01-2023

code
URJC

Desarrollo de Aplicaciones Web

Colección de ejercicios, ejemplos y proyecto final

Universidad Rey Juan Carlos

Micael Gallego
Correo: micael.gallego@urjc.es
Twitter: @micael_gallego

Francisco Gortázar
Correo: francisco.gortazar@urjc.es
Twitter: @fgortazar

Michel Maes
michel.maes@urjc.es

Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Índice

- Parte 1. Tecnologías web básicas de cliente
- Parte 2 – Tecnologías web de servidor
- Parte 3 – Despliegues de aplicaciones web
- Parte 4. Tecnologías web avanzadas de cliente
- Proyecto (Diseña e implementa una aplicación web)

code
URJC

Desarrollo de Aplicaciones Web

Parte 1. Tecnologías web básicas de cliente

code URJC

Desarrollo de Aplicaciones Web

Tema 2 – Maquetación: HTML y CSS

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 1

- Crea una página HTML (sin incluir estilos)
- El contenido puede ser de cualquier temática:
  - Una empresa, un grupo de música, un departamento de la universidad, un producto software, un teléfono móvil, una asociación cultural, etc…
- El contenido puede ser inventado o real (obtenido de una página de Internet)

Ejercicio 1

- La página debe incluir, al menos:
  - Varias secciones con subsecciones
  - Fotografías/Imágenes
  - Listas de elementos
  - Una o varias tablas
  - Hiperenlaces

Índice de Ejercicios

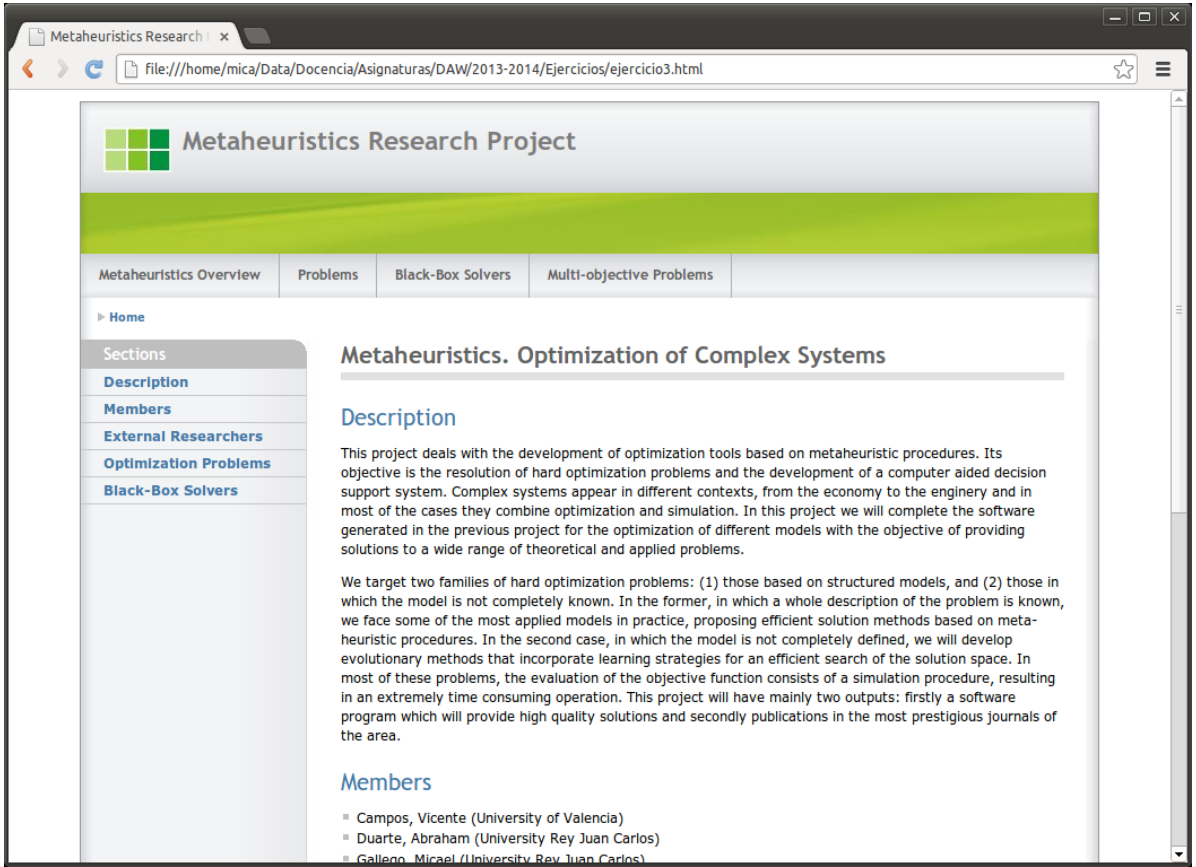
Ejercicio 2

- Crea una página web con HTML y CSS con el siguiente botón

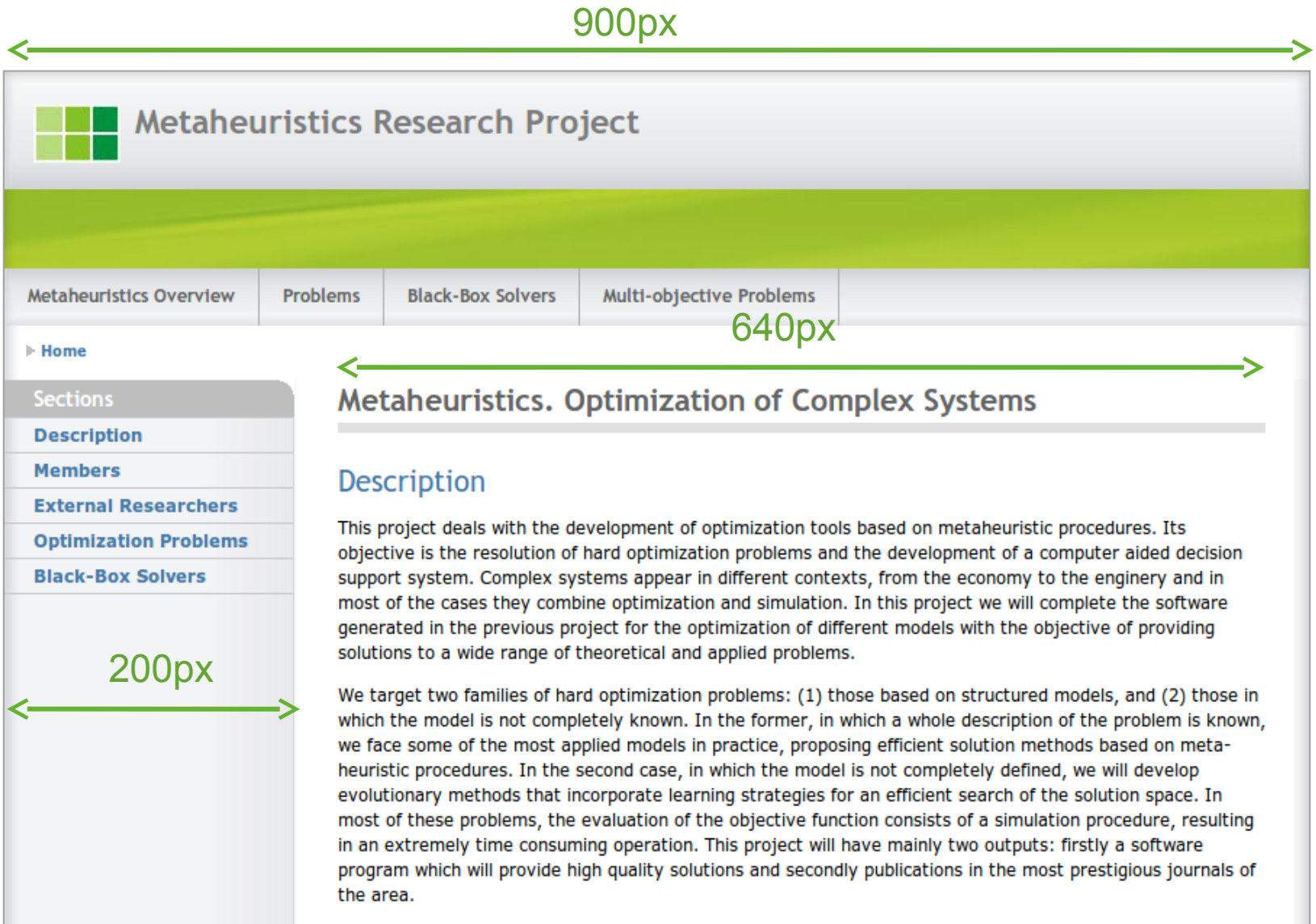
Índice de Ejercicios

CSS
Ejercicio 3

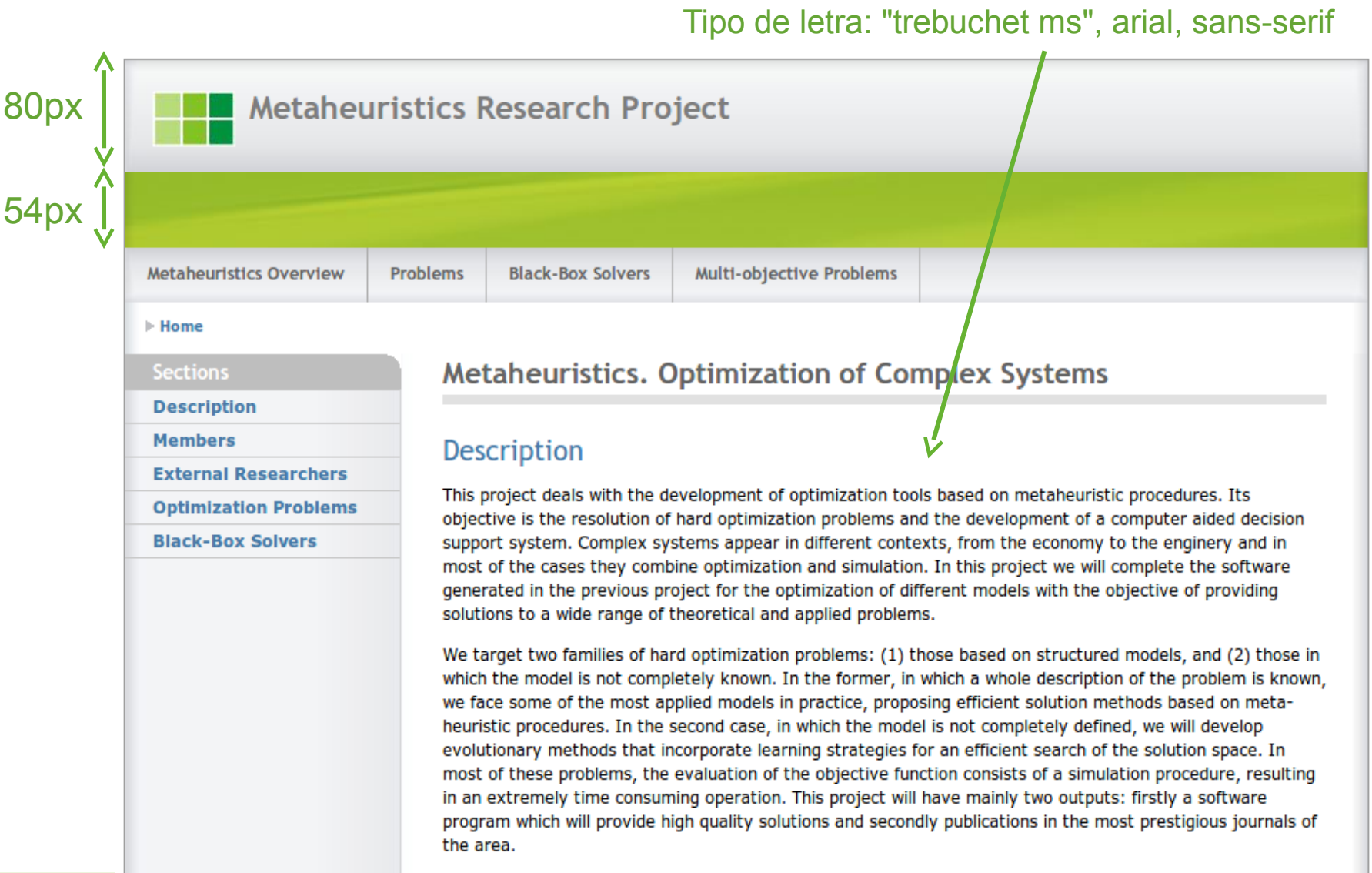
- Aplica el siguiente estilo a la página del ejercicio 1



Ejercicio 3

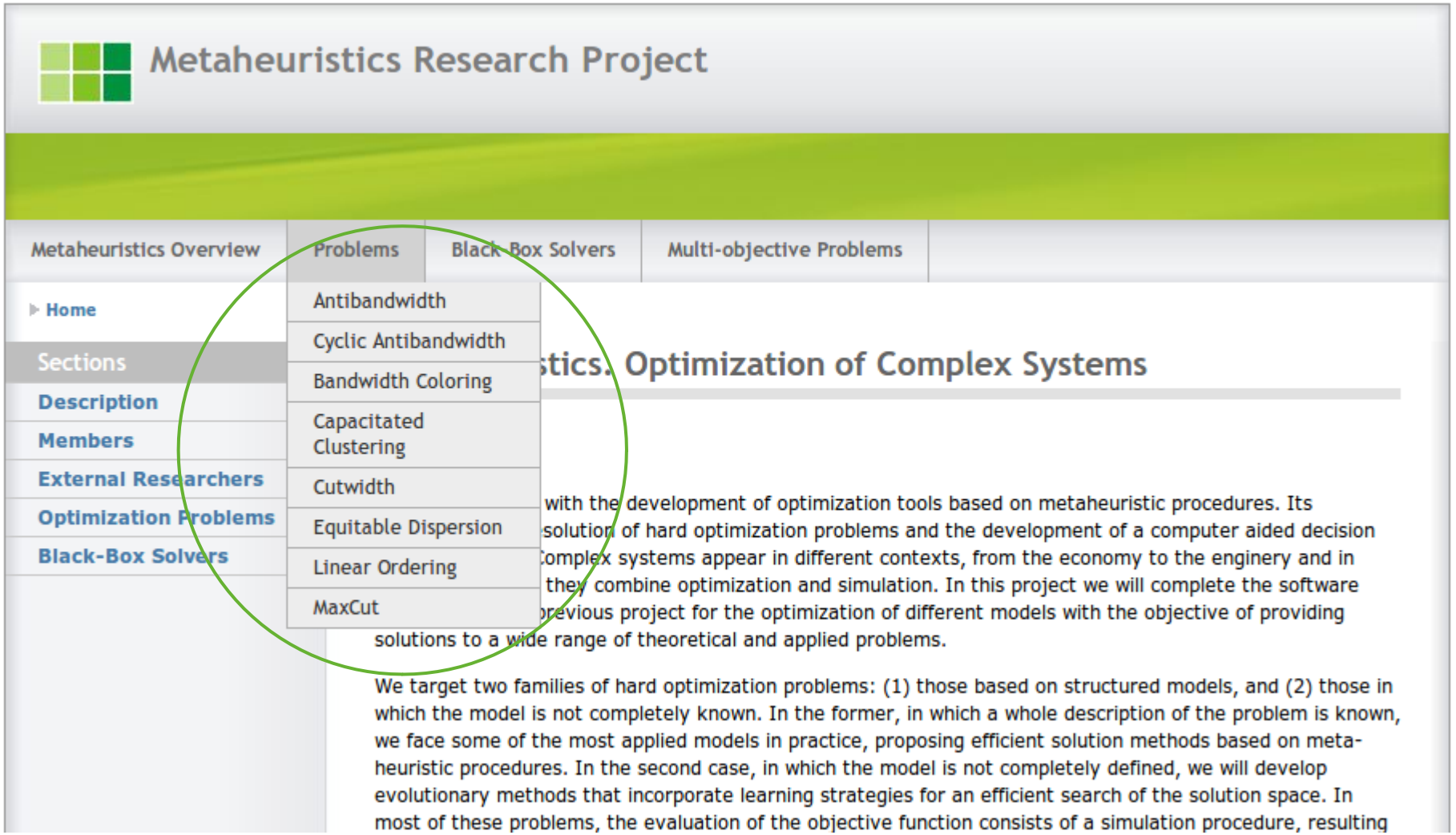

900px
Metaheuristics Research Project
Metaheuristics Overview
Problems
Black-Box Solvers
Multi-objective Problems
640px
Home
Sections
Description
Members
External Researchers
Optimization Problems
Black-Box Solvers
200px
Metaheuristics. Optimization of Complex Systems
Description
This project deals with the development of optimization tools based on metaheuristic procedures. Its objective is the resolution of hard optimization problems and the development of a computer aided decision support system. Complex systems appear in different contexts, from the economy to the enginery and in most of the cases they combine optimization and simulation. In this project we will complete the software generated in the previous project for the optimization of different models with the objective of providing solutions to a wide range of theoretical and applied problems.
We target two families of hard optimization problems: (1) those based on structured models, and (2) those in which the model is not completely known. In the former, in which a whole description of the problem is known, we face some of the most applied models in practice, proposing efficient solution methods based on meta-heuristic procedures. In the second case, in which the model is not completely defined, we will develop evolutionary methods that incorporate learning strategies for an efficient search of the solution space. In most of these problems, the evaluation of the objective function consists of a simulation procedure, resulting in an extremely time consuming operation. This project will have mainly two outputs: firstly a software program which will provide high quality solutions and secondly publications in the most prestigious journals of the area.

CSS
Ejercicio 3

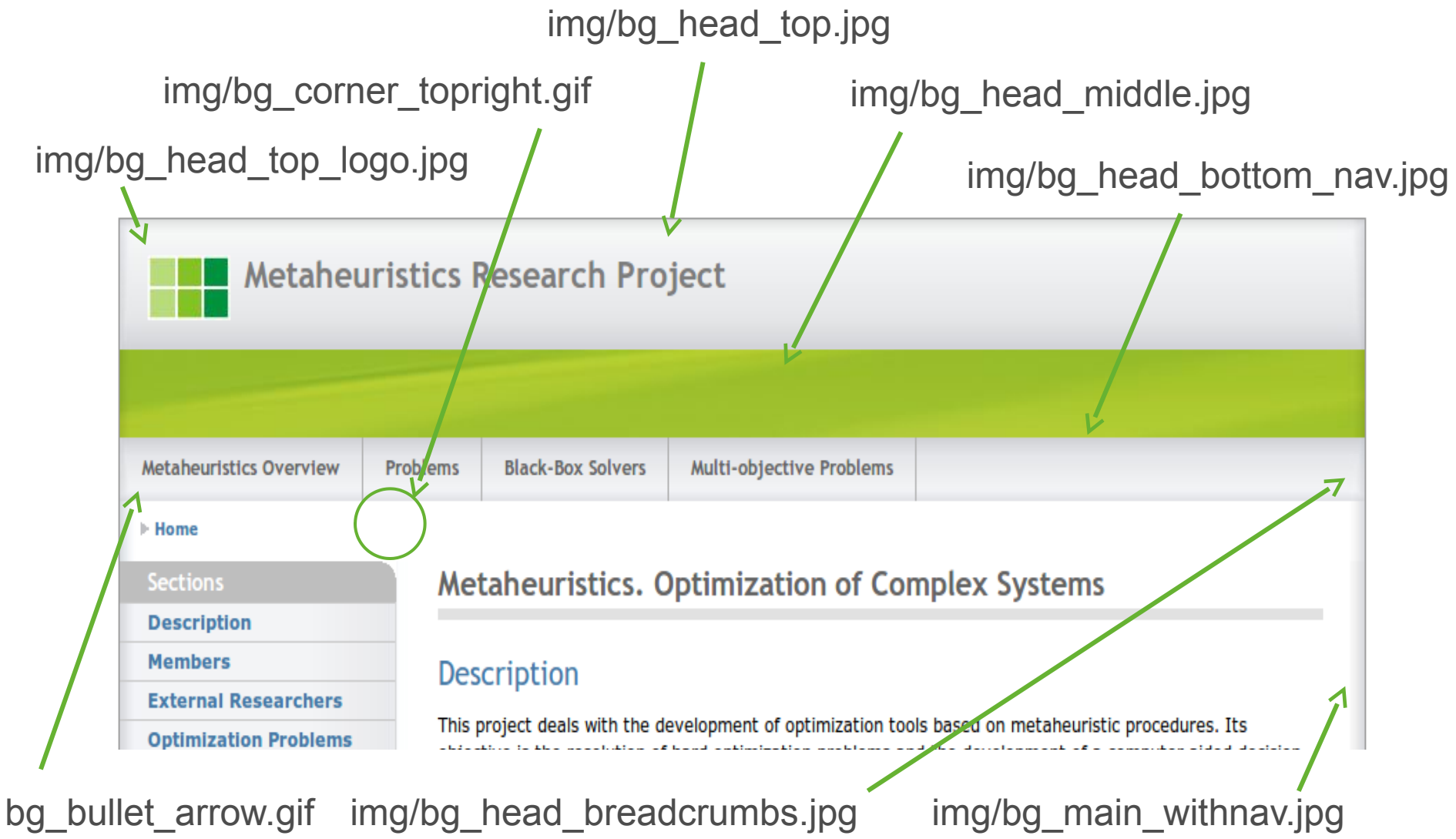


CSS
Ejercicio 3

code URJC



CSS
Ejercicio 3



CSS
Ejercicio 3

img/bg_foot.jpg

Ejercicio 3

- **Imágenes de fondo:**
  - Propiedad background
  - Se puede poner un color por si no carga la imagen
  - Se indica la ruta de la imagen con la función url(...)
  - Se indica la repetición de la imagen
  - Otras propiedades de la imagen

```
background: url(img/bg_corner_topright.gif) no-repeat;
background: transparent url(img/...) no-repeat 0 50%;
```

http://devdocs.io/css/background

Ejercicio 3

- **Otros detalles CSS:**
  - Padding y margin con valor cero a todos los elementos de la página
  - Tamaño de texto a todos los elementos del body a 0.6em y tipo de letra sans-serif
  - Posicionamiento en profundidad: z-index: 1
  - Quitar los puntos de una lista:
    - list-style-type: none;

code URJC

Desarrollo de Aplicaciones Web

Tema 3 – Introducción a JavaScript

Índice de Ejercicios

Índice de Ejercicios

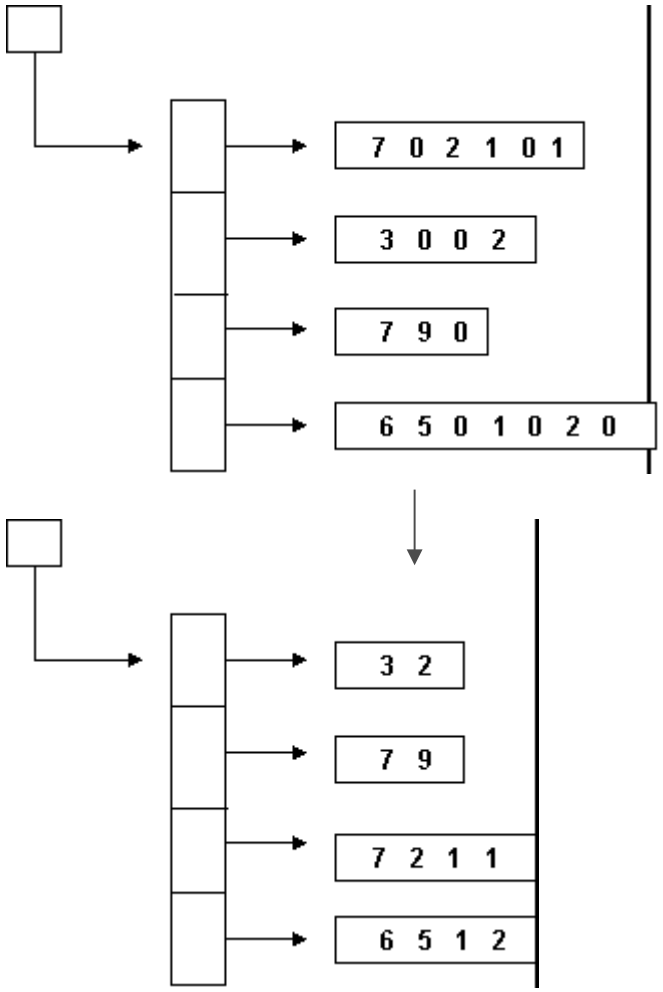
Ejercicio 1

- Crear una función que **reciba un array como parámetro** y devuelva un array de dos elementos
- El primer elemento apuntará a un **array con los números pares** del array que se pasa como parámetro
- El segundo elemento apuntará a un **array con los números impares** del array que se pasa como parámetro
- Para probar la función se implementarán varias llamadas con **diferentes arrays** y el resultado se mostrará en la **consola del navegador**

Índice de Ejercicios

Ejercicio 2

- Crear una función que **reciba un array bidimensional, le quite los ceros y ordene las filas de menor a mayor longitud**
- Para probar la función se implementarán varias llamadas con **diferentes arrays** y el resultado se mostrará en la **consola del navegador**



Ejercicio 2

- Para ordenar los arrays por tamaño se puede usar el **algoritmo de la burbuja** (*bubble sort*)
- El algoritmo consiste en comparar cada elemento del array con el siguiente, intercambiándolos de posición si están en el orden equivocado
- El array se recorre hasta que todos los elementos estén ordenados (un elemento siempre es menor que el siguiente)

Índice de Ejercicios

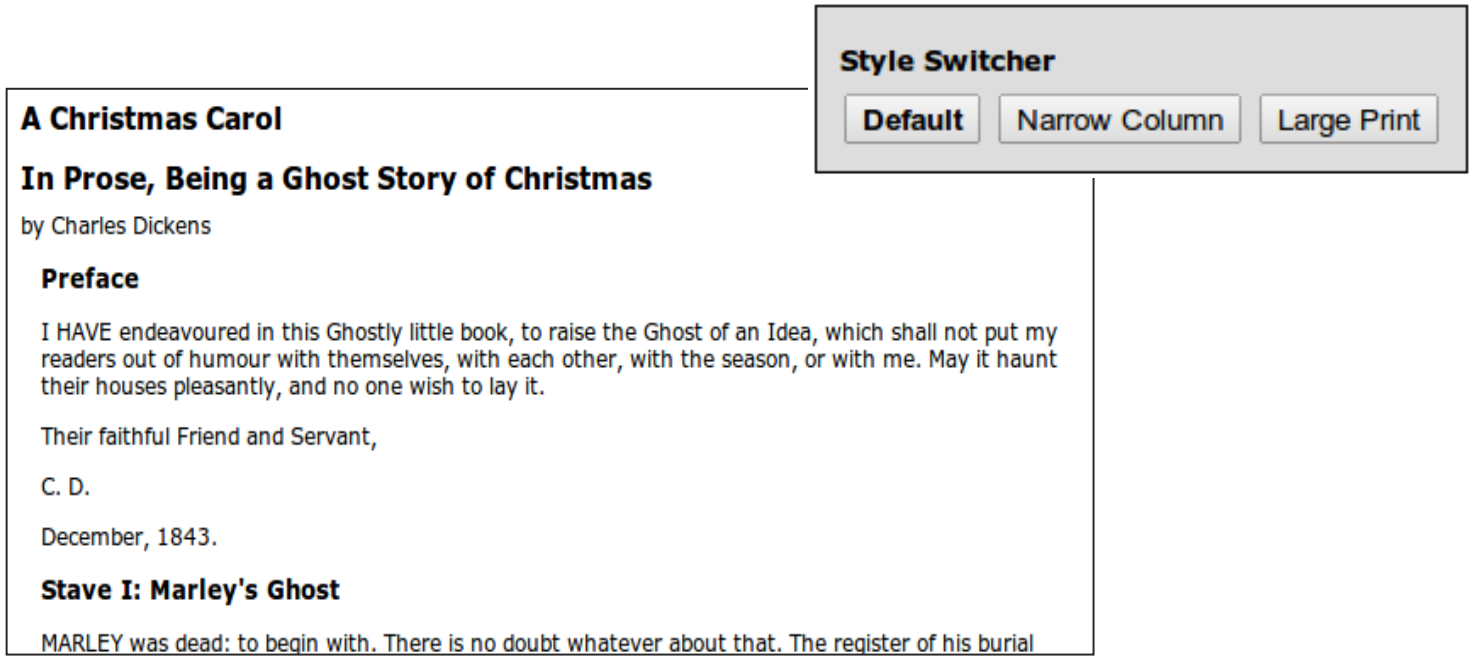
Ejercicio 3

- Crea una página con campo de texto y un botón
- Cada vez que se pulse el botón, se añadirá el contenido del cuadro de texto a la página (sin borrar el contenido previo)

Índice de Ejercicios

Ejercicio 4

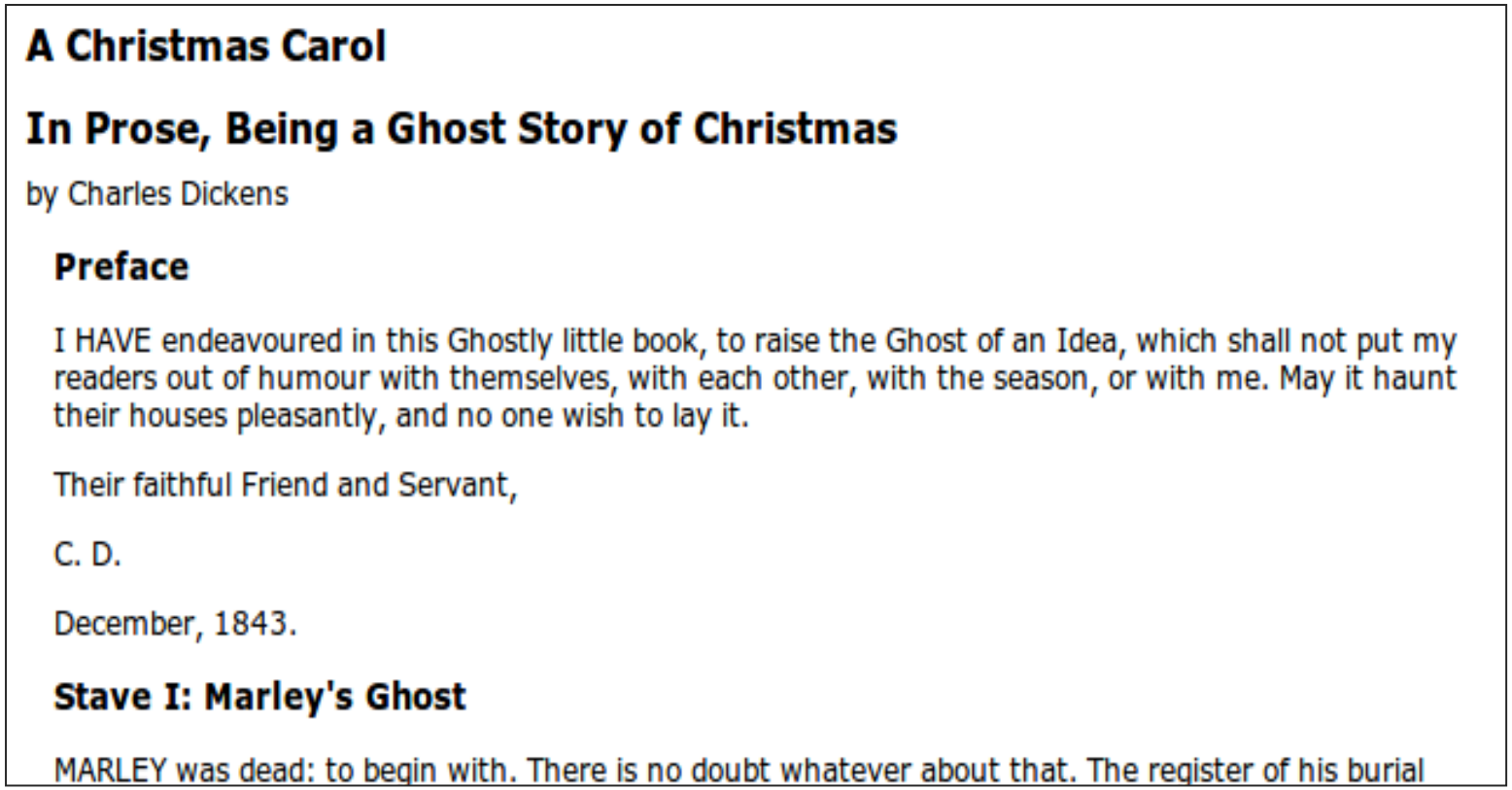
- Se desea implementar una **barra de herramientas** para modificar los **estilos de un documento**



Ejercicio 4

code URJC

- Documento de ejemplo (descargar)



A Christmas Carol

In Prose, Being a Ghost Story of Christmas

by Charles Dickens

Preface

I HAVE endeavoured in this Ghostly little book, to raise the Ghost of an Idea, which shall not put my readers out of humour with themselves, with each other, with the season, or with me. May it haunt their houses pleasantly, and no one wish to lay it.

Their faithful Friend and Servant,

C. D.

December, 1843.

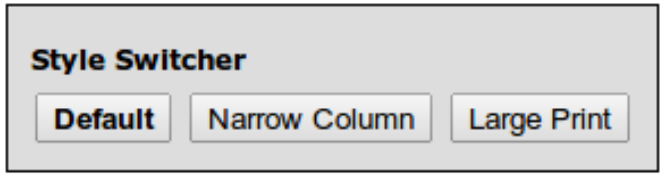
Stave I: Marley's Ghost

MARLEY was dead: to begin with. There is no doubt whatever about that. The register of his burial

Ejercicio 4

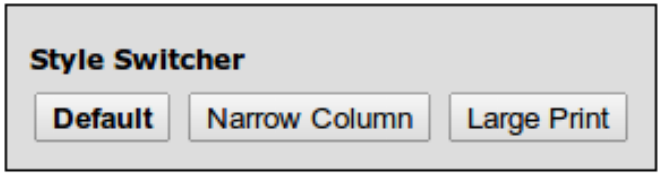
- **Barra de herramientas**

```
<div id="switcher" class="switcher">
  <h3>Style Switcher</h3>
  <button id="switcher-default">
      Default
  </button>
  <button id="switcher-narrow">
      Narrow Column
  </button>
  <button id="switcher-large">
      Large Print
  </button>
</div>
```



Ejercicio 4

- **Barra de herramientas**
  - **Botones**
    - **Default:** Volver al estilo normal
    - **Large print:** Aplicar el estilo CSS "large" al body (y quitar los demás)
    - **Narrow Column:** Aplicar el estilo CSS "narrow" al body (y quitar los demás)
  - **Opción seleccionada**
    - La **opción seleccionada** debería reflejarse mostrando el texto del botón en **negrita (estilo "selected")**



Ejercicio 4

- **Barra de herramientas**
  - Al cargar el documento, la barra de herramientas debería aparecer **minimizada** (usando la clase CSS "hidden")
  - Al pulsar en ella, debe cambiar de modo **minimizado** a **normal** (y viceversa)

Ejercicio 4

- **Barra de herramientas**
  - Las teclas también pueden configurar el estilo:
    - D: Default
    - N: Narrow
    - L: Large

Ejercicio 4

code URJC

- **Utilidades JavaScript / jQuery necesarias**
  - **$(...).is("button"):** Indica si el elemento tiene el nombre de la etiqueta indicado
  - **"sss-www".split('-'):** divide el String en un array de Strings usando el '-'
  - **$(document).keyup(...):** Configura una función que se ejecutará cuando se pulse una tecla
  - **String.fromCharCode(event.which):** Devuelve la tecla pulsada como un String

01-2023

code URJC

Desarrollo de Aplicaciones Web

Tema 4 – JavaScript Avanzado

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 1

- Implementa la clase **Intervalo**
  - Atributos: limiteInf, limiteSup
  - Métodos: incluido(num), longitud, toString

- Implementa un script de prueba que haga uso de varios objetos de esa clase

Índice de Ejercicios

Ejercicio 2

- Crear una jerarquía de herencia para representar **figuras geométricas**
- De cada figura se debe conocer:
  - **Color**
  - **Perímetro**
  - **Área**
  - **Ratio entre el área y el perímetro (area / per)**
- Las figuras concretas que tiene que contemplar el programa son **círculos** y **rectángulos**

Ejercicio 2

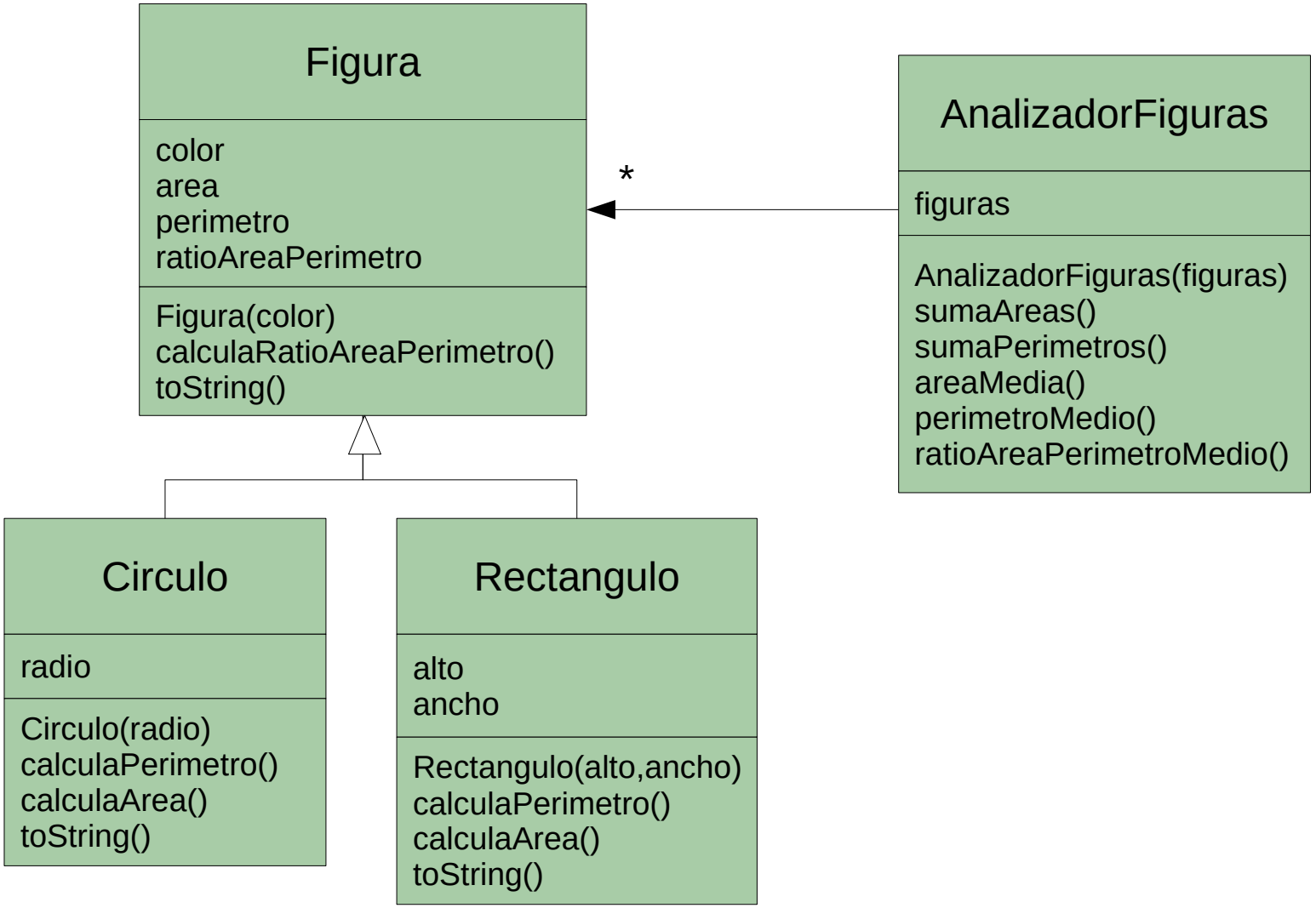


Figura
color
area
perimetro
ratioAreaPerimetro
Figura(color)
calculaRatioAreaPerimetro()
toString()
*
AnalizadorFiguras
figuras
AnalizadorFiguras(figuras)
sumaAreas()
sumaPerimetros()
areaMedia()
perimetroMedio()
ratioAreaPerimetroMedio()
Circulo
radio
Circulo(radio)
calculaPerimetro()
calculaArea()
toString()
Rectangulo
alto
ancho
Rectangulo(alto,ancho)
calculaPerimetro()
calculaArea()
toString()

Ejercicio 2

- Implementar la clase **AnalizadorFiguras** que permita analizar un array de figuras.
- Los **análisis** serán:
  - Suma total de áreas
  - Suma total de perímetros
  - Área media
  - Perímetro medio
  - Ratio area perímetro medio
- Se deberá crear un **script** que:
  - Construya un array con diferentes figuras de distintos tipos y con distintos valores
  - Ejecute todos los análisis del AnalizadorFiguras
  - Muestre el resultado en la página HTML

Ejercicio 2

- **Círculo**
  - Área: Math.PI * radio * radio
  - Perímetro: 2 * Math.PI * radio
- **Rectángulo**
  - Área: largo * ancho
  - Perímetro: 2 * ancho + 2 * largo

Índice de Ejercicios

Ejercicio 3

- Crear una jerarquía de herencia para representar **figuras geométricas**
- De cada figura se debe conocer:
  - **Color**
  - **Perímetro**
  - **Área**
  - **Ratio entre el área y el perímetro (area / per)**
- Las figuras concretas que tiene que contemplar el programa son **círculos** y **rectángulos**

Ejercicio 3

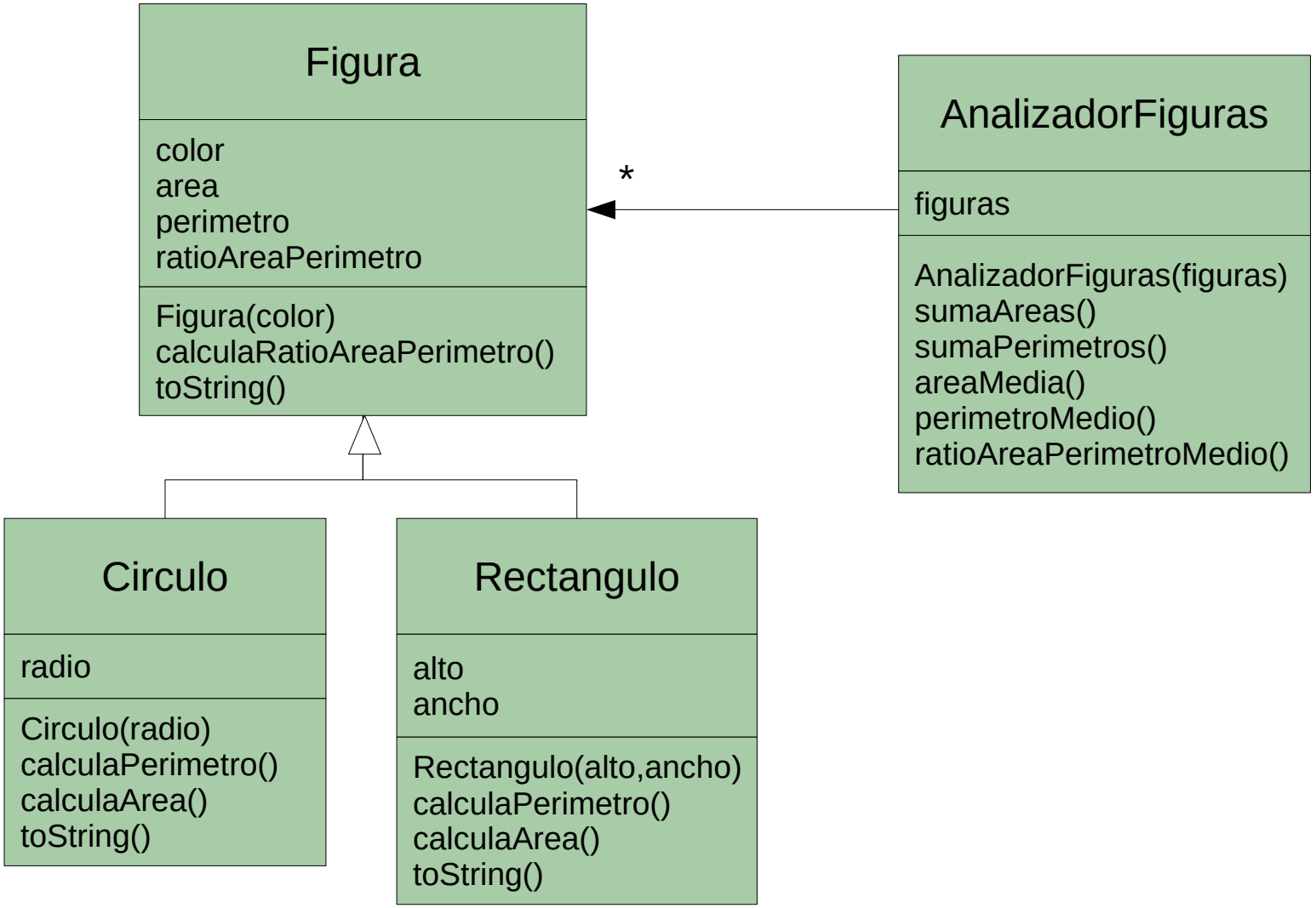


Figura
color
area
perimetro
ratioAreaPerimetro
Figura(color)
calculaRatioAreaPerimetro()
toString()
*
AnalizadorFiguras
figuras
AnalizadorFiguras(figuras)
sumaAreas()
sumaPerimetros()
areaMedia()
perimetroMedio()
ratioAreaPerimetroMedio()
Circulo
radio
Circulo(radio)
calculaPerimetro()
calculaArea()
toString()
Rectangulo
alto
ancho
Rectangulo(alto,ancho)
calculaPerimetro()
calculaArea()
toString()

Ejercicio 3

- Implementar la clase **AnalizadorFiguras** que permita analizar un array de figuras.
- Los **análisis** serán:
  - Suma total de áreas
  - Suma total de perímetros
  - Área media
  - Perímetro medio
  - Ratio area perímetro medio
- Se deberá crear un **script** que:
  - Construya un array con diferentes figuras de distintos tipos y con distintos valores
  - Ejecute todos los análisis del AnalizadorFiguras
  - Muestre el resultado en la página HTML

Ejercicio 3

- **Círculo**
  - Área: Math.PI * radio * radio
  - Perímetro: 2 * Math.PI * radio
- **Rectángulo**
  - Área: largo * ancho
  - Perímetro: 2 * ancho + 2 * largo

Índice de Ejercicios

Ejercicio 4

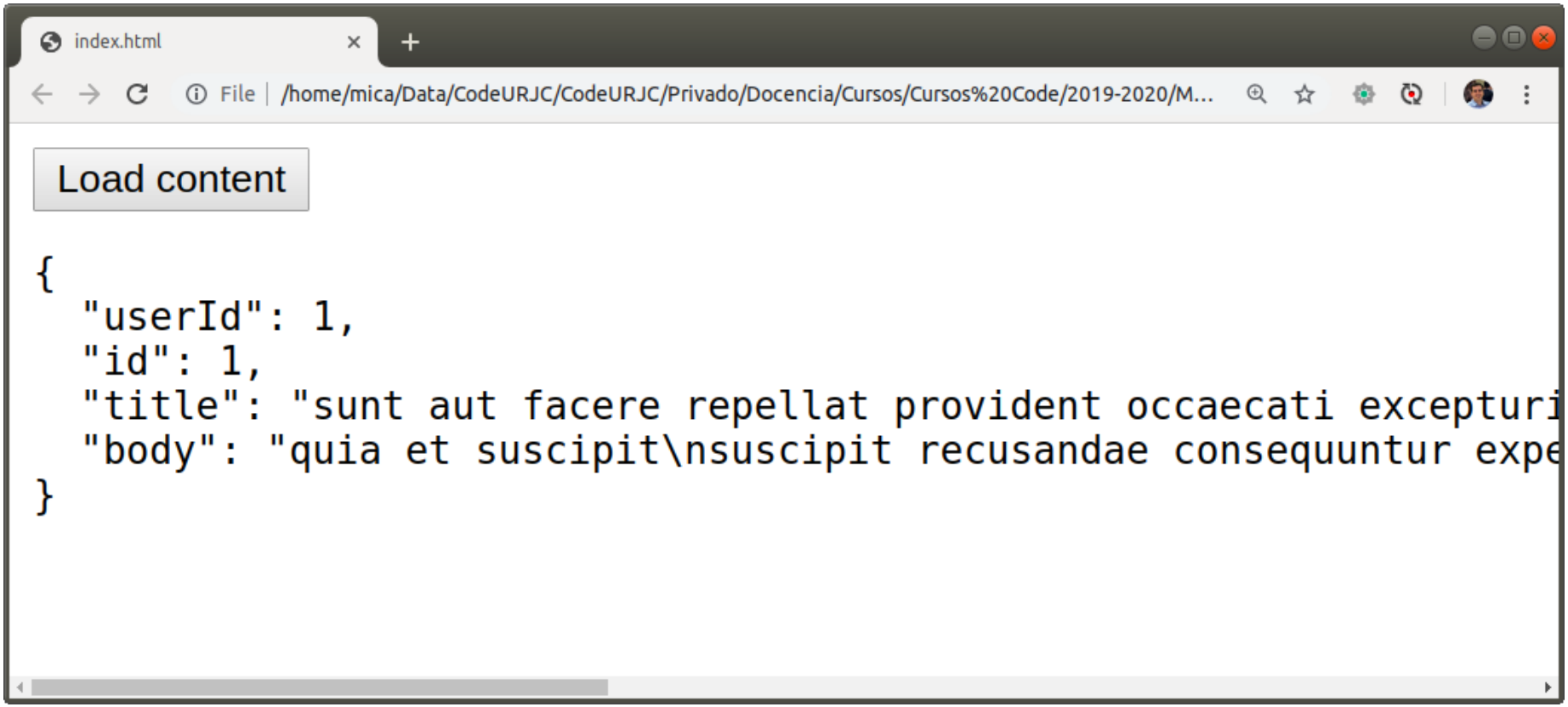
- Fetch es una versión mejorada de XMLHttpRequest para hacer peticiones AJAX
- Está basada en promesas, lo que simplifica su uso
- Implementa una web que muestre un botón y cargue el contenido de la URL cuando el botón se pulse
- El contenido es un JSON con información de un post

https://jsonplaceholder.typicode.com/posts/1

https://developers.google.com/web/updates/2015/03/introduction-to-fetch

https://developer.mozilla.org/es/docs/Web/API/Fetch_API/Utilizando_Fetch

Ejercicio 4


index.html
File | /home/mica/Data/CodeURJC/CodeURJC/Privado/Docencia/Cursos/Cursos%20Code/2019-2020/M...
Load content
{
"userId": 1,
"id": 1,
"title": "sunt aut facere repellat provident occaecati excepturi
"body": "quia et suscipit\nsuscipit recusandae consequuntur expe
}

Índice de Ejercicios

Ejercicio 5

- Supongamos que la función fetch no existe en el browser
- Vamos a implementar una versión simiplicada de fetch llamada “load”
- No vamos a basar en el XMLHttpRequest del browser

Ejercicio 5

- Funcionamiento de **XMLHttpRequest**

```
var req = new XMLHttpRequest();
req.open("GET", "http://..", true);
req.onreadystatechange = function() {
  if (req.readyState == 4) {
     if(req.status == 200)
        console.log(req.responseText);
     else
        console.log("Error loading page");
  }
};

req.send();
```

https://developer.mozilla.org/es/docs/Web/API/XMLHttpRequest/Using_XMLHttpRequest

Ejercicio 5

- Se pide crear una función **load(...)** que reciba la URL como parámetro y devuelva una promesa

```
function load(url){ ... }

load("http://..")
  .then(text =>{
     console.log(text);
  })
  .catch(error => {
    console.log(error);
  })
```

- Cuando exista un error en la carga, debería lanzarse un error desde la promesa

Índice de Ejercicios

Ejercicio 6

- Convierte el ejercicio del fetch (ejercicio 4) para que use async / await

code
URJC

Desarrollo de Aplicaciones Web

Parte 2 – Tecnologías web de servidor

code
URJC

Desarrollo de Aplicaciones Web

Tema 1.1 – Spring y Maven

Índice de Ejemplos

- Ejemplo 1 (Crear proyecto Spring)

Índice de Ejemplos

- Ejemplo 1 (Crear proyecto Spring)

Crear proyecto Spring

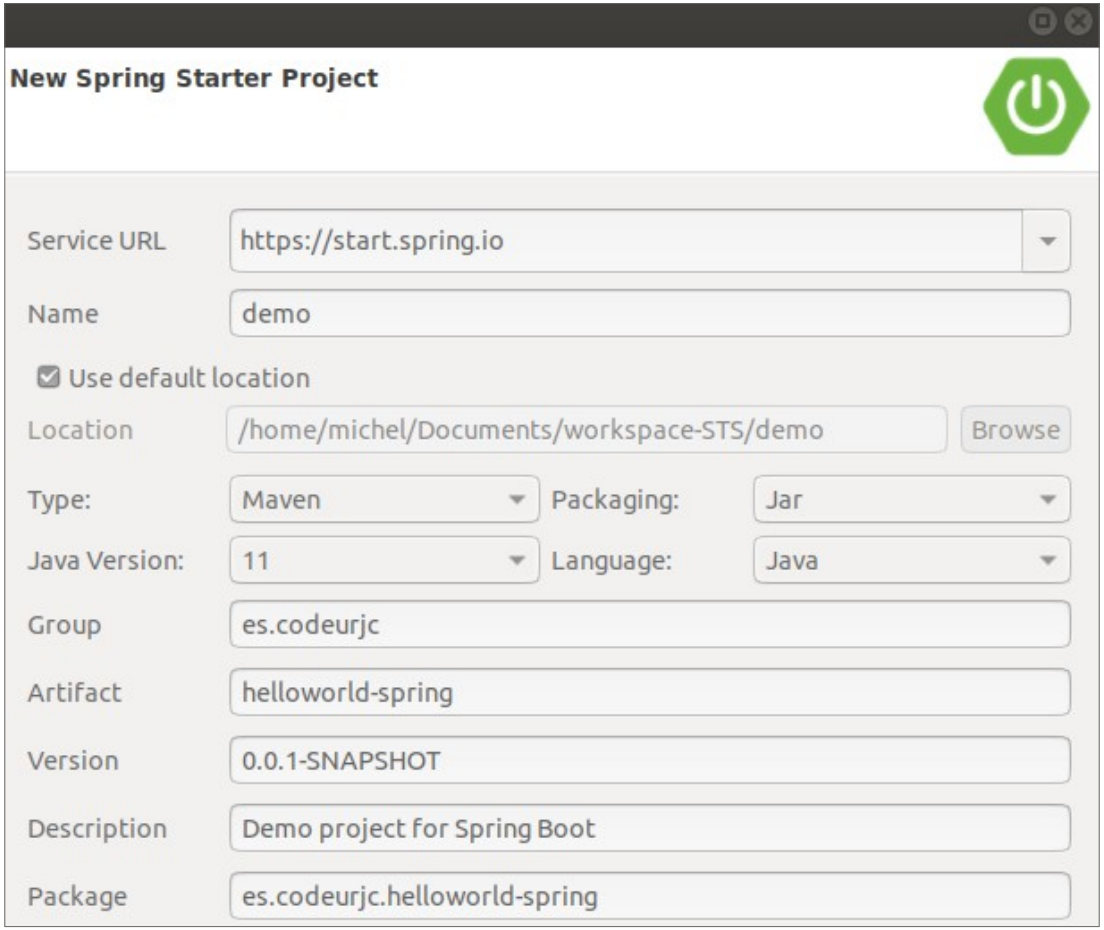
code URJC

- **Eclipse STS**
  - File > New project > Spring Starter Project
  - Selecciona la versión de Java (8, 11 o 15)
  - Indicar el nombre del proyecto:
    - GroupId: es.codeurjc
    - ArtifactId: helloworld-spring
  - Se seleccionan las librerías que se quieren usar

Crear proyecto Spring

code URJC

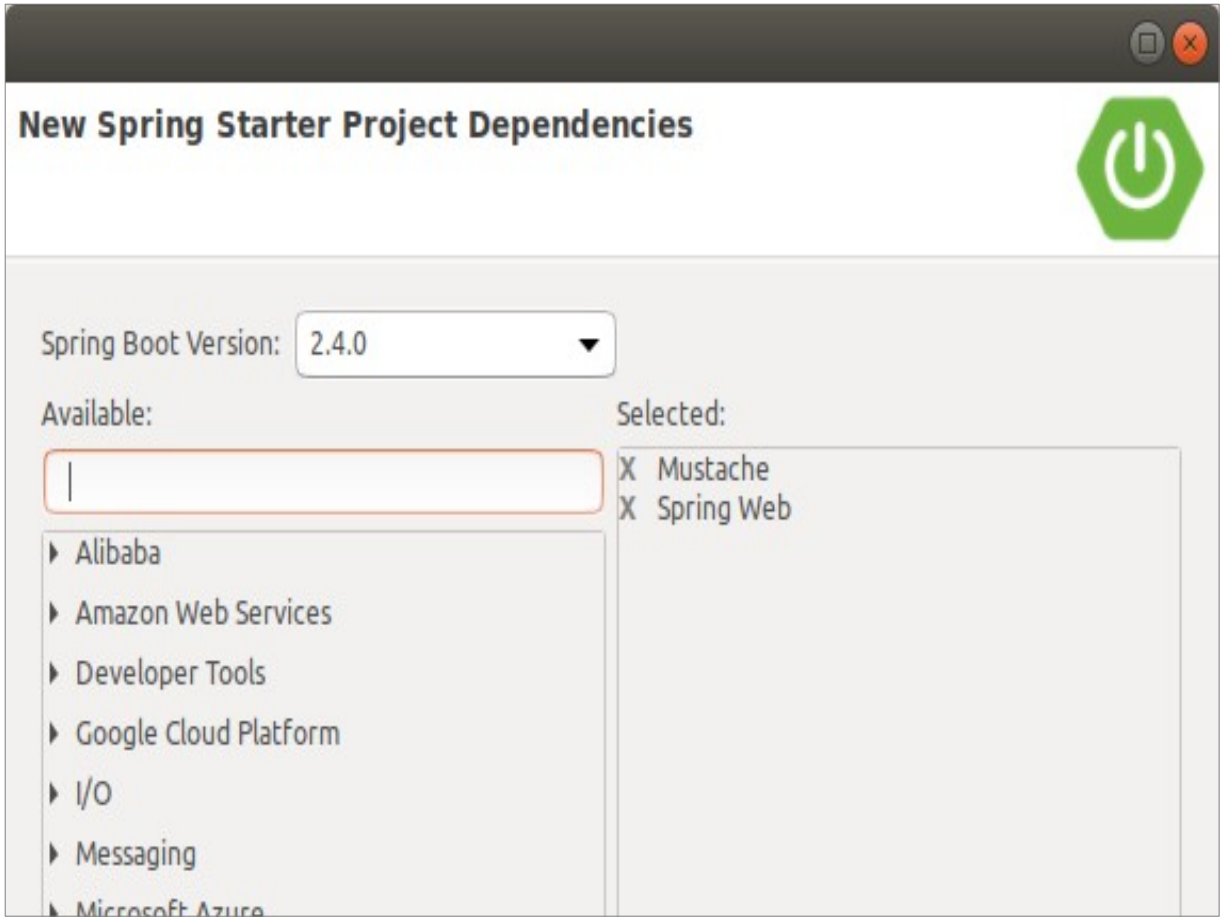
- Eclipse STS



Crear proyecto Spring

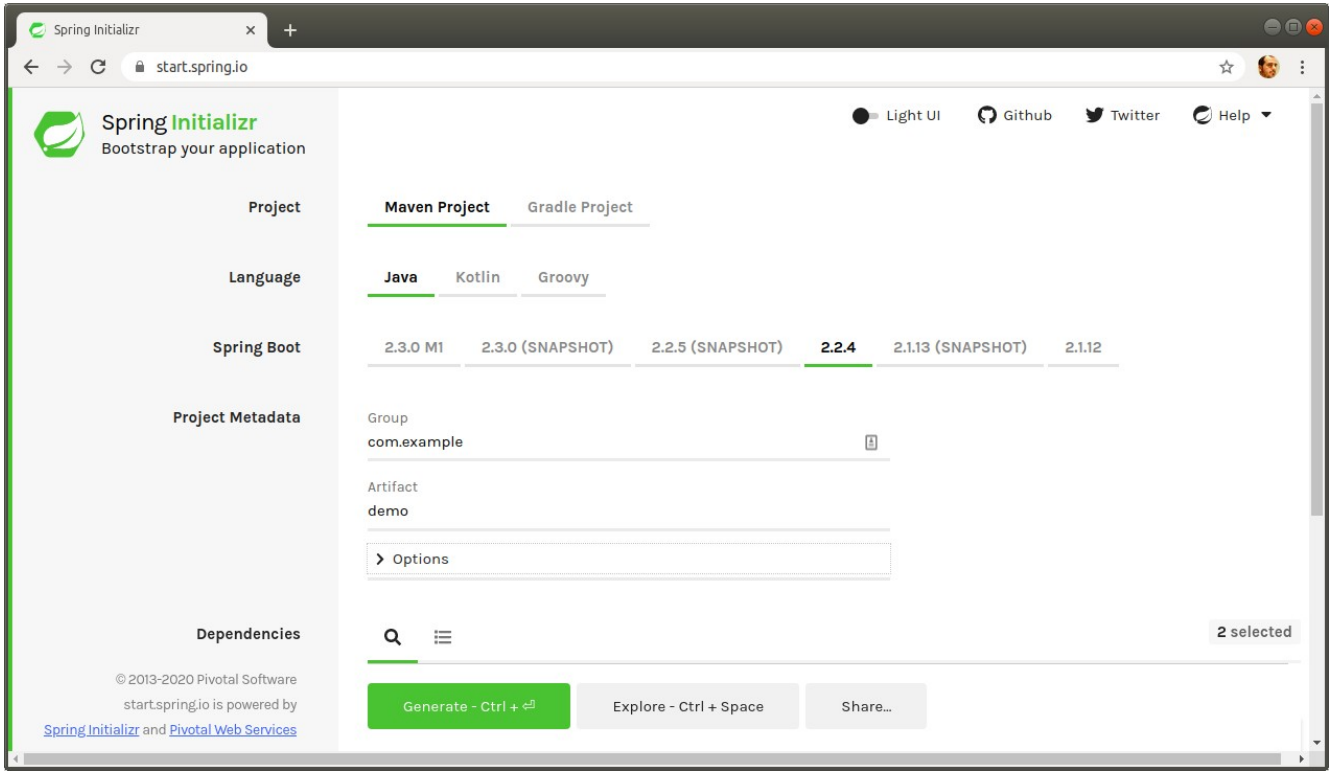
code URJC

- Eclipse STS



Crear proyecto Spring

- Desde la web



https://start.spring.io

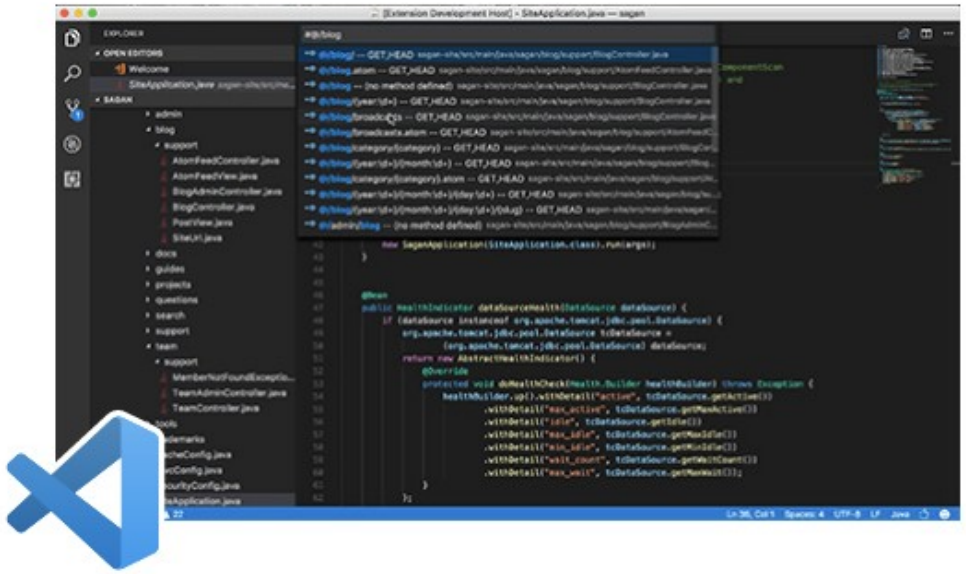
Crear proyecto Spring

- Visual Studio Code

Spring Tools 4 for Visual Studio Code

Free. Open source.

SPRING TOOLS 4
VSCode Marketplace

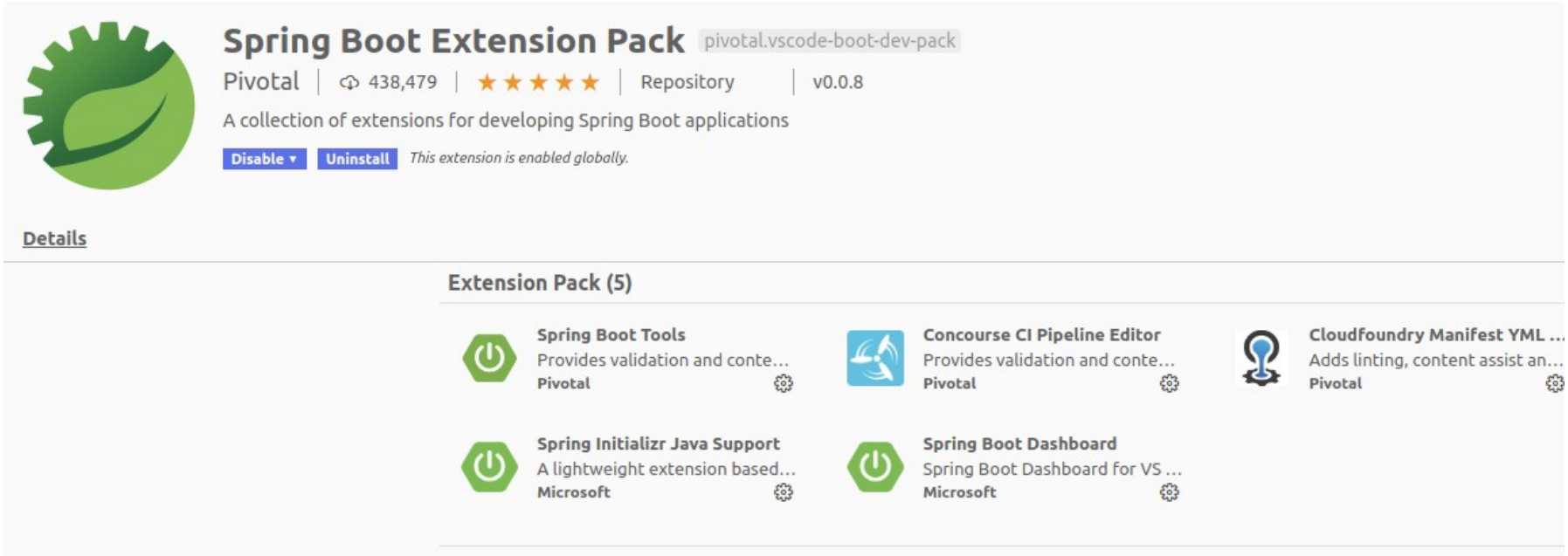


Instalamos las extensiones de Spring en nuestro VSCode

https://marketplace.visualstudio.com/items?itemName=Pivotal.vscode-boot-dev-pack

Crear proyecto Spring

- Visual Studio Code



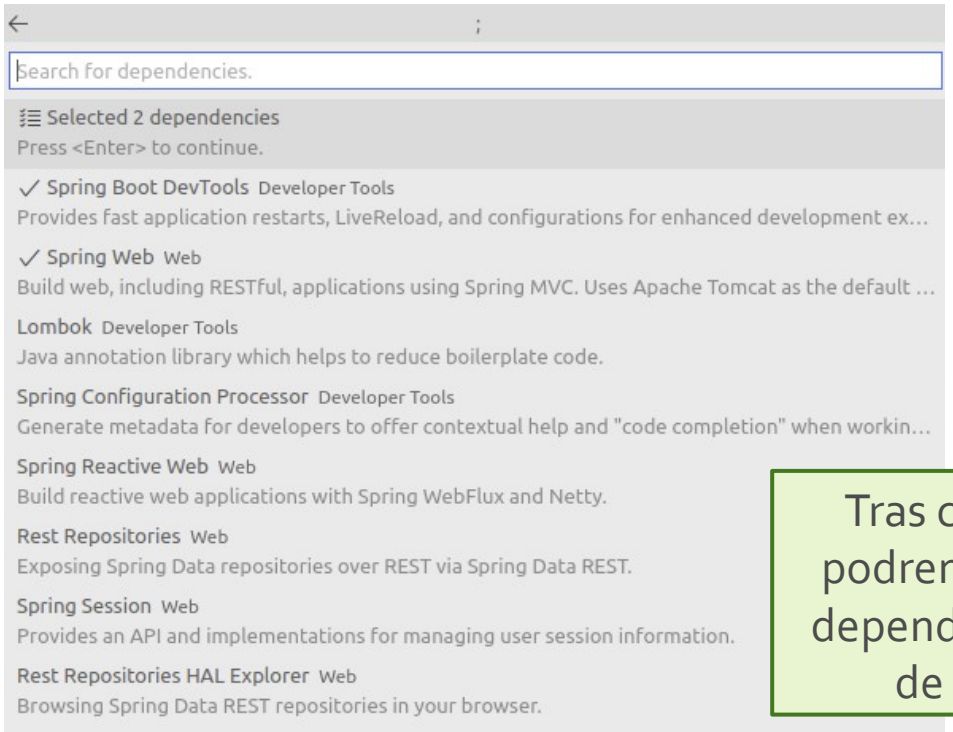
Crear proyecto Spring

code URJC

- **Visual Studio Code**
  - View > Command Palette (o Ctrl + Shift + P)
  - Introducimos "Spring" y seleccionamos "Create a Maven Project"
  - Nos pedirá incluir información adicional:
    - Versión de Spring Boot: *2.3.4*
    - Lenguaje: *Java*
    - GroupId: *es.codeurjc* y ArtifactId: *helloword-vscode*
    - Empaquetamiento: *JAR*
    - Versión: *1.8 u 11*

Crear proyecto Spring

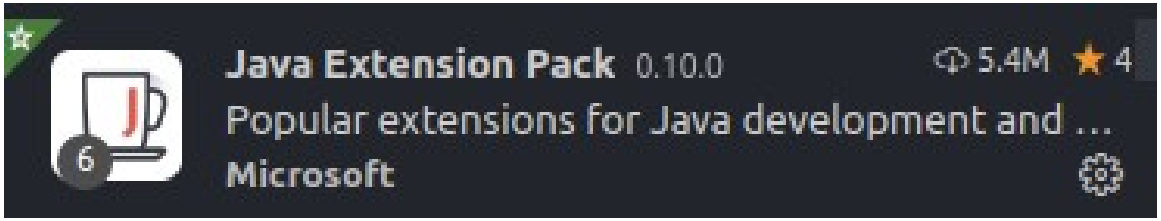
- **Visual Studio Code**
  - Nos permite, además, seleccionar las dependencias básicas



Tras crear el proyecto, podremos añadir nuevas dependencias al pom.xml de forma manual

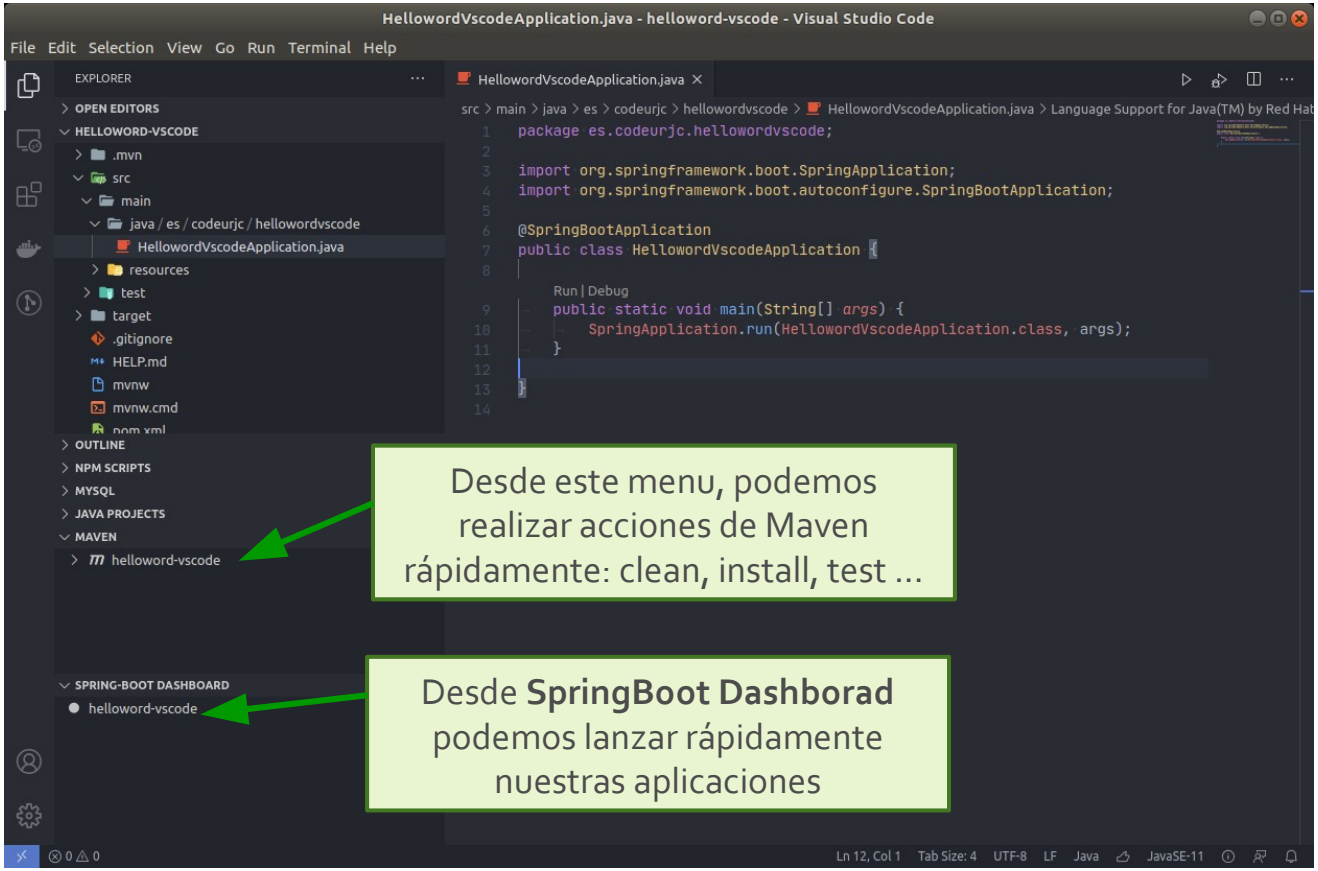
Crear proyecto Spring

- **Visual Studio Code**
  - El pack de extensiones incluye numerosas funcionalidades, como **Spring Dashboard**, que nos permite visualizar las aplicaciones en el entorno y lanzarlas de forma sencilla.
  - Es muy recomendable tener instalada la extensión de ***Java Extension Pack*** (desde esta extensión, también pueden instalarse las de Spring Boot).



Crear proyecto Spring

- Visual Studio Code



code
URJC

Desarrollo de Aplicaciones Web

Tema 1.2 – Aplicaciones Web con Spring

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Índice de Ejemplos

code URJC

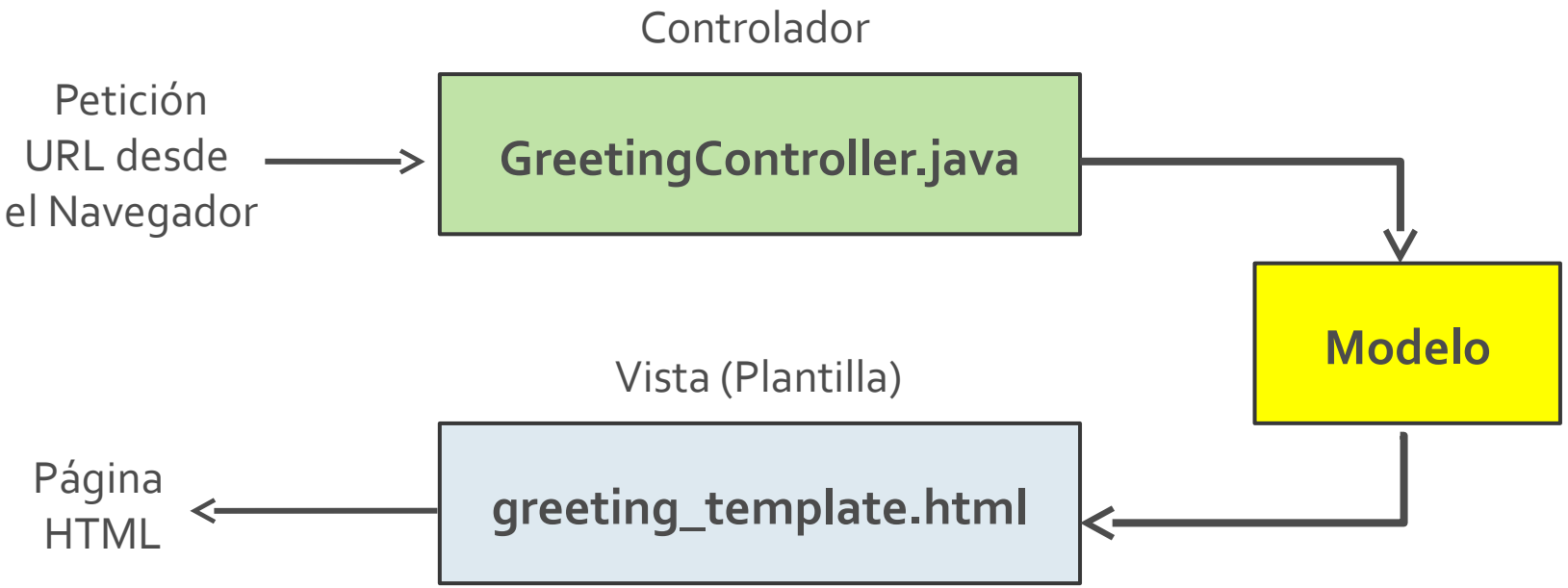
- **Ejemplo 1 (Aplicación básica Spring)**
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Spring MVC

code URJC

ejem1

- **Aplicación web básica Spring MVC**



http://spring.io/guides/gs/serving-web-content/

Spring MVC

code URJC

ejem1

- **Controlador:** Clase encargada de atender peticiones web
  - 1) **Manipulan los datos** que llegan con la petición (hacen peticiones a la BBDD, utilizan servicios externos…)
  - 2) **Obtienen los datos** que se visualizará en la página **(modelo)**
  - 3) Deciden qué **plantilla generará el HTML** partiendo de esos datos **(vista)**

Spring MVC

code URJC

ejem1

- Controlador

GreetingController.java

```
@Controller
public class GreetingController {

    @GetMapping("/greeting")
    public String greeting(Model model) {

        model.addAttribute("name", "World");

        return "greeting_template";
    }
}
```

Se indica qué **URL** debe llevar la petición para **ejecutar el controlador**

Se añade al parámetro model **la información** que será visualizada en la página web

El método devuelve el **nombre de la plantilla** que será usada para generar el HTML partiendo del modelo

Spring MVC

code URJC

ejem1

- **Vista (Plantillas, *Templates*)**
  - Las vistas en **Spring MVC** se implementan como plantillas HTML definidas en base a la información del modelo
  - Existen diversas tecnologías de plantillas: **JSP (estándar), Mustache**, Thymeleaf, FreeMarker, etc…
  - Nosotros usaremos **Mustache**

Spring MVC

code URJC

ejem1

- **Vista (Plantillas)**

En las plantillas se indican los elementos del modelo para que sean sustituidos por sus valores al generar el HTML

greeting_template.html

```
<html>
<body>
    <p>Hello, {{name}}</p>
</body>
</html>
```

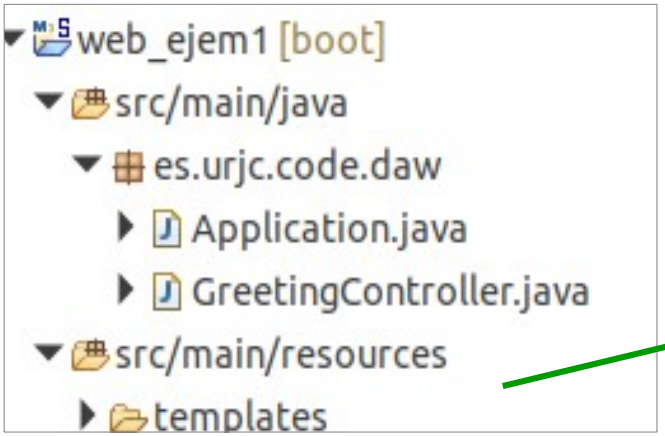
Plantilla implementada con la librería **Mustache**

https://mustache.github.io/

Spring MVC

ejem1

- Para poder usar la **extensión .html** en las plantillas lo tenemos que configurar



application.properties

```
spring.mustache.suffix=.html
```

- Si no se configura, las plantillas deberían tener la **extensión .mustache**.

Spring MVC

- **pom.xml**

Proyecto **padre** para aplicaciones **SpringBoot**

Dependencias necesarias para implementar aplicaciones web **Spring MVC** con **Mustache** y **SpringBoot**

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>

      <groupId>es.urjc.code.daw</groupId>
      <artifactId>ejem1</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <packaging>jar</packaging>

      <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>2.4.0</version>
            <relativePath/>
      </parent>

      <properties>
            <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
            <java.version>1.8</java.version>
      </properties>

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

</project>
```

Spring MVC

code URJC

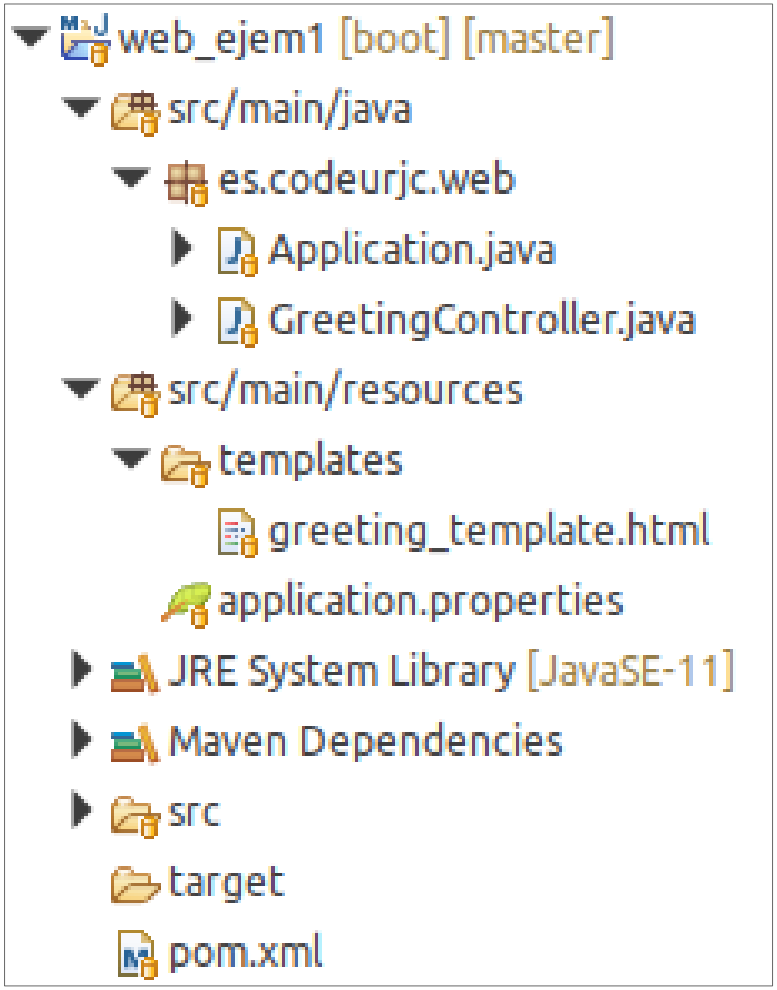
ejem1

- **Aplicación principal**
  - La aplicación se ejecuta como una **app Java normal**
  - Botón derecho proyecto > Run as… > Java Application…

```
@SpringBootApplication
public class Application {

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

Spring MVC

- Estructura de la aplicación



ejem1

Spring MVC

code URJC

ejem1

- Ejecución de la application


```
<html>
<body>
<p>Hello, World!</p>
</body>
</html>
```

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- **Ejemplo 2 (Técnicas para facilitar la depuración)**
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Spring MVC

code URJC

ejem2

- **Técnicas para facilitar la depuración**
  - Mostrar información detallada de las peticiones web en el log

application.properties

```
logging.level.org.springframework.web=DEBUG
```

Spring MVC

code URJC

ejem2

- **Técnicas para facilitar la depuración**
  - Usar el log de Spring

```
@Controller
public class SampleLogController {

  private Logger log = LoggerFactory.getLogger(SampleLogController.class);

  @GetMapping("/page_log")
  public String page(Model model) {

    log.trace("A TRACE Message");
    log.debug("A DEBUG Message");
    log.info("An INFO Message");
    log.warn("A WARN Message");
    log.error("An ERROR Message");

    return "page";
  }
}
```

Spring MVC

code URJC

ejem2

- **Técnicas para facilitar la depuración**
  - Usar el log de Spring
    - Por defecto los niveles de DEBUG y TRACE no se muestran

```
2020-11-30 02:23:08.115  INFO 76993 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet '
2020-11-30 02:23:08.116  INFO 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet
2020-11-30 02:23:08.116 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Detected StandardServletMultipartResolv
2020-11-30 02:23:08.116 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Detected AcceptHeaderLocaleResolver
2020-11-30 02:23:08.116 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Detected FixedThemeResolver
2020-11-30 02:23:08.117 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Detected org.springframework.web.servle
2020-11-30 02:23:08.118 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Detected org.springframework.web.servle
2020-11-30 02:23:08.118 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : enableLoggingRequestDetails='false': re
2020-11-30 02:23:08.118  INFO 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 2 ms
2020-11-30 02:23:08.127 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : GET "/page_log", parameters={}
2020-11-30 02:23:08.131 DEBUG 76993 --- [nio-8080-exec-1] s.w.s.m.m.a.RequestMappingHandlerMapping : Mapped to es.codeurjc.web.SampleLogCont
2020-11-30 02:23:08.144  INFO 76993 --- [nio-8080-exec-1] es.codeurjc.web.SampleLogController      : An INFO Message
2020-11-30 02:23:08.144  WARN 76993 --- [nio-8080-exec-1] es.codeurjc.web.SampleLogController      : A WARN Message
2020-11-30 02:23:08.144 ERROR 76993 --- [nio-8080-exec-1] es.codeurjc.web.SampleLogController      : An ERROR Message
2020-11-30 02:23:08.151 DEBUG 76993 --- [nio-8080-exec-1] o.s.w.s.v.ContentNegotiatingViewResolver : Selected 'text/html' given [text/html, 
2020-11-30 02:23:08.159 DEBUG 76993 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed 200 OK
```

https://www.baeldung.com/spring-boot-logging

https://docs.spring.io/spring-boot/docs/2.4.0/reference/htmlsingle/#boot-features-logging

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- **Ejemplo 2 (Generación de HTML con Mustache)**
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Generación de HTML con Mustache

code URJC

ejem2

- **Mustache**
  - Un formato de plantillas para generar contendo HTML muy sencillo
  - Existen implementaciones para diferentes lenguajes: Java, JavaScript, Ruby, C++, Rust, ASP, C…

http://mustache.github.io/

Generación de HTML con Mustache

code URJC

ejem2

- Los **controladores** generan los datos que las **plantillas** usan para generar el **HTML final**

GreetingController.java

```
@Controller
public class GreetingController {

  @GetMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name", "World");

    return "greeting_template";
  }
}
```

greeting_template.html

```
<html>
<body>
    <p>Hello, {{name}}</p>
</body>
</html>
```

Generación de HTML con Mustache

code URJC

ejem2

- Los **controladores** generan los datos que las **plantillas** usan para generar el **HTML final**

GreetingController.java

```
@Controller
public class GreetingController {

  @GetMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name", "world");

    return "greeting_template";
  }
}
```

greeting_template.html

```
<html>
<body>
    <p>Hello, {{name}}</p>
</body>
</html>
```

Generación de HTML con Mustache

ejem2

- Los **controladores** generan los datos que las **plantillas** usan para generar el **HTML final**

GreetingController.java

```
@Controller
public class GreetingController {

  @GetMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name", "World");

    return "greeting_template";
  }
}
```

greeting_template.html

```
<html>
<body>
    <p>Hello, {{name}}</p>
</body>
</html>
```

Generación de HTML con Mustache

code URJC

ejem2

- **Funcionalidades básicas**
  - Uso de atributos del modelo
  - Generación de HTML si una expresión es true
  - Generación de listas o tablas HTML con el contenido de objetos del modelo de tipo lista
  - Uso de cabecera y pie comunes a varias páginas

Generación de HTML con Mustache

ejem2

```
@GetMapping("/list_objects")
public String iterationObj(Model model) {

  List<Person> people = new ArrayList<>();
  people.add(new Person("Pepe","Pérez"));
  people.add(new Person("Juan","González"));
  people.add(new Person("Romón","Lucas"));

  model.addAttribute("people", people);

  return "list_obj_template";
}
```

```
public class Person {

    private String name;
    private String surname;

    public Person(String name,
        String surname){
        this.name = name;
        this.surname = surname;
    }

    public String getName() {
        return name;
    }

    public String getSurname() {
        return surname;
    }
}
```

```
<html>
<body>
    <p>People in list:</p>
    <ul>
    {{#people}}
        <li>{{name}} {{surname}}</li>
    {{/people}}
    </ul>
</body>
</html>
```

Se puede acceder a getter de los objetos en las iteraciones

Generación de HTML con Mustache

code URJC

ejem2

- **Cabecera y pie común en todas las plantillas**
  - Las plantillas pueden incluir el contenido de otras plantillas con `{{>plantilla}}`

page.html

```
{{>header}}

<p>Main content</p>

{{>footer}}
```

header.html

```
<html>
<body>
<h1>Welcome {{userName}}</h1>
```

footer.html

```
<p>Footer</p>
</body>
</html>
```

Generación de HTML con Mustache

code URJC

ejem2

- **Cabecera y pie común en todas las plantillas**
  - Los atributos del modelo se definen en un @ControllerAdvice con @ModelAttribute

```
@ControllerAdvice
public class DefaultModelAttributes {

    @ModelAttribute("userName")
    public String userName() {
        return "Juan";
    }
}
```

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- **Ejemplo 3 (Proceso de formularios y enlaces)**
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Proceso de formularios y enlaces

code URJC

ejem3

- **Creación de formularios en HTML**
  - La etiqueta **<form>** contiene los elementos del formulario
  - Puede contener otros elementos HTML

```
<form action='url_controlador'>

</form>
```

  - **action:** URL del controlador que será ejecutado al enviar los datos al servidor pulsando el botón de enviar (*submit*)

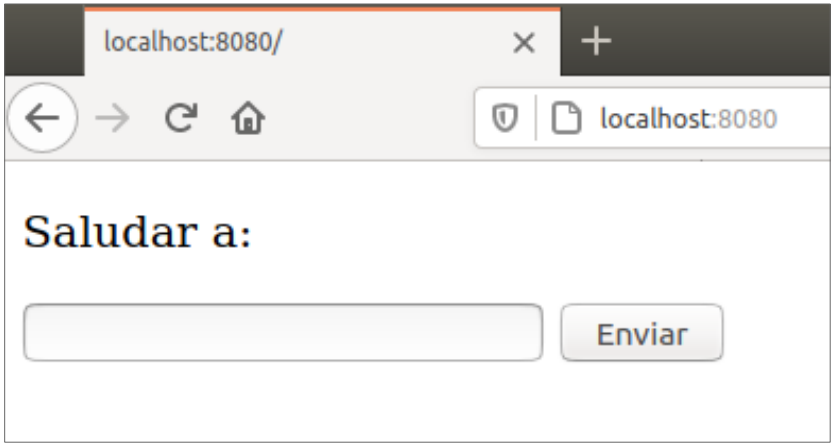
Proceso de formularios y enlaces

code URJC

ejem3

- Creación de formularios en HTML

```
<html>
<body>

<form action="greeting">

  <p>Saludar a:</p>
  <input type='text' name='userName'/>

  <input type='submit' value='Enviar'/>
</form>

</body>
</html>
```



Proceso de formularios y enlaces

code URJC

ejem3

- **Creación de formularios en HTML**
  - Tiene que existir al menos un **botón para enviar** los datos del formulario

```
<form action='url_controlador'>
  ...
  <input type='submit' value='Enviar'>
</form>
```

Botón con texto

```
<form action='url_controlador'>
  ...
  <input type='image' src='imagen.png'>
</form>
```

Botón gráfico

Proceso de formularios y enlaces

code URJC

ejem3

- **Creación de formularios en HTML**
  - Campo de texto

```
<input type='text' name='nombreParametro'>
```

  - Área de texto

```
<textarea name='nombreParametro' rows=5 cols=40>
Texto del cuadro de texto </textarea>
```

  - Casilla de verificación (checkbox)

```
<input type='checkbox' name='nombreParametro'
value='valorOpcion'>Texto Opción
```

Proceso de formularios y enlaces

code URJC

ejem3

- **Acceso a los datos desde el controlador**
  - Los valores se recogen con parámetros del método del controlador con anotaciones **@RequestParam**

src/main/resources/static/index.html

```
<form action="greeting">
  <p>Saludar a :</p>
  <input type='text' name='userName'/>
  <input type='submit' value='Enviar'/>
</form>
```

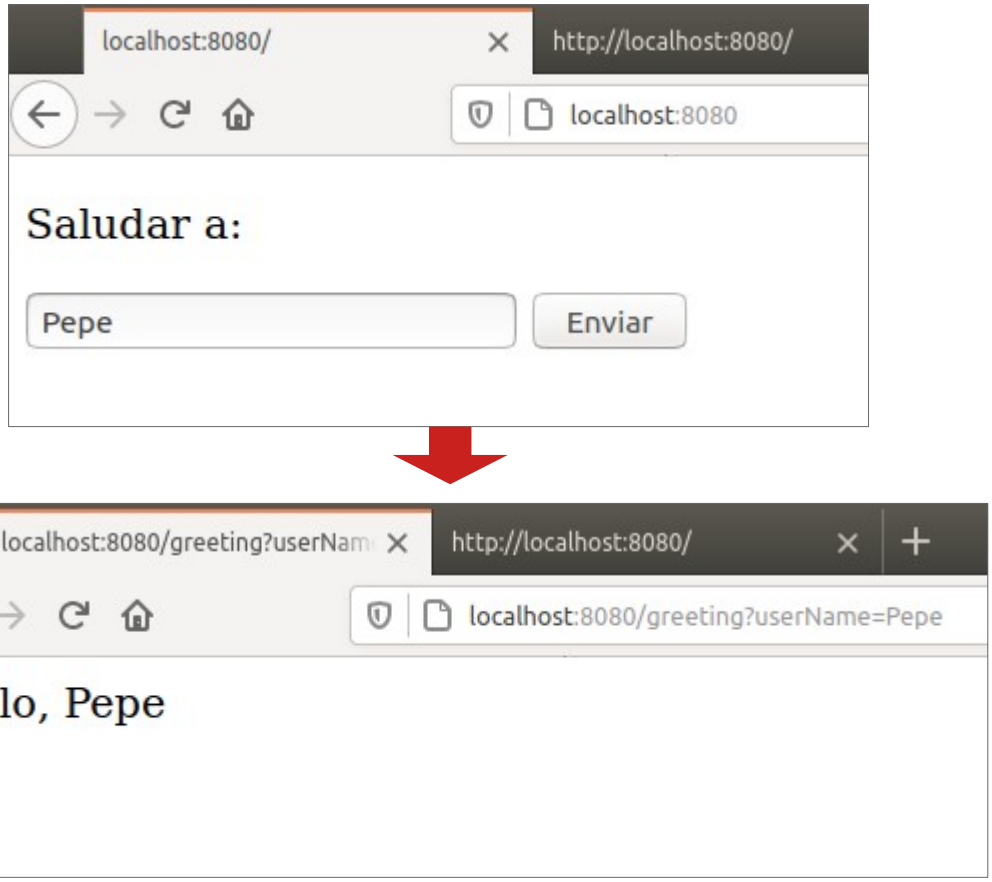
Parámetro con el valor del campo de texto del formulario

```
@RequestMapping("/greeting")
public String greeting(Model model, @RequestParam String userName) {

  model.addAttribute("name", userName);

  return "greeting_template";
}
```

Proceso de formularios y enlaces

ejem3

- Acceso a los datos desde el controlador



Índice de Ejemplos

code URJC

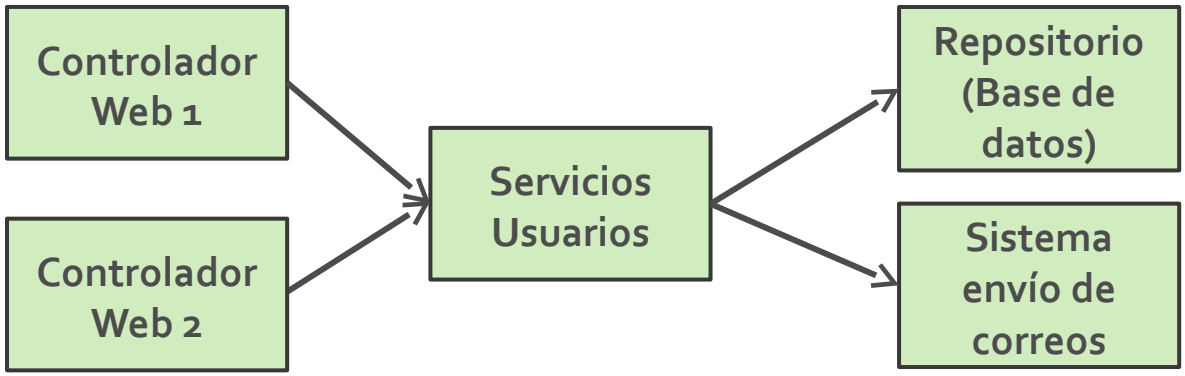
- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- **Ejemplo 4 (Inyección de dependencias)**
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Inyección de dependencias

code URJC

ejem4

- Las aplicaciones se suelen dividir en **módulos de alto nivel**
- Algunos **módulos** ofrecen servicios a otros módulos
- **Ejemplo:** Diseño modular de una aplicación web con SpringMVC



Inyección de dependencias

code URJC

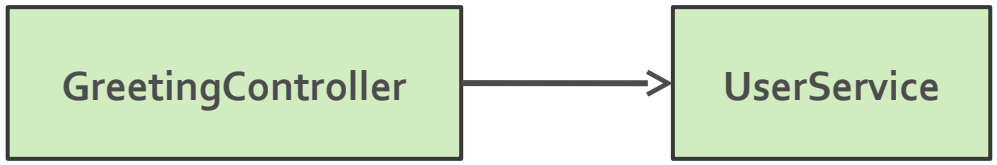
ejem4

- ¿Cómo se **implementa un módulo**?
- ¿Cómo se **conecta un módulo** a otro módulo?
- La **inyección de dependencias** es una técnica que permite especificar un módulo y sus dependencias
- Cuando se inicia la aplicación, el framework crea todos los módulos e **inyecta las dependencias** en los módulos que las necesitan
- **Spring** dispone de un sistema de inyección de dependencias interno

Inyección de dependencias

code URJC

ejem4



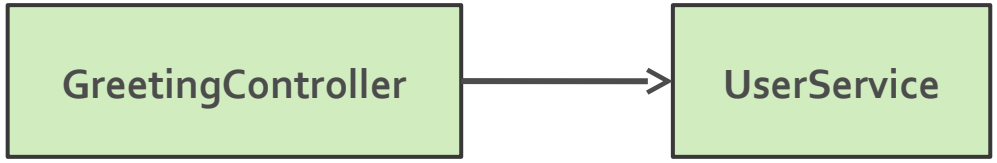

```
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @GetMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name",
       usersService.getNumUsers()+" users");

    return "greeting_template";
  }
}
```

```
@Component
public class UsersService {

  public int getNumUsers(){
    return 5;
  }
}
```

Inyección de dependencias

code URJC

ejem4




```java
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @GetMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name",
       usersService.getNumUsers()+" users");

    return "greeting_template";
  }
}
```

```java
@Component
public class UsersService {

  public int getNumUsers(){
    return 5;
  }
}
```

Inyección de dependencias

code URJC

ejem4

- A los módulos de la aplicación Spring se los denomina **beans** o **componentes**
- Para que una clase se considere un componente, tiene que anotarse con **@Component @Controller o @Service**
- Si un componente depende otro, puede poner la anotación **@Autowired** (auto enlazado) en un atributo, un constructor o un método setter

https://docs.spring.io/spring-framework/docs/5.3.0/reference/html/core.html#beans

Inyección de dependencias

code URJC

ejem4

```
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @GetMapping("/greeting")
  public String greeting(Model model) { ... }
}
```

```
@Controller
public class GreetingController {

  private UsersService usersService;

  public GreetingController(UsersService usersService){
    this.usersService = usersService;
  }

  @GetMapping("/greeting")
  public String greeting(Model model) { ... }
}
```

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- **Ejemplo 5 (Inyección de dependencias)**
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Inyección de dependencias

code URJC

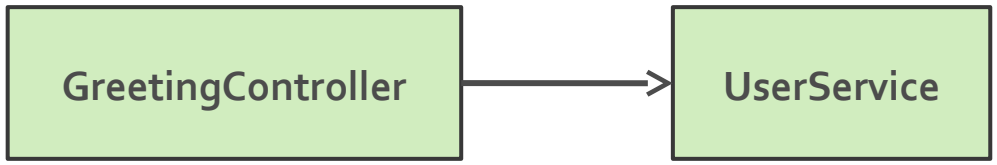
ejem5

- En las aplicaciones **SpringBoot**, la clase principal de la aplicación se utiliza para **configurar los componentes**
- Por cada componente que se quiera **configurar**:
  - Se **quita la anotación** @Component del componente
  - Se **añade un método** en la clase principal anotado con **@Bean** que devuelva un nuevo componente con la configuración requerida

Inyección de dependencias

code URJC

ejem5




```
@Controller
public class GreetingController {

  @Autowired
  private UsersService usersService;

  @GetMapping("/greeting")
  public String greeting(Model model) {

    model.addAttribute("name",
       usersService.getNumUsers()+" users");

    return "greeting_template";
  }
}
```

```
public class UserService {

  private int numUsers;

  public UserService(int numUsers){
    this.numUsers = numUsers;
  }

  public int getNumUsers() {
    return numUsers;
  }
}
```

Inyección de dependencias

code URJC

ejem5

En la clase de la aplicación se configura el componente

Se implementa un método anotado con **@Bean** que devuelve el componente ya configurado

```
@SpringBootApplication
public class Application {

    @Bean
    public UserService usersService(){
        return new UserService(10);
    }

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

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- **Ejemplo 6 (Objeto HttpSession)**
- Ejemplo 7 (Componente especifico para cada usuario)
- Ejemplo 8 (Imágenes)

Objeto HttpSession

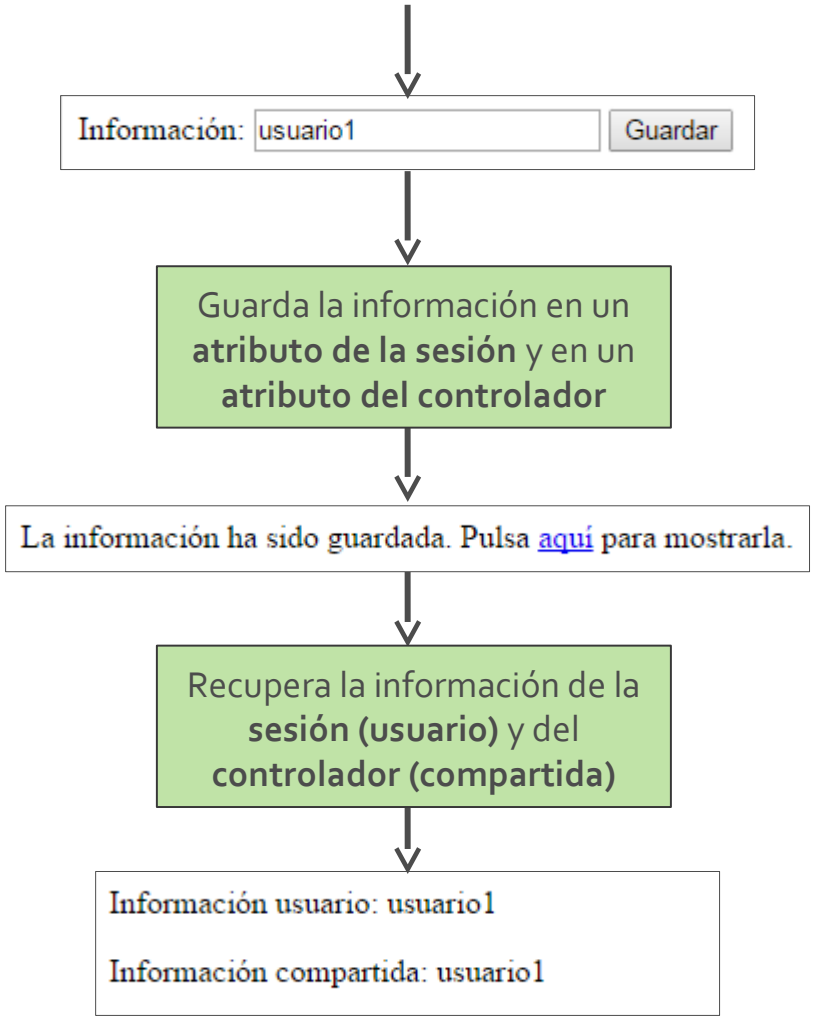
- La sesión se representa como un objeto del interfaz **javax.servlet.http.HttpSession**
- El framework Spring es el encargado de crear un **objeto de la sesión** diferente para cada **usuario**
- Para acceder al **objeto de la sesión** del usuario que está haciendo una petición, basta incluirlo como parámetro en el método del controlador

```
@GetMapping("/ruta_controlador")
public String procesarFormulario(HttpSession sesion, ...) {
    Object info = ...;
    sesion.setAttribute("info", info);
    return "template";
}
```

Objeto HttpSession

ejem6

- Ejemplo



Objeto HttpSession

code URJC

ejem6

- **Ejemplo**
  - Una aplicación recoge la información de un formulario y la guarda en dos lugares:
    - **Atributo del controlador (compartida)**
    - **Atributo de la sesión (usuario).**
  - Una vez guardada la información, se puede acceder a ella y generar una página
  - Si **dos usuarios** visitan esta página a la **misma vez**, se puede ver cómo la información del controlador es compartida (la que guarda el último usuario es la que se muestra), pero la que se guarda en la sesión es diferente para cada usuario
  - Para simular dos usuarios en el mismo ordenador, se puede usar el **modo normal** y el **modo incógnito** de Google Chrome.

Objeto HttpSession

ejem6

```
@Controller
public class SesionController {

	private String infoCompartida;

	@PostMapping("/procesarFormulario")
	public String procesarFormulario(@RequestParam String info, HttpSession sesion) {

		sesion.setAttribute("infoUsuario", info);
		infoCompartida = info;

		return "resultado_formulario";
	}

	@GetMapping("/mostrarDatos")
	public String mostrarDatos(Model model, HttpSession sesion) {

		String infoUsuario = (String) sesion.getAttribute("infoUsuario");

		model.addAttribute("infoUsuario", infoUsuario);
		model.addAttribute("infoCompartida", infoCompartida);

		return "datos";
	}
}
```

HttpSession
como parámetro

Objeto HttpSession

code URJC

ejem6

- **Métodos de HttpSession**
  - **void setAttribute(String name, Object value):** Asocia un objeto a la sesión identificado por un nombre
  - **Object getAttribute(String name):** Recupera un objeto previamente asociado a la sesión
  - **boolean isNew():** Indica si es la primera página que solicita el usuario. Si la sesión es nueva.
  - **void invalidate():** Cierra la sesión del usuario borrando todos sus datos. Si visita nuevamente la página, será considerado como un usuario nuevo.
  - **void setMaxInactiveInterval(int segundos):** Configura el tiempo de inactividad para cerrar automáticamente la sesión del usuario.

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- **Ejemplo 7 (Componente especifico para cada usuario)**
- Ejemplo 8 (Imágenes)

Componente específico para cada usuario

code URJC

ejem7

- En Spring existe una forma de más **alto nivel** para asociar información al **usuario**
- Consiste en crear un **@Component** especial que se asociará a cada usuario y hacer **@Autowire** del mismo en el controlador que se utilice
- Internamente Spring hace bastante **magia** para que la información se gestione de forma adecuada

Componente específico para cada usuario

ejem7

- Componente para el usuario

```
@Component
@SessionScope
public class Usuario {

    private String info;

    public void setInfo(String info) {
        this.info = info;
    }

    public String getInfo() {
        return info;
    }
}
```

La anotación **@SessionScope** hace que haya **una instancia del componente** por cada usuario

Componente específico para cada usuario

code URJC

ejem7

```
@Controller
public class SesionController {

	@Autowired
	private Usuario usuario;

	private String infoCompartida;

	@PostMapping("/procesarFormulario")
	public String procesarFormulario(@RequestParam String info) {

		usuario.setInfo(info);
		infoCompartida = info;

		return "resultado_formulario";
	}

	@GetMapping("/mostrarDatos")
	public String mostrarDatos(Model model) {

		String infoUsuario = usuario.getInfo();

		model.addAttribute("infoUsuario", infoUsuario);
		model.addAttribute("infoCompartida", infoCompartida);

		return "datos";
	}
}
```

Se accede al objeto usuario con **@Autowired** (inyección de dependencias)

Se utilizan métodos del objeto

Índice de Ejemplos

code URJC

- Ejemplo 1 (Aplicación básica Spring)
- Ejemplo 2 (Técnicas para facilitar la depuración)
- Ejemplo 2 (Generación de HTML con Mustache)
- Ejemplo 3 (Proceso de formularios y enlaces)
- Ejemplo 4 (Inyección de dependencias)
- Ejemplo 5 (Inyección de dependencias)
- Ejemplo 6 (Objeto HttpSession)
- Ejemplo 7 (Componente especifico para cada usuario)
- **Ejemplo 8 (Imágenes)**

Imágenes

code URJC

ejem8

- En las aplicaciones web el usuario puede **subir imágenes** y cualquier otro tipo de fichero
- Esas imágenes se **guardan en el disco** duro (fuera de la carpeta static) o en la **base de datos**
- La **carpeta static** no está disponible cuando la aplicación está en producción (fuera del IDE)

Imágenes

code URJC

ejem8

- Formulario para subir imágenes

```
<form action="/upload_image" method="post" enctype="multipart/form-data">

  <p>Image name: </p>
  <input type='text' name='imageName'/>

  <p>Image file:</p><input type='file' name='image' accept=".jpg, .jpeg"/>

  <input type='submit' value='Save'/>
</form>
```

Imágenes

code URJC

ejem8

- Controlador para subir imágenes

```
@PostMapping("/upload_image")
public String uploadImage(@RequestParam String imageName,
    @RequestParam MultipartFile image) throws IOException {

    this.imageName = imageName;

    Files.createDirectories(IMAGES_FOLDER);

    Path imagePath = IMAGES_FOLDER.resolve("image.jpg");

    image.transferTo(imagePath);

    return "uploaded_image";
}
```

Imágenes

code URJC

ejem8

- Plantilla para mostrar la imagen

```
<html>
<body>

<h1>{{imageName}}</h1>

<img src="/download_image"></img>

</body>
</html>
```

Imágenes

code URJC

ejem8

- Controlador para mostrar la imagen

```
@GetMapping("/image")
public String viewImage(Model model) {

     model.addAttribute("imageName", imageName);

     return "view_image";
}

@GetMapping("/download_image")
public ResponseEntity<Object> downloadImage(Model model)
     throws MalformedURLException {

     Path imagePath = IMAGES_FOLDER.resolve("image.jpg");

     Resource image = new UrlResource(imagePath.toUri());

     return ResponseEntity.ok()
          .header(HttpHeaders.CONTENT_TYPE, "image/jpeg")
          .body(image);
}
```

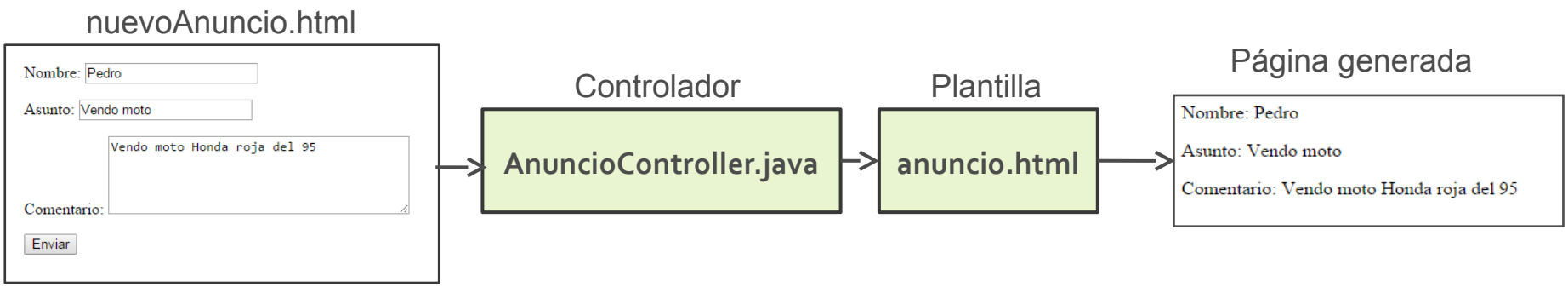
Índice de Ejercicios

Índice de Ejercicios

Ejercicio 1

- Crear una página **html estática** que muestre un **formulario** para enviar al servidor información de nuevos anuncios.
- En el formulario debe aparecer:
  - Campo de texto para el **nombre** del usuario
  - Campo de texto para el **asunto** del mensaje
  - Área de texto para el **cuerpo** del mensaje
  - **Botón de envío**
- Implementar un **controlador** que sea ejecutado al pulsar el botón de envío del formulario y recoja los datos del formulario
- Diseñar una **plantilla** que muestre el anuncio que se ha enviado al servidor

Ejercicio 1


nuevoAnuncio.html
Nombre: Pedro
Asunto: Vendo moto
Vendo moto Honda roja del 95
Comentario:
Enviar
Controlador
AnuncioController.java
Plantilla
anuncio.html
Página generada
Nombre: Pedro
Asunto: Vendo moto
Comentario: Vendo moto Honda roja del 95

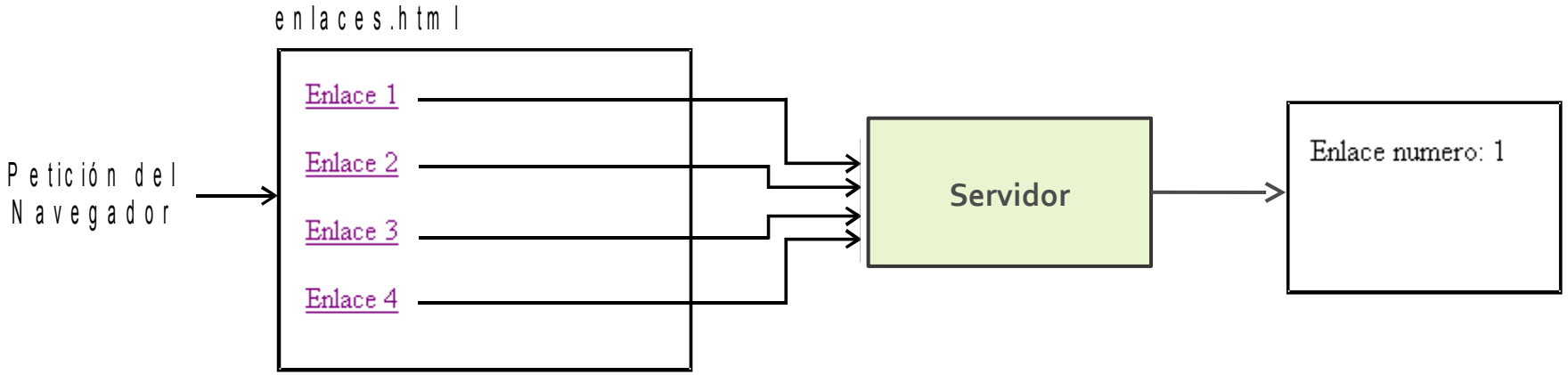
Índice de Ejercicios

Ejercicio 2

code URJC

- Crear una página **html** con cuatro enlaces
- Todos los enlaces hacen referencia a un mismo controlador
- En la URL de cada enlace incluimos un parámetro llamado "**nenlace**" con valores 1,2,3 y 4
- Implementar un **controlador** que sea llamado al pulsar cualquiera de los enlaces
- Diseñar una **plantilla** que muestre el número del enlace que ha sido pulsado

Ejercicio 2


enlaces.html
Enlace 1
Enlace 2
Enlace 3
Enlace 4
Petición del Navegador
Servidor
Enlace numero: 1

Índice de Ejercicios

Ejercicio 3

- Implementa la misma funcionalidad que el ejercicio 2 pero incluye la información en la propia URL en vez de cómo parámetros

Índice de Ejercicios

Ejercicio 4

- Crear una aplicación web para gestionar un tablón de anuncios con varias páginas
- La página principal muestra los anuncios existentes (sólo nombre y asunto) y un enlace para insertar un nuevo anuncio
- Si pulsamos en la cabecera de un anuncio se navegará a una página nueva que muestre el contenido completo del anuncio

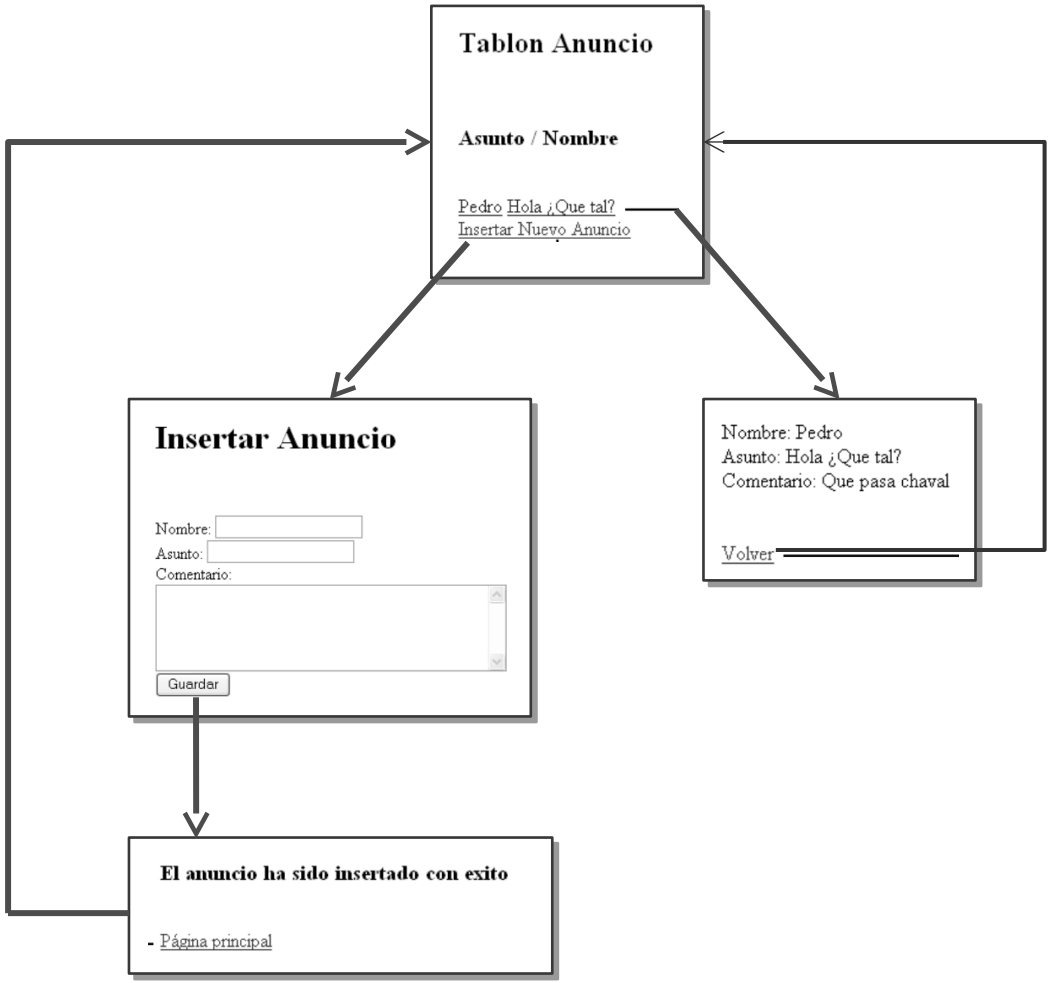
Ejercicio 4

- Si se pulsa el enlace para añadir el anuncio se navegará a una nueva página que contenga un formulario
- Al enviar el formulario se guardará el nuevo anuncio y se mostrará una página indicando que se ha insertado correctamente y un enlace para volver
- En la página del anuncio se podrá borrar

Ejercicio 4

- **Implementación**
  - Se recomienda usar un único controlador con varios métodos (cada uno atendiendo una URL diferente)
  - El controlador tendrá como atributo una lista de objetos Post
  - Ese atributo será usado desde los diferentes métodos

Ejercicio 4


Tablon Anuncio
Asunto / Nombre
Pedro Hola ¿Que tal?
Insertar Nuevo Anuncio
Insertar Anuncio
Nombre:
Asunto:
Comentario:
Guardar
Nombre: Pedro
Asunto: Hola ¿Que tal?
Comentario: Que pasa chaval
Volver
El anuncio ha sido insertado con exito
- Página principal

Índice de Ejercicios

Ejercicio 5

code URJC

- Estructura el tablón **de mensajes** para que el código esté separado en dos clases principales (*beans*):
  - **PostController:** Gestión de peticiones web
  - **PostService:** Gestión de los mensajes
- Implementa una gestión de alumnos que evite problemas de accesos simultáneos (dos usuarios quieren borrar el mismo mensaje)

Índice de Ejercicios

Ejercicio 6

- Modificar el Tablón de mensajes (ejercicio 5) para que la primera vez que un usuario acceda a la página principal le salga un mensaje de Bienvenida (creación de la sesión)
- En las siguientes visitas a la página principal no tiene que aparecer el mensaje

Ejercicio 6

- Cuando el usuario cree un anuncio por primera vez en la sesión, introducirá su nombre.
- Cuando vaya a crear más anuncios durante la sesión, el nombre le debe aparecer ya escrito (aunque con la posibilidad de modificarlo)
- Además, cada vez que vaya a incluir un anuncio se le debe indicar cuántos anuncios que lleva creados en la sesión

Índice de Ejercicios

Ejercicio 7

code URJC

- Añade imágenes al tablón de mensajes
- Sólo habrá una imagen por anuncio
- Se puede usar el id del anuncio como parte del nombre del fichero de la imagen para evitar colisiones y facilitar el acceso

01-2023

code
URJC

Desarrollo de Aplicaciones Web

Tema 2 – APIs REST con Spring

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Índice de Ejemplos

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Clientes de APIs REST

code URJC

front-ejem1

- **Cliente JavaScript**
  - Las aplicaciones web con **AJAX** o con arquitectura **SPA,** implementadas con **JavaScript**, usan servicios **REST** desde el **navegador**
  - Se pueden usar APIs REST usando la **API estándar** del browser o **librerías externas (jQuery)**

Clientes de APIs REST

code URJC

front-ejem1

- **Cliente JavaScript: jQuery**
  - Muestra en la consola el resultado de la API REST

script.js

```
$(document).ready(function(){

  $.ajax({
    url:"https://www.googleapis.com/books/v1/volumes?q=intitle:java"
  }).done(function(data) {
    console.log(data);
  });

});
```

http://api.jquery.com/jquery.ajax/

Clientes de APIs REST

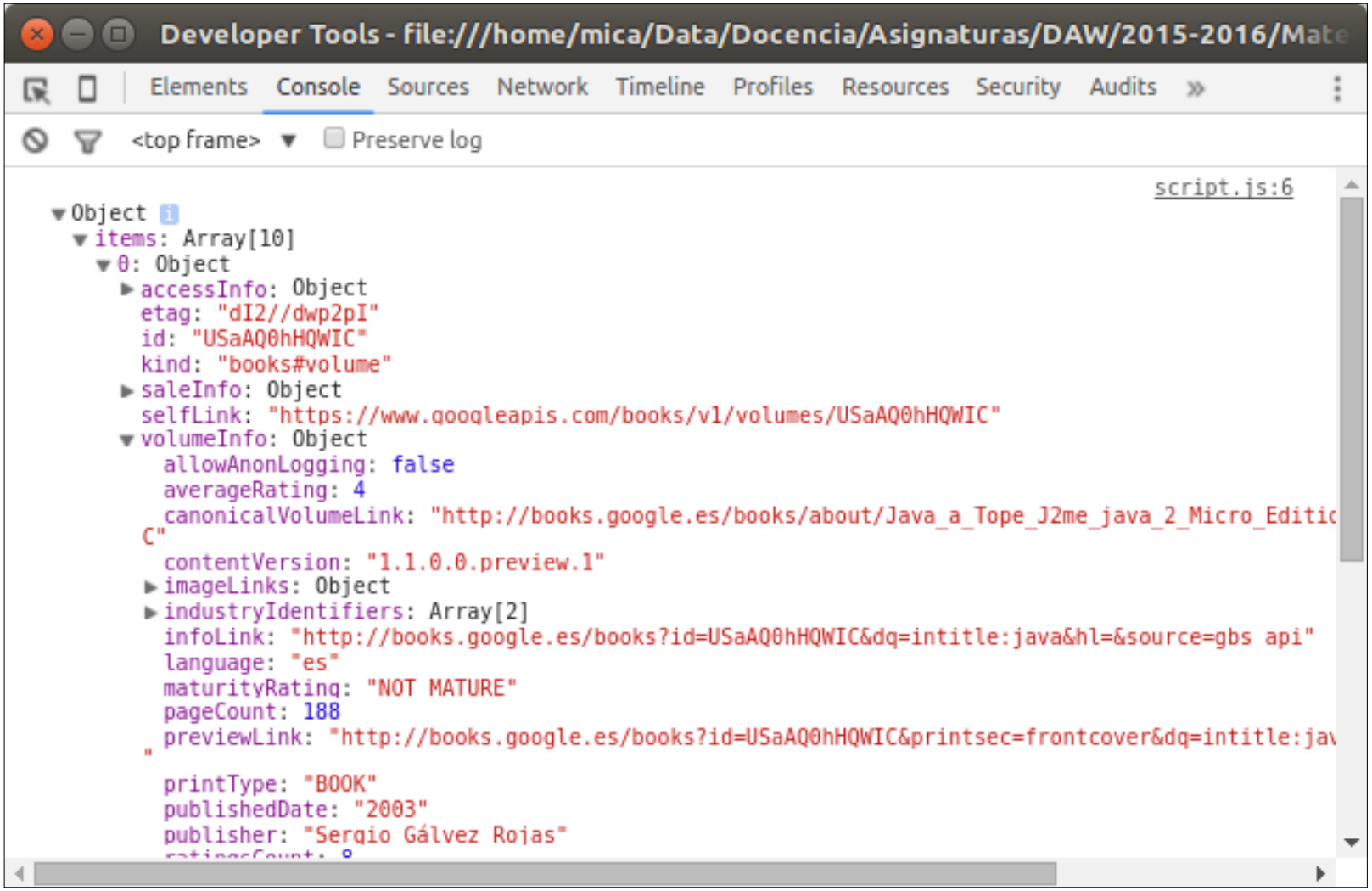
code URJC

- Cliente JavaScript: jQuery

front-ejem1

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js">
    </script>
    <script src="script.js"></script>
  </head>
  <body>
  </body>
</html>
```

Clientes de APIs REST



Índice de Ejemplos

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Clientes de APIs REST

code URJC

front-ejem2

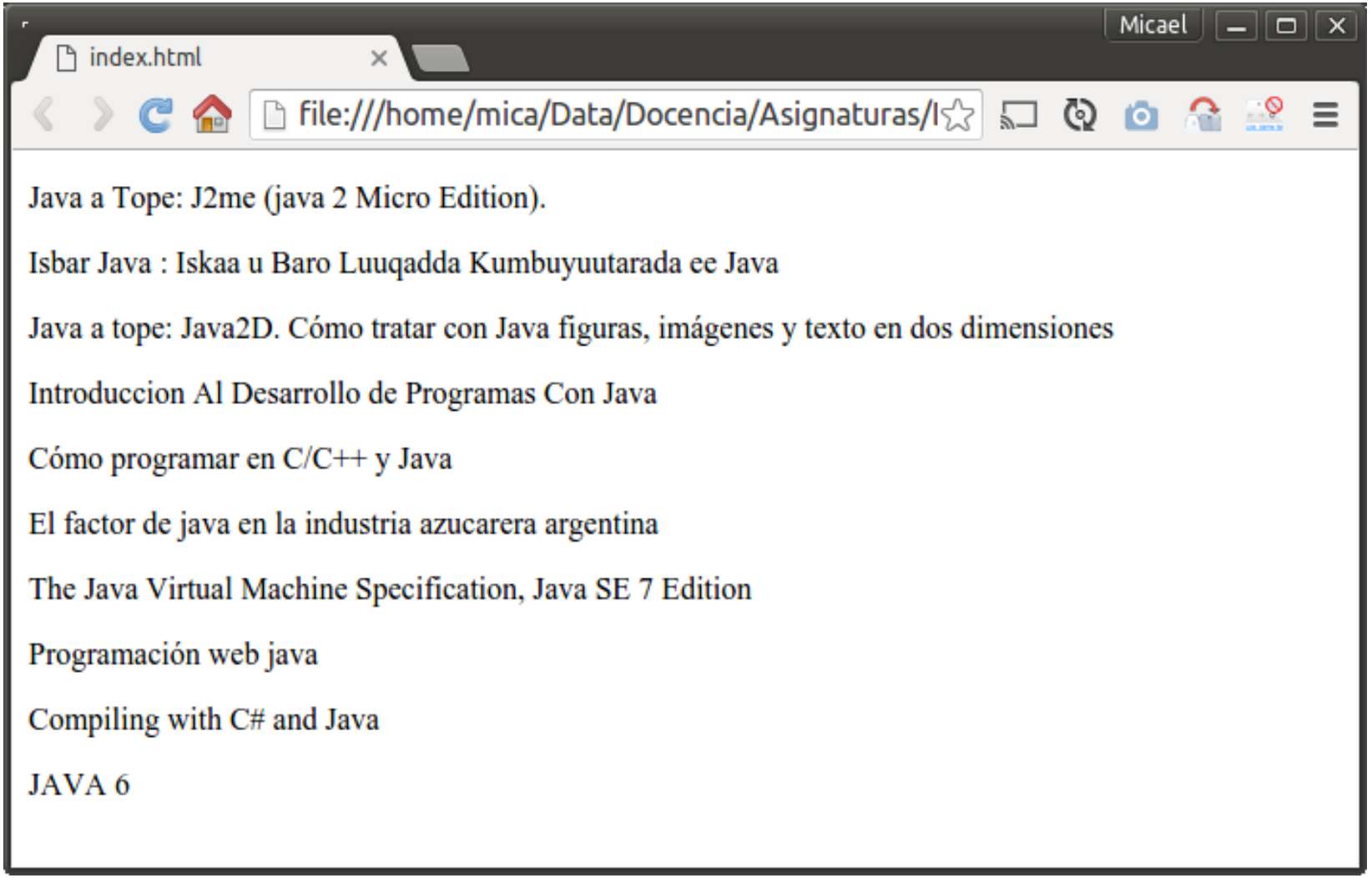
- **Cliente JavaScript: jQuery**
  - Muestra en la página los títulos de los libros

script.js

```
$(document).ready(function(){
  $.ajax({
    url:"https://www.googleapis.com/books/v1/volumes?q=intitle:java"
  }).done(function(data) {
    for(var i=0; i<data.items.length; i++){
      $("body").append(
        "<p>"+data.items[i].volumeInfo.title+"</p>");
    }
  });
});
```

http://api.jquery.com/jquery.ajax/

Clientes de APIs REST



Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- **Ejemplo 1 (Clientes API REST)**
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Clientes de APIs REST

code URJC

ejem1

- **API REST Posts**
  - Ofrece de operaciones de consulta, creacion, modificación y borrado
  - Las propiedades de un post son: user, title y text
  - La API rest está implementada en el proyecto **ejem1**

Clientes de APIs REST

code URJC

ejem1

- **API REST Posts**
  - Consulta de posts
    - Method: GET
    - URL: http://127.0.0.1:8080/posts/
    - Result:

