GRAILS

ADOLFO SANZ DE DIEGO

ABRIL 2013

1 ACERCA DE

1.1 AUTOR

- **Adolfo Sanz De Diego**
  - Correo: asanzdiego@gmail.com
  - Twitter: [@asanzdiego](http://twitter.com/asanzdiego)
  - Linkedin: http://www.linkedin.com/in/asanzdiego
  - Blog: http://asanzdiego.blogspot.com.es

1.2 LICENCIA

- **Este obra está bajo una licencia:**
  - Creative Commons Reconocimiento-CompartirIgual 3.0
- **El código fuente de los programas están bajo una licencia:**
  - GPL 3.0

2 INTRODUCCIÓN

2.1 ¿GRAILS?

- Grails no sólo es un framework de desarrollo web, sino que es una **plataforma completa de desarrollo**:
  - Contenedor/servidor web
  - Gestor de base de datos
  - Scaffolding
  - Empaquetado de la aplicación (war)
  - Realización de tests (unitarios, de integración, funcionales)
  - Extensible con plugins

2.2 PARADIGMAS

- Se basa en los paradigmas:
  - **CoC** (Convención sobre Configuración)
  - **DRY** (Don’t Repeat Yourself)
  - **MVC** (Modelo Vista Controlador)

2.3 GORM

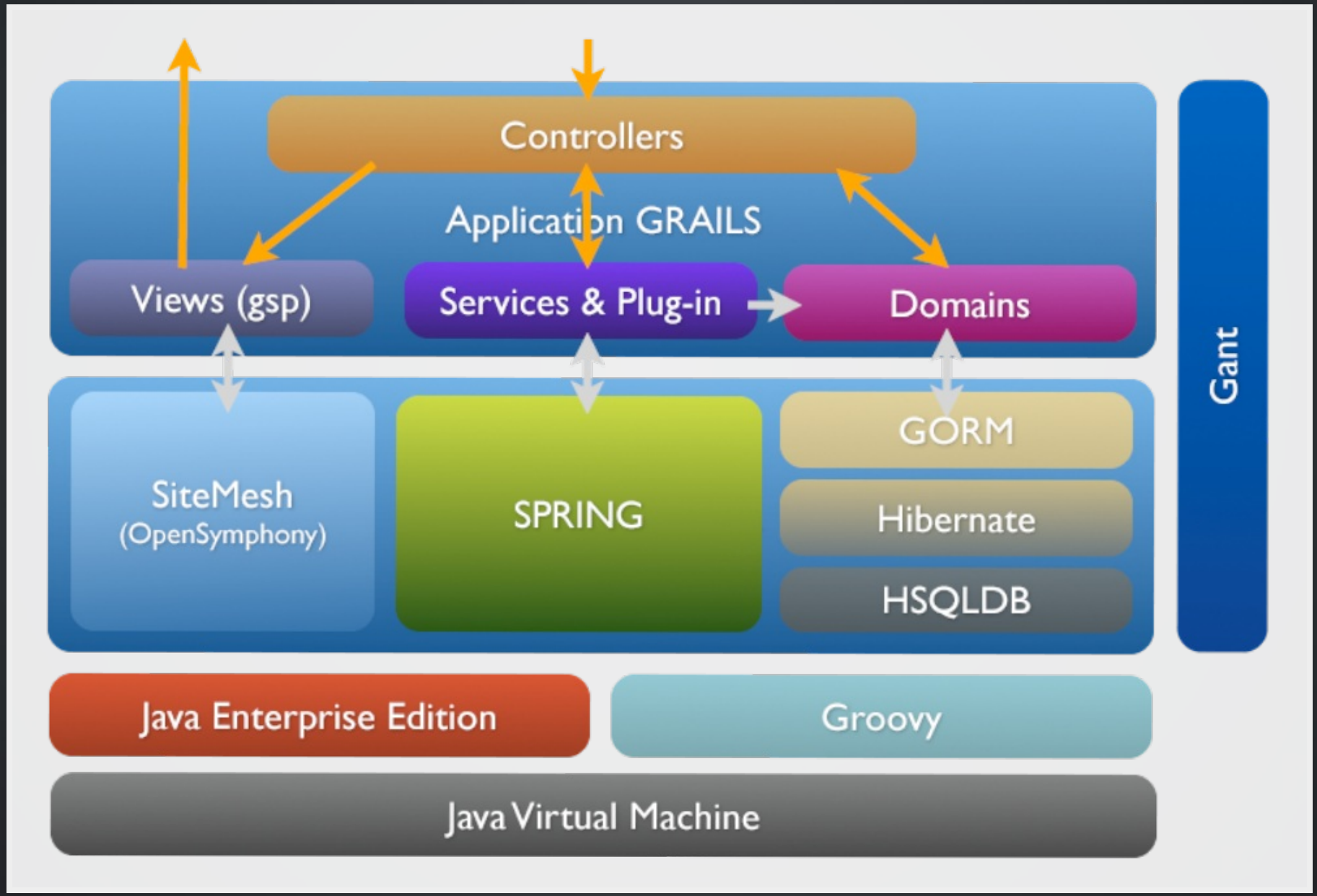
- GORM (Grails Object Relational Mapping) sirve para el **mapeo objeto-relacional**:
  - Uno a uno
  - Uno a muchos
  - Muchos a muchos

2.4 PLUGINS

- Existen multitud de plugins que **extienden la plataforma**:
  - Seguridad
  - AJAX
  - Búsqueda
  - Informes
  - etc.
- Se pueden crear plugins **internos para funcionalidades comunes** entre varias aplicaciones.

2.5 NIH (NOT INVENTED HERE)

2.6 ARQUITECTURA



3 INSTALACIÓN Y CONFIGURACIÓN

3.1 JDK

1. Descargar.
2. Instalar/Descomprimir.
3. Variable de entorno y añadir al path.

```
export JAVA_HOME="~/Java/jdk"
export PATH=$PATH":"$JAVA_HOME"/bin"
```

3.2 GROOVY-SDK

1. Descargar.
2. Instalar/Descomprimir.
3. Variable de entorno y añadir al path.

```
export GROOVY_HOME="~/Java/groovy"
export PATH=$PATH":"$GROOVY_HOME"/bin"
```

3.3 GRAILS

1. Descargar.
2. Instalar/Descomprimir.
3. Variable de entorno y añadir al path.

```
export GRAILS_HOME="~/Java/groovy"
export PATH=$PATH":"$GRAILS_HOME"/bin"
```

3.4 PROBANDO

```
$ grails --version
Grails version: 2.2.1
```

4 GETTING STARTED

4.1 CREATE GRAILS PROJECT

- Por línea de comandos:

```
$ grails create-app my-project
```

- En el GGTS:

```
File > New > Grails Project
```

- Línea de comandos en el GGTS:

```
Control + Alt + Shift + G
```

4.2 DIRECTORIOS (I)

```
%PROJECT_HOME%
+ grails-app    -> ficheros de la aplicación grails
+ lib           -> bibliotecas
+ scripts       -> scripts
+ src
  + groovy      -> otros ficheros groovy opcionales
  + java        -> otros ficheros java opcionales
+ test          -> clases de test
+ web-app
  + css         -> archivos CSS
  + images      -> archivos de imágenes
  + js          -> archivos JavaScript
  + WEB-INF     -> otros ficheros de la aplicación web
```

4.3 DIRECTORIOS (II)

```
%PROJECT_HOME%
+ grails-app
  + conf        -> archivos de configuración
    + hibernate -> archivos de configuración de hibernate
    + spring    -> archivos de configuración de spring
  + controllers -> controladores
  + domain      -> clases de dominio
  + i18n        -> ficheros de internacionalización
  + services    -> servicios
  + taglib      -> bibliotecas de etiquetas
  + util        -> clases de utilidades
  + views       -> vistas
    + layouts   -> layouts
```

4.4 CREATE A DOMAIN CLASS

```
$ grails create-domain-class org.example.Libro
```

```
package org.example

class Libro {
  String titulo
  String author

  static constraints = {
    titulo(blank: false)
    author(blank: false)
  }
}
```

4.5 CREATE A CONTROLLER

```
$ grails create-controller org.example.Libro
```

```
package org.example

class LibroController {
    def scaffold = Libro
}
```

4.6 CREATING TEST DATA

- grails-app/conf/BootStrap.groovy

```
import org.example.Libro
class BootStrap {
  def init = { servletContext ->
    // Check whether the test data already exists.
    if (!Libro.count()) {
      new Libro(
        author: "S. King",
        titulo: "The Shining").save(failOnError: true)
      new Libro(
        author: "J. Patterson",
        titulo: "Along Came a Spider").save(failOnError: true)
    }
  }

  def destroy = {
  }
}
```

4.7 START GRAILS

```
$ grails run-app
```

5 SCAFFOLDING

5.1 DEFINICIÓN

- **Generación automática de código** para las cuatro operaciones básicas de cualquier aplicación (CRUD):
  - **C**reate
  - **R**ead
  - **U**pdate
  - **D**elete

5.2 DINÁMICO

- En el controlador:

```
def scaffold = true // si se sigue la convención de nombrado
```

```
def scaffold = DomainClass // si no se sigue la convención de nombrado
```

5.3 ESTÁTICO

- Genera el controlador:

```
grails generate-controller org.example.Libro
```

- Genera la vista:

```
grails generate-views org.example.Libro
```

- Genera el controlador y la vista:

```
grails generate-all org.example.Libro
```

5.4 TEMPLATES

- Podemos extraer las templates de generación de código con el siguiente comando:

```
grails install-templates
```

- Se pueden ver y modificar para su uso en la carpeta:

```
%PROJECT_HOME%
+ src
  + templates
    + artifacts
    + scaffolding
    + testing
    + war
```

6 VALIDACIÓN

6.1 CLASES DE DOMINIO

```
class User {

 String email
 String password
 Integer age
 String twitter

 static constraints = {
   email(email:true, blank:false, unique:true)
   password(size:5..15, blank:false)
   age(range:18..99)
   twitter(url:true, nullable:true)
 }
}
```

6.2 CONTROLADORES

```
def user = new User(params)
if(user.validate()) {
  // do something with user
}
else {
  user.errors.allErrors.each {
    println it
  }
}
```

6.3 VISTAS

```
<g:hasErrors bean="${user}">
 <ul>
  <g:eachError var="err" bean="${user}">
   <li>${err}</li>
  </g:eachError>
 </ul>
</g:hasErrors>
```

7 CRUD

7.1 CREATE

```
def p = new Persona(nombre: "Fred", edad: 40)
p.save()
```

7.2 READ

```
def p = Persona.get(unaPersona.id)
assert 1 == p.id
```

7.3 UPDATE

```
def p = Persona.get(unaPersona.id)
p.nombre = "Bob"
p.save()
```

7.4 DELETE

```
 def p = Persona.get(unaPersona.id)
p.delete()
```

8 GORM

8.1 AGREGACIÓN (UNIDIRECCIONAL)

```
class Cara {
  Nariz nariz
}

class Nariz {
  ...
}
```

8.2 AGREGACION (BIDIRECCIONAL)

```
class Cara {
  Nariz nariz
}

class Nariz {
  static belongsTo = [face:Face]
}
```

```
new Cara(nose:new Nariz()).save() // guarda ambos: Cara y Nariz
new Nariz(face:new Cara()).save() // da error
Face.get(faceId).delete() // borra ambos: Cara y Nariz
```

8.3 UNO A UNO (FOREIGN KEY)

```
class Cara {
  static hasOne = [nose:Nose]

  // opcional, pero buena práctica
  static constraints = {
    nariz unique: true
  }
}
class Nariz {
  Cara cara // crea una FK en la tabla de Nariz
}
```

8.4 UNO A MUCHOS (I)

```
class Autor {
  static hasMany = [ libros : Libro ]
  String nombre
}
class Libro {
  String titulo
}
```

- Cascada al salvar y al actualizar pero no al borrar.

8.5 UNO A MUCHOS (II)

```
class Autor {
  static hasMany = [ libros : Libro ]
  String nombre
}
class Libro {
  static belongsTo = [ author: Autor ]
  String titulo
}
```

- Con belongsTo cascada al salvar, al actualizar y al borrar.

8.6 MUCHOS A MUCHOS

```
class Autor {
  static hasMany = [libros:Libro]
  String nombre
}

class Libro {
  static belongsTo = Autor
  static hasMany = [authors:Autor]
  String titulo
}
```

- El belongsTo marca el "propietario" de la relación, en este caso el Autor.
- Al salvar un Autor, se salvarán sus Libros, pero no al revés.
- Recomiendan usar 2 relaciones uno a muchos, mejor que una relación muchos a muchos.

9 QUERING

9.1 LISTADOS

- Todos los elementos:

```
def libros = Libro.list()
```

- Paginación:

```
def libros = Libro.list(offset:10, max:20)
```

- Ordenación

```
def libros = Libro.list(sort:"title", order:"asc")
```

9.2 POR ID

```
def libro = Libro.get(23)
```

```
def libros = Libro.getAll(23, 93, 81)
```

9.3 FINDBY Y FINDALLBY

```
.find[All]By([Property][Comparator][And|Or])?[Property][Comparator]
```

9.4 COMPARADORES (I)

- **InList** - Busca el valor dentro de la lista pasada por parámetro.
- **LessThan** - Menor que el valor pasado por parámetro.
- **LessThanEquals** - Menor o igual que el valor pasado por parámetro.
- **GreaterThan** - Mayor que el valor pasado por parámetro.
- **GreaterThanEquals** - Mayor o igual que el valor pasado por parámetro.

