

Unidad 2 Introducción SPRING 5

Indice

1. Entorno de desarrollo.....	2
1.1. Creación de un proyecto web con Spring Boot:.....	2
2. SPRING BOOT.....	3
2.1 Uso de SPRING INITIALIZR.....	3
3. Estructura de un proyecto con SPRING BOOT.....	4
4. Cómo funciona una aplicación web.....	5
4.1 Métodos de petición http.....	5
5. Patrones de diseño.....	7
6. Estructura de un proyecto web.....	11
6.1. Controladores.....	11
6.2. Modelo.....	12
6.3. Vistas.....	13

1. Entorno de desarrollo

- Descargar Spring Tools 4 para Eclipse y descomprimir.
`spring-tool-suite-4-4.12.0.RELEASE-e4.21.0-linux.gtk.x86_64.tar.gz`
- Crear un workspace asociado.

- Podemos usar Spring desde Eclipse instalando el plug-in correspondiente.

Uso de Spring Tools 4: eclipse + plugin de Spring

Eclipse: Ir a Eclipse MarketPlace y busca Spring Tool

- Spring permite desplegar aplicaciones web tanto en jar como en war. Una de las ventajas de Spring Boot es que dispone de un servidor TomCat embebido que permite desplegar jar.

- Usamos war cuando queremos publicar en un servidor externo o cuando usamos JSP.

1.1. Creación de un proyecto web con Spring Boot:

- Añadir dependencias.
- `SpringBootWebApplication` como clase principal de la aplicación. Sobre la anotación hacer ctrl+clic

➤ Desde la pestaña New- Spring Starter Project

- Type: administrador de dependencias MAVEN
- Packaging: empaquetado del proyecto en spring se usa jar que es más portable (para uso en un servidor local o externo)

En Spring Boot:

- el formato jar se usa para web sobre todo con plantillas thymeleaf y api REST.
- Jar tiene la ventaja que está más optimizado para HTML5, no se compila a un servlet.
- war es más óptimo para publicar en un servidor externo.

- Java: Version 13 u 8.

- Group ID estructura de paquetes `com.daw.springboot.rest`

- Package raíz o base de la aplicación: dominio.nombreEmpresa.organizacion.com.daw.springboot.rest

➤ Siguiente ventana:

- Spring Boot Version Usar una estable(no snapshot o M2) 2.2.5
- Librerías o dependencias (Developer Tools):
 - Spring Web (JSP, REST, JSON, Thymeleaf)
 - Thymeleaf
 - Spring Boot DevTools (permite actualizar el servidor embebido, en el despliegue cualquier cambio que realicemos en el código).

Si al crear el proyecto hay fallos con algunas dependencias:

- cerramos Spring o Eclipse.
- Nos vamos a la carpeta del usuario `file:///home/nombreUsuario` y eliminamos la carpeta `.m2`
- Abrimos Spring y actualizamos proyecto con Maven

2. SPRING BOOT

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

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 terceros allá donde sea posible.
- ▶ Sin generar código o configuración XML.

2.1 Uso de SPRING INITIALIZR

Es un generador rápido de proyectos.

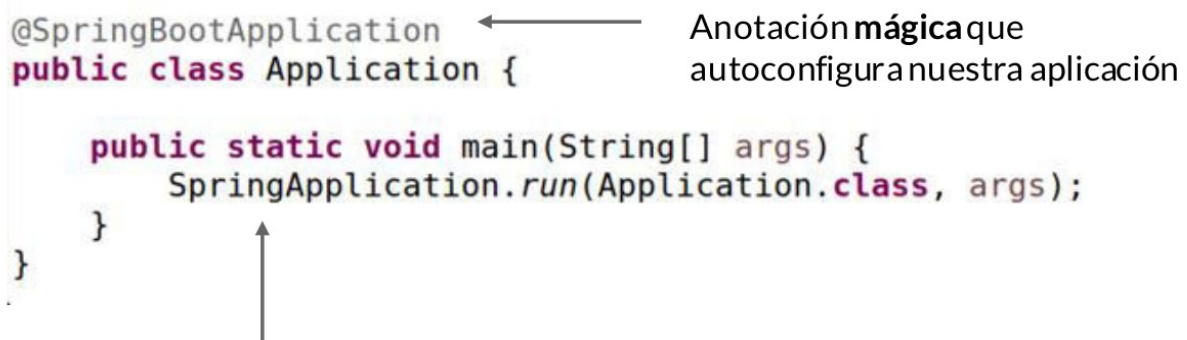
Simplemente vía HTML (<https://start.spring.io/>) o a través de un API (usando nuestro IDE)

Una vez definido el proyecto se puede descargar.

