaciones escritas en Groovy al estilo de Ruby on Rails o de PHP con Simphony. Tiene bastantes características interesantes, como es su compatibilidad con Java (podemos usar libremente las clases de java en nuestros proyectos), tipado fuerte y débil y una sintaxis rápida y sencilla, que utiliza bastantes convenios que a aquellos que venimos de Java al principio nos chocan.

Grails es un servidor de aplicaciones que permite ejecutar aplicaciones escritas en Groovy. Algunas de sus características relevantes son:

- Usa convención frente a configuración en para saber qué tiene que hacer. Por ejemplo si una clase Book.grrovy está en grails-app/domain, ya sabe que esa clase declara una entidad del modelo de datos.
- Está preparado para ejecutar aplicaciones que siguen el patrón MVC modelo vista controlador.
- El código fuente es interpretado, basado en ficheros escritos en lenguaje Groovy y páginas web generadas mediante plantillas GSP, que son bastante parecidas a los tradicionales JSPs de J2EE.
- Arquitectura basada en plugins escritos en Groovy. Esto le permite extender su funcionalidad.
- Acceso directo a todos los paquetes java estándar, con soporte para Maven
- Gestor de ciclo de vida basado en Gradle, que es bastante compatible con Maven, aunque no sigue su misma filosofía.
- Soporte nativo para mapeo de bases de datos a Groovy mediante el GORM (basado en Hibernate), con gestión automática de la base de datos subyacente.
- Sistema de generación de modelo, vista y controladores para implementar un CRUD integrado en Grails
- Y por último, las aplicaciones generadas en Grails se pueden empaquetar como WAR y desplegar en un servidor de aplicaciones estándar o en un Tomcat.

2. Requisitos iniciales

El ejemplo lo voy a realizar sobre Ubuntu 14.04 Desktop, que podemos instalar en una máquina virtual usando VirtualBox con mucha facilidad. Ya sabéis, las pruebas con gaseosa y en casa.

3. Generación del proyecto base mediante el plugin Archetype de Maven

Desde Ubuntu, en principio instalamos Maven 3 con este comando:

```
$ sudo apt-get install openjdk-7-jdk
$ sudo apt-get install maven
$ mvn --version
Warning: JAVA_HOME environment variable is not set.
Apache Maven 3.0.5
Maven home: /usr/share/maven
Java version: 1.7.0_55, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-openjdk-i386/jre
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "3.13.0-29-generic", arch: "i386", family: "unix"
```

Si no lo teníamos instalado nos instalará todo el JDK para Java y dejará todo listo para comenzar a crear nuestra aplicación Fácil. Limpio. Rápidoâ€¦. Pero tiene un problema: en Ubuntu 14.04 nos instala Apache Maven 3.0.5, que no nos vale, ya que el plugin de Grails para Maven requiere Maven 3.1 o posterior. Por ello vamos a instalar una más reciente, por ejemplo Maven 3.2.1 sobre nuestro Ubuntu.

Para instalar una versión más reciente hay que hacer varios pasos:

```
$ cd ~
$ wget http://apache.rediris.es/maven/maven-3/3.2.1/binaries/apache-maven-3.2.1-bin.tar.gz
$ sudo mkdir -p /usr/local/apache-maven
$ sudo mv apache-maven-3.2.1-bin.tar.gz /usr/local/apache-maven/
$ cd /usr/local/apache-maven/
$ sudo tar -xzvf apache-maven-3.2.1-bin.tar.gz
$ cd ~
$ nano .profile
```

Ahora editamos el .profile y le añadimos estas líneas al final:

```
export M2_HOME=/usr/local/apache-maven/apache-maven-3.2.1
export M2=$M2_HOME/bin
export MAVEN_OPTS="-Xms256m -Xmx512m"
export PATH=$M2:$PATH
```

Cerramos la sesión y volvemos a iniciarla, para coger los cambios del .profile. Y probamos el nuevo Maven:

```
$ mvn --version
Apache Maven 3.2.1 (ea8b2b07643dbb1b84b6d16e1f08391b666bc1e9; 2014-02-14T18:37:52+01:00)
Maven home: /usr/local/apache-maven/apache-maven-3.2.1
Java version: 1.7.0_55, vendor: Oracle Corporation
Java home: /usr/lib/jvm/java-7-openjdk-i386/jre
Default locale: es_ES, platform encoding: UTF-8
OS name: "linux", version: "3.13.0-29-generic", arch: "i386", family: "unix"
```

4. Creación del proyecto base desde el arquetipo Maven

Para comenzar creamos un directorio para el proyecto:

```
$ mkdir -p ~/workspaces/workspaceGrailsdemo
$ cd ~/workspaces/workspaceGrailsdemo
```

NOTA: esta estructura de carpetas está especialmente diseñada para trabajar con Eclipse y Netbeans. Mi consejo es:

- Meter todos vuestros desarrollos en la carpeta workspaces
- Por cada proyecto o desarrollo crear dentro de ella una carpeta workspacenombreproyecto y dentro de ella las carpetas de los proyectos
- Esto último facilita mucho abrir los proyectos en Eclipse y luego importarlos

Mediante el uso del arquetipo Grails para Maven podemos generar de manera sencilla el proyecto base. El arquetipo es generado mediante un plugin de Maven que se descarga automáticamente del repositorio central de Maven. Yo voy a usar en este tutorial una versión concreta del arquetipo, pero se pueden usar otras versiones, es cuestión de ir probando lo que nos funciona.

Nota importante: la versión elegida del plugin es muy importante, ya que no todas funcionanâ€¦. Por ejemplo yo he probado la versión 2.3.8 y con pruebas sencillas no he logrado levantar la aplicación con la configuración por defecto. La versión 2.2.4 inicializa y ejecuta la aplicación sin problemas, por lo que para tomar contacto con Grails es la apropiada.

Podemos saber la lista de versiones disponibles consultando directamente el directorio del plugin en el repositorio central de Maven. ¿no sabes cómo buscarlo? La ruta para encontrar un artefacto o plugin en el repositorio central de Maven es:

http://repo1.maven.org/maven/GROUP-ID/ARTIFACT-ID

Donde:

- GROUP-ID es el groupId convirtiendo los puntos por barras d, por ejemplo si el gropuId es org.grails, el GROUP-ID será org/grails
- El ARTIFACT-ID es el artifactId que tenemos que usar.
- NOTA: esto vale para CUALQUIER dependencia Maven

Y el comando a ejecutar para este arquetipo es:

```
$ mvn archetype:generate -DarchetypeGroupId=org.grails -DarchetypeArtifactId=grails-maven-archetype -DarchetypeVersion=2.2.4 -Dgrou
[INFO] Scanning for projects...
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building Maven Stub Project (No POM) 1
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] >>> maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom >>>
```

Últimos Tutoriales del Autor

» Menu.bat Una forma cómoda de ejecutar comandos y aplicaciones, por ejemplo para Maven

» Ejemplo JSF 2.0 con MyFaces Trinidad, Ajax y Maven

» Acciones JSF basadas en clases internas de Java

» Rendimiento en espacio y transferencia de un servidor Subversion

» Instalación de subversion

```
[INFO]
[INFO] <<< maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom <<<
[INFO]
[INFO] --- maven-archetype-plugin:2.2:generate (default-cli) @ standalone-pom ---
[INFO] Generating project in Interactive mode
[INFO] Archetype repository missing. Using the one from [org.grails:grails-maven-archetype:2.3.8] found in catalog remote
[INFO] Using property: groupId = es.com.gonzalezalmiron.grailsdemo
[INFO] Using property: artifactId = grailsdemo
Define value for property 'version':  1.0-SNAPSHOT: :
[INFO] Using property: package = es.com.gonzalezalmiron.grailsdemo
Confirm properties configuration:
groupId: es.com.gonzalezalmiron.grailsdemo
artifactId: grailsdemo
version: 1.0-SNAPSHOT
package: es.com.gonzalezalmiron.grailsdemo
 Y: : y
[INFO] ----------------------------------------------------------------------------
[INFO] Using following parameters for creating project from Old (1.x) Archetype: grails-maven-archetype:2.2.4
[INFO] ----------------------------------------------------------------------------
[INFO] Parameter: groupId, Value: es.com.gonzalezalmiron.grailsdemo
[INFO] Parameter: packageName, Value: es.com.gonzalezalmiron.grailsdemo
[INFO] Parameter: package, Value: es.com.gonzalezalmiron.grailsdemo
[INFO] Parameter: artifactId, Value: grailsdemo
[INFO] Parameter: basedir, Value: /home/usuario/workspaces/workspaceGrailsdemo
[INFO] Parameter: version, Value: 1.0-SNAPSHOT
[INFO] project created from Old (1.x) Archetype in dir: /home/usuario/workspaces/workspaceGrailsdemo/grailsdemo
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 19.921s
[INFO] Finished at: Tue Jun 10 11:07:09 CEST 2014
[INFO] Final Memory: 10M/30M
[INFO] ------------------------------------------------------------------------
```

Si no te has dado cuenta ya, la ruta del arquetipo en Maven es:

http://repo1.maven.org/maven2/org/grails/grails-maven-archetype/

Y veremos una carpeta por cada versión del plugin.

4.1. Inicializando la aplicación

Tras la ejecución del plugin de archetype lo único que tenemos en grailsdemo es un pom.xml y una carpeta src casi vacía, salvo el web.xml que hay en src/main/webapp/WEB-INF. Parece poco, pero como siempre hay que terminar de trabajar en Maven. Inicializamos el proyecto.

```
~/workspaces/workspaceGrailsdemo/grailsdemo$ mvn initialize
 [INFO] Scanning for projects...

â€¦. Descarga un montón de paquetes desde el repositorio centralâ€¦..


[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building A custom grails project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- grails-maven-plugin:2.2.4:validate (default-validate) @ grailsdemo ---
[INFO] No Grails application found - skipping validation.
[INFO]
[INFO] --- grails-maven-plugin:2.2.4:init (default-init) @ grailsdemo ---
[INFO] Cannot read application info, so initialising new application.
[WARNING] Grails Start with out fork

|Loading Grails 2.2.4
|Configuring classpath
|Running pre-compiled script
.
|Environment set to development
.......................................
|Created Eclipse project files.
.
|Created Grails Application at /home/usuario/workspaces/workspaceGrailsdemo/grailsdemo
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 17.625s
[INFO] Finished at: Tue Jun 10 11:11:38 CEST 2014
[INFO] Final Memory: 21M/51M
[INFO] ------------------------------------------------------------------------
```

4.2. Instalando las plantillas utilizadas por defecto

Este paso es un poco curioso. Lo que hace es instalar las plantillas en la carpeta /src/templates que luego se usan para generar los GSPs. Conviene hacerlo en este momento, pues luego cuando se instala el plugin de Fields ya no se puede hacer (por lo menos a mi me ha dado problemas):

```
$ mvn grails:install-templates
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for es.com.gonzalezalmiron.grailsdemo:grailsdemo:grails-
[WARNING] 'dependencies.dependency.(groupId:artifactId:type:classifier)' must be unique: org.grails.plugins:cache:zip -> duplicate d
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 143, column 15
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-surefire-plugin is missing. @ line 97, column 17
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
```

```
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] Using the builder org.apache.maven.lifecycle.internal.builder.singlethreaded.SingleThreadedBuilder with a thread count of 1
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building A custom grails project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- grails-maven-plugin:2.2.4:install-templates (default-cli) @ grailsdemo ---
[WARNING] Grails Start with out fork

|Loading Grails 2.2.4
|Configuring classpath
|Running pre-compiled script
.
|Environment set to development
......
|Templates installed successfully
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 8.722 s
[INFO] Finished at: 2014-06-11T12:50:24+01:00
[INFO] Final Memory: 29M/247M
[INFO] ------------------------------------------------------------------------
$ dir src/templates/
artifacts  scaffolding  testing  wa
$ dir src/templates/artifacts/
Controller.groovy   Filters.groovy     ScaffoldingController.groovy  Service.groovy  Tests.groovy
DomainClass.groovy  hibernate.cfg.xml  Script.groovy                 TagLib.groovy   WebTest.groovy
$ dir src/templates/scaffolding/
Controller.groovy  create.gsp  edit.gsp  _form.gsp  list.gsp  renderEditor.template  show.gsp  Test.groovy
```

El uso de las plantillas lo veremos más adelante.

5. Compilando y probando la aplicación de ejemplo

Ejecutando la aplicación:

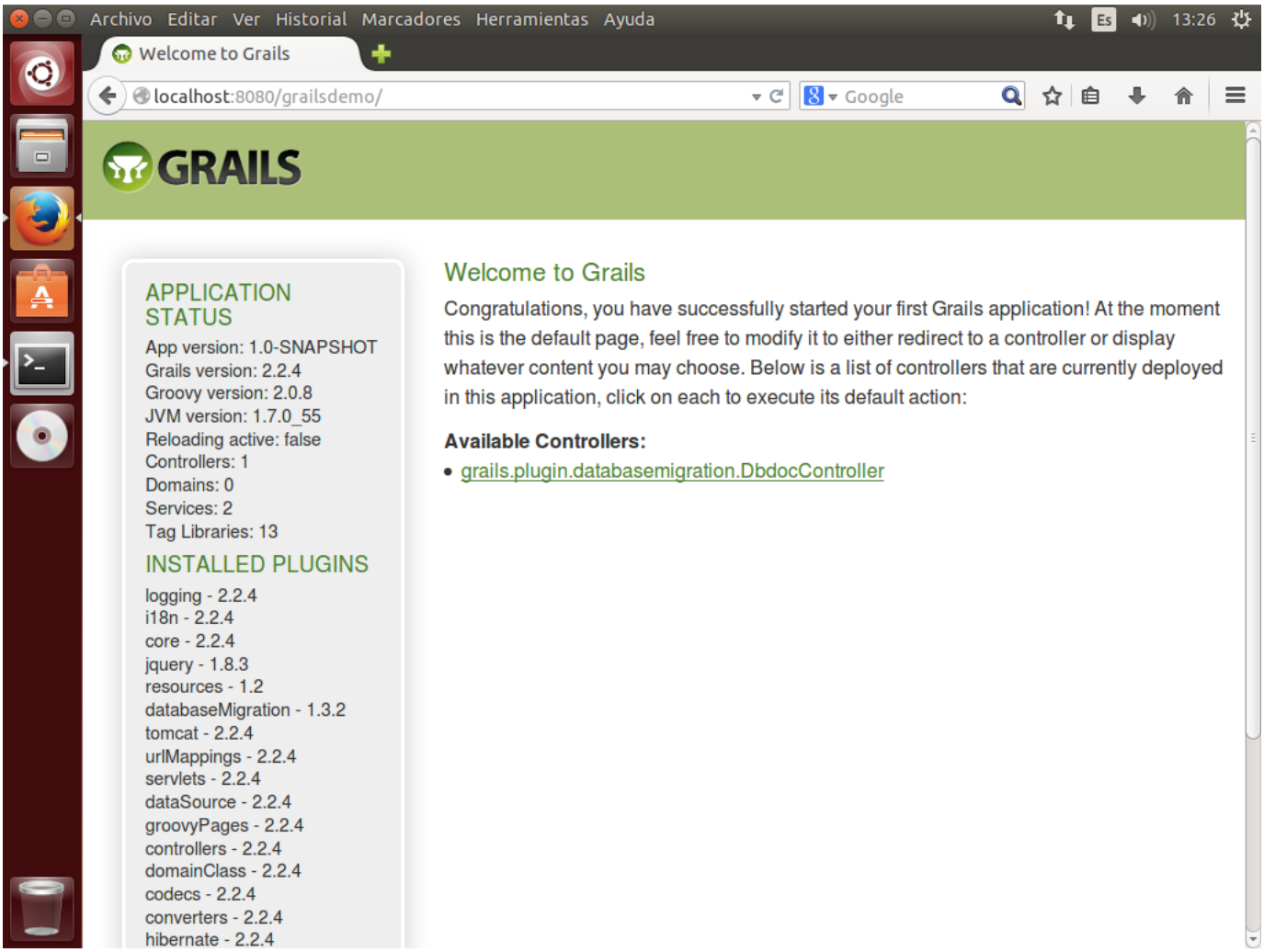
```
$ mvn grails:run-app
[INFO] Scanning for projects...
[WARNING]
[WARNING] Some problems were encountered while building the effective model for es.com.gonzalezalmiron.grailsdemo:grailsdemo:grails-
[WARNING] 'dependencies.dependency.(groupId:artifactId:type:classifier)' must be unique: org.grails.plugins:cache:zip -> duplicate d
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-compiler-plugin is missing. @ line 143, column 15
[WARNING] 'build.plugins.plugin.version' for org.apache.maven.plugins:maven-surefire-plugin is missing. @ line 97, column 17
[WARNING]
[WARNING] It is highly recommended to fix these problems because they threaten the stability of your build.
[WARNING]
[WARNING] For this reason, future Maven versions might no longer support building such malformed projects.
[WARNING]
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building A custom grails project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- grails-maven-plugin:2.2.4:run-app (default-cli) @ grailsdemo ---
[WARNING] Grails Start with out fork

|Loading Grails 2.2.4
|Configuring classpath
|Running pre-compiled script
.
|Environment set to development
..............................
|Packaging Grails application
|Installing zip tomcat-2.2.4.zip...
...
|Installed plugin tomcat-2.2.4
.............
|Installing zip hibernate-2.2.4.zip...
...
|Installed plugin hibernate-2.2.4
...
|Installing zip jquery-1.8.3.zip...
...
|Installed plugin jquery-1.8.3
...
|Installing zip cache-1.0.1.zip...
...
|Installed plugin cache-1.0.1
...
|Installing zip resources-1.2.zip...
...
|Installed plugin resources-1.2
...
|Installing zip database-migration-1.3.2.zip...
...
|Installed plugin database-migration-1.3.2
......
|Compiling 119 source files

..
|Compiling 8 source files
.........
|Running Grails application
```

```
|Server running. Browse to http://localhost:8080/grailsdemo
```

Podría ser que el Tomcat7 (puede ser otro servidor o aplicación) ya estuviera ocupando el puerto, por lo que debemos primero pararlo:

```
$ sudo lsof â€“i

Buscamos el proceso que ocupa el pueto 8080. En mi caso era el Tomcat7

$ sudo service tomcat7 stop
$ mvn grails:run-app

â€¦. Grails hace su trabajo

|Loading Grails 2.2.4
|Configuring classpath
|Running pre-compiled script
.
|Environment set to development
.............................
|Packaging Grails application
............
|Running Grails application
|Server running. Browse to http://localhost:8080/grailsdemo
```

Y vemos el resultado:



6. El plugin de Grails para Maven

Si instalamos Grails como servidor de aplicaciones y entorno de generación de aplicaciones, hay un montón de acciones que se pueden ejecutar desde Grails. Estas acciones suelen tener un equivalente Maven para ser ejecutadas desde el plugin de Maven.

En la documentación de Grails está toda la documentación del plugin de Grails para Mavenâ€¦ pero ojo, es bastante incompleta. Hay que averiguar muchas cosas mediante ensayo y error.

```
$ mvn grails:help
[INFO] ------------------------------------------------------------------------
[INFO] Building A custom grails project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- grails-maven-plugin:2.2.4:help (default-cli) @ grailsdemo ---
[INFO] org.grails:grails-maven-plugin:2.2.4

Maven plugin for GRAILS applications
  This plugin allows you to integrate GRAILS applications into maven 2 builds.

This plugin has 37 goals:
```

```
grails:clean
  Cleans a Grails project.

grails:config-directories
  Set sources/tests directories to be compatible with the directories layout
  used by grails.

grails:console
  Runs a Grails console inside the current project.

grails:create-controller
  Creates a new controller.

grails:create-domain-class
  Creates a new domain class.

grails:create-integration-test
  Creates a new Grails integration test which loads the whole Grails environment
  when run.

grails:create-pom
  Creates a creates a maven 2 POM for an existing Grails project.

grails:create-script
  Creates a Grails Gant Script.

grails:create-service
  Creates a new service class.

grails:create-tag-lib
  Creates a new GSP tag library.

grails:create-unit-test
  Creates a new Grails unit test. A unit test requires that you mock out access
  to dynamic methods, but executes a lot quicker.

grails:exec
  Executes an arbitrary Grails command.

grails:generate-all
  Generates a CRUD interface (controller + views) for a domain class.

grails:generate-controller
  Generates a CRUD controller for a specified domain class.

grails:generate-views
  Generates the CRUD views for a specified domain class.

grails:help
  Display help information on grails-maven-plugin.
  Call
    mvn grails:help -Ddetail=true -Dgoal=
  to display parameter details.

grails:init
  Validate consistency between Grails and Maven settings.

grails:init-plugin
  Validate consistency between Grails and Maven settings.

grails:install-templates
  Installs the artifact and scaffolding templates.

grails:list-plugins
  Lists the available plugins.

grails:maven-clean
  Cleans a Grails project and jars in lib directory.

grails:maven-compile
  Compiles a Grails project.

grails:maven-functional-test
  Runs a Grails application's functional tests.

grails:maven-grails-app-war
  Creates a WAR archive and register it in maven. This differs from the
  MvnWarMojo in that it makes the WAR file the build artifact for the
  'grails-app' packaging. The standard 'maven-war' goal is designed for the
  normal 'war' packaging.

  So why have two versions? Well, version 1.0 of the plugin was released with
  the war packaging as the default for Grails projects. That hasn't worked out
  so well, but we still need to support it, so we need one war target for the
  war packaging type, and one for the grails-app packaging type (which is now
  preferred over war).

grails:maven-test
  Runs a Grails applications unit tests.

grails:maven-war
  Creates a WAR archive for the project and puts it in the usual Maven place.

grails:package
  Packages the Grails application into the web-app folder for running.

grails:package-plugin
```

```
    Packages the Grails plugin.

grails:run-app
  Runs a Grails application in Jetty.

grails:run-app-https
  Runs a Grails application in Jetty with HTTPS listener.

grails:run-war
  Runs a Grails application in Jetty from its WAR.

grails:set-version
  Set the grails application version from the Maven POM version.

grails:test-app
  Runs a Grails applications unit tests and integration tests.

grails:upgrade
  Upgrades a Grails application to the version of this plugin.

grails:validate
  Validate consistency between Grails and Maven settings.

grails:validate-plugin
  Validate consistency between Grails and Maven settings.

grails:war
  Creates a WAR archive.
```

Para saber qué propiedades podemos definir antes de llamar a una de las tareas del plugin de Grails usaremos este comando:

```
$ mvn help:describe -Ddetail -DartifactId=grails-maven-plugin -DgroupId=org.grails
```

Si queremos la ayuda de un goal complete usamos el parámetro goal:

```
$ mvn grails:help -Dgoal= create-domain-class  -Ddetail
```

Por ejemplo para grails:create-domain-class:

```
grails:create-domain-class
  Description: Creates a new domain class.
  Implementation: org.grails.maven.plugin.GrailsCreateDomainClassMojo
  Language: java

  Available parameters:

    activateAgent
      User property: activateAgent
      Whether to activate the reloading agent (forked mode only) for this
      command

    basedir (Default: ${basedir})
      Required: true
      The directory where is launched the mvn command.

    domainClassName
      User property: domainClassName
      The name for the domain class to create.

    env
      User property: grails.env
      The Grails environment to use.

    extraClasspathEntries
      Extra classpath entries as a comma separated list of file names. For
      entries with a comma in their name, use backslash to escape. INTERNAL
      This parameter is not meant to be used externally. It is used by IDEs
      that require extra classpath entries to execute grails commands.

    fork (Default: false)
      User property: fork
      Whether the JVM is forked for executing Grails commands

    forkDebug (Default: false)
      User property: forkDebug
      Whether the JVM is forked for executing Grails commands

    forkedVmArgs
      List of arguments passed to the forked VM

    forkMaxMemory (Default: 1024)
      User property: forkMaxMemory
      Whether the JVM is forked for executing Grails commands

    forkMinMemory (Default: 512)
      User property: forkMinMemory
      Whether the JVM is forked for executing Grails commands

    forkPermGen (Default: 256)
      User property: forkPermGen
      Whether the JVM is forked for executing Grails commands

    grailsBuildListener
      Fully qualified classname of a grails build listener to attach to the
      Grails command

    grailsEnv
```

```
      User property: environment
      The Grails environment to use.

    grailsHome
      User property: grailsHome
      The path to the Grails installation.

    grailsWorkDir (Default: ${project.build.directory}/work)
      User property: grails.grailsWorkDir
      The Grails work directory to use.

    nonInteractive (Default: false)
      Required: true
      User property: nonInteractive
      Whether to run Grails in non-interactive mode or not. The default is to
      run interactively, just like the Grails command-line.

    pluginsDir (Default: ${basedir}/plugins)
      Required: true
      User property: pluginsDirectory
      The directory where plugins are stored.

    showStacktrace (Default: false)
      User property: showStacktrace
      Turns on/off stacktraces in the console output for Grails commands.
```

Error importante: si al ejecutar un commando de Grails Maven retorna un error de este estilo:

```
[INFO] BUILD FAILURE
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 15.058s
[INFO] Finished at: Mon Jun 09 14:20:22 CEST 2014
[INFO] Final Memory: 26M/62M
[INFO] ------------------------------------------------------------------------
[ERROR] Failed to execute goal org.grails:grails-maven-plugin:2.2.4:create-domain-class (default-cli) on project grailsdemo: Unable
```

Es muy probable que nos falte definir parámetros para el comando de Grails. En Grails, se pedirían por pantalla dichos parámetros. Pero dado que la versión evaluada tiene algunos parámetros con el modo interactivo desactivado, hay que definirlos antes de ejecutar el goal del plugin de Grails.

6.1. Añadir un plugin de Grails al proyecto

En Grails hay muchos plugins disponibles para realizar un montón de tareas. La lista completa de plugins disponibles la podemos generar con:

```
$ mvn grails:list-plugins
```

Para añadir el plugin de Grails, simplemente se mete en el pom.xml la dependencia del plugin, por ejemplo:

```
        org.grails.plugins
        authentication
        2.0.1
        zip
        runtime



        org.grails.plugins
        file-uploader
        1.2.1
        zip
        runtime
```

Al intentar ejecutar la aplciación, se descargará e instalará el plugin de Grails.

7. Introducción al patrón MVC y al CRUD en Grails

Grails está preparado para crear aplicaciones que siguen el patrón MVC de manera sencilla. El Proceso es el siguiente:

1. Creamos el modelo. Para ello vamos a definir las entidades mediante clases Groovy que definirán los datos que se van a persistir en la base de datos
2. Luego generamos los controladores asociados a dichas entidades. Se generarán automáticamente las acciones necesarias para implementar el patrón CRUD (Consltas, altas, bajas y modificaciones)
3. Luego generamos las vistas que podrán mostrar la información de dicha entidad

Generemos una entidad simple: Book:

```
$ mvn grails:create-domain-class -DdomainClassName=Book
â€¦

|Loading Grails 2.2.4
|Configuring classpath
|Running pre-compiled script
.
|Environment set to development
.................
|Created file grails-app/domain/grailsdemo/Book.groovy
..
|Compiling 1 source files
......
|Created file test/unit/grailsdemo/BookTests.groovy
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 17.534s
```

```
[INFO] Finished at: Mon Jun 09 13:41:30 CEST 2014
[INFO] Final Memory: 29M/70M
[INFO] ------------------------------------------------------------------------
```

Y ahora veremos que se ha creado una clase groovy en la carpeta de clases del dominio grails-app/domain/grailsdemo

```
$ dir grails-app/domain/grailsdemo/
Book.groovy
$ nano grails-app/domain/grailsdemo/Book.groovy

package grailsdemo

class Book {

    static constraints = {
    }
}
```

Vamos a generar el controlador (ojo con el nombre de la entidad, ya que las genera dentro del paquete grailsdemo)

```
$ mvn grails:generate-controller -DdomainClassName=grailsdemo.Book
[INFO] Scanning for projects...

â€¦

[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building A custom grails project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- grails-maven-plugin:2.2.4:generate-controller (default-cli) @ grailsdemo ---
[WARNING] Grails Start with out fork

|Loading Grails 2.2.4
|Configuring classpath
|Running pre-compiled script
.
|Environment set to development
.............................
|Packaging Grails application
......................
|Generating controller for domain class grailsdemo.Book
|Finished generation for domain class grailsdemo.Book
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 16.739s
[INFO] Finished at: Tue Jun 10 12:25:30 CEST 2014
[INFO] Final Memory: 38M/91M
[INFO] ------------------------------------------------------------------------
```

Y ahora generamos la vista:

```
$ mvn grails:generate-views -DdomainClass=grailsdemo.Book
[INFO] Scanning for projects...

â€¦

[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Building A custom grails project 1.0-SNAPSHOT
[INFO] ------------------------------------------------------------------------
[INFO]
[INFO] --- grails-maven-plugin:2.2.4:generate-views (default-cli) @ grailsdemo ---
[WARNING] Grails Start with out fork

|Loading Grails 2.2.4
|Configuring classpath
|Running pre-compiled script
.
|Environment set to development
.............................
|Packaging Grails application
....
|Compiling 1 source files
.........
|Packaging Grails application
..........
|Generating views for domain class grailsdemo.Book
|Finished generation for domain class grailsdemo.Book
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 39.272s
[INFO] Finished at: Tue Jun 10 12:28:23 CEST 2014
[INFO] Final Memory: 34M/88M
[INFO] ------------------------------------------------------------------------
```

Los ficheros generados son:

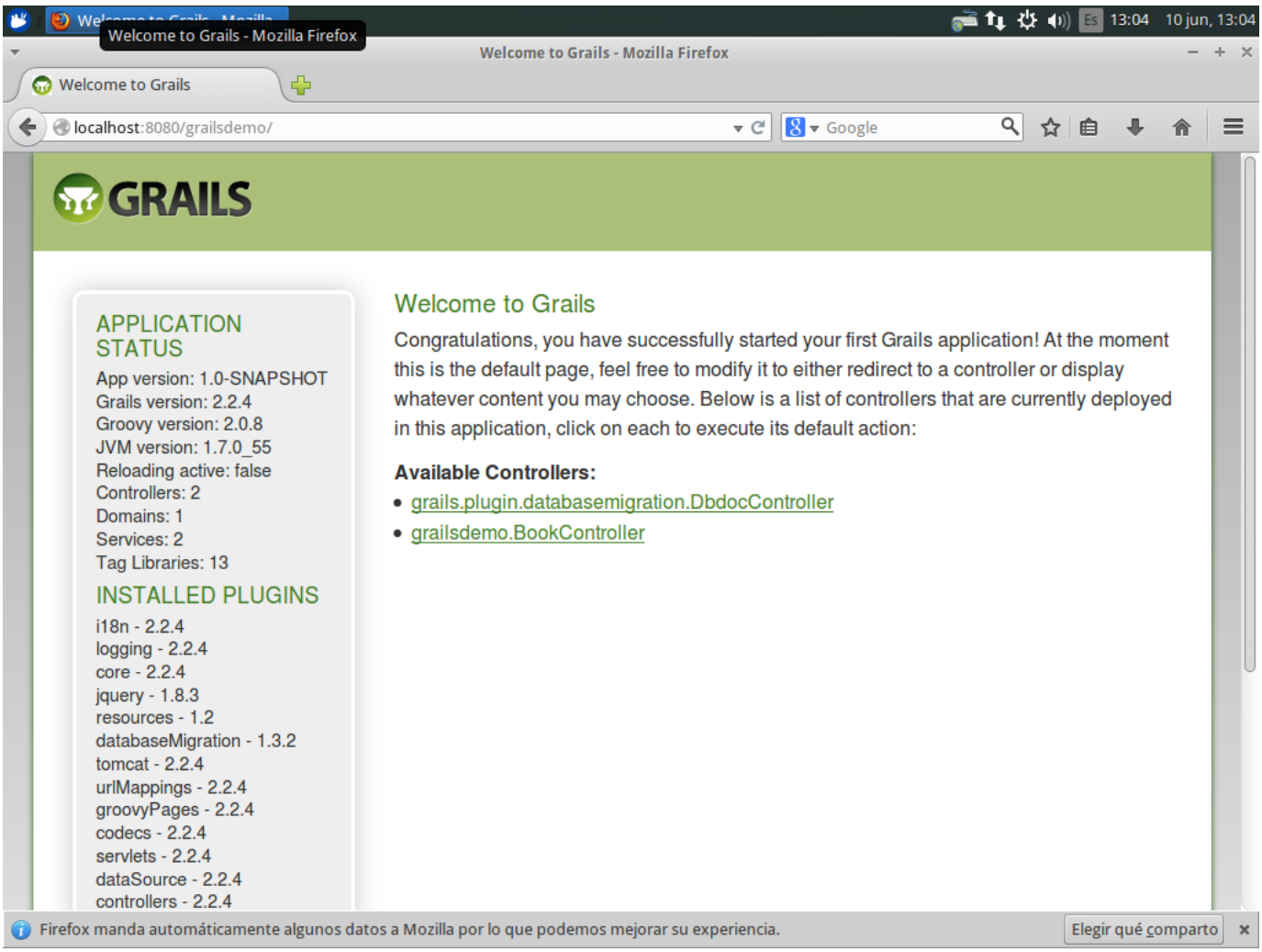
- en grails-app/views/book/
  - create.gsp
  - edit.gsp
  - _form.gsp
  - list.gsp
  - show.gsp
- En grails-app/controllers/grailsdemo/
  - BookController.groovy

- En grails-app/domain/grailsdemo/
  - Book.groovy

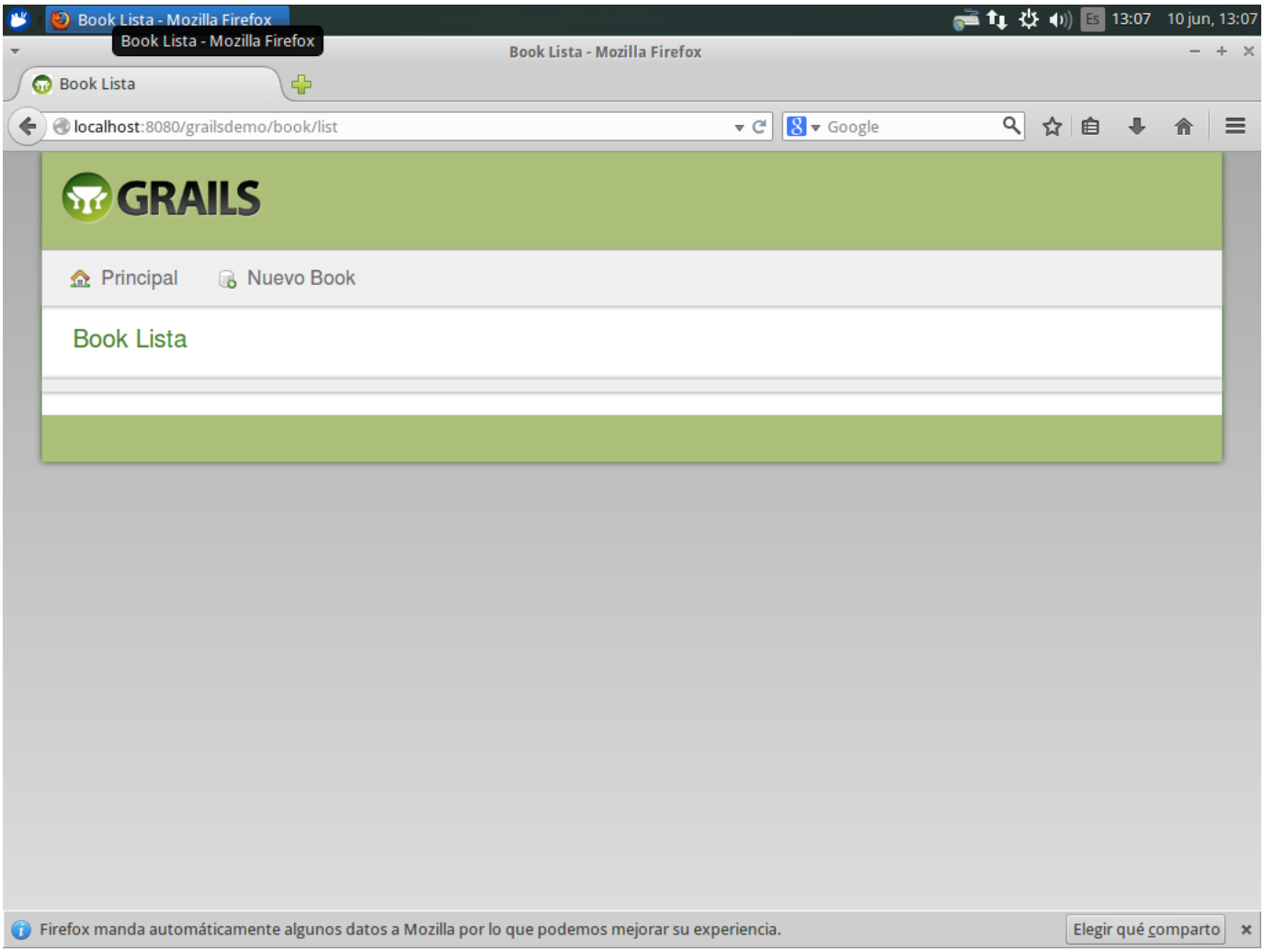
Las vistas generadas son GSPs, que son parecidos a los JSPs estándares, pero incluyen etiquetas Groovy para generar los elementos de la vista.

El controlador está escrito en Groovy, por lo que hay que entender algunos de los conveniso del lenguaje Grroovy para ver cómo funciona.

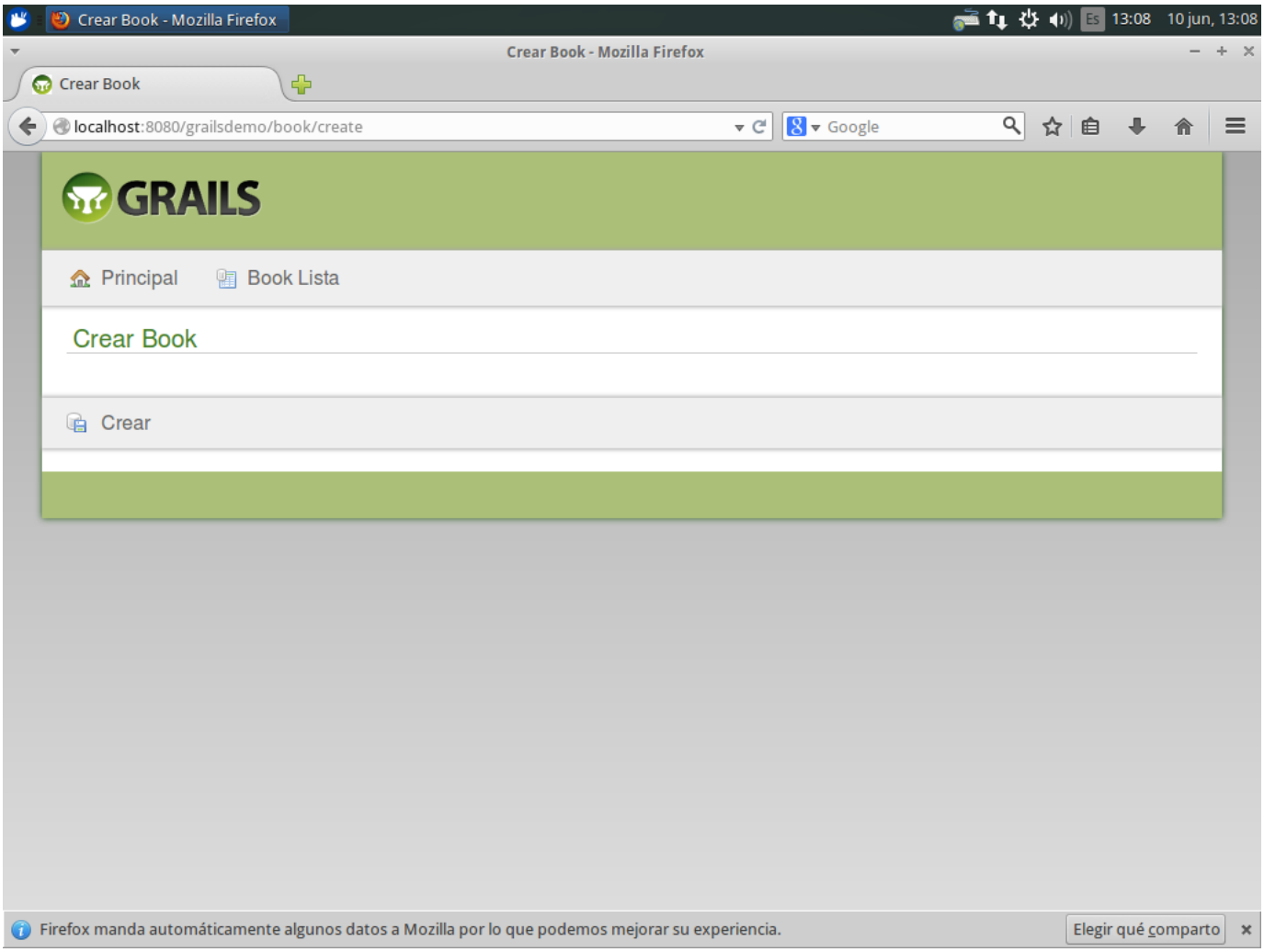
Ahora ejecutamos de nuevo la aplicación con `$ mvn grails:run-app` y al abrir el navegador ya aparece el nuevo controlador.



Pinchamos en el controlador:



Y si pinchamos en Crear Book:



Ahora vamos a añadir nuevos campos a la entidad. Modificamos el fichero Book.groovy

```
class Book {

    static constraints = {
    }

    String titulo;
    String autor;
    Date   fechaPublicaon;
}
```

Y regeneramos las vistas con:

```
$ mvn grails:generate-views -DdomainClass=grailsdemo.Book
```

Al ir a crear un nuevo libro aparecen ya los campos.

7.1. Las plantillas de los formularios

¿Cómo genera Grails los ficheros GSPs cuando usamos grails:generate-views? El misterio está en los ficheros de plantillas. Pero la plantilla es a su vez un GSP, luego el motor de Grails genera un GSP a partir de un GSP de plantilla. El proceso exacto es bastante curioso. Para explicarlo mejor vamos a ver una plantilla y un GSP generado a partir de ella:

```
<% import grails.persistence.Event %>
<%=packageName%>
<html>
        <head>
                <meta name="layout" content="main">
                <g:set var="entityName" value="\${message(code: '${domainClass.propertyName}.label', default: '${className}')}" />
                <title><g:message code="default.list.label" args="[entityName]" /></title>
        </head>
        <body>
                <a href="#list-${domainClass.propertyName}" class="skip" tabindex="-1"><g:message code="default.link.skip.label" de
                <div class="nav" role="navigation">
                        <ul>
                                <li><a class="home" href="\${createLink(uri: '/')}"><g:message code="default.home.label"/></a></li>
                                <li><g:link class="create" action="create"><g:message code="default.new.label" args="[entityName]" 
                        </ul>
                </div>
                <div id="list-${domainClass.propertyName}" class="content scaffold-list" role="main">
                        <h1><g:message code="default.list.label" args="[entityName]" /></h1>
                        <g:if test="\${flash.message}">
                        <div class="message" role="status">\${flash.message}</div>
                        </g:if>
                        <table>
                                <thead>
                                        <tr>
                                        <%  excludedProps = Event.allEvents.toList() << 'id' << 'version'
```

```
                                        allowedNames = domainClass.persistentProperties*.name << 'dateCreated' << 'lastUpdat
                                        props = domainClass.properties.findAll { allowedNames.contains(it.name) && !excluded
                                        Collections.sort(props, comparator.constructors[0].newInstance([domainClass] as Obje
                                        props.eachWithIndex { p, i ->
                                                if (i < 6) {
                                                        if (p.isAssociation()) { %>
                                        <th><g:message code="${domainClass.propertyName}.${p.name}.label" default="${p.natu
                                <%      } else { %>
                                        <g:sortableColumn property="${p.name}" title="\${message(code: '${domainClass.prope
                                <%  }   }   } %>
                                        </tr>
                                </thead>
                                <tbody>
                                <g:each in="\${${propertyName}List}" status="i" var="${propertyName}">
                                        <tr class="\${(i % 2) == 0 ? 'even' : 'odd'}">
                                        <%  props.eachWithIndex { p, i ->
                                                        if (i == 0) { %>
                                        <td><g:link action="show" id="\${${propertyName}.id}">\${fieldValue(bean: ${property
                                        <%      } else if (i < 6) {
                                                        if (p.type == Boolean || p.type == boolean) { %>
                                        <td><g:formatBoolean boolean="\${${propertyName}.${p.name}}" /></td>
                                        <%          } else if (p.type == Date || p.type == java.sql.Date || p.type == java.sql.Time
                                        <td><g:formatDate date="\${${propertyName}.${p.name}}" /></td>
                                        <%          } else { %>
                                        <td>\${fieldValue(bean: ${propertyName}, field: "${p.name}")}</td>
                                        <%  }   }   } %>
                                        </tr>
                                </g:each>
                                </tbody>
                        </table>
                        <div class="pagination">
                                <g:paginate total="\${${propertyName}Total}" />
                        </div>
                </div>
        </body>
</html>
```

Ahora vemos el fichero generado:

```
<%@ page import="grailsdemo.Book" %>

<html>
        <head>
                <meta name="layout" content="main">
                <g:set var="entityName" value="${message(code: 'book.label', default: 'Book')}" />
                <title><g:message code="default.list.label" args="[entityName]" /></title>
        </head>
        <body>
                <a href="#list-book" class="skip" tabindex="-1"><g:message code="default.link.skip.label" default="Skip to content…
                <div class="nav" role="navigation">
                        <ul>
                                <li><a class="home" href="${createLink(uri: '/')}"><g:message code="default.home.label"/></a></li>
                                <li><g:link class="create" action="create"><g:message code="default.new.label" args="[entityName]" /
                        </ul>
                </div>
                <div id="list-book" class="content scaffold-list" role="main">
                        <h1><g:message code="default.list.label" args="[entityName]" /></h1>
                        <g:if test="${flash.message}">
                        <div class="message" role="status">${flash.message}</div>
                        </g:if>
                        <table>
                                <thead>
                                        <tr>

                                                <g:sortableColumn property="autor" title="${message(code: 'book.autor.label', defaul

                                                <g:sortableColumn property="fechaPublicaon" title="${message(code: 'book.fechaPublic

                                                <g:sortableColumn property="titulo" title="${message(code: 'book.titulo.label', defa

                                        </tr>
                                </thead>
                                <tbody>
                                <g:each in="${bookInstanceList}" status="i" var="bookInstance">
                                        <tr class="${(i % 2) == 0 ? 'even' : 'odd'}">

                                                <td><g:link action="show" id="${bookInstance.id}">${fieldValue(bean: bookInstance, f

                                                <td><g:formatDate date="${bookInstance.fechaPublicaon}" /></td>

                                                <td>${fieldValue(bean: bookInstance, field: "titulo")}</td>

                                        </tr>
                                </g:each>
                                </tbody>
                        </table>
                        <div class="pagination">
                                <g:paginate total="${bookInstanceTotal}" />
                        </div>
                </div>
        </body>
</html>
```

Observar que entre otras cosas:

- Los \$ se cambian por $ en el fichero GSP generado
- Los ${variable} se sustituyen por su valor en el GSP generado
- Se usan sentencias de control g:if y g:each para generar elementos en el GSP generado.

Este último punto es muy importante, ya que si queremos personalizar un GSP generado hay que modificar las plantillas y añadir las exclusiones o código a medida que necesitemos... en la propia plantilla.

Una pregunta curiosa. Por defecto ¿cuántas columnas tiene como máximo un listado estándar en Grails?

Por cierto, ¿cuál es el orden de las columnas en los listados?

7.2. Un ejemplo sencillo de plantilla modificada list.gsp

Os voy a mostrar una plantilla list.gsp que incluye elementos a medida:

```
<% import grails.persistence.Event %>
<%=packageName%>

<!DOCTYPE html>
<html>
    <head>
        <meta name="layout" content="main">
        <g:set var="entityName" value="\${message(code: '${domainClass.propertyName}.label', default: '${className}')}" />
        <title><g:message code="default.list.label" args="[entityName]" /></title>
        <resource:dateChooser />
        <resource:autoComplete skin="default" />
        <export:resource />
    </head>
    <body>
        <a href="#list-${domainClass.propertyName}" class="skip" tabindex="-1"><g:message code="default.link.skip.label" default="Sk


        <g:if test="\${'${propertyName}' == 'autorInstance'}">

            <g:form action="searchBy" controller="autor">
                <div id="autocom">
                    <g:message code="default.search.label" args="['cliente', 'comercial']" />
                    <richui:autoComplete name="dato" action="completeFilter" class="required" />
                    <g:actionSubmitImage value="Buscar"  action="searchBy" src="\${resource(dir: 'images/skin', file: 'lupa.png')}"
                </div>
            </g:form>

        </g:if>

        <g:else>

            <g:form action="filterLike" controller="${domainClass.propertyName}">
                <div id="autocom">
                    <g:message code="default.filter.label" args="['${domainClass.propertyName}']" />
                    <richui:autoComplete name="dato" action="completeFilter" class="required" />
                    <g:actionSubmitImage value="Buscar"  action="filterLike" src="\${resource(dir: 'images/skin', file: 'lupa.png')}
                </div>
            </g:form>
        </td>
```

8. El plugin de Fields

En Grails, al mecanismo utilizado para generar el patrón CRUD se le llama scaffolding (andamiaje). Este mecanismo utiliza un sistema de plantillas para gerarlas vistas.

Estas plantillas analizan la clase entidad del dominio y generan la vista durante la etapa "grails:generate-views".

Pero el sistema de plantillas utilizado por defecto es muy básico. Es recomendable utilizar el plugin Fieds de Grails para tener más flexibilidad a la hora de generar las vistas.

8.1. Instalación del plugin de Fields

Se mete la dependencia en el pom.xml y se vuelve a compilar:

```
        org.grails.plugins
        fields
        1.3
        zip
        runtime
```

Luego hacemos $ mvn grails:run-app para que se instale el plugin de fields.

Si volvemos a intentar regenerar la vista veremos cómo se descarga e instala el plugin automáticamente.

Para instalar las plantillas del plugin hay que hacer:

```
$ mvn grails:exec -Dcommand="install-form-fields-templates"

â€¦


|Loading Grails 2.2.4
|Configuring classpath
.
|Environment set to development
....
|Copying templates from /home/usuario/workspaces/workspaceGrailsdemo/grailsdemo/plugins/fields-1.3
.......
|Template installation complete
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 26.476s
[INFO] Finished at: Tue Jun 10 13:35:58 CEST 2014
```

```
[INFO] Final Memory: 25M/61M
[INFO] ------------------------------------------------------------------------
```

Las plantillas se generan en /src/templates/scaffolding

El plugin de fields nos permite modificar el comportamiento de cada campo, pudiéndole asignar a un campo un trozo de código simplemente ubicando el código de plantilla en el lugar adecuado. Por ejemplo podemos modificar el campo fechaPublicacion de la entidad Book. Para ello creamos el fichero de plantilla de dicho campo en el lugar adecuado:

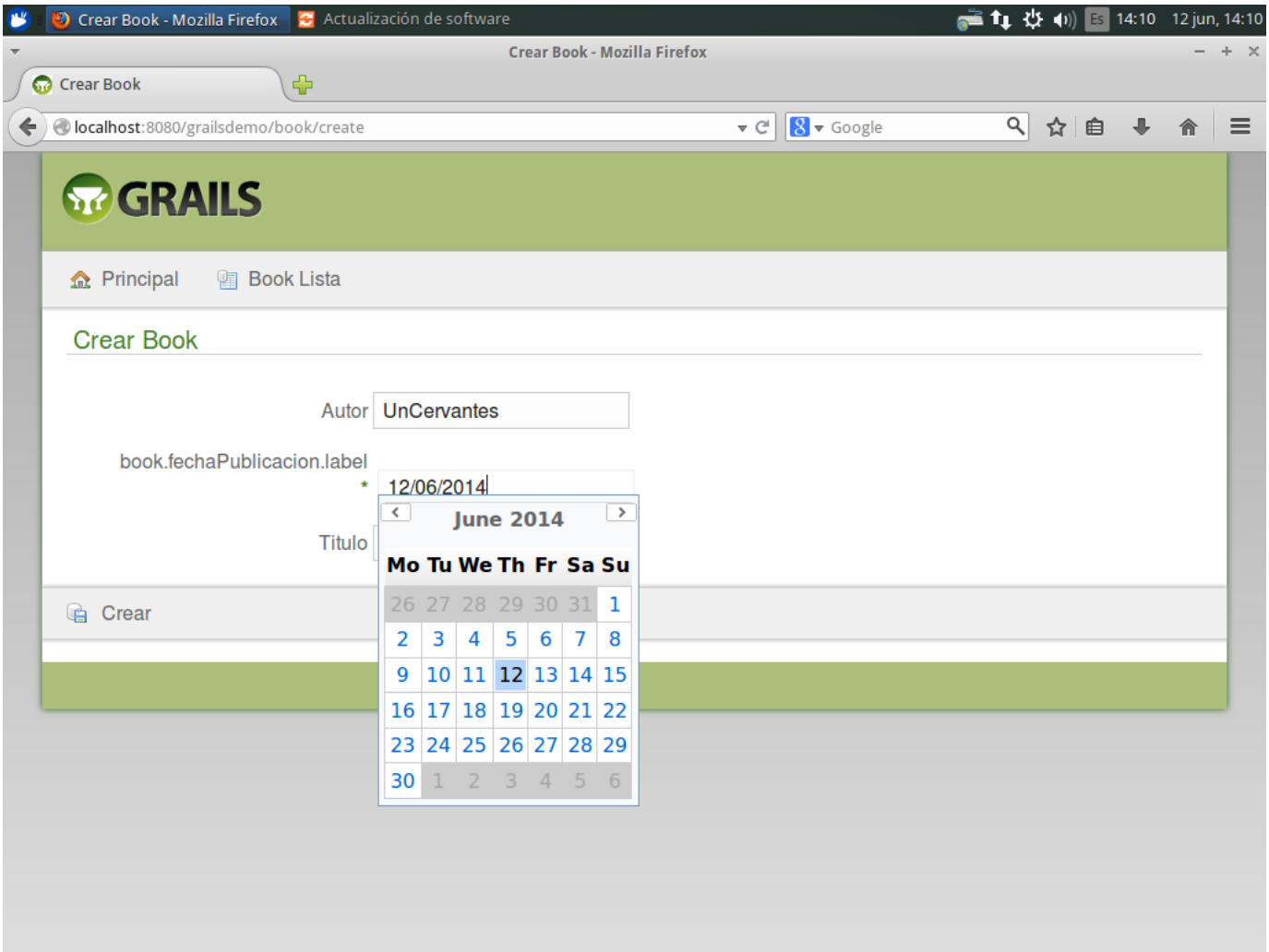
```
$ mkdir grails-app/views/book/fechaPublicacion
$ nano grails-app/views/book/fechaPublicacion/_field.gsp
<div class="fieldcontain ${hasErrors(bean: bookInstance, field: 'fechaPublicacion', 'error')} required">
        <label for="fechaInicio">
                <g:message code="book.fechaPublicacion.label" />
                <span class="required-indicator">*</span>
        </label>
        <richui:dateChooser name="fechaInicio" format="dd/MM/yyyy" firstDayOfWeek="Mo" value="${bookInstance?.fechaPublicacion}" />
</div>
```

Os dejo como tarea:

- Instalar el plugin richui
- Añadir la etiqueda al principio del fichero src/templates/scaffolding/create.gsp

```
<%=packageName%>
```

- Regenerar la vista
- Ver que se genera un nuevo campo fecha de publicación



Ayuda: Esta es la dependencia que yo he usado:

```
            org.grails.plugins
            richui
            0.8
            zip
            runtime
```

9. Los controladores, la persistencia y el lenguaje Groovy

Para el patrón MVC necesitamos un modelo, que es nuestra entidad Book, un conjunto de vistas, que son los GSP generados por Grails y nos queda nada más analizar el contolador, que es una clase Groovy.

Ya que hemos generado el controlador, listemos su contenido:

```
$ cat grails-app/controllers/grailsdemo/BookController.groovy
package grailsdemo

import org.springframework.dao.DataIntegrityViolationException

class BookController {

    static allowedMethods = [save: "POST", update: "POST", delete: "POST"]

    def index() {
        redirect(action: "list", params: params)
    }

    def list(Integer max) {
        params.max = Math.min(max ?: 10, 100)
        [bookInstanceList: Book.list(params), bookInstanceTotal: Book.count()]
    }

    def create() {
        [bookInstance: new Book(params)]
    }

    def save() {
        def bookInstance = new Book(params)
        if (!bookInstance.save(flush: true)) {
            render(view: "create", model: [bookInstance: bookInstance])
            return
        }

        flash.message = message(code: 'default.created.message', args: [message(code: 'book.label', default: 'Book'), bookInstance.
        redirect(action: "show", id: bookInstance.id)
    }

    def show(Long id) {
        def bookInstance = Book.get(id)
        if (!bookInstance) {
            flash.message = message(code: 'default.not.found.message', args: [message(code: 'book.label', default: 'Book'), id])
            redirect(action: "list")
            return
        }

        [bookInstance: bookInstance]
    }

    def edit(Long id) {
        def bookInstance = Book.get(id)
        if (!bookInstance) {
            flash.message = message(code: 'default.not.found.message', args: [message(code: 'book.label', default: 'Book'), id])
            redirect(action: "list")
            return
        }

        [bookInstance: bookInstance]
    }

    def update(Long id, Long version) {
        def bookInstance = Book.get(id)
        if (!bookInstance) {
            flash.message = message(code: 'default.not.found.message', args: [message(code: 'book.label', default: 'Book'), id])
            redirect(action: "list")
            return
        }

        if (version != null) {
            if (bookInstance.version > version) {
                bookInstance.errors.rejectValue("version", "default.optimistic.locking.failure",
                          [message(code: 'book.label', default: 'Book')] as Object[],
                          "Another user has updated this Book while you were editing")
                render(view: "edit", model: [bookInstance: bookInstance])
                return
            }
        }

        bookInstance.properties = params

        if (!bookInstance.save(flush: true)) {
            render(view: "edit", model: [bookInstance: bookInstance])
            return
        }

        flash.message = message(code: 'default.updated.message', args: [message(code: 'book.label', default: 'Book'), bookInstance.
        redirect(action: "show", id: bookInstance.id)
    }

    def delete(Long id) {
        def bookInstance = Book.get(id)
        if (!bookInstance) {
            flash.message = message(code: 'default.not.found.message', args: [message(code: 'book.label', default: 'Book'), id])
            redirect(action: "list")
            return
        }

        try {
```

```
            bookInstance.delete(flush: true)
            flash.message = message(code: 'default.deleted.message', args: [message(code: 'book.label', default: 'Book'), id])
            redirect(action: "list")
        }
        catch (DataIntegrityViolationException e) {
            flash.message = message(code: 'default.not.deleted.message', args: [message(code: 'book.label', default: 'Book'), id])
            redirect(action: "show", id: id)
        }
    }
}
```

¿Cómo funciona el controlador? Es como en otros frameworks MVC. La ruta solicitada por el navegador comienza por el nombre del controlador y la acción solicitada, y los parámetros añadidos a la ruta estarán disponibles dentro del controlador. Dentro del controlador se debe invocar a un método. Así http://localhost:8080/grailsdemo/book/ invoca a la acción por defecto, que es index() y ésta a su vez redirige al list(), y http://localhost:8080/grailsdemo/book/create invoca a la acción créate del controlador BookController.groovy

Si observamos las acciones encontraremos detalles importantes al estar usando Groovy, que hay que tener en cuenta si venimos de Java:

- El estilo de escritura es muy simple, no hay muchos ";""
- Groovy tiene más operadores que Java, como ?. y otros.
- Las variables en Groovy pueden ser débil o fuertemente tipadas.
- Al estilo de javascript, las variables no tipadas se definen con "def"
- A una variable se le puede asignar un valor o un trozo de código. Esto último es lo que se denominan "closures" en Groovy.
- Además es muy normal definir la "closure " e invocarla a la vez, como por ejemplo en estre fragmento de código:

```
def results = book.list (max: params.max, offset: params.offset) {
    eq("user",paramTarea2)
    if (params.order) {
        order(params.sort, params.order)
    } else {
        order("description", "asc")
    }
}
```

- Todos los métodos devuelven siempre un valor, que el valor de la última sentencia o expresión ejecutada
- Cuando se ejecuta un método del controlador, la variable params contiene los parámetros de la petición.
- Una lista se crea simplemente encerrando un conjunto de valores entre corchetes cuadrados: ["cadena1", "cadena2", "cadena3"]
- Un mapa se crea simplemente encerrando un conjunto de pares clave:valor entre corchetes cuadrados: [nombre: "Diego", apellido:"Rojo"]
- Si el método del controlador devuelve un mapa, éste se añadirá al conjunto de variables disponibles por la vista, como en el método edit() del controlador de Book
- Ojo con las vistas, ya que los GSPs se parecen a los JSPs estándares de java, pero son más potentes, ya que las expresiones ${ } admiten cualquier expresión Groovy, incluso llamadas a métodos.
- Todo el acceso a la base de datos lo haremos a través de GORM, que es el mapeador de Grails, basado en Hibernate. Además GORM utiliza un avanzado sistema de "proxis dinámicos" que simplifica la ejecución de consultas sencillas, como los típicos findByAutor()
- Nota: todo el sistema de persistencia por defecto trabaja en memoria, pero se puede guardar en una base de datos simplemente cambiando el DataSource.groovy

10. Conclusión

En este tutorial hemos realizado una presentación de Grails, usando como herramienta de desarrollo simplemente Maven. Hemos visto entre otros:

- Como se crea una aplicación en Grails
- Como se crea un patrón CRUD en Grails
- Una ligera introducción al patrón MVC y su implementación en Grails
- Algunas técnicas de Grails y Groovy

Y ahora llega la pregunta del millón ¿cuándo debo usar Grails en vez de Java con JSF y los frameworks estándares?

- Para un desarrollo corto, con un modelo de datos sencillo y un flujo de pantallas basado en CRUDs, Grails nos da la oportunidad de desarrollar la aplicación en poco tiempo, si lo dominamos.
- Realmente desarrollar en Groovy también tiene su propia curva de aprendizaje. Se pueden hacer cosas bastante interesantes en poco tiempo, pero adaptar el estilo visual de la aplicación a lo que a nosotros nos gusta lleva su tiempo.
- Si el desarrollo es largo, los frameworks de Java suelen facilitar los desarrollos (por ejemplo usando Primefaces o similares) y es algo a lo que la mayor parte de desarrolladores Java están ya acostumbrados.