9.5 COMPARADORES (II)

- **Like** - Equivalente al like de SQL.
- **Ilike** - Similar a Like sólo que no es sensible a las mayúsculas.
- **NotEqual** - No es igual al valor pasado por parámetro.
- **Between** - Entre dos valores (necesita dos parámetros).
- **IsNotNull** - Valor no nulo (no requiere ningún parámetro).
- **IsNull** - Valor nulo (no requiere ningún parámetro)

9.6 EJEMPLOS

```
class Libro {
    String titulo
    Date fecha
}
```

```
def libro = Libro.findByTitulo("The Stand")
def libros = Libro.findAllByTituloLike("Harry Pot%")
libros = Libro.findAllByFechaBetween(primeraFecha, segundaFecha)
libros = Libro.findAllByFechaGreaterThan(someDate)
libros = Libro.findAllByTituloOrFechaLessThan("%titulo buscado%", fechaBuscada)
```

10 SERVICIOS

10.1 DEFINICIÓN

- Se utilizan cuando necesitamos **transacciones** o cuando utilizamos varias **clases de dominio**

10.2 CREACIÓN

```
$ grails create-service org.example.Libro
```

```
package org.example

class LibroService {

  def doSomething() {

    // do domething
  }
}
```

10.3 TRANSACCIONALIDAD

- **Por defecto son transaccionales**, para deshabilitarlo:

```
static transactional = false
```

10.4 SCOPE

- **Por defecto son singleton**, pero podemos usar otros scopes:
  - **prototype** - Una instancia por cada inyección.
  - **request** - Una instancia por cada request.
  - **flash** - Una instancia para la request actual y la siguiente.
  - **flow** - Una instancia por cada webflow.
  - **conversation** - Una instancia por cada conversacion de un webflow.
  - **session** - Una instancia por cada sesión.
  - **singleton** - Una única instancia (por defecto).

```
static scope = "flow"
```

10.5 INYECCIÓN

- Los servicios se pueden inyectar en los controladores.

```
class LibroController {

 def libroService
 ...
}
```

11 CONFIGURACIÓN LOG4J

11.1 LOGGING LEVELS

1. off
2. fatal
3. error
4. warn
5. info
6. debug
7. trace
8. all

11.2 ARTEFACTOS

- conf/Config.groovy

```
log4j = {

    // warn a todos los artefactos de nuestra aplicacion
    warn "grails.app"

    // debug a un controlador específico alojado en el paquete por defecto
    debug "grails.app.controllers.YourController"

    // debug a una clase de dominio específica
    debug "grails.app.domain.org.example.Book"

    // error a todos los taglibs
    error "grails.app.taglib"

    // info a tos los servicios
    info "grails.app.services"
}
```

12 CONFIGURACIÓN SPRING

12.1 IYECCIÓN NORMAL

- conf/spring/resources.groovy

```
beans = {
    rules(org.example.Rules) {
        deltaAge = 5
        deltaHeight = 0.1
    }
}
```

12.2 IYECCIÓN TESTS

- conf/spring/resources.groovy

```
defineBeans {
    rules(org.example.Rules) {
        deltaAge = 5
        deltaHeight = 0.1
    }
}
```

13 TESTING

13.1 UNIT TEST

- Tienen que ser rápidos, no se ejecutan en el servidor, utilizan mocks.
- Utilizan las anotaciones @TestFor y @Mock

13.2 INTEGRATION TEST

- Se ejecutan en el servidor con datos reales.
- Podemos utilizar el BootStrap para meter datos en la base de datos.

14 SPRING SECURITY

14.1 INSTALACIÓN

- conf/BuildConfig.groovy

```
...
plugins {
  ...
  compile ':spring-security-core:1.2.7.3'
  ...
}
```

14.2 CONFIGURACIÓN

```
grails refresh-dependencies
```

```
grails s2-quickstart org.example User Role
```

14.3 USO

- Se usa la anotación @Secured(['ROLE_NAME']) tanto a nivel de clase como a nivel de método.
- Se pueden usar tambien las siguientes reglas:
  - **IS_AUTHENTICATED_ANONYMOUSLY**: cualquiera puede entrar, incluso sin hacer login
  - **IS_AUTHENTICATED_REMEMBERED**: sólo usuarios con login pueden entrar
  - **IS_AUTHENTICATED_FULLY**: obliga a hacer login aunque tengas la cookie de remember