3. Estructura de un proyecto con SPRING BOOT

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

(con CTRL+Click podemos editarla)



```
@SpringBootApplication
public class Application {

    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

← Anotación mágica que autoconfigura nuestra aplicación

↑

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

4. Cómo funciona una aplicación web

4.1 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.
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
CSS3
JS, jQuery

Thymeleaf

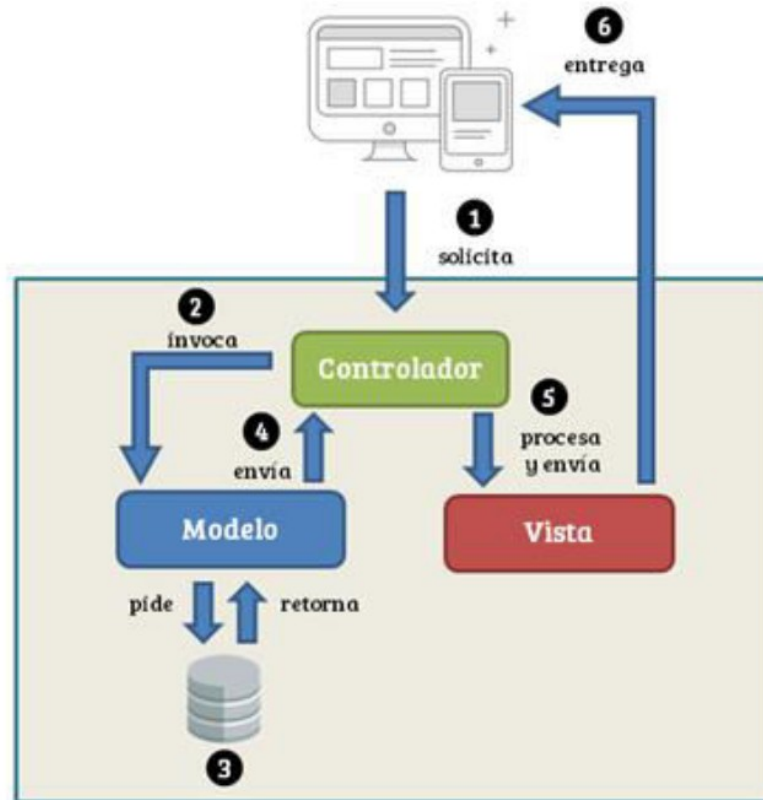
Spring Boot
Spring Web MVC
Spring Data JPA
Hibernate

H2

5. Patrones de diseño

► Una posible solución correcta a un problema de diseño dentro de un contexto, y que se presenta frecuentemente.

PATRÓN MVC: MODELO - VISTA - CONTROLADOR



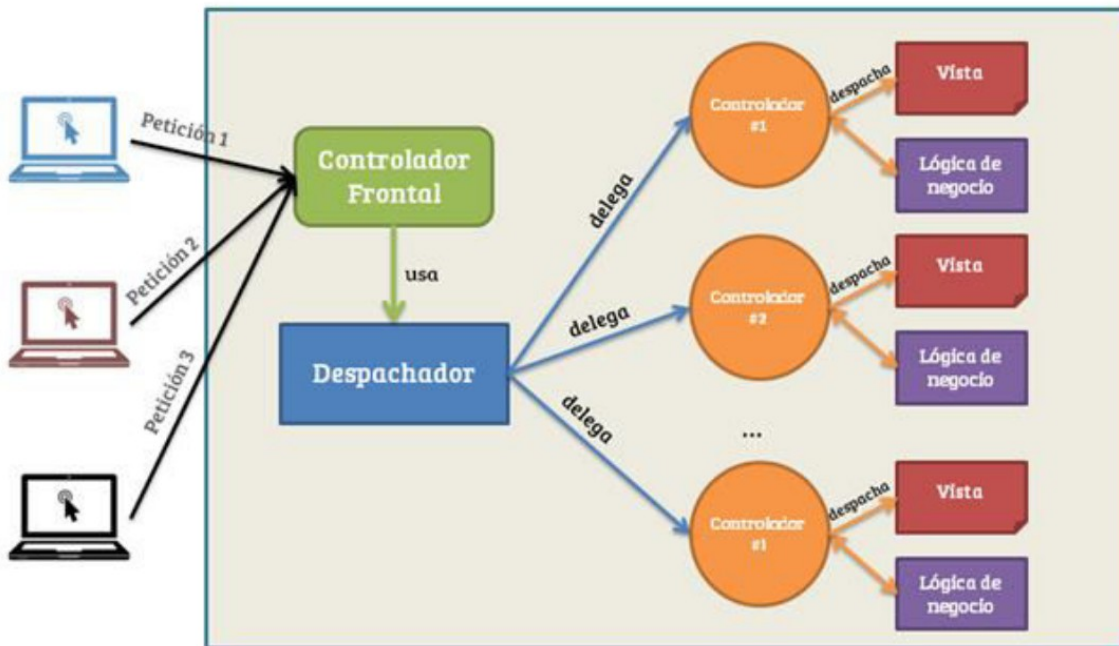
VENTAJAS

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

DESVENTAJAS

- Complejidad
- Coste de actualizaciones frecuentes.

PATRÓN FRONT CONTROLLER



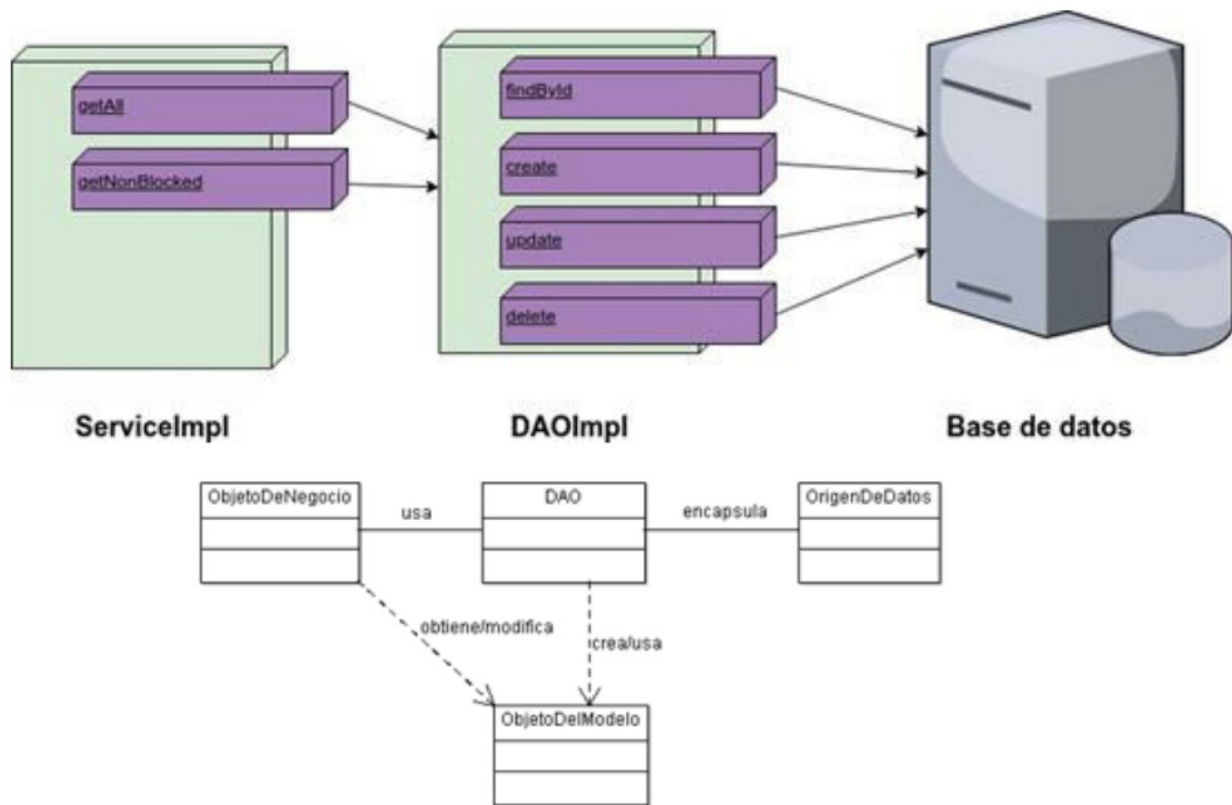
VENTAJAS

- 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



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.

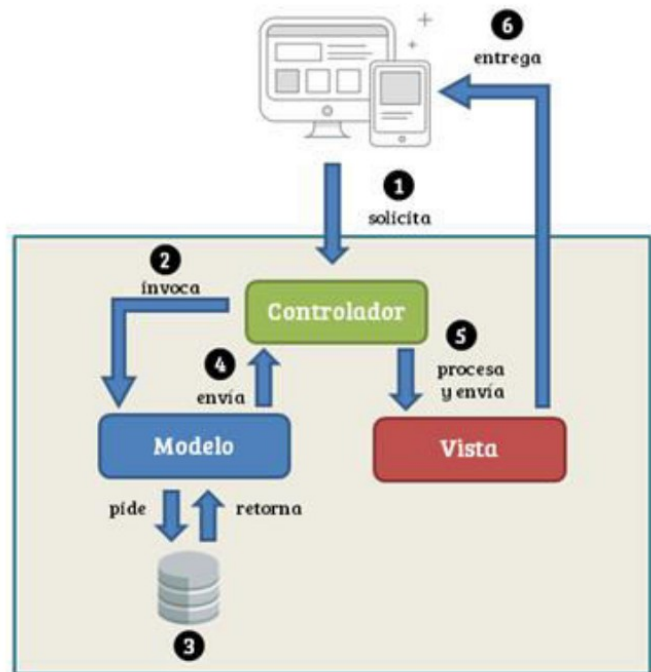
6. Estructura de un proyecto web

ESTRUCTURA SEGÚN MVC

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

Podemos añadir

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



6.1. Controladores

- ▶ Clases con métodos que atenderán las peticiones desde el navegador.
- ▶ Acceden al modelo, y lo retornan a la vista

¿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

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

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

En nuestra plantilla Thymeleaf, podemos utilizar los datos recibidos.

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

6.2. Modelo

- Entidades: clases que modelan los objetos de nuestro modelo de negocio.
- Repositorios (almacenes)
- Servicios (Lógica de negocio)

6.3. Vistas

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