

CURSO DE SPRING BOOT 2 Y SPRING MVC 5



MIS DATOS

- ▶ Luis Miguel López Magaña
- ▶ 15 años desarrollando aplicaciones Java
(Java SE, Java EE, Spring, Hibernate,
Android, ...)
- ▶ Profesor de FP desde hace 10 años.

REQUISITOS PARA ESTE CURSO

- ▶ Java SE 8.
- ▶ Metodología de programación orientada a objetos.
- ▶ Spring Core

Y mejor si además sabes

- ▶ Algo de Java EE 7.
- ▶ Maven.
- ▶ Patrones de diseño.

¿QUÉ ME VA A APORTEAR SPRING?

- ▶ Reconocer las ventajas del uso de Spring Boot.
- ▶ Conocer la estructura de un proyecto Spring Web MVC.
- ▶ Conocer cómo acceder a datos utilizando Spring Data JPA
- ▶ Aplicar sólida seguridad a nuestras aplicaciones web.

CONTENIDOS

1. Spring Boot
2. Spring WebMVC
3. Spring Data JPA
4. Proyecto de Ejemplo

PRÁCTICAS

Practicaremos la sintaxis de cada una de las diferentes lecciones. Durante las lecciones centrales, realizaremos un miniproyecto.

Además, para finalizar el curso realizaremos paso a paso un proyecto completo que integre la mayor parte de los conocimientos del curso.

¿QUÉ SERÉ CAPAZ AL FINAL DEL CURSO?

- ▶ Aprenderás a configurar aplicaciones rápidamente con Spring Boot.
- ▶ Serás capaz de customizar propiedades de configuración.
- ▶ Conocerás cómo configurar aplicaciones web con Spring Boot, Spring Data JPA y Thymeleaf.

¿QUÉ SERÉ CAPAZ AL FINAL DEL CURSO?

- ▶ Serás capaz de implementar aplicaciones que hacen uso de formularios
- ▶ Aprenderás a crear aplicaciones web sólidas.
- ▶ Podrás conectar con una base de datos relacional mediante Spring Data JPA.
- ▶ Aprenderás a desarrollar las diferentes capas de una aplicación web.

¿QUÉ CURSOS PUEDO REALIZAR AL TERMINAR ESTE?

- ▶ Curso de Desarrollo Web Java EE
- ▶ Curso de Hibernate
- ▶ Curso de introducción a Thymeleaf
- ▶ Curso de SQL desde cero.
- ▶ ...

CURSO DE SPRING BOOT 2 Y SPRING MVC 5





INTRODUCCIÓN A SPRING BOOT

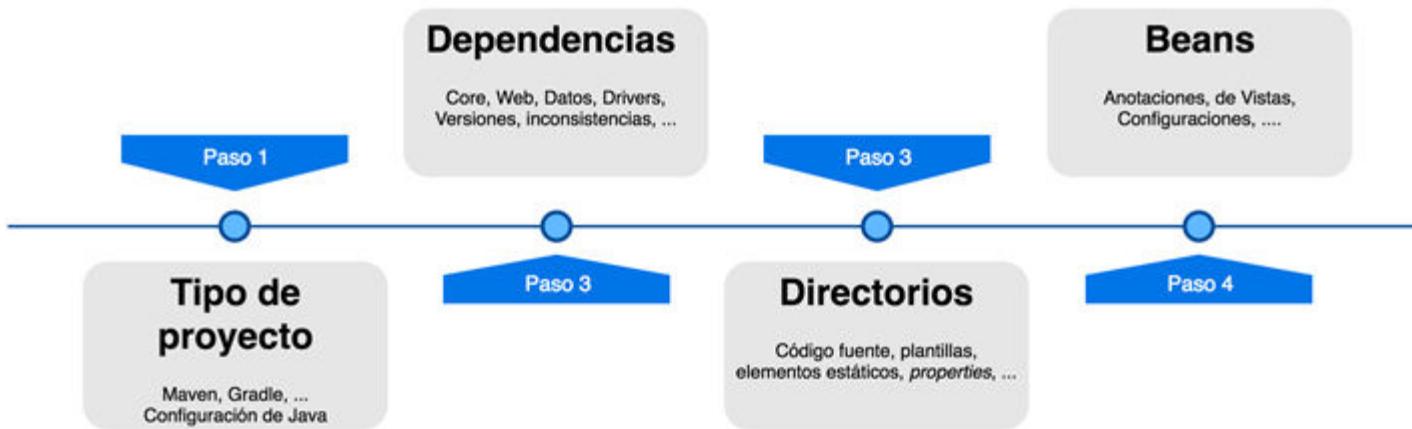


MOTIVACIÓN

Crear un proyecto web con Spring es ciertamente complicado.

Imagen extraída de Freepik

PASOS A SEGUIR



(1) TIPO DE PROYECTO

- ▶ ¿Maven? ¿Gradle? ¿Según IDE?
- ▶ La creación y configuración de un proyecto puede que varíe en función del tipo (o arquetipo) que seleccionemos.
- ▶ En este punto posiblemente tengamos también que configurar bien Java (o Kotlin, o Groovy).

(2) DEPENDENCIAS

- ▶ Escoger la versión concreta de Spring.
- ▶ Añadir las dependencias (por ejemplo Maven) necesarias.
- ▶ ¿Se me olvida alguna?
- ▶ Son las versiones compatibles.

(3) ESTRUCTURA DE DIRECTORIOS

- ▶ Código fuente
- ▶ Plantillas
- ▶ Elementos estáticos (img, js, css,...)
- ▶ Ficheros de configuración y properties
- ▶ Test unitarios
- ▶ Scripts de base de datos
- ▶ ...

(4) CONFIGURACIÓN E INICIALIZACIÓN DE BEANS COMPLEJOS

- ▶ Escaneo de componentes
- ▶ Anotaciones
- ▶ Configuración del *dispatcher servlet*
- ▶ UTF-8
- ▶ Motor de plantillas
- ▶ ...



Imagen extraída de Freepik



BOOT

Al rescate

Imagen extraída de Freepik

SPRING BOOT

Facilita la creación de aplicaciones basadas en Spring **independientes** y **listas para usar**, con un **mínimo esfuerzo**.



Imagen extraída de Freepik

CARACTERÍSTICAS DE SPRING BOOT

- ▶ Creación de aplicaciones Spring independientes
- ▶ Con servidor embebido (Tomcat, Jetty, ...)
- ▶ Dependencias iniciales que facilitan la configuración de componentes.
- ▶ Configuración automática de librerías de 3ºs allá donde sea posible.
- ▶ Sin generar código o configuración XML

VERSIÓN ACTUAL DE SPRING BOOT

2.1.0 **CURRENT** **GA**

2.1.1 **SNAPSHOT**

2.0.7 **SNAPSHOT**

2.0.6 **GA**

1.5.18 **SNAPSHOT**

1.5.17 **GA**

<https://spring.io/projects/spring-boot>

REQUISITOS DE SPRING BOOT 2.1.0.RELEASE

- ▶ Java 8 o 9
- ▶ Spring Framework 5.1.2.RELEASE o superior
- ▶ Maven 3.3 o superior
- ▶ Tomcat 9.0 (Servlet 3.1+, soporte para 4.0)

CONVENCIÓN SOBRE CONFIGURACIÓN



DIFICULTADES AL CONFIGURAR UN PROYECTO SPRING

- Librerías y versiones
- Configuración de componentes.
-



“

La convención sobre configuración es un patrón de diseño de software usado por muchos frameworks que trata de minimizar el número de decisiones que un programador tiene que tomar al usar dicho framework, pero sin perder la flexibilidad.

CoC

- ▶ Si la convención sobre configuración es suficiente para lograr un determinado comportamiento, el programador no tiene que configurar nada explícitamente.
- ▶ Si la convención no es suficiente para obtener el comportamiento deseado, configuramos los aspectos necesarios.

EJEMPLO

- ▶ Una clase llamada *Ventas* se almacenará en una tabla en base de datos llamada *Ventas*.
- ▶ Si quiero que la tabla se llame *VentaProductos*, entonces lo puedo configurar explícitamente.

MÁS EJEMPLOS

- ▶ Si Spring Boot detecta en el classpath la dependencia de H2 (base de datos) y no detecta la configuración de conexión (*datasource*) ejecuta automáticamente una conexión en memoria.
- ▶ Si Spring Boot detecta en el classpath a Thymeleaf (motor de plantillas), por defecto espera que estás estén en la carpeta *templates*, con extensión **.html**

USO DE SPRING INITIALIZR



NUESTRO ENTORNO

- ▶ Java 8 (Oracle JDK)
- ▶ Spring Tool Suite 4 (sobre eclipse)



ÚLTIMA VERSIÓN: SPRING TOOL SUITE 4

- ▶ Bundle junto a Eclipse
- ▶ Plugin para Visual Studio Code
- ▶ Plugin para Atom.io

<https://spring.io/tools>



Spring Tools | 4

PASO 1: COMPROBAR JDK

- ▶ Desde la línea de comandos:

```
java -version
```

```
user@ubuntu:~$ java -version
No se ha encontrado la orden «java», pero se puede instalar con:
sudo apt install default-jre
sudo apt install openjdk-11-jre-headless
sudo apt install openjdk-8-jre-headless
```

PASO 1: COMPROBAR JDK

- ▶ A día de hoy, un plugin (*surefire*) tiene una incidencia abierta que hace que no funcione en ubuntu con JDK.
- ▶ Sin embargo, sí funciona con Oracle JDK

PASO 1: COMPROBAR JDK

- ▶ Pasos a seguir:

```
$ sudo add-apt-repository ppa:webupd8team/java  
$ sudo apt update  
$ sudo apt install oracle-java8-installer
```

PASO 2: INSTALAR MAVEN

- ▶ Pasos a seguir:

```
$ sudo apt install maven
```

PASO 3: DESCARGAR SPRING TOOL SUITE



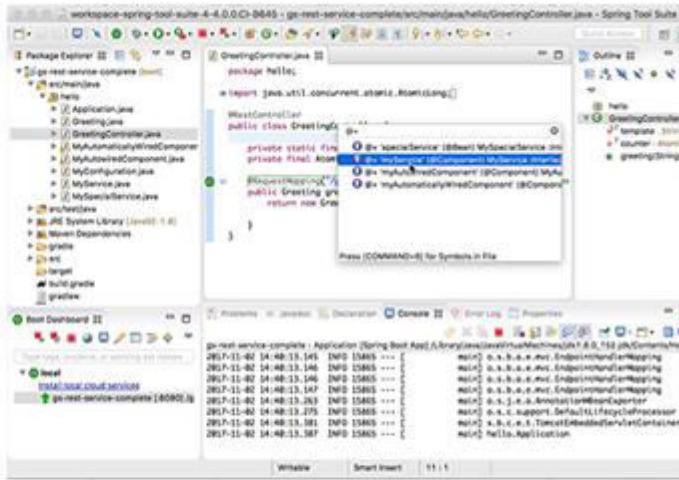
Spring Tools 4 for Eclipse

The all-new Spring Tool Suite 4.
Free. Open source.

[Download STS4
\(for Windows 64-bit\)](#)

[Download STS4
\(for macOS 64-bit\)](#)

[Download STS4
\(for Linux 64-bit\)](#)



<https://spring.io/tools>

PASO 4: DESCOMPRIMIR Y ENLAZAR

- ▶ Desde la línea de comandos:

```
$ cd ~/Descargas
```

```
$ sudo mv spring-.....tar.gz /opt
```

```
$ cd /opt
```

```
$ sudo tar zxvf spring-...tar.gz
```

PASO 4: DESCOMPRIMIR Y ENLAZAR

```
$ sudo ln -s  
    /opt/sts-4...RELEASE/SpringToolSuite4  
    /usr/local/bin/sts
```

PASO 5 (OPCIONAL): CREAR ACCESO DIRECTO

- ▶ Desde la línea de comandos:

```
sudo gedit /usr/share/applications/sts.desktop
```

[Desktop Entry]

Name=Spring Tool Suite 4

Comment=Spring Tool Suite 4 para Ubuntu

Exec=/usr/local/bin/sts

Icon=/opt/sts-4.0.1.RELEASE/icon.xpm

StartupNotify=true

Terminal=false

Type=Application

Categories=IDE;Development;Java;

Y... LISTO



SPRING INITIALIZR

- ▶ Generador rápido de proyectos
- ▶ Vía HTML (<https://start.spring.io/>) o a través de un API (usando nuestro IDE)

SPRING INITIALIZR bootstrap your application now

Generate a with and Spring Boot

Project Metadata

Artifact coordinates
Group:
Artifact:

Dependencies

Add Spring Boot Starters and dependencies to your application.
Search for dependencies

Selected Dependencies

Don't know what to look for? Want more options? [Switch to the full version.](#)

SPRING INITIALIZR

- ▶ Generador rápido de proyectos
- ▶ Vía HTML (<https://start.spring.io/>) o a través de un API (usando nuestro IDE)

SPRING INITIALIZR bootstrap your application now

Generate a with and Spring Boot

Project Metadata

Artifact coordinates
Group:
Artifact:

Dependencies

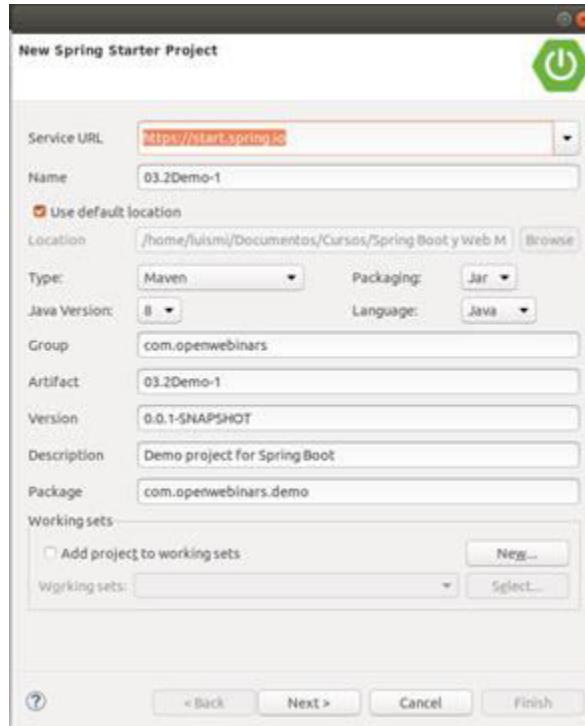
Add Spring Boot Starters and dependencies to your application.
Search for dependencies

Selected Dependencies

Don't know what to look for? Want more options? [Switch to the full version.](#)

SPRING INITIALIZR

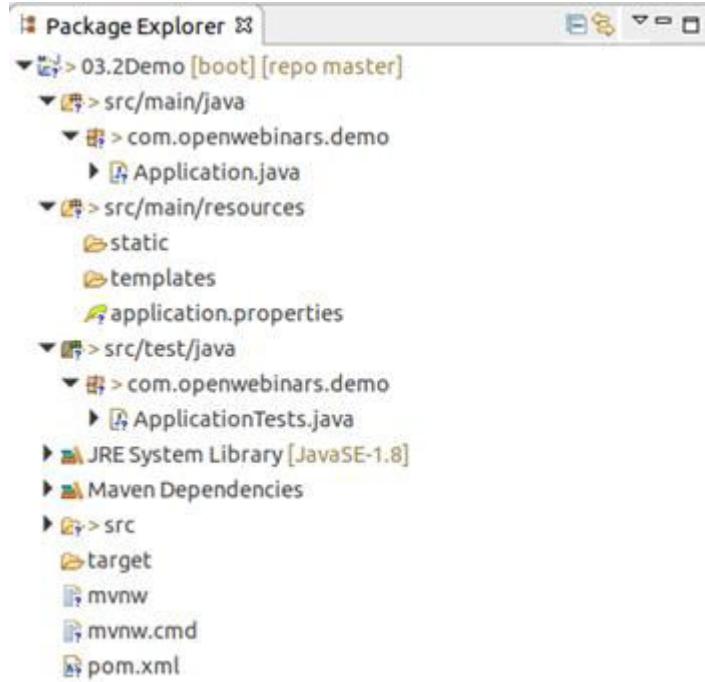
File > New >
Spring Starter Project



ESTRUCTURA DE UN PROYECTO CON SPRING BOOT



PROYECTO GENERADO CON SPRING INITIALIZR



POM.XML

- ▶ Datos de nuestro proyecto
- ▶ *Pom parent*
- ▶ Propiedades
- ▶ Dependencias
- ▶ Plugins

CÓDIGO FUENTE

```
▼ 📂 > src/main/java
  ▼ 📂 > com.openwebinars.demo ← Nuestro código de aplicación
    ➤ 📜 Application.java

▼ 📂 > src/main/resources ← Recursos: imágenes, plantillas,
  static                                ficheros de propiedades, sql, ...
  templates
  📜 application.properties

▼ 📂 > src/test/java
  ▼ 📂 > com.openwebinars.demo ← Test
    ➤ 📜 ApplicationTests.java
```

LIBRERÍAS

- ▶ JRE System Library [JavaSE-1.8] ←
- ▶ Maven Dependencies ←
 - ▶ spring-boot-starter-web-2.1.0.RELEASE.jar - /home/luismi/.m2/repo...
 - ▶ spring-boot-starter-2.1.0.RELEASE.jar - /home/luismi/.m2/repositi...
 - ▶ spring-boot-2.1.0.RELEASE.jar - /home/luismi/.m2/repository/org...
 - ▶ spring-boot-autoconfigure-2.1.0.RELEASE.jar - /home/luismi/.m2/...
 - ▶ spring-boot-starter-logging-2.1.0.RELEASE.jar - /home/luismi/.m2/...
 - ▶ logback-classic-1.2.3.jar - /home/luismi/.m2/repository/ch/qos/lo...
 - ▶ logback-core-1.2.3.jar - /home/luismi/.m2/repository/ch/qos/log...
 - ▶ log4j-to-slf4j-2.11.1.jar - /home/luismi/.m2/repository/org/apache...
 - ▶ log4j-api-2.11.1.jar - /home/luismi/.m2/repository/org/apache/lo...
 - ▶ jul-to-slf4j-1.7.25.jar - /home/luismi/.m2/repository/org/slf4j/jul-t...
 - ▶ javax.annotation-api-1.3.2.jar - /home/luismi/.m2/repository/java...
 - ▶ snakeyaml-1.23.jar - /home/luismi/.m2/repository/org/yaml/snake...
 - ▶ spring-boot-starter-json-2.1.0.RELEASE.jar - /home/luismi/.m2/repo...
 - ▶ jackson-databind-2.9.7.jar - /home/luismi/.m2/repository/com/fasterxml...

Nuestro versión de la JVM

Librerías asociadas a cada una de las dependencias Maven que hemos incluido antes.

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>

    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

CLASE MAIN

```
@SpringBootApplication ← Anotación mágica que  
public class Application { autoconfigura nuestra aplicación  
  
    public static void main(String[] args) {  
        SpringApplication.run(Application.class, args);  
    }  
}
```



Comienza a ejecutar Spring, el servidor de Tomcat y tras eso nuestra aplicación.

EJECUCIÓN DEL PROYECTO

Varias posibilidades

- ▶ Desde el IDE: *Run As > Spring Boot App*
- ▶ Desde el terminal
 - ▶ Ejecutar: mvn spring-boot:run
 - ▶ Generar jar:
 - ▶ mvn package
 - ▶ cd target
 - ▶ java -jar project-name.jar

SPRING BOOT Y STARTERS



POM.XML

- ▶ Datos de nuestro proyecto
- ▶ *Pom parent*
- ▶ Propiedades
- ▶ Dependencias
- ▶ Plugins

POM JERÁRQUICO

- ▶ El pom.xml de nuestros proyectos se basa en otro.
 - ▷ <https://github.com/spring-projects/spring-boot/blob/master/spring-boot-project/spring-boot-starters/spring-boot-starter-parent/pom.xml>
- ▶ Incluye los elementos más comunes a cualquier proyecto.

STARTER

- ▶ Descriptores de dependencia fáciles de usar.
- ▶ *Ventanilla única* para agregar a un proyecto las tecnologías de Spring y otras relacionadas.
- ▶ Nos libera de añadir dependencias concreta una a una.

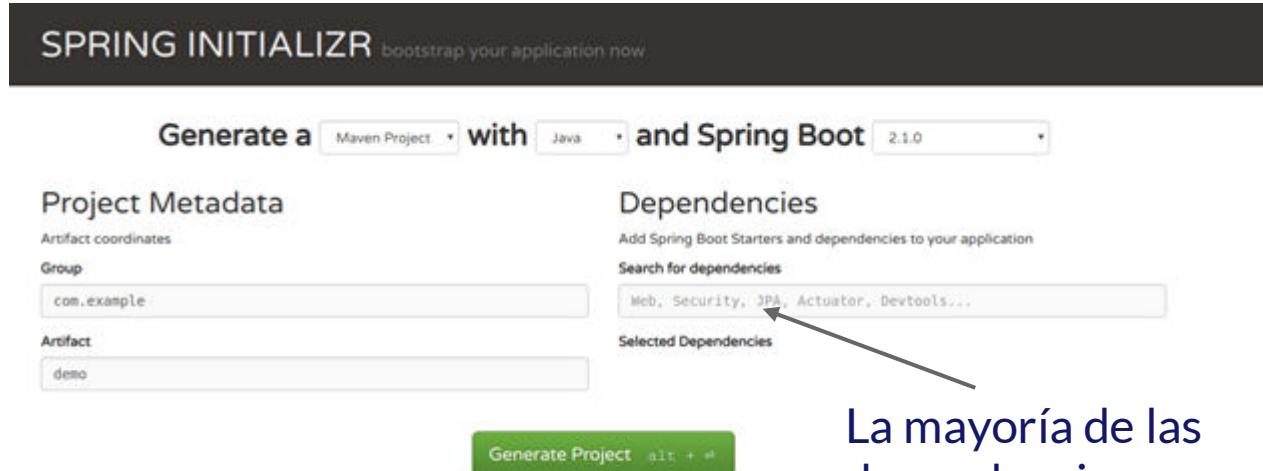
EJEMPLO DE STARTER

Si nuestro proyecto va a ser una aplicación web que accede a una base de datos relacional, tan solo tenemos que añadir.

- ▶ `spring-boot-starter-data-jpa`
- ▶ `spring-boot-starter-web`

Esto añade múltiples ficheros .jar así como diferentes elementos de configuración.

AÑADIR STARTERs DURANTE LA DEFINICIÓN DEL PROYECTO

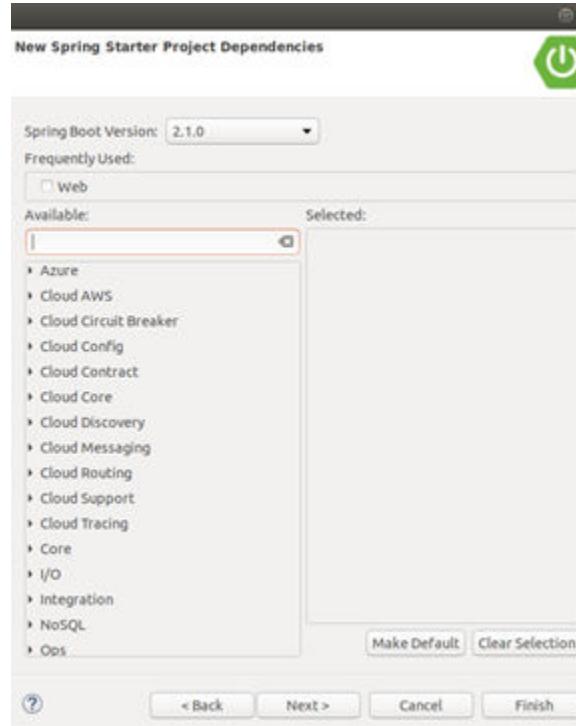


The screenshot shows the Spring Initializer web interface. At the top, it says "SPRING INITIALIZER bootstrap your application now". Below that, it asks "Generate a [Maven Project] with [Java] and Spring Boot [2.1.0]". The "Project Metadata" section has "Artifact coordinates" fields for "Group" (containing "com.example") and "Artifact" (containing "demo"). The "Dependencies" section has a search bar with "Web, Security, JPA, Actuator, Devtools..." and a "Selected Dependencies" list below it. A green button at the bottom says "Generate Project alt + ⌘".

La mayoría de las dependencias que añadimos aquí son starters.

AÑADIR STARTERs DURANTE LA DEFINICIÓN DEL PROYECTO

Al igual que si la añadimos por esta vía.



AÑADIR STARTERs DURANTE EL DESARROLLO DEL PROYECTO

- ▶ Si los queremos añadir más adelante, debe ser directamente en el pom.xml
- ▶ Fácilmente localizables en [mvnrepository](#).

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
</dependencies>
```

STARTERS MÁS USUALES

Todos los paquetes comienzan con `spring-boot-starter-`

Nombre	Descripción
<code>web</code>	Aplicaciones web, API RESTful, MVC
<code>data-jpa</code>	Spring Data Jpa con Hibernate
<code>thymeleaf</code>	Thymeleaf como motor de plantillas
<code>test</code>	Test con JUnit, Hamcrest y Mockito
<code>actuator</code>	Funciones de monitorización y administración para producción

<https://docs.spring.io/spring-boot/docs/current/reference/html/using-boot-build-systems.html#using-boot-starter>

ESTRUCTURA DEL CÓDIGO



ESTRUCTURA DEL CÓDIGO CON SPRING BOOT

- ▶ Spring Boot no nos obliga estructurar el código de una forma específica
- ▶ Sí podemos seguir una serie de buenas prácticas.

USO DEL PAQUETE default

- ▶ En Java, una clase sin paquete se considera que está en el paquete por defecto (default)
- ▶ Si usamos la anotación `@SpringBootApplication` (o similar), podemos encontrarnos con problemas.
- ▶ Es muy recomendable siempre usar una estructura de paquetes como un dominio a la inversa: `com.openwebinars.proyecto`.

UBICACIÓN DE LA CLASE **MAIN**

- ▶ Será habitual que tengamos una clase principal, anotada con **@SpringBootApplication**.
- ▶ Esta clase debería estar en el paquete raíz, ya que delimita el paquete (y subpaquetes) donde se escaneará automáticamente para buscar componentes.

UBICACIÓN DE LA CLASE MAIN

```
com
+- example
  +- myapplication ←
    +- Application.java
    |
    +- customer
      |   +- Customer.java
      |   +- CustomerController.java
      |   +- CustomerService.java
      |   +- CustomerRepository.java
      |
    +- order
      +- Order.java
      +- OrderController.java
      +- OrderService.java
      +- OrderRepository.java
```

Si la clase Application está anotada con @SpringBootApplication, y creamos una componente que esté en un paquete fuera de esta jerarquía, no será detectado por el escaneo automático. ¡CUIDADO! Es un error muy frecuente.

AUTOCONFIGURACIÓN Y USO DE `@SPRINGBOOTAPPLICATIO N`



CONFIGURACIÓN DE UNA APLICACIÓN SPRING

- ▶ Múltiples elementos de configuración.
- ▶ Clases anotadas con @Configuration.
- ▶ Uso de properties.

AUTOCONFIGURACIÓN CON SPRING BOOT

- ▶ Se obtiene gracias a `@SpringBootApplication` (o `@EnableAutoconfiguration`).
- ▶ Configura nuestra aplicación en función de las dependencias jar que encuentre en el *classpath*.
 - ▷ Si añadimos la dependencia de H2 sin configuración, autoconfigura una base de datos en memoria.

CÓMO VER LA CONFIGURACIÓN POR DEFECTO DE UN PROYECTO

- ▶ mvn --debug spring-boot:run

CÓMO DESHABILITAR UN ELEMENTO ESPECÍFICO DE LA AUTOCONFIGURACIÓN

- ▶ Si nos interesa que no se aplique un elemento específico de autoconfiguración, lo podemos deshabilitar con `@EnableAutoConfiguration(exclude=...)`

```
import org.springframework.boot.autoconfigure.*;
import org.springframework.boot.autoconfigure.jdbc.*;
import org.springframework.context.annotation.*;

@Configuration
@EnableAutoConfiguration(exclude={DataSourceAutoConfiguration.class})
public class MyConfiguration {
}
```

CÓMO DESHABILITAR UN ELEMENTO ESPECÍFICO DE LA AUTOCONFIGURACIÓN

- ▶ También lo podemos hacer mediante la propiedad `spring.autoconfigure.exclude`, que acepta una lista de clases (preferiblemente, nombres cualificados: `paquete.nombre`).

CÓMO FUNCIONA UNA APLICACIÓN WEB



¿QUÉ ES UNA APLICACIÓN WEB?

- ▶ Herramienta accesible a través de una red (intranet o internet) mediante un navegador web.



VENTAJAS

- ▶ No requiere de instalación ni actualización por el usuario.
- ▶ Es multiplataforma y multidispositivo.
- ▶ Fácil acceso: internet democrático.
- ▶ Ahorro de tiempo (cliente y programador)
- ▶ Ahorro de espacio (al cliente)
- ▶ Colaboración (compartir entre usuarios)

Imágenes extraídas de Freepik



DESVENTAJAS

- ▶ Menor velocidad y capacidad de procesamiento.
- ▶ Menos funcionalidades que aplicaciones nativas.
- ▶ Disponibilidad: posible falta de conectividad, caída de servidores,...

BASADAS EN EL PROTOCOLO HTTP

- ▶ Transferencia de hipertexto (WWW)
- ▶ Versión actual: 1.1 - 2.0
- ▶ Sin estados



PROTOCOLO HTTP

- ▶ Esquema de petición - respuesta

```
GET http://www.dom.com HTTP/1.1
Host: www.dom.com
Accept: application/xhtml+xml
User-Agent: ....
```

```
HTTP/1.1 200 OK
Content-Length: 12345
Content-Type: application/xhtml+xml
...
```



ACCESO A UNA APLICACIÓN WEB: URLs

- ▶ Podemos acceder a una aplicación web a través de su URL
- ▶ URL: localizador unificado de recursos

protocolo://maquina:puerto/camino/al/recurso

Ejemplos: <http://www.openwebinars.net>,
<http://www.google.com>,
<https://es.wikipedia.org>

MÉTODOS DE PETICIÓN HTTP

- ▶ También conocidos como verbos
- ▶ Indica qué operación queremos realizar con el recurso.

GET

solicita al servidor que envíe el recurso identificado por la URL

HEAD

pide al servidor que envíe una respuesta idéntica a la que enviaría con GET, pero sin el cuerpo de la respuesta.

POST

envía datos al servidor para que sean procesados por el recurso identificado por la URL. Los datos se deben incluir en el cuerpo de la petición.

MÉTODOS DE PETICIÓN HTTP

PUT

envía un recurso determinado (un archivo) al servidor.

DELETE

solicita la eliminación de un recurso.

TRACE

solicita al servidor que le envíe un mensaje de respuesta. Se usa para diagnosticar problemas de conexión.

OPTIONS

pide al servidor que le indique los métodos HTTP que soporta para una determinada URL.

PATCH

se emplea para modificar parcialmente un recurso ya existente en el servidor.

CÓDIGOS DE RESPUESTA HTTP

1XX

Respuestas informativas. Indica que la petición ha sido recibida y se está procesando.

2XX

Respuesta correctas. Indica que la petición se ha procesado correctamente (200 OK, 201 Created,...)

3XX

Respuestas de redirección. Indica que el cliente necesita realizar más acciones para finalizar la petición.

4XX

Errores causados por el cliente. Indica que ha habido un error en el procesado de la petición a causa de que el cliente ha hecho algo mal.

(400 Bad Request, 404 Not Found, ...)

5XX

Errores causados por el servidor. Indica que ha habido un error en el procesado de la petición a causa de un fallo en el servidor.

TECNOLOGÍAS QUE USAREMOS CON LAS APLICACIONES WEB



HTML5 Thymeleaf
CSS3
JS, jQuery

Spring Boot
Spring Web MVC
Spring Data JPA
Hibernate

H2

ALGUNOS PATRONES DE DISEÑO



“

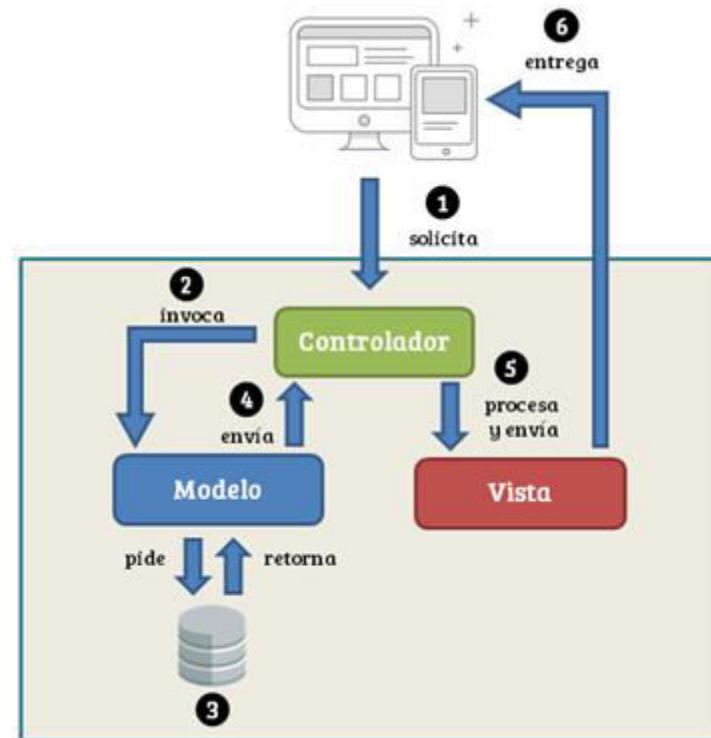
Christopher Alexander (arquitecto).
The Timeless Way of Building (1979)

Cada patrón describe un problema que ocurre infinidad de veces en nuestro entorno, así como la solución al mismo, de tal modo que podemos utilizar esta solución un millón de veces más adelante sin tener que volver a pensarla otra vez.

PATRÓN DE DISEÑO

- ▶ Una posible solución correcta a un problema de diseño dentro de un contexto, y que se presenta frecuentemente.
- ▶ Auge a partir del libro *Design Patterns* escrito por el grupo Gang of Four (GoF) (Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides)

PATRÓN MVC: MODELO - VISTA - CONTROLADOR



PATRÓN MVC: MODELO - VISTA - CONTROLADOR

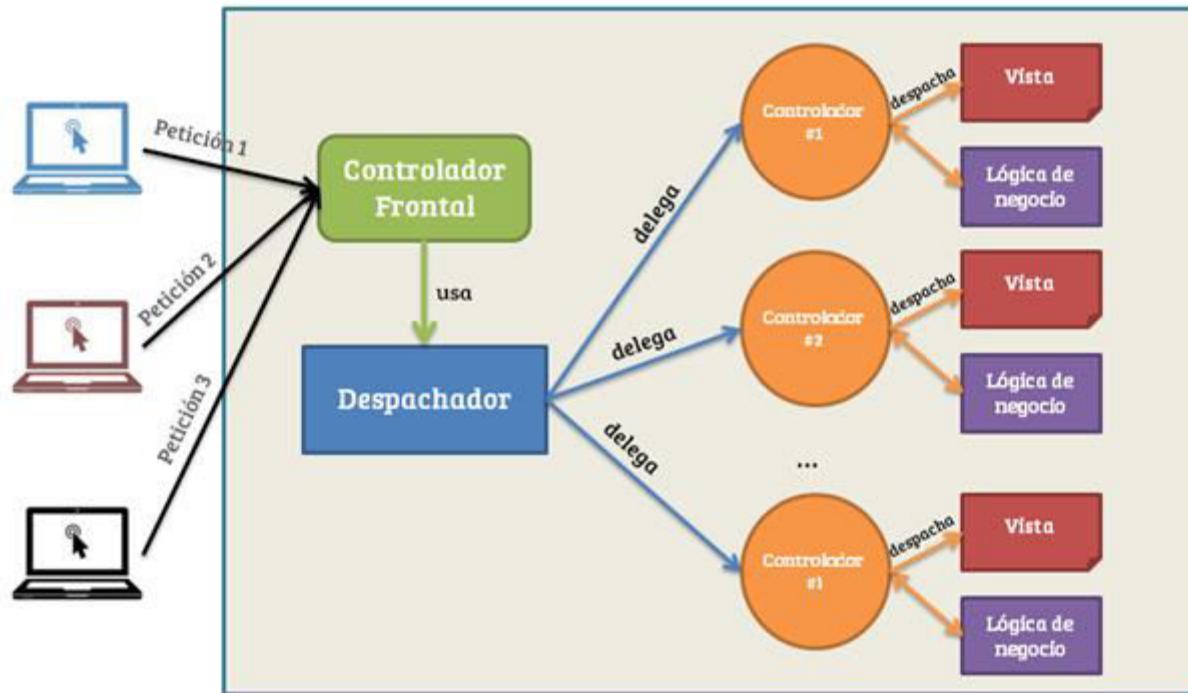
VENTAJAS

- Adaptación al cambio.
- Soporte para múltiples tipos de vistas.

DESVENTAJAS

- Complejidad
- Coste de actualizaciones frecuentes.

PATRÓN FRONT CONTROLLER



PATRÓN FRONT CONTROLLER

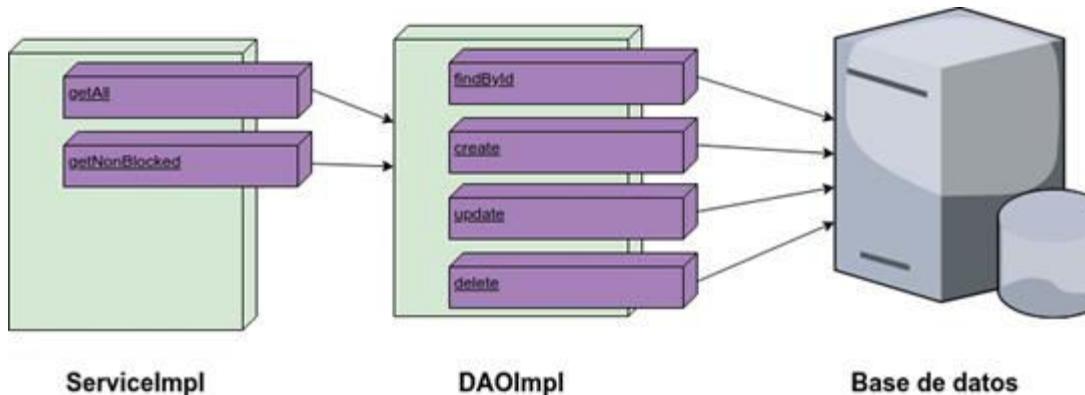
VENTAJAS

- Centralización en un único punto de gestión de peticiones.
- Aumento de la reusabilidad.
- Mejora de la seguridad.

DESVENTAJAS

- Disminución de la velocidad de respuesta (efecto embudo).

PATRÓN DAO: DATA ACCESS OBJECT

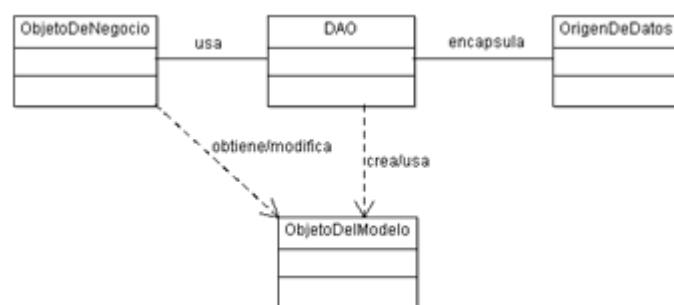


ORACLE
D A T A B A S E

Microsoft®
SQL Server®

PostgreSQL

MySQL®



PATRÓN DAO: DATA ACCESS OBJECT

VENTAJAS

- Adaptación al cambio.
- Un objeto de negocio no tiene que conocer el destino de la información.

DESVENTAJAS

- Complejidad.
- Configuración adicional.
- Rendimiento en aplicaciones críticas.

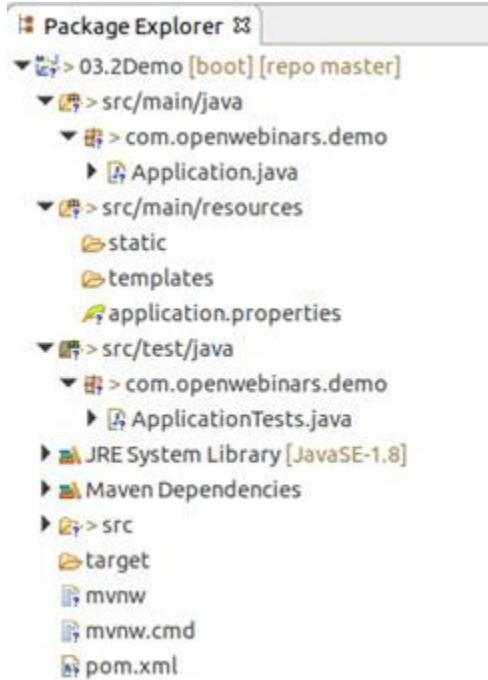
ESTRUCTURA DE UN PROYECTO WEB



APLICACIÓN WEB



PROYECTO DE SPRING BOOT GENERADO CON SPRING INITIALIZR



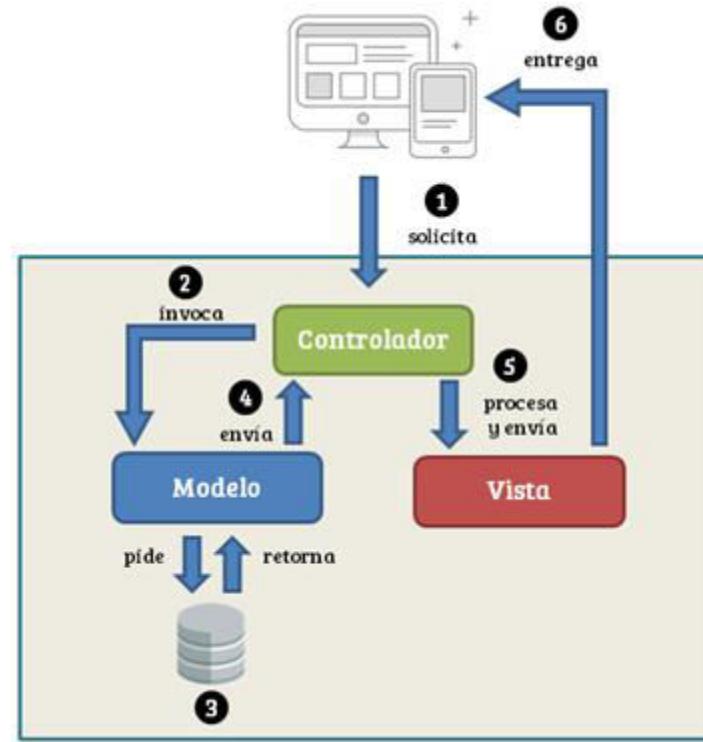
¿Qué elementos va a incluir un proyecto para el desarrollo de una aplicación web?
El patrón MVC nos dará pistas.

ESTRUCTURA SEGÚN MVC

- ▶ Controladores
- ▶ Modelo
 - ▶ Entidades
 - ▶ Repositorios
 - ▶ Servicios
- ▶ Vistas

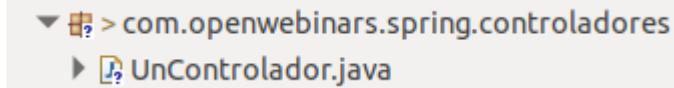
Podemos añadir

- ▶ Configuración
- ▶ Seguridad
- ▶ Utilidades
- ▶ ...



CONTROLADORES

- ▶ Clases con métodos que atenderán las peticiones desde el navegador.
- ▶ Acceden al modelo, y lo retornan a la vista.
- ▶ Al menos un paquete para almacenarlos.



▼ > com.openwebinars.spring.controladores
► UnControlador.java

A screenshot of a Java IDE showing a package structure. The package 'com.openwebinars.spring.controladores' is expanded, revealing a single file named 'UnControlador.java'. The file is represented by a blue folder icon with a question mark inside.

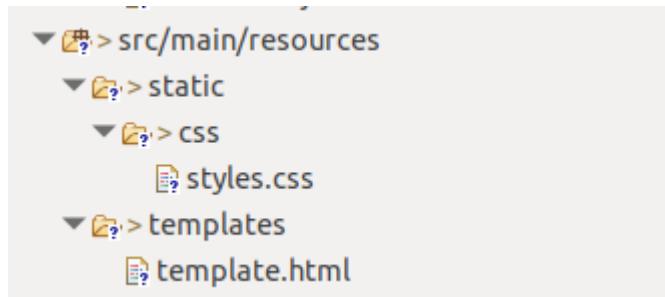
MODELO

- ▶ Entidades: clases que modelan los objetos de nuestro modelo de negocio.
- ▶ Repositorios (almacenes)
- ▶ Servicios (Lógica de negocio)
- ▶ Un paquete por cada tipo



VISTAS

- ▶ Plantillas (normalmente, HTML) que, tras ser procesadas, serán visualizadas por el navegador.
- ▶ Recursos estáticos (CSS, imágenes, JS, ...)



Plantillas y recursos pueden ser organizados en carpetas, según nos convenga.

¿Y EL RESTO DE PAQUETES?

- ▶ Configuración
- ▶ Seguridad
- ▶ Utilidades

CONFIGURACIÓN

- ▶ Clases anotadas con @Configuration
- ▶ Modifican aspectos de configuración por defecto.
- ▶ Ejemplo: configuración de las propiedades de la base de datos.

SEGURIDAD

- ▶ Útil si usamos Spring Security
- ▶ Clase que extiende a WebSecurityConfigurerAdapter
- ▶ Si nuestro manejo de seguridad es complejo, podemos tener subpaquetes (modelo, servicios, configuración, filtros, ...)

UTILIDADES

- ▶ Clases auxiliares que nos sirven para algunas tareas.
- ▶ Pueden ser o no componentes Spring.

CONCEPTOS DE JAVA EE NECESARIOS



“

Una aplicación web con Spring MVC
no deja de ser una aplicación web Java.

Por tanto, necesitamos conocer
algunos conceptos de este modelo de
desarrollo de aplicaciones.

SERVLET

- ▶ Clase que nos permite gestionar peticiones/resuestas de una aplicación servidora.
- ▶ Usualmente usados en contextos web bajo el protocolo HTTP.
- ▶ Se ejecutan en un contenedor de servlets (por ejemplo, Tomcat)

EJEMPLO DE UN SERVLET

```
1 import java.io.IOException;
2 import java.io.PrintWriter;
3 import javax.servlet.http.HttpServlet;
4 import javax.servlet.http.HttpServletRequest;
5 import javax.servlet.http.HttpServletResponse;
6 import javax.servlet.annotation.WebServlet;
7
8 @WebServlet("/index")
9 public class ServletExample extends HttpServlet{
10
11     public void doGet(HttpServletRequest request, HttpServletResponse response)
12         throws IOException{
13         PrintWriter out = response.getWriter();
14         out.println("<html>");
15         out.println("<body>");
16         out.println("<h1>Hello Servlet Get</h1>");
17         out.println("</body>");
18         out.println("</html>");
19     }
20 }
```

WEB.XML: DESCRIPTOR DE DESPLIEGUE

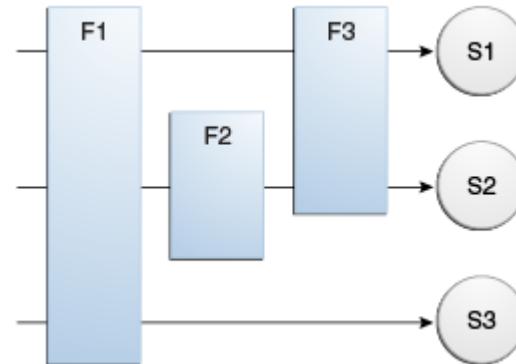
- ▶ Fichero que indica cómo desplegar en el servidor los componentes de la aplicación (servlets entre otros)
- ▶ Formato XML
- ▶ Ruta /WEB-INF/web.xml

WEB.XML: DESCRIPTOR DE DESPLIEGUE

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_9" version="3.0" xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
        http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
    <display-name>Servlet 3.0 application</display-name>
    <servlet>
        <servlet-name>welcome</servlet-name>
        <servlet-class>WelcomeServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>welcome</servlet-name>
        <url-pattern>/hello.welcome</url-pattern>
    </servlet-mapping>
</web-app>
```

FILTER

- ▶ Los filtros permiten transformar peticiones (sobre todo) o respuestas HTTP
 - ▶ También nos permiten alterar el *circuito* normal que seguiría una petición/respuesta en función del contenido de estas.
 - ▶ Por ejemplo, seguridad, cambiar codificación a UTF-8,
- ...



FILTER

```
@WebFilter("/AuthenticationFilter")
public class AuthenticationFilter implements Filter {

    public void init(FilterConfig fConfig) throws ServletException {
    }

    public void doFilter(ServletRequest request, ServletResponse response, FilterChain chain) throws IOException, ServletException {
        HttpServletRequest req = (HttpServletRequest) request;
        HttpServletResponse res = (HttpServletResponse) response;

        String uri = req.getRequestURI();
        this.context.log("Requested Resource::"+uri);

        HttpSession session = req.getSession(false);

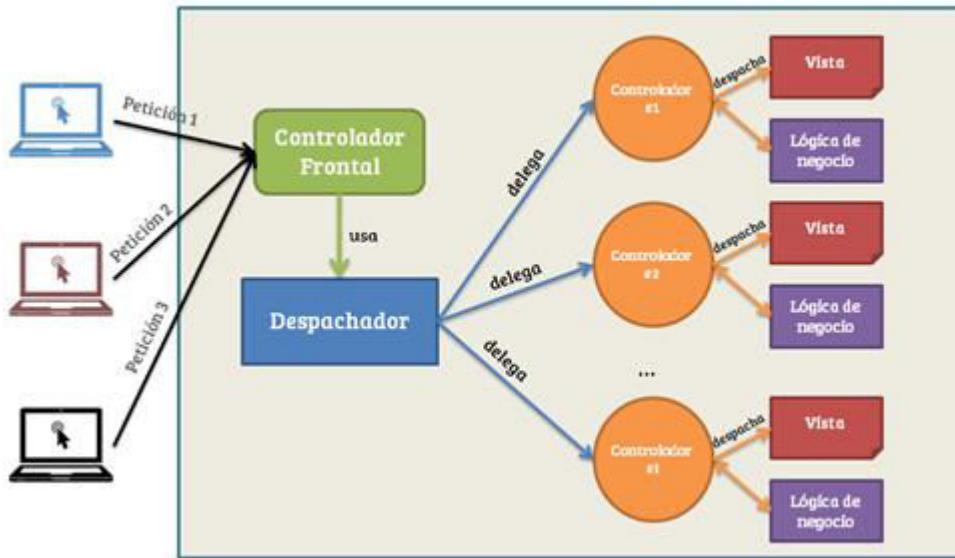
        if(session == null && !(uri.endsWith("html") || uri.endsWith("LoginServlet"))){
            this.context.log("Unauthorized access request");
            res.sendRedirect("login.html");
        }else{
            // pass the request along the filter chain
            chain.doFilter(request, response);
        }
    }

    public void destroy() {
        //close any resources here
    }
}
```

¿Y CÓMO USA TODO ESTO SPRING MVC?

DISPATCHER SERVLET

- ▶ Implementación del patrón **Front Controller**



DISPATCHER SERVLET

- ▶ Despacha todas las peticiones (delegando en otras clases).
- ▶ Necesita ser mapeado (descriptor de despliegue o JavaConfig).
- ▶ *Si usamos Spring Boot, esto no es necesario, él se encarga de esta configuración.*

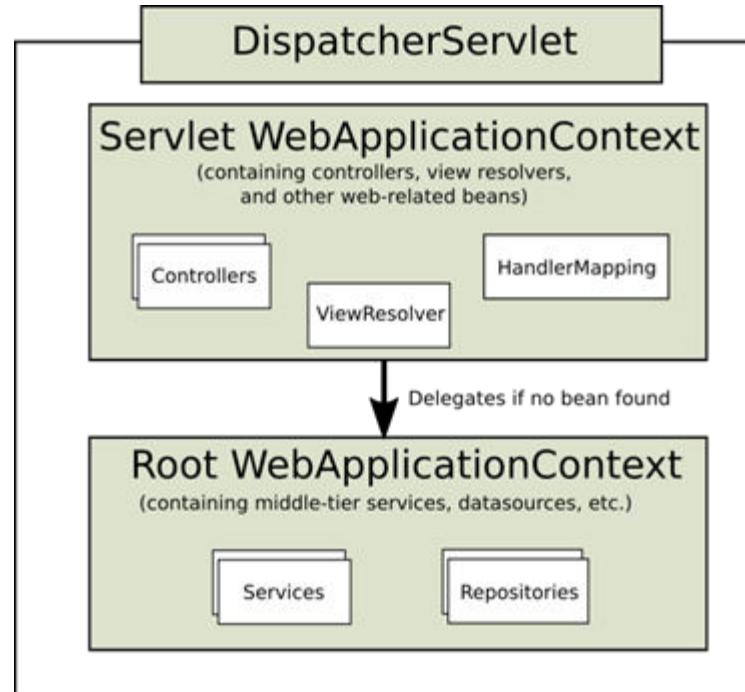
<https://docs.spring.io/spring/docs/5.1.2.RELEASE/spring-framework-reference/web.html#mvc-servlet>

CONTEXTOS

- ▶ Una aplicación Spring necesita un *ApplicationContext*: contenedor de inversión de control.
- ▶ Para apps web tenemos *WebApplicationContext*.

CONTEXTOS

Usualmente (y así trabajaremos durante el curso) solo necesitaremos un *WebApplicationContext*, pero podemos tener una jerarquía.



ALGUNOS BEANS ESPECIALES

Colaboran con el DispatcherServlet

- ▶ *HandlerMapping* y *HandlerAdapter*: mapeo de peticiones a métodos de un controlador.
- ▶ *ViewResolver*: transforma nombres de vistas en vistas con las que renderizar una respuesta.

¿Y QUÉ HACE SPRING BOOT POR NOSOTROS?

CONFIGURACIÓN DE DISPATCHER SERVLET

Sin Spring Boot, tendríamos que hacer esto
manualmente.

```
import org.springframework.web.WebApplicationInitializer;

public class MyWebApplicationInitializer implements WebApplicationInitializer {

    @Override
    public void onStartup(ServletContext container) {
        XmlWebApplicationContext appContext = new XmlWebApplicationContext();
        appContext.setConfigLocation("/WEB-INF/spring/dispatcher-config.xml");

        ServletRegistration.Dynamic registration = container.addServlet("dispatcher",
new DispatcherServlet(appContext));
        registration.setLoadOnStartup(1);
        registration.addMapping("/");
    }
}
```

CONFIGURACIÓN DE DISPATCHER SERVLET

Sin Spring Boot, tendríamos que hacer esto
manualmente.

```
public class MyWebAppInitializer extends
AbstractAnnotationConfigDispatcherServletInitializer {

    @Override
    protected Class<?>[] getRootConfigClasses() {
        return null;
    }

    @Override
    protected Class<?>[] getServletConfigClasses() {
        return new Class<?>[] { MyWebConfig.class };
    }

    @Override
    protected String[] getServletMappings() {
        return new String[] { "/" };
    }
}
```

CONFIGURACIÓN DE DISPATCHER SERVLET

Sin Spring Boot, tendríamos que hacer esto
manualmente.

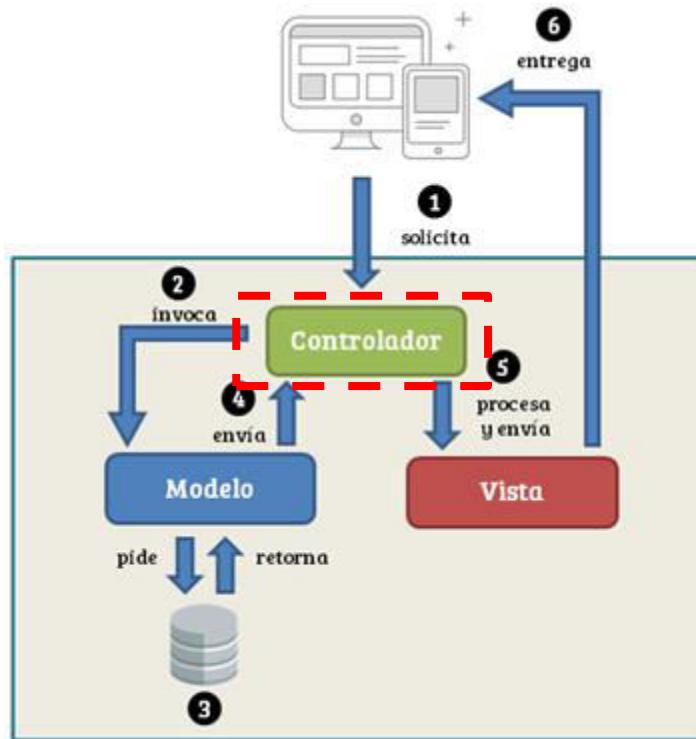
```
public class MyWebAppInitializer extends AbstractDispatcherServletInitializer {  
  
    // ...  
  
    @Override  
    protected Filter[] getServletFilters() {  
        return new Filter[] {  
            new HiddenHttpMethodFilter(), new CharacterEncodingFilter() };  
    }  
}
```



CONTROLADORES

CONTROLADOR

Clase que se va a encargar de atender peticiones y derivarnos a una vista adecuada.



¿CÓMO SE CREA UN CONTROLADOR?

- ▶ Clase POJO Java (*Plain Old Java Object*)
- ▶ Anotación @Controller
- ▶ Métodos anotados con @RequestMapping o sus derivados
 - ▷ @GetMapping
 - ▷ @PostMapping
 - ▷ @PutMapping
 - ▷ @DeleteMapping
 - ▷ @PatchMapping

ESTRUCTURA BÁSICA DE UN MÉTODO DEL CONTROLADOR

```
@Controller  
public class MainController {  
  
    @GetMapping("/")  
    public String welcome() {  
        return "index";  
    }  
}
```

La clase es un pojo, no tiene que extender a ninguna otra obligatoriamente, tan solo anotada con `@Controller`

La anotación `@GetMapping` indica que este método se invoca cuando se produce una petición GET a /

El método puede tener un nombre cualquiera, y no tiene por qué recibir parámetros

El método devuelve un `String`. Ruta de la plantilla sin la extensión (que se supone es `.html`)

CÓMO ENVIAR DATOS A LA VISTA

La clase *Model* es un *Map*, que nos permite *pasar* objetos del controlador a la vista.

```
@GetMapping("/")
public String welcome(Model model) {
    model.addAttribute("mensaje", "¡Hola a todos!");
    return "index";
}
```

En nuestra plantilla Thymeleaf, podemos utilizar los datos recibidos.

```
<h1 th:text="${mensaje}">Mensaje</h1>
```

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html>

POSIBLES ARGUMENTOS DE UN MÉTODO DEL CONTROLADOR

- ▶ *java.util.Map, org.springframework.ui.Model,
org.springframework.ui.ModelMap*

Nos permite pasar datos a la vista

- ▶ *@ModelAttribute*

Permite acceder a un objeto del modelo
(útil con formularios)

POSIBLES ARGUMENTOS DE UN MÉTODO DEL CONTROLADOR

- ▶ *@RequestBody*
Permite acceder a un objeto presente en el cuerpo de la petición
- ▶ *HttpEntity<?>*
Permite acceder a la petición (encabezado y cuerpo)

POSIBLES TIPOS DE RETORNO EN UN MÉTODO DEL CONTROLADOR

- ▶ *String*: es el más usual en las últimas versiones de Spring. Se trata del nombre de la plantilla, que será resuelto por el *ViewResolver* configurado (Spring Boot + Thyemeleaf en el classpath)
- ▶ *@ResponseBody* + cualquier tipo de dato: se convierte el valor devuelto a través del conversor configurado.

POSIBLES TIPOS DE RETORNO EN UN MÉTODO DEL CONTROLADOR

- ▶ *HttpEntity<?>, ResponseEntity<?>*: se devuelve la respuesta HTTP completa (cabeceras y cuerpo).
- ▶ *ModelAndView*: modelo + vista en un solo objeto.

¿CUÁNTOS MÉTODOS PUEDE TENER UN CONTROLADOR?

- ▶ Puede tener cuantos necesitemos.
- ▶ No hay un límite determinado. Lo establece el diseño de clases de nuestra aplicación.

```
@Controller
public class MainController {

    @GetMapping("/")
    public String welcome(Model model) {
        model.addAttribute("mensaje", "¡Hola a todos!");
        return "index";
    }

    @GetMapping("/saludo")
    public String saludoCompleto(Model model) {
        model.addAttribute("saludo",
            "Seguro que has visto otras plataformas con miles de cursos, pero en OpenWebinars
        return "saludo";
    }
}
```

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

CONTROLADOR DE UNA WEB CLÁSICA

- ▶ Las webs clásicas de pequeños negocios solían incluir:
 - ▷ **Portada.** Información sobre la organización (quiénes somos)
 - ▷ **Qué hacemos**
 - ▷ Dónde estamos e información de contacto.

CONTROLADOR DE UNA WEB CLÁSICA

- ▶ Crea un nuevo proyecto, añadiendo los starters de WEB y THYMELEAF.
- ▶ Crea un controlador con 3 métodos, para que atiendan a las rutas `/`, `/que`, `/contacto`
- ▶ Crea las plantillas necesarias (`index.html`, `que.html` y `contacto.html`).
- ▶ Puedes pasar el contenido de las plantillas a través de un objeto `Model`.

CÓMO ENVIAR DATOS A LA VISTA

La clase *Model* es un *Map*, que nos permite *pasar* objetos del controlador a la vista.

```
@GetMapping("/")
public String welcome(Model model) {
    model.addAttribute("mensaje", "¡Hola a todos!");
    return "index";
}
```

En nuestra plantilla Thymeleaf, podemos utilizar los datos recibidos.

```
<h1 th:text="${mensaje}">Mensaje</h1>
```

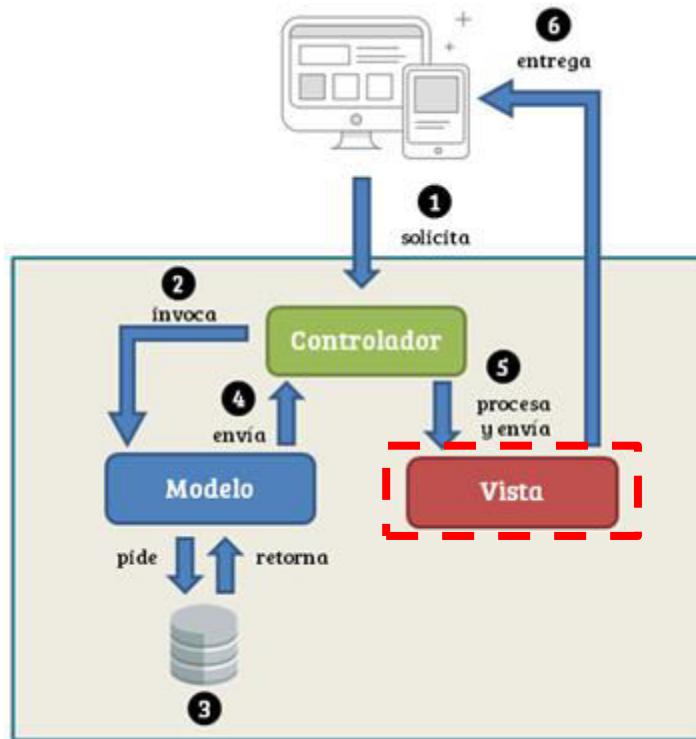
<https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/ui/Model.html>



VISTAS EN SPRING MVC

VISTA

Es la parte encargada de renderizar las plantillas que visualizará el usuario.



“

Spring MVC está diseñado para ser **independiente** de la tecnología de la **vista**. Podemos verlo como piezas de un puzzle que encajan. Así podemos seleccionar la tecnología más adecuada.

POSIBLES TECNOLOGÍAS EN LA VISTA

Thymeleaf

Motor de plantillas que enfatiza el uso de plantillas naturales. Su integración con Spring es muy sencilla, e inmediata con Spring Boot.

FreeMarker

Apache FreeMarker es un motor de plantillas para generar contenido HTML. Integración sencilla con Spring.

Groovy Markup

Motor de plantillas para generar contenido XML-like. Requiere del uso de Groovy 2.3.1+

JSP + JSTL

Tecnología Java que permite crear páginas web dinámicas. A nivel de rendimiento es equivalente al uso de Servlets.

Tiles

Apache Tiles es un marco de desarrollo de plantillas que ha sido muy popular por su uso junto a Struts.

Otras (JSR-223)

Handlebars, Mustache, React, EJS, ... o cualquier otro sistema que pueda correr sobre un motor de scripting JSR-233.

CARACTERÍSTICAS DE THYMELEAF

- ▶ Es un motor de plantillas: plantilla + modelo = resultado.

modelo

nombre = Pepe

apellidos=Pérez Pérez

plantilla

<div ...>

 <p>\${nombre}</p>

 <p>\${apellidos}</p>

</div>

resultado

<div ...>

 <p>Pepe</p>

 <p>Pérez Pérez</p>

</div>

CARACTERÍSTICAS DE THYMELEAF

- ▶ Natural Templating

```
<html>
<head>
    <title>Hola mundo</title>
</head>
<body>
    <p th:text="${saludo}">Hola Mundo</p>
</body>
</html>
```

Esta etiqueta es propia de HTML. El atributo no lo es, y el navegador lo descarta, pero no da error.

Si trabajamos con esta plantilla de forma estática (sin pasar por el motor), podemos ver un texto por defecto.

CARACTERÍSTICAS DE THYMELEAF

- ▶ **Procesador:** un objeto que aplica una transformación a un determinado artefacto (texto, etiqueta, comentario,...)
- ▶ **Dialectos:** conjunto de procesadores.
- ▶ Spring posee un dialecto propio (*SpringStandardDialect*).
- ▶ Nos permite utilizar SpEL (Spring Expression Language) en lugar del lenguaje de expresiones por defecto.

CÓMO CONFIGURAR THYMELEAF COMO MOTOR DE PLANTILLAS

- ▶ La forma más sencilla: Spring Boot + starter de Thymeleaf

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

¿QUÉ HACE SPRING BOOT POR NOSOTROS PARA CONFIGURAR THYMELEAF?

- ▶ Configura varios beans
 - ▷ *ViewResolver (ThymeleafViewResolver)* Encargado de resolver las vistas.
 - ▷ *TemplateEngine (SpringTemplateEngine)*
Indica el dialecto Thymeleaf y habilita el lenguaje de expresiones EL.
 - ▷ *TemplateResolver (SpringResourceTemplateResolver)*
Permite acceder físicamente a las plantillas, indicando ruta y sufijo

¿QUÉ HACE SPRING BOOT POR NOSOTROS PARA CONFIGURAR THYMELEAF?

```
@Configuration
public class ThymeleafConfiguration {

    @Bean
    public SpringResourceTemplateResolver templateResolver(){
        SpringResourceTemplateResolver templateResolver = new SpringResourceTemplateResolver();
        templateResolver.setApplicationContext(this.applicationContext);
        templateResolver.setPrefix("/WEB-INF/templates/");
        templateResolver.setSuffix(".html");
        templateResolver.setTemplateMode(TemplateMode.HTML);
        templateResolver.setCacheable(true);
        return templateResolver;
    }

    @Bean
    public SpringTemplateEngine templateEngine(){
        SpringTemplateEngine templateEngine = new SpringTemplateEngine();
        templateEngine.setTemplateResolver(templateResolver());
        templateEngine.setEnableSpringELCompiler(true);
        return templateEngine;
    }

    @Bean
    public ThymeleafViewResolver viewResolver(){
        ThymeleafViewResolver viewResolver = new ThymeleafViewResolver();
        viewResolver.setTemplateEngine(templateEngine());
        viewResolver.setOrder(1);
        viewResolver.setViewNames(new String[] {".html", ".xhtml"});
        return viewResolver;
    }
}
```

CÓMO MODIFICAR LA CONFIGURACIÓN POR DEFECTO DE THYMELEAF

- ▶ *application.properties*
 - ▷ spring.thymeleaf.cache
 - ▷ spring.thymeleaf.check-template
 - ▷ spring.thymeleaf.check-template-location
 - ▷ spring.thymeleaf.enabled
 - ▷ spring.thymeleaf.encoding
 - ▷ spring.thymeleaf.mode
 - ▷ spring.thymeleaf.prefix
 - ▷ spring.thymeleaf.suffix

PROFUNDIZAR EN THYMELEAF

Accede a nuestro [Curso de Introducción a Thymeleaf](#) y profundiza en esta tecnología para sacar más partido a este curso.

Curso de Introducción a Thymeleaf

Aprende a usar Thymeleaf, el mejor motor de plantillas para utilizar junto a Spring y empieza a disfrutar de las ventajas del natural templating con este curso.

★★★★★ 4.4 (9 valoraciones) 4 horas y 8 minutos

CONOCIMIENTOS Y HABILIDADES QUE ADQUIERES REALIZANDO ESTE CURSO

- Aprenderás a crear plantillas web.
- Conocerás cómo configurar aplicaciones web con Spring Boot, Spring Data JPA y Thymeleaf.
- Serás capaz de manejar formularios.
- Aprenderás a manejar listas y bucles para representar datos.
- Serás capaz de cambiar los estilos CSS de forma condicional.
- Aplicarás técnicas de validación y manejo de mensajes de error.

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

CAMBIO DE PROPIEDADES DE THYMELEAF

- ▶ Prueba a *jugar* con el fichero *application.properties*, cambiando el valor de alguna de las propiedades de configuración.
- ▶ Por ejemplo, *spring.thymeleaf.cache=false* te permitirá realizar cambios en la plantilla y poder visualizarlos sin tener que *relanzar* el proyecto.

CAMBIO DE PROPIEDADES DE THYMELEAF

- ▶ Si la extensión `.html` no te gusta, puedes probar a cambiar el sufijo a `.htm`

RECOGER PARÁMETROS DE UNA PETICIÓN HTTP



PARÁMETROS EN UNA PETICIÓN

Es una forma de pasar información en la petición.

Dos tipos

- ▶ En el *path* de la URL
- ▶ En la parte *query* de la URL



@REQUESTPARAM

- ▶ Una URL tiene dos partes
 - ▷ Path: `http://www.openwebinars.net/`
 - ▷ Query: `?categoria=backend`
- ▶ La parte **Query** viene delimitada por **?**
- ▶ Pares `name=value`
- ▶ Si hay más de uno, `n1=v1&n2=v2&n3=v3`

@REQUESTPARAM

- Con @RequestParam注入amos el valor en una variable

www.dominio.com/?name=Luismi

```
@GetMapping("/")
public String welcome(@RequestParam("name") String name, Model model) {
    model.addAttribute("nombre", name);
    return "index";
}
```

- El tipo de dato suele ser *String*, pero puede tomar otros valores (*int, long, Date, ...*)

@REQUESTPARAM

- ▶ Por defecto, *required=true*
- ▶ Si realizamos la petición sin parámetro, obtenemos error 400 (Bad Request).
- ▶ Alternativas
 - ▷ *required=false + defaultValue*
 - ▷ Uso de *Optional*.

@PATHVARIABLE

- ▶ También podemos extraer partes del Path.
- ▶ Podemos declarar variables entre llaves
{var}
/pedidos/{clientId}
- ▶ Pueden estar en medio de una ruta
/pedido/{pedidold}/articulos

@PATHVARIABLE

- ▶ Con @PathVariable injectamos el valor en una variable.

```
@GetMapping("/saludo/{name}")
public String saludo(@PathVariable String name, Model model) {
    model.addAttribute("nombre", name);
    return "index";
}
```

- ▶ Si el nombre de la variable es diferente del parámetro, lo injectamos con @PathVariable("name")

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

PASO DE MÁS DE UN PARÁMETRO CON @REQUESTPARAM

- ▶ Prueba a modificar el controlador welcome para que reciba dos parámetros en la query

`/?firstName=Pepe&lastName=Perez`

- ▶ Captura los dos parámetros y muéstralos a través de la plantilla.

`${'Hola' + firstName + ' ' + lastName}`

PASO DE MÁS DE UN PARÁMETRO CON @PATHVARIABLE

- ▶ Prueba a modificar el controlador saludo para que reciba dos parámetros en el path
/saludo/{firstName}/{lastName}
- ▶ Captura los dos parámetros y muéstralos a través de la plantilla.
 \${'Hola' + firstName + ' ' + lastName}

CONTENIDO ESTÁTICO



ENTENDEMOS POR CONTENIDO ESTÁTICO...

- ▶ Ficheros.html
- ▶ Imágenes
- ▶ Css
- ▶ Javascript
- ▶ Fichero binarios
- ▶ ...

¿DÓNDE ALMACENAMOS EL CONTENIDO ESTÁTICO?

- ▶ Carpeta */src/main/resources/static*



¿CÓMO ACCEDO AL CONTENIDO ESTÁTICO?

- ▶ Spring Boot viene al rescate
 - ▶ Sirve el contenido en
 - ▷ */static*
 - ▷ */public*
 - ▷ */resources*
 - ▷ */META-INF/resources*
- en la raíz de nuestro contexto.

CAMBIOS EN LA CONFIGURACIÓN

- ▶ Podemos cambiar el mapeo a la ruta raíz por otra que nos interese más
`spring.mvc.static-path-pattern=/resources/**`
- ▶ También se puede cambiar la localización de los recursos a otra carpeta
`spring.resources.static-locations`

WEBJARS: RECURSOS ESTÁTICOS COMO ARCHIVOS .JAR

RECURSOS ESTÁTICOS UTILIZADOS CON FRECUENCIA

- ▶ jQuery
- ▶ Bootstrap
- ▶ moment.js
- ▶ ...



¿PODEMOS AÑADIR CON MAVEN ESTOS RECURSOS?

- ▶ La respuesta es sí: webjars
- ▶ <https://www.webjars.org/>
- ▶ Ventajas
 - ▷ No dependemos de la disponibilidad de un CDN.
 - ▷ Velocidad de carga acorde a nuestra aplicación web.

¿CÓMO CONFIGURARLO?

- ▶ Spring Boot viene de nuevo al rescate
- ▶ Todo contenido que esté en una ruta /webjars/** es servido desde el fichero .jar correspondiente como contenido estático de forma automática.

DEPENDENCIAS MÁS USADAS

```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>bootstrap</artifactId>
  <version>3.3.7-1</version>
</dependency>
```

Este es el más mágico de los 3,
pues nos va a permitir disfrutar
de versión agnóstica

```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>jquery</artifactId>
  <version>3.3.1-1</version>
</dependency>
```

```
<dependency>
  <groupId>org.webjars</groupId>
  <artifactId>webjars-locator</artifactId>
  <version>0.34</version>
</dependency>
```

CAMBIOS EN LAS RUTAS DE NUESTRA PLANTILLA

```
<link href="css/bootstrap.min.css" rel="stylesheet">
<link href="/webjars/bootstrap/css/bootstrap.min.css" rel="stylesheet">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js">
</script>
<script src="/webjars/jquery/jquery.min.js"></script>

<script src="js/bootstrap.min.js"></script>
<script src="/webjars/bootstrap/js/bootstrap.min.js"></script>
```

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

SITIO WEB ESTÁTICO

- ▶ Si tienes conocimientos de HTML5, CSS3 y Javascript, puedes probar a crear un sitio web estático dentro de la carpeta `/src/main/resources/static` y comprobar que puedes invocarlo desde el navegador.

SITIO WEB ESTÁTICO

- ▶ También puedes probar a añadir elementos a las plantillas de ejemplos anteriores, como encabezados, menús,... utilizando estilos CSS propios. Puedes almacenar estos estilos en `/src/main/resources/static/css/styles.css`

INTRODUCCIÓN A LA CREACIÓN DE FORMULARIOS



FORMULARIOS CON SPRING MVC

- ▶ Spring ofrece funcionalidades para el manejo de formularios.
- ▶ Se realiza a través de un objeto, llamado *Command Object*, que es el *bean* que servirá para almacenar la información recogida en el formulario.
- ▶ Este objeto debe tener tantos atributos (con getters y setters) como campos tenga el formulario.

FORMULARIOS CON SPRING MVC

- ▶ En algunas ocasiones, podremos usar como *command object* objetos (entidades) de nuestro modelo.

FLUJO DE UN FORMULARIO CON SPRING MVC

(1) Enviar objeto al formulario

- @GetMapping
- Añadimos al modelo un *command object* ¿vacío?
- Nos dirigimos a la plantilla del formulario.

(2) Formulario

- Tomamos del contexto el *command object*.
- Asociamos cada uno de sus atributos a los campos correspondientes del formulario.
- Dirigimos la acción del formulario hacia (3)

(3) Procesar los datos

- @PostMapping
- Recogemos el objeto enviado desde el formulario, ya lleno de datos.
- Aplicamos la lógica de negocio correspondiente.
- Redirigimos hacia otro controlador (¿de listado?)

FLUJO DE UN FORMULARIO CON SPRING MVC

1

/empleado/new

model.addAttribute(...)

EmpleadoController

E:Empleado

3

/empleado/new/submit

...submit(@ModelAttribute("empleadoForm")
Empleado empleado)

EmpleadoController

2

Nuevo empleado

ID
0

Nombre
Nombre

Email
correo@correo.com

Teléfono
954000000

E:Empleado

action=/empleado/new/submit

FORMULARIOS CON SPRING MVC + THYMELEAF

```
<form action="#"
```

Controlador al que dirigir el envío del formulario

```
    th:action="@{/empleado/new/submit}"
```

```
    th:object="${empleadoForm}"
```

Command object que se corresponderá con los campos del formulario.

```
    method="post">
```

```
...
```

```
</form>
```

```
<input type="text"
```

Nombre de la propiedad de la clase Empleado a la que asociamos el campo del formulario

```
    id="elID" th:field="*{campo}">
```

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

OTROS CONTROLES DE FORMULARIO

- ▶ Visita nuestro [Curso de introducción a Thymeleaf](#), para aprender a añadir diferentes tipos de campos de formulario, como los *check* y los *radio*.

OTROS CONTROLES DE FORMULARIO

- ▶ Añade
 - ▶ Una nueva propiedad a empleado, que sea booleana: *directivo*. Modifica el código necesario de getters, setters,
....
 - ▶ Añade un campo check al formulario que permita indicar si un empleado es un directivo o no.

OTROS CONTROLES DE FORMULARIO

- ▶ Modifica
 - ▶ Los datos de ejemplo, indicando que alguno de los empleados sí sea directivo.
 - ▶ El listado de empleados, para que aparezca una nueva columna, indicando si un empleado es o no directivo. ¿Eres capaz de hacerlo con un icono?

FORMULARIOS: EDICIÓN Y VALIDACIÓN



1.

FORMULARIOS PARA EDITAR REGISTROS

FORMULARIOS DE EDICIÓN

- ▶ Idénticos a los de edición
- ▶ El *command object* debe llevar datos, en lugar de estar vacío.
- ▶ El servicio debe permitirnos rescatar los datos para editar.
- ▶ El url incluirá el id del dato a editar.

FLUJO DE UN FORMULARIO CON SPRING MVC

(1) Enviar objeto al formulario

- @GetMapping
- Añadimos al modelo un **command object con los datos a editar**. Lo rescatamos desde el servicio usando el id.
- Nos dirigimos a la plantilla del formulario.

(2) Formulario

- Tomamos del contexto el **command object**.
- Asociamos cada uno de sus atributos a los campos correspondientes del formulario. Así, el **navegador los visualiza**.
- Dirigimos la acción del formulario hacia (3)

(3) Procesar los datos

- @PostMapping
- Recogemos el objeto enviado desde el formulario, ya lleno de datos.
- Aplicamos la lógica de negocio correspondiente.
- Redirigimos hacia otro controlador (¿de listado?)

FLUJO DE UN FORMULARIO CON SPRING MVC

1

/empleado/edit/{id}

model.addAttribute(...)

EmpleadoController

E:Empleado

3

/empleado/edit/submit

...submit(@ModelAttribute("empleadoForm")
Empleado empleado)

EmpleadoController

2

Nuevo empleado

ID
0

Nombre
Nombre

Email
correo@correo.com

Teléfono
954000000

E:Empleado

action=/empleado/edit/submit

2.

VALIDACIÓN DE DATOS CON SPRING

VALIDACIÓN CON SPRING

- ▶ Spring permite usar el estándar *JSR-303/JSR-380 Bean Validation API*.
- ▶ Spring Boot configura por defecto la implementación de este estándar realizada por *hibernate*.
- ▶ Permite realizar la validación añadiendo anotaciones en nuestras clases modelo.

ANOTACIONES DE VALIDACIÓN

- ▶ @NotNull: el atributo no puede ser nulo
- ▶ @Min, @Max: mayor o igual (o menor o igual) que un valor determinado.
- ▶ @NotEmpty: el atributo no puede estar vacío (Strings, colecciones, arrays,...)
- ▶ @Email: el atributo debe ser un email válido.
- ▶ @Size: el atributo tiene que tener un tamaño según el indicado.

<https://beanvalidation.org/2.0/spec/#builtinconstraints>

CUSTOMIZACIÓN DE MENSAJES DE VALIDACIÓN

- ▶ Podemos modificar el mensaje por defecto.
- ▶ Varias estrategias
 - ▷ Cadena *hardcodeada* en la anotación.
 - ▷ **Ficheros de properties.**
 - ▷ Necesitamos configuración algunos beans
 - ▷ Nos permite internacionalización de mensajes de error.

CUSTOMIZACIÓN DE MENSAJES DE VALIDACIÓN

► Configuración

```
@Configuration  
public class MyConfig {  
  
    @Bean  
    public MessageSource messageSource() {  
        ReloadableResourceBundleMessageSource messageSource  
            = new ReloadableResourceBundleMessageSource();  
  
        messageSource.setBasename("classpath:errors");  
        messageSource.setDefaultEncoding("UTF-8");  
        return messageSource;  
    }  
  
    @Bean  
    public LocalValidatorFactoryBean getValidator() {  
        LocalValidatorFactoryBean bean = new LocalValidatorFactoryBean();  
        bean.setValidationMessageSource(messageSource());  
        return bean;  
    }  
}
```

Indicamos dónde debe buscar el fichero de properties.

Indicamos que use ese fichero para la validación.

CUSTOMIZACIÓN DE MENSAJES DE VALIDACIÓN

- ▶ Cambio de valor

```
public class Empleado {  
  
    @Min(value=1, message="{empleado.id.mayorquecero}")  
    private long id;
```

- ▶ Nuevo mensaje de error

empleado.id.mayorquecero=El ID del empleado debe ser un numero entero positivo

FORMULARIOS CON THYMELEAF

Accede a nuestro [Curso de Introducción a Thymeleaf](#) y aprende cómo utilizar otros campos de formulario más complejos.

Curso de Introducción a Thymeleaf

Aprende a usar Thymeleaf, el mejor motor de plantillas para utilizar junto a Spring y empieza a disfrutar de las ventajas del natural templating con este curso.

★★★★★ 4.4 (9 valoraciones) 4 horas y 8 minutos

CONOCIMIENTOS Y HABILIDADES QUE ADQUIERES REALIZANDO ESTE CURSO

- Aprenderás a crear plantillas web.
- Conocerás cómo configurar aplicaciones web con Spring Boot, Spring Data JPA y Thymeleaf.
- Serás capaz de manejar formularios.
- Aprenderás a manejar listas y bucles para representar datos.
- Serás capaz de cambiar los estilos CSS de forma condicional.
- Aplicarás técnicas de validación y manejo de mensajes de error.

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

VALIDACIÓN DE OTROS CAMPOS DEL FORMULARIO

- ▶ Siguiendo el esquema de esta lección, añade el código necesario para visualizar los errores de validación de los demás campos, añadiendo un mensaje personalizado de error.

SUBIDA DE FICHEROS



1.

MULTIPART CON SPRING

MULTIPART

- ▶ Tipo de mensaje que permite que una petición tenga varias partes delimitadas, con su correspondiente *Content-Type*.

```
MIME-version: 1.0
Content-type: multipart/mixed; boundary="frontera"

This is a multi-part message in MIME format.
--frontera
Content-type: text/plain

Este es el cuerpo del mensaje
--frontera
Content-type: application/octet-stream
Content-transfer-encoding: base64

PGh0bWw+CiAgPGh1YWQ+CiAgPC9oZWFlPgogIDxib2R5PgogICAgPHA+RXN0ZSB1cyB1bCBjdWVycG8gZGVsIG1lbNhamU8L3A+CiAgPC9ib2R5Pgo8L2h0bWw+Cic=\r--frontera--
```

MULTIPART

- ▶ De esta forma, podemos enviar los datos del formulario y datos binarios, a la vez.
<form enctype="multipart/form-data">
- ▶ El formulario debe incluir un campo de subida de ficheros:
<input type="file" .../>
- ▶ Spring soporta perfectamente la gestión de peticiones multiparte.

MULTIPART CON SPRING

- ▶ Cuando Spring procesa una petición multiparte, nos deja acceder a ella (o ellas) a través de `@RequestParam`

```
@PostMapping("/form")
public String handleFormUpload(@RequestParam("name") String name,
    @RequestParam("file") MultipartFile file) {
```

- ▶ La clase `MultipartFile` tiene métodos convenientes para permitirnos procesar el fichero.

¿DÓNDE LO ALMACENAMOS?

- ▶ Propio proyecto
 - ▷ Fácil para aprender
 - ▷ No es buena práctica en producción
- ▶ Servicio de almacenamiento externo
 - ▷ GridFS
 - ▷ Nube (*Amazon, Azure, Drive...*)
 - ▷ Si son imágenes, servicios específicos, como *imgur*.

2.

SERVICIO DE ALMACENAMIENTO

NUESTRO EJEMPLO

- ▶ Tomamos un ejemplo de Spring
<https://spring.io/guides/gs/uploading-files/>
- ▶ La idea es crear un servicio de almacenamiento estándar (interface) y una implementación en local.

INTERFACE StorageService

```
public interface StorageService {  
    void init();  
    String store(MultipartFile file, long id);  
    Stream<Path> loadAll();  
    Path load(String filename);  
    Resource loadAsResource(String filename);  
    void deleteAll();  
}
```

Métodos que debería proporcionarnos un servicio de almacenamiento de ficheros.

Algunos están modificados sobre el ejemplo.

CLASE **FileSystemStorageService**

- ▶ Almacenamiento en nuestro sistema de ficheros.
- ▶ Método *store*
 - ▶ Recibe el multiparte y el ID del nuevo empleado.
 - ▶ Suponemos que estamos subiendo el avatar.
 - ▶ Lo almacenamos como ID.ext (por ejemplo, 5.png, o 4.jpg).

CLASE **FileSystemStorageService**

- ▶ Método *load*
 - ▷ Devuelve la ruta de un fichero desde su nombre.
- ▶ Método *loadAsResource*
 - ▷ Recibe el nombre de un fichero.
 - ▷ Busca el fichero, y lo devuelve como una instancia de *Resource* (envoltorio conveniente para un fichero)

CLASE `FileSystemStorageService`

- ▶ Método `init`
 - ▷ Inicializa el sistema de ficheros
- ▶ Método `deleteAll`
 - ▷ Elimina todos los ficheros

La clase `FileSystemStorageService` utiliza algunas clases de **error** y de **configuración**, y necesita ser **invocado al lanzar el proyecto**.

CLASES DE ERROR

- ▶ *StorageException*
Error general de almacenamiento.
- ▶ *StorageFileNotFoundException*
Fichero no encontrado.

CLASES DE CONFIGURACIÓN

- ▶ *StorageProperties*
 - ▶ Anotada con `@ConfigurationProperties`
 - ▶ Envoltorio conveniente para uso de properties.
 - ▶ Inyectable mediante `@Autowired`
 - ▶ Tenemos que añadir `@EnableConfigurationProperties` (`StorageProperties.class`) en una clase con `@Configuration`

INVOCAR AL LANZAR EL PROYECTO

- ▶ Creación de un *CommandLineRunner* en la clase *Application*.
- ▶ Permite encapsular un código que se ejecutará en el lanzamiento de la aplicación.
- ▶ Usamos una expresión lambda (muy habitual en este contexto).

3.

**CÓMO INTEGRARLO
CON NUESTRA
APLICACIÓN**

CÓMO INTEGRARLO EN NUESTRO PROYECTO

- ▶ Necesitamos que, cuando creamos un nuevo empleado, el fichero se almacene.
- ▶ Antes de terminar la petición, necesitamos la ruta del fichero, para guardarla junto al empleado
- ▶ En la vista, añadiremos lo necesario para que visualizar el avatar del usuario.
- ▶ El fichero no será obligatorio, así que mostraremos un avatar aleatorio para los que no tengan ninguno.

MÉTODO `serveFile`

- ▶ Método especial que será capaz de devolvernos el fichero como respuesta a una petición.
- ▶ Nos aísla tener que configurar el almacenamiento estático para obtener los ficheros.

MÉTODO `serveFile`

`@ResponseBody` indica que el método no nos llevará a una vista, sino que devolverá un recurso.

```
@GetMapping("/files/{filename:.+}")
@ResponseBody
public ResponseEntity<Resource> serveFile(@PathVariable String filename) {
    Resource file = storageService.loadAsResource(filename);
    return ResponseEntity.ok().body(file);
}
```

Curva que apunta a la linea de código `return ResponseEntity.ok().body(file);`.
Explicación: `ResponseEntity` nos permite montar la respuesta HTTP, definiendo un código 200 OK, y la imagen como cuerpo de la misma.

La interfaz `Resource` es un envoltorio de un recurso (fichero binario, fichero en el classpath, ...)

Usamos una expresión *glob* que, además, asigna el valor a la variable `filename`

Cargamos nuestra imagen como un `Resource`

CAMBIOS EN EL CONTROLADOR

- ▶ Al crear un nuevo empleado queremos
 - ▶ Almacenar el fichero.
 - ▶ Almacenar la ruta del fichero con la información del nuevo empleado.

CAMBIOS EN EL CONTROLADOR

- ▶ [MvcUriComponentsBuilder](#)
 - ▷ Nos permite armar una URI/URL
- ▶ *fromMethodName* lo hace usando un método, en nuestro caso anotado con la ruta /file/filename.
- ▶ Sustituye la variable por el nombre concreto del fichero.
- ▶ Obtenemos la URI completa en una cadena de caracteres.

CAMBIOS EN LA PLANTILLA

- ▶ Visualización del avatar
 - ▷ Una nueva columna en la tabla
- ▶ Avatar por defecto
 - ▷ Si la imagen es nula
 - ▷ Utilizamos el servicio
<http://avatars.adorable.io/> que nos permite obtener un avatar a partir de un email.

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

IMPLEMENTAR LA OPCIÓN DE EDITAR

- ▶ En este proyecto hemos implementado la subida de un avatar para un nuevo empleado.
- ▶ Sin embargo, no hemos añadido esta funcionalidad al editar un registro ya existente.
- ▶ Te animamos a que lo intentes por tu cuenta.

CAMBIAR EL TIPO DE AVATAR

- ▶ Puedes jugar en la visualización del avatar por defecto, utilizando otros sistemas, como por ejemplo
<https://es.gravatar.com/>

CAMBIAR EL ALMACENAMIENTO (PRO)

- ▶ Puedes probar a utilizar otro sistema de almacenamiento, por ejemplo **imgur**.
- ▶ Es gratuito, y nos permite subir imágenes a través de su API.
- ▶ Las subidas públicas no requieren de autenticación, tan solo que nuestra aplicación esté registrada.
- ▶ Más info en <https://apidocs.imgur.com/>

HACIENDO UNA APLICACIÓN WEB SEGURA



SEGURIDAD CON SPRING = SPRING SECURITY

- ▶ Ofrece servicios de seguridad a aplicaciones Java EE.
- ▶ Integración muy sencilla con proyectos Spring MVC a través de Spring Boot.
- ▶ Dos procesos:
 - ▷ **Autenticación:** ¿quién eres?
 - ▷ **Autorización:** ¿para qué tienes permiso?

SPRING SECURITY EN NUESTRO POM.XML

```
<dependency>
<groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-security</artifactId>
</dependency>
```

“

Learning by doing: vamos a aprender a utilizar Spring Security utilizándolo en nuestro proyecto.

PASO 1: CONFIGURACIÓN DE LA SEGURIDAD

- ▶ Paquete *seguridad*
- ▶ Extiende a *WebSecurityConfigurerAdapter*
- ▶ Anotada con *@EnableWebSecurity*

```
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
}
```

PASO 2: CONFIGURAR EL MÉTODO DE AUTENTICACIÓN

- ▶ Autenticación: ¿tú quién eres?
- ▶ En este primer ejemplo lo hacemos en memoria.
- ▶ Tampoco codificamos las contraseñas.
- ▶ Un solo usuario *admin/admin/admin*.

```
@Override  
protected void configure(AuthenticationManagerBuilder auth) throws Exception {  
    auth  
        .inMemoryAuthentication()  
        .passwordEncoder(NoOpPasswordEncoder.getInstance())  
        .withUser("admin")  
        .password("admin")  
        .roles("ADMIN");  
}
```

HASTA AHORA...

Solo con este código ya tenemos

- ▶ Requisito de autenticación en todas las URLs
- ▶ Generación de un formulario de *login*
- ▶ Permitir el acceso a un usuario con Username *admin* y Password *admin*
- ▶ Permitir hacer logout
- ▶ Prevención de ataques *CSRF*, entre otras (*Session Fixation, X-XSS-Protection, Clickjacking,...*)

HASTA AHORA...

- ▶ Integración con métodos del api Servlet
(HttpServletRequest)
 - ▷ *getRemoteUser()*
 - ▷ *getUserPrincipal()*
 - ▷ *isUserInRole(...)*
 - ▷ *login(...)*
 - ▷ *logout()*

ADEMÁS...

- ▶ La clase de configuración ha registrado un filtro de Servlet especial, llamado *springSecurityFilterChain*.
- ▶ Es una cadena de filtros responsable de toda la seguridad (proteger las URLs, validar usuario y contraseña, redirigir al formulario de login,...)

PASO 3: AUTORIZACIÓN

- ▶ Autorización: *¿para qué tienes permiso?*
- ▶ Lo configuramos mediante un objeto de tipo *HttpSecurity*.
- ▶ Uso de method chain.

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
}
```

PASO 4: LOGIN

- ▶ Modificamos lo necesario para
 - ▷ Cambiar el formulario de login y hacer uno propio
 - ▷ Cambiar la ruta a `/login`
 - ▷ Permitir que cualquiera acceda al formulario

PASO 4: LOGIN

```
@Override  
protected void configure(HttpSecurity http) throws Exception {  
    http  
        .authorizeRequests()  
            .anyRequest().authenticated()  
            .and()  
        .formLogin()  
            .loginPage("/login")  
            .permitAll();  
}
```

PASO 4: LOGIN

- ▶ Para customizar el formulario, necesitaríamos un controlador que nos lleve a la plantilla de *login*.
- ▶ Si un controlador tan solo sirve para llevarnos a la plantilla, lo podemos simplificar.

PASO 4: LOGIN

- ▶ Creamos una clase de configuración, que implementa *WebMvcConfigurer*.
- ▶ Podemos mapear una ruta a una plantilla.

```
@Override  
public void addViewControllers(ViewControllerRegistry registry) {  
    registry.addViewController("/login");  
}
```

PASO 4: LOGIN (PLANTILLA)

- ▶ Creamos una nueva plantilla (*login.html*)
- ▶ Nos podemos basar en otro ejemplo de bootstrap.

The image shows a clean, modern login form. At the top, the text "Please sign in" is centered. Below it are two stacked input fields: the top one is labeled "Email address" and the bottom one is labeled "Password". Underneath these fields is a small checkbox labeled "Remember me". At the bottom of the form is a large, prominent blue button with the white text "Sign in".

PASO 4: LOGIN (PLANTILLA)

Para que funcione

- ▶ Apuntamos a las webjars
- ▶ Modificamos el formulario de login
 - ▶ `th:action="@{/login}"`
 - ▶ Los campos deben tener
`name="username"` y
`name="password".`

PASO 5: AUTORIZAR PETICIONES

- ▶ Nuestra seguridad es tan restrictiva que no permite servir ni los estilos css.
- ▶ Añadimos rutas para las que permitiremos el acceso siempre.
- ▶ Lo hacemos mediante el método `antMatchers(...)`, que acepta un `varargs` de `String` con rutas (podemos usar `glob`).

PASO 6: LOGOUT

- ▶ Permitimos que un usuario autenticado realice el logout para
 - ▷ Invalidar la sesión
 - ▷ Limpiar el contexto de seguridad (*SecurityContextHolder*)
 - ▷ Redirigir al login (`/login?logout`)
- ▶ Añadimos los elementos necesarios a la plantilla.

PASO 6: LOGOUT

¡ATENCIÓN! Desde las últimas versiones de Spring, la petición logout es POST (no GET).

- ▶ **Motivo:** uso de CSRF
- ▶ **Solución:**
 - ▶ Deshabilitar CSRF (no recomendado)
 - ▶ Realizar una petición POST (fácil)

PASO 6: LOGOUT

- ▶ De esta forma, no tenemos que deshabilitar CSRF, y aprovechamos los estilos de bootstrap sin modificaciones.

```
<ul class="nav navbar-nav navbar-right">
  <li class="dropdown">
    <a href="#" class="dropdown-toggle" data-toggle="dropdown" role="button" aria-haspopup="true" aria-expanded="false">
      <ul class="dropdown-menu">
        <li>
          <a href="javascript:document.getElementById('logoutForm').submit()">Salir</a>
        </li>
      </ul>
    </li>
  </ul>
</div><!-- .nav-collapse -->
</div>
</nav>
<form th:action="@{/logout}" method="POST" id="logoutForm">
</form>
```

PASO 7: MOSTRAR EL NOMBRE DEL USUARIO LOGUEADO

- ▶ Usual que podamos ver los datos del usuario logueado.
- ▶ Lo podemos sacar del objeto SecurityContextHolder y pasarlo a través del modelo.
- ▶ O podemos tomarlo directamente en la plantilla.

PASO 7: MOSTRAR EL NOMBRE DEL USUARIO LOGUEADO

- ▶ Thymeleaf ofrece una librería de extras con Spring Security (v. 3, 4 y 5).
- ▶ Añadimos un nuevo espacio de nombres:

```
<html  
    xmlns:th="http://www.thymeleaf.org"  
    xmlns:sec="http://www.thymeleaf.org/extras/spring-security">
```

- ▶ Podemos visualizar el nombre:
`<div sec:authentication="name"></div>`
`<div th:text="#{authentication.name}"></div>`

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

AÑADIR INFORMACIÓN DE ERROR

- ▶ Al intentar un login fallido, la url a la que nos remite es /login?error.
- ▶ Intenta añadir algún elemento de interfaz de usuario (qué tal un *alert* de bootstrap) en el que se muestre un mensaje de error.
- ▶ Para obtener un *request param* llamado error thymeleaf podemos usar la expresión \${param.error}

MODIFICAR LA AUTORIZACIÓN

- ▶ En el ejemplo, tanto el listado de empleados como el alta de uno nuevo está autenticado.
- ▶ ¿Serías capaz de que el listado de empleados fuera de acceso público, pero no el alta de un nuevo empleado o la edición de uno existente?



MANEJO DE SESIONES

1.

HTTPSESSION

HTTP ES UN PROTOCOLO...

- ▶ **SIN ESTADOS**
- ▶ Cuando realiza una petición, no puede recordar nada de lo anterior.
- ▶ Necesita un mecanismo auxiliar para que un dato sobreviva más allá de una petición.
- ▶ A dicho mecanismo se le conoce como **sesiones**.

HTTPSESSION

- ▶ Interfaz que ofrece Java EE
- ▶ Es un Map en el que se almacenan pares clave - objeto.
- ▶ Cuando creamos una sesión, se envía una cookie, que el navegador mantiene y reenvía en todas las peticiones siguientes. (*JSESSIONID*).
- ▶ La sesión es individual: cada usuario tiene la suya; los datos no se comparten entre sesiones.

PARA QUÉ SE USAN LAS SESIONES

- ▶ Spring Security la usa cuando nos logueamos.
 - ▷ Así nos permite navegar entre páginas que requieren autenticación sin volver a loguearnos.
- ▶ Lógica de nuestra app
 - ▷ Carrito de la compra
 - ▷ Asistentes (proceso a realizar en varios pasos).

USO DE **HTTPSESSION** CON SPRING

- ▶ Podemos `@Autowired` el objeto `HttpSession` en cualquier controlador.
- ▶ Podemos pasarlo como argumento de cualquier método en un controlador.

DESVENTAJAS DE HTTPSESSION

- ▶ Dependemos de la implementación del servidor (contenedor de servlets) sobre el que desplegamos nuestra app.
- ▶ Problemas de escalabilidad
 - ▷ En un grupo de servidores, podemos sincronizar sesiones entre ellos → mayor cargo.
 - ▷ Integración continua: un despliegue de una nueva *release* provoca pérdida de sesiones.

¡SPRING AL RESCATE!

SPRING SESSION

- ▶ Integración transparente con *HttpSession*.
- ▶ Independencia de la implementación del servidor de aplicaciones.
- ▶ Sesiones en cluster
- ▶ Múltiples sesiones en un solo navegador
- ▶ Uso en APIs RESTful
- ▶ ...

2. SPRING SESSION

SPRING SESSION 2.0

- ▶ API de Spring que permite el manejo de sesiones en cluster sin estar atado al contenedor de la aplicación.
- ▶ Módulos (entre otros)



Spring Session Core

Funcionalidades principales y comunes.

Spring Session Data Redis

Almacenamiento de sesiones en Redis.

Spring Session Data JDBC

Almacenamiento en una base de datos relacional.

¿QUÉ ES REDIS?

- ▶ Motor de base de datos NoSQL en memoria.
- ▶ Almacenamiento de hashes (*key/value*)
- ▶ Implementada en C. Redis Lab (BSD License)
- ▶ Almacenamiento de sesiones, caché,...
- ▶ Rendimiento *extremo* comparado con otros SGBD.
- ▶ Replicación *master/slave* en árbol.

3.

INTEGRACIÓN DE NUESTRO PROYECTO CON SPRING SESSION

MODIFICACIONES

- ▶ Instalar redis (vía docker)
- ▶ Actualizar pom.xml
- ▶ Configuración (*application.properties*)
- ▶ **Spring Boot se encarga de lo demás.**

INSTALAR REDIS (VÍA DOCKER)

- ▶ **Docker**: gestor de contenedores
- ▶ **Docker hub**: repositorio de imágenes prefabricadas.
- ▶ Hay una imagen con la última versión estable de *redis*.

<https://docs.docker.com/samples/library/redis>

L

```
$ docker run -d --name myredis -p 6379:6379 redis
```

ACTUALIZAR POM.XML

- ▶ Añadimos dos nuevas dependencias

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-redis</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.session</groupId>
    <artifactId>spring-session-data-redis</artifactId>
</dependency>
```

CONFIGURACIÓN

- ▶ Gracias a Spring Boot, la configuración se puede hacer vía *application.properties*
`spring.session.store-type=redis`
- ▶ Spring Boot se encarga de crear un filtro, *springSessionRepositoryFilter*, encargado de la gestión de los objetos *HttpSession*.

OTROS ASPECTOS DE CONFIGURACIÓN

- ▶ *server.servlet.session.timeout*
Duración del timeout de una sesión.
- ▶ *spring.session.redis.flush-mode=on-save*
Modo de volcado de datos
- ▶ *spring.session.redis.namespace=spring:session*
Espacio de nombre usado para las claves

OTROS ASPECTOS DE CONFIGURACIÓN

- ▶ *spring.redis.host=localhost*
URL/IP del servidor Redis
- ▶ *spring.redis.password*
Contraseña
- ▶ *spring.redis.port=6379*
Puerto del servidor Redis

¿CÓMO COMPROBAMOS QUE FUNCIONA?

- ▶ Ejecutamos la aplicación
- ▶ Nos logueamos
- ▶ Abrimos una consola y ejecutamos

```
$ docker exec -ti myredis bash  
root@.....# redis-cli keys '*'
```

```
1) "spring:session:expirations:1542834540000"  
2) "spring:session:index:org.springframework.session.FindByIndexNameSessionRepository.PR  
INCIPAL_NAME_INDEX_NAME:admin"  
3) "spring:session:expirations:1542834600000"  
4) "spring:session:sessions:02e57ce5-734d-4983-9a52-7c2efc9b116d"  
5) "spring:session:sessions:expires:02e57ce5-734d-4983-9a52-7c2efc9b116d"
```

¿CÓMO COMPROBAMOS QUE FUNCIONA?

- ▶ Probamos a borrar la sesión, y comprobamos que nos la aplicación nos vuelve a llevar al login.

```
root@.....# redis-cli del  
spring:session:sessions:02e57ce5-734d-....
```

```
1) "spring:session:expirations:1542834540000"  
2) "spring:session:index:org.springframework.session.FindByIndexNameSessionRepository.PR  
INCIPAL_NAME_INDEX_NAME:admin"  
3) "spring:session:expirations:1542834600000"  
4) "spring:session:sessions:02e57ce5-734d-4983-9a52-7c2efc9b116d"  
5) "spring:session:sessions:expires:02e57ce5-734d-4983-9a52-7c2efc9b116d"
```

INTRODUCCIÓN A SPRING DATA



1.

PERSISTENCIA DE DATOS

PERSISTENCIA DE LA INFORMACIÓN

- ▶ La mayoría de las aplicaciones necesitan persistir la información.
- ▶ Significa que esta sobreviva más allá de una sesión o un determinado tiempo.
- ▶ La solución nos la aportan las **bases de datos**.

BASES DE DATOS

- ▶ Dos grandes modelos: **relacional** y **NoSQL**

Relacional

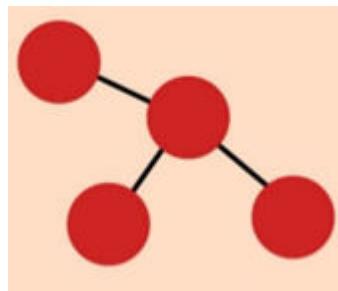
- Tablas, columnas, filas
- Garantía frente a duplicidad.
- Integridad referencial
- Difícilmente escalables

NoSQL

- Diferentes modelos de datos.
- Velocidad, flexibilidad.
- Escalabilidad
- Volumen de datos.

DESFASE OBJETO - RELACIONAL

- ▶ En Java usamos objetos (n-dimensiones)
- ▶ Las bases de datos, tablas (2-dimensiones)
- ▶ ¿Cómo salvar este desfase?



Objetos en memoria



Tablas en la base de datos relacional

DESFASE OBJETO - RELACIONAL

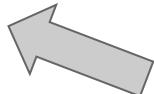
```
PreparedStatement ps = con
    .prepareStatement("SELECT * FROM empleados WHERE id = ?");
ps.setInt(1, id);
ResultSet rs = ps.executeQuery();
Empleado result = null;
if (rs.next()) {
    result = new Empleado(rs.getInt("id"),
        rs.getString("nombre"),
        rs.getString("apellidos")
        rs.getDate("fecha_nacimiento").toLocalDate(),
        rs.getFloat("sueldo"));
}
rs.close();
ps.close();
return result;
```

SOLUCIÓN AL DESFASE OBJETO - RELACIONAL: ORM

- ▶ *Object-Relational Mapping*
- ▶ Una pieza de software se encarga de traducir objetos en filas y viceversa.



HIBERNATE



MyBatis

eclipse)link

UNA PALABRA SOBRE JPA

- ▶ Java Persistence API
- ▶ Desde Java se propone un API estándar para tratar la persistencia de datos.
- ▶ Java no ofrece ninguna implementación concreta.
- ▶ La mayoría de ORMs comerciales, sí lo hacen.

Si JPA es el baile, Hibernate es el bailarín.

UNA PALABRA MÁS SOBRE JPA

- ▶ Una facilidad para especificar cómo nuestros objetos Java se relaciona con el esquema de una base de datos
- ▶ Una api sencillo para realizar las operaciones CRUD
- ▶ Un lenguaje y un API para realizar consultas sobre los datos (JPQL).
- ▶ Elementos de optimización (actualización de entidades, captura de asociaciones, caché,...)

2. **SPRING DATA**

Y a todo esto,
¿qué hay de
Spring?

¿QUÉ HACEMOS DESDE SPRING?

- ▶ Se puede usar directamente JDBC
- ▶ Se puede utilizar directamente Hibernate
- ▶ Podemos usar JPA sobre Hibernate
- ▶ También tenemos disponible **Spring Data**.



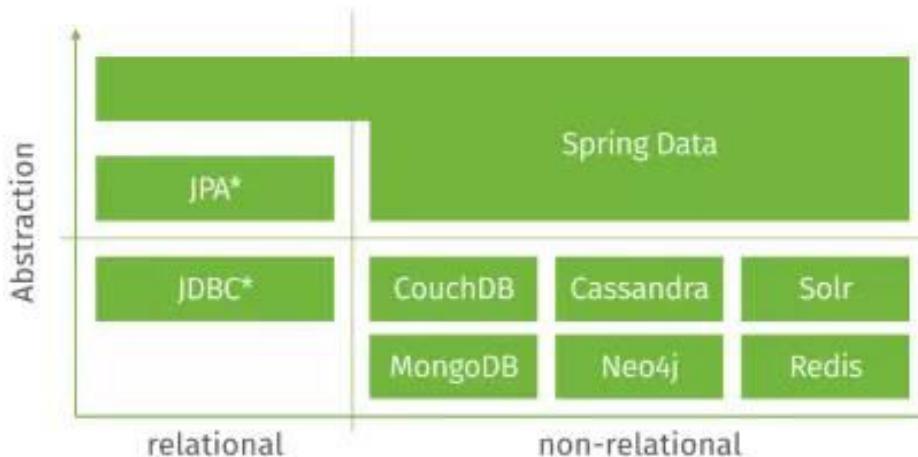
Spring Data

SPRING DATA

- ▶ Ofrece un modelo consistente de programación para acceder a datos.
- ▶ Facilita el uso de bases de datos relacionales y NoSQL
- ▶ Proyecto *paraguas* para muchos subproyectos.

SPRING DATA

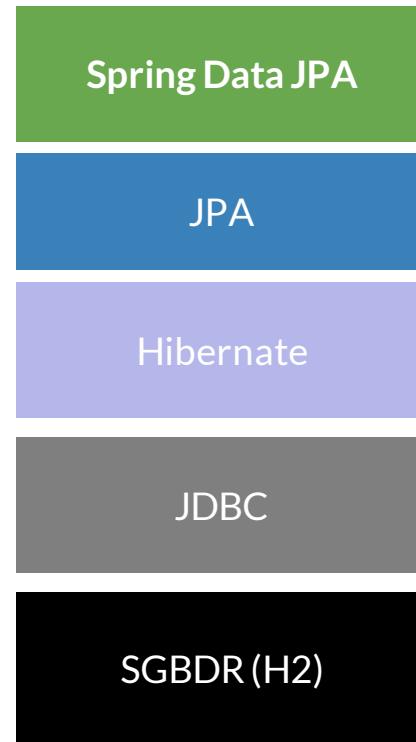
Spring Data



<https://spring.io/projects/spring-data>

SPRING DATA JPA

- ▶ Integra todas las funcionalidades de JPA con Spring Data
- ▶ Será con el subproyecto que trabajemos.
- ▶ Nuestra SGBD será H2.
 - ▷ Embebible (Maven)
 - ▷ ACID
 - ▷ Ligero



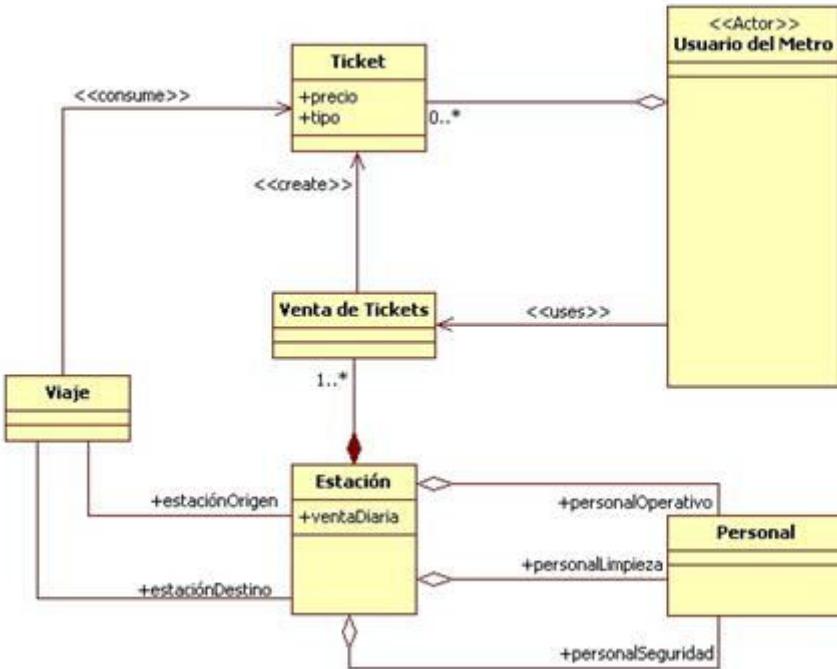
DEPENDENCIAS

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
</dependency>
```



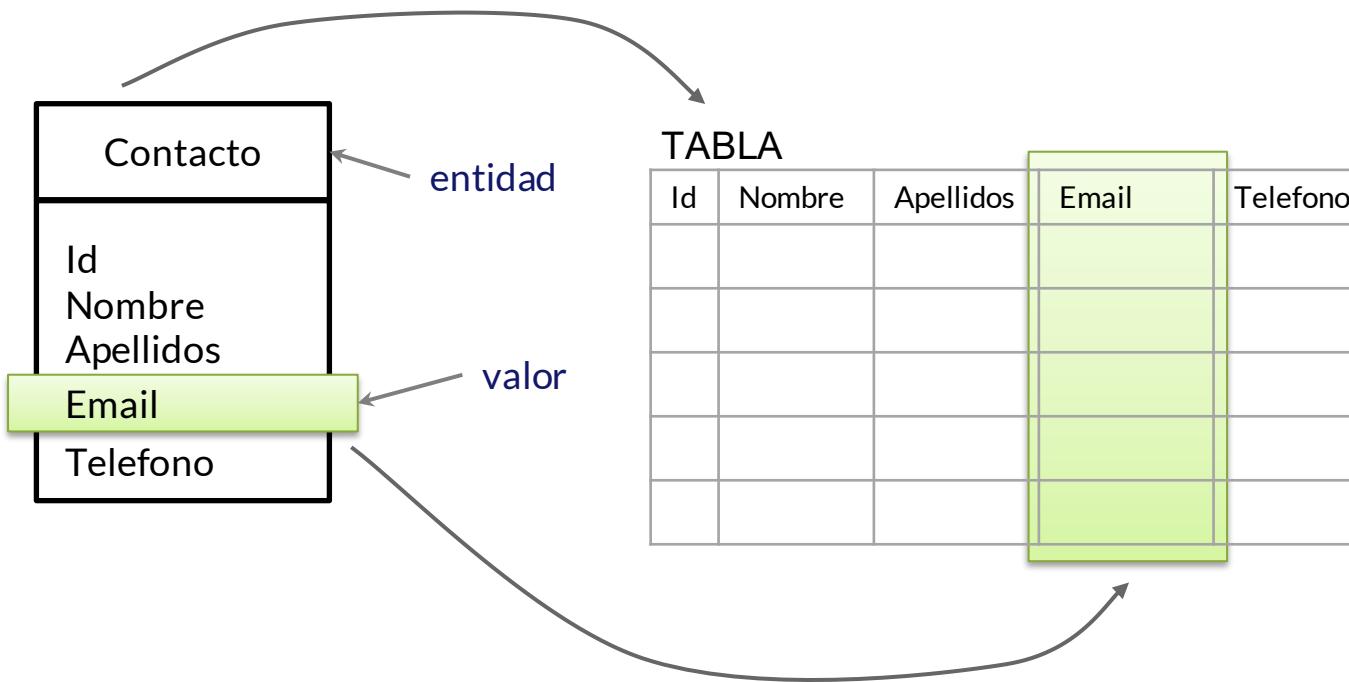
SPRING DATA JPA ENTIDADES

MODELO DE DOMINIO DE UN SISTEMA



- ▶ Representa los conceptos propios del problema a resolver.
- ▶ Vocabulario y conceptos clave.
- ▶ **Entidades y relaciones.**

MAPEO DE OBJETOS A TABLAS



ENTIDAD

- ▶ Clase (*pojo*) Java con `@Entity`
- ▶ Debe tener (al menos) un atributo que lo identifique (concepto de PK de bases de datos) con `@Id`. Anotación sobre propiedad o método *getter*.
- ▶ También se trasladarán a la base de datos como columnas el resto de propiedades que tenga la clase.

ENTIDAD

```
@Entity
public class Producto {

    @Id @GeneratedValue
    private Long Id;

    private String nombre;
    private String descripcion;
    private float pvp;

    //... resto de métodos y atributos
}
```

CONTROL DE NOMBRES

- ▶ *@Entity* se mapea con una tabla que se llame igual que la clase.
- ▶ Con un *@Table* adicional, podemos cambiar el nombre.

```
@Entity
@Table (name="PROD")
public class Producto {

    //... resto de métodos y atributos
}
```

CONTROL DE NOMBRES

- ▶ Los atributos también se mapean con su nombre
- ▶ Con un `@Column`, podemos cambiar el nombre.

```
@Entity
@Table(name="PROD")
public class Producto {

    @Column(name="prod_nombre")
    private String nombre;
    //... resto de métodos y atributos
}
```

ANOTACIÓN `@Column`

- ▶ Además `@Column` nos permite definir otras propiedades:
 - ▶ `Nullable`: si permite almacenar nulos
 - ▶ `Name`: nombre de la columna en la BD
 - ▶ `Insertable, updatable`: define si la entidad puede ser o no insertable o actualizable.
 - ▶ `Length`: tamaño que tendrá el campo en la BD.

ELECCIÓN DEL IDENTIFICADOR (CLAVE PRIMARIA)

- Es habitual utilizar campos *artificiales*, con el mismo nombre (*id*) y de tipo entero (*long*).
- JPA se puede encargar de generarlos (@*GeneratedValue*).
 - **AUTO:** Hibernate escoge la mejor estrategia para el SGBD elegido.
 - **SEQUENCE:** se utiliza una secuencia
 - **IDENTITY:** se utiliza un campo autonumérico
 - **TABLE:** se utiliza una tabla extra especial.

MAPEO DE VALORES.

TIPOS DE DATOS

- ▶ Tipos básicos
- ▶ Envoltorios de tipos básicos (*Long, Double, ...*)
- ▶ *String, BigInteger, BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, byte[], Byte[], char[], o Character[]*
- ▶ *java.io.Serializable* (representación en bytes)*
- ▶ *@Embeddable*

El resto de tipos de datos generarán por defecto un error.

MAPEO DE VALORES. ASOCIACIONES

- ▶ JPA nos permite asociar dos entidades.
- ▶ Debemos conocer la multiplicidad de dicha asociación.
 - ▶ @ManyToOne: muchos a uno
 - ▶ @OneToMany: uno a muchos
 - ▶ @ManyToMany: muchos a muchos
 - ▶ @OneToOne: uno a uno.

MAPEO DE VALORES. ASOCIACIONES

```
@Entity  
public class Producto {  
  
    @Id @GeneratedValue  
    private Long Id;  
  
    //... resto de atributos  
  
    @ManyToOne  
    private Categoria categoria;  
}
```

ENTIDADES JPA

Accede a nuestro
[Curso de
Hibernate y JPA](#)
para saber más
sobre entidades y
asociaciones



The screenshot shows a course page with the following details:

- Estás en:** Inicio / Cursos de Programación y Sistemas / Backend / Curso Online de Hibernate y JPA
- Título:** Curso Online de Hibernate y JPA
- Descripción:** Para realizar persistencia de datos sin usar SQL directamente podemos hacer uso de un ORM como Hibernate. En este curso conocerás la persistencia de objetos en Java con Hibernate y Java Persistence API de forma práctica.
- Valoración:** ★★★★ 4.1 (98 valoraciones)
- Duración:** 7 horas y 40 minutos

TEMARIO

Contenido	Duración
X INTRODUCCIÓN	45M
▶ Presentación del profesor y del curso	3M
▶ Introducción	19M

EN NUESTRO PROYECTO

- ▶ Vamos a transformar nuestra clase modelo en una entidad.

```
@Entity  
public class Empleado {  
  
    @Id @GeneratedValue  
    @Min(value=0, message="{empleado.id.mayorquecero}")  
    private long id;
```

- ▶ Escogemos siempre las anotaciones de *javax.persistence.**, y no las específicas de Hibernate.

EN NUESTRO PROYECTO

- ▶ Para ver resultados, necesitamos esperar a la siguiente lección.



SPRING DATA JPA: REPOSITORIOS

REPOSITORY

- ▶ Interfaz principal de Spring Data.
- ▶ Toma una entidad y su tipo de id como argumentos.
- ▶ *CrudRepository* es un buen punto de partida para empezar.
 - ▷ Operaciones CRUD
 - ▷ count()
 - ▷ existsById
 - ▷

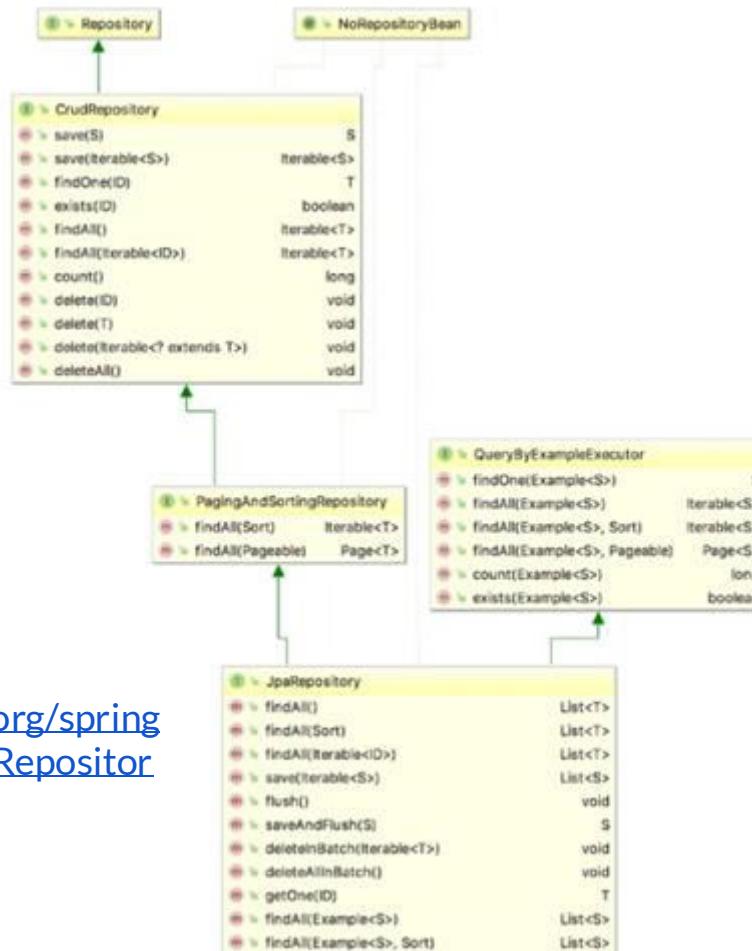
REPOSITORIOS

- ▶ Para usarlo, tan solo tenemos que extender el repositorio.

```
public interface EmpleadoRepository  
    extends CrudRepository<Empleado, Long> {  
  
}
```

- ▶ No tenemos que implementar ningún tipo de método para tener toda la funcionalidad de CrudRepository.

REPOSITORIOS



<https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/framework/data/repository/CrudRepository.html>

CRUDREPOSITORY

Métodos CRUD básicos

- ▶ *save(...)*: almacena una entidad
- ▶ *saveAll(...)*: almacena varias entidades
- ▶ *findOne(...)*: devuelve una entidad por su PK
- ▶ *findAll()*: devuelve todas las entidades en un iterable
- ▶ *count()*: número total de entidades
- ▶ *delete(...)*: elimina una entidad
- ▶ *exists(...)*: verifica si existe una entidad por su PK.

PAGINGANDSORTINGREPOSITORY

Añade al anterior:

- ▶ *findAll(Sort sort)*: devuelve todos según un orden
- ▶ *findAll(Pageable pageable)*: devuelve todos por páginas (usado en paginación de resultados).

JPAREPOSITORY

Añade al anterior:

- ▶ `findAll()`, `findAll(Sort sort)`: devuelve un `List<T>`
- ▶ `save(...)`: devuelve en un `List<T>`.
- ▶ `flush()`: descarga todas las tareas pendientes a la base de datos.
- ▶ `saveAndFlush(...)`: almacena la entidad y descarga los cambios a la base de datos.
- ▶ `deleteInBatch(...)`: elimina un `Iterable` de entidades.

INTEGRACIÓN CON NUESTRO PROYECTO

- ▶ Refactorizar *EmpleadoService* a *EmpleadoServiceMemory*
- ▶ Crear interfaz *EmpleadoService*
- ▶ Crear clase *EmpleadoServiceDB*
- ▶ Cargar datos de ejemplo en un *CommandLineRunner*
- ▶ Configurar algunas *properties*
- ▶ Actualizar la seguridad
- ▶ ¿Actualizar el formulario?

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

MODIFICAR EL FORMULARIO

- ▶ Nuestro formulario incluye un campo para insertar el ID de un nuevo empleado.
- ▶ También visualiza el ID de un empleado a modificar.
- ▶ Dicho ID no tiene necesidad de visualizarse.

MODIFICAR EL FORMULARIO

- ▶ Trata de modificar dicho campo por uno oculto, y comprueba que la inserción y edición sigue funcionando correctamente.

SPRING DATA JPA: CONSULTAS BÁSICAS



CONSULTAS

Spring Data JPA ofrece varios mecanismos

- ▶ Derivadas del nombre del método
- ▶ @Query
 - ▶ Consultas JPQL
 - ▶ Consultas nativas SQL
- ▶ Consultas con QueryDSL
- ▶ Consultas con Query By Example
- ▶ Consultas con Criteria API
- ▶ ...

1.

DERIVADAS DEL NOMBRE DEL MÉTODO

CONSULTAS DERIVADAS DEL NOMBRE DEL MÉTODO

- ▶ Definimos métodos en nuestro repositorio.
- ▶ Dichos métodos no necesitan implementación (se la da Spring Data).
- ▶ El nombre del método debe seguir una serie de reglas para que de él se pueda derivar la consulta.

ESTRUCTURA DE LA CONSULTA

La consulta debe comenzar por

- ▶ *find...By*
 - ▶ *read...By*
 - ▶ *query...By*
 - ▶ *count...By*
 - ▶ *get...By*
-
- ▶ *By* es el delimitador para definir los criterios.

TRADUCCIÓN DE LA CONSULTA

List<Person> findByLastname(String Lastname)

- ▶ Repositorio sobre *Person*.
- ▶ Incluye una propiedad llamada *Lastname*.
- ▶ Queremos aquellas instancias de *Person* cuyo *Lastname* coincide con el que pasamos como argumento.

`SELECT * FROM PERSON WHERE LASTNAME = '...';`

ELEMENTOS TÍPICOS DE CONSULTAS

- ▶ *Distinct*
- ▶ *And, or*
- ▶ *Between*
- ▶ *IsNull, IsNotNull, NotNull*
- ▶ *Like, NotLike*
- ▶ *In, Not in*
- ▶ *Containing*
- ▶ *OrderBy*
- ▶ ...

EJEMPLOS DE CONSULTAS

- ▶ `List<...> findByEmailAddressAndLastname(
 EmailAddress emailAddress, String Lastname)`
- ▶ `List<...> findByStartDateBetween(
 Date start, Date end)`
- ▶ `List<...> findByLastnamesIn(
 Collection<String>
 Lastnames)`

LIMITACIÓN DE LOS RESULTADOS DE UNA CONSULTA

- ▶ *First, Top*
Person findTopByOrderByAgeDesc()
- ▶ Podemos usar Optional (Java 8) para aquellas que devuelven un solo resultado
Optional<Person>
findTopByOrderByAgeDesc()
- ▶ Se pueden acompañar de un número
List<Person>
findTop10ByOrderByAgeDesc()

CONVIRTIENDO RESULTADOS DE CONSULTAS EN UN STREAM

- ▶ Spring Data permite devolver los resultados como un Stream de Java 8, para poder procesarlos después.

```
Stream<User> readAllByFirstnameNotNull();
```

EN NUESTRO PROYECTO

Algunos cambios para trabajar

- ▶ Comentamos la validación del campo ID.
- ▶ Modificamos el formulario (¡ojo! Es una propuesta de ejercicio de la lección anterior. Si aún no lo has intentado, trata de conseguir modificar el código por tu cuenta)

EN NUESTRO PROYECTO

Implementemos un buscador

- ▶ Añadir datos de ejemplo aleatorios (al menos, 40 empleados).
- ▶ Añadir un formulario de búsqueda en el listado.
- ▶ Implementar una consulta.
- ▶ Modificar el controlador para mostrar los resultados.

2.

CONSULTAS CON @QUERY

JPQL

- ▶ *Java Persistence Query Language*
- ▶ Lenguaje de consulta de JPA
- ▶ Inspirado en SQL y HQL (*Hibernate Query Language*)
- ▶ Orientado a objetos
- ▶ Consultas SELECT, UPDATE, WHERE

@QUERY

- ▶ Nos permite lanzar una consulta JPQL a través de un método.

```
@Query("select p from Persona p where u.nombre = ?1")
List<Persona> findByNombre(String nombre);
```

- ▶ Pasamos argumentos con `?1, ?2, ..., ?n` o parámetros con nombre `:nombre, :apellidos`.
- ▶ Más flexibilidad que el método anterior.
- ▶ Recomendado si venimos del mundo SQL.

@QUERY

- ▶ También podemos usar SQL mediante consultas nativas.

```
@Query(value="select * from Persona where nombre = ?1",
        nativeQuery=true)
List<Persona> findByNombre(String nombre);
```

3.

OTROS MÉTODOS DE CONSULTA

QUERYDSL

- ▶ Framework que permite la construcción de consultas.
- ▶ Estáticas, tipadas y a través de un API.
- ▶ Integrable no solo con Spring Data JPA, sino con otros módulos (por ejemplo, Spring Data MongoDB)
- ▶ Cambios en el pom.xml

QUERYDSL

- ▶ Extendemos *QuerydslPredicateExecutor*

```
public interface QuerydslPredicateExecutor<T>
{
    Optional<T> findById(Predicate predicate);
    Iterable<T> findAll(Predicate predicate);
    long count(Predicate predicate);
    boolean exists(Predicate predicate);
    // ... más métodos
}
```

QUERYDSL

- ▶ Extendemos *QuerydslPredicateExecutor*

```
public interface UserInterface extends  
    CrudRepository<User, Long>,  
    QuerydslPredicateExecutor<User> {  
}
```

QUERYDSL

- ▶ Podemos usar predicados para consultar

```
Predicate predicate =  
    user.firstname.equalsIgnoreCase("pepe")  
        .and(user.lastname.startsWithIgnoreCase("s"));  
  
userRepository.findAll(predicate);
```

QUERYBYEXAMPLE

- ▶ Incluido dentro de Spring Data JPA
- ▶ *CrudRepository* +
QueryByExampleExecutor
- ▶ *JpaRepository*
- ▶ La idea subyacente es buscar dando un ejemplo (*Example<S>*) de lo que queremos encontrar.

QUERYBYEXAMPLE

- ▶ Búsqueda sencilla

```
Empleado filter = new Empleado();
empleado.setTelefono("954000000");
```

```
Example<Empleado> example = Example.of(filter);
List<Empleado> result =
    repositorio.findAll(example);
```

QUERYBYEXAMPLE

- ▶ Búsqueda algo más compleja

```
Empleado filter = new Empleado();
empleado.setNombre("mar");
ExampleMatcher matcher = ExampleMatcher.matching()
    .withStringMatcher(StringMatcher.CONTAINING
)
    .withIgnoreCase();
Example<Empleado> example =
    Example.of(filter, matcher);
List<Empleado> result =
    repositorio.findAll(example);
```

ALGO PARA PRACTICAR

Un ejercicio para hacer por tu cuenta

MÁS CONSULTAS

- ▶ Trata de incluir más consultas en el repositorio, y añade la lógica necesaria en el servicio y el controlador para poder visualizar los resultados.
- ▶ Por ejemplo:
 - ▶ modifica los teléfono para que sean aleatorios, y haz una búsqueda específica por teléfono.
 - ▶ Busca a los empleados que tengan nulos en algún campo.

CONSULTAS JPA

Accede a nuestro [Curso de Hibernate y JPA](#) para saber más sobre consultas, sobre todo si utilizas más de una entidad.

The screenshot shows a course page with the following details:

- URL:** [https://www.cursoonline.com/cursos/hibernate-jpa](#)
- Page Title:** Curso Online de Hibernate y JPA
- Breadcrumbs:** Estás en: [Inicio](#) / [Cursos de Programación y Sistemas](#) / [Backend](#) / [Curso Online de Hibernate y JPA](#)
- Description:** Para realizar persistencia de datos sin usar SQL directamente podemos hacer uso de un ORM como Hibernate. En este curso conocerás la persistencia de objetos en Java con Hibernate y Java Persistence API de forma práctica.
- Rating:** ★★★★ 4.1 (98 valoraciones)
- Duration:** 7 horas y 40 minutos
- Table of Contents (TEMARIO):**
 - X INTRODUCCIÓN** 45M
 - ▶ [Presentación del profesor y del curso](#) 3M
 - ▶ [Introducción](#) 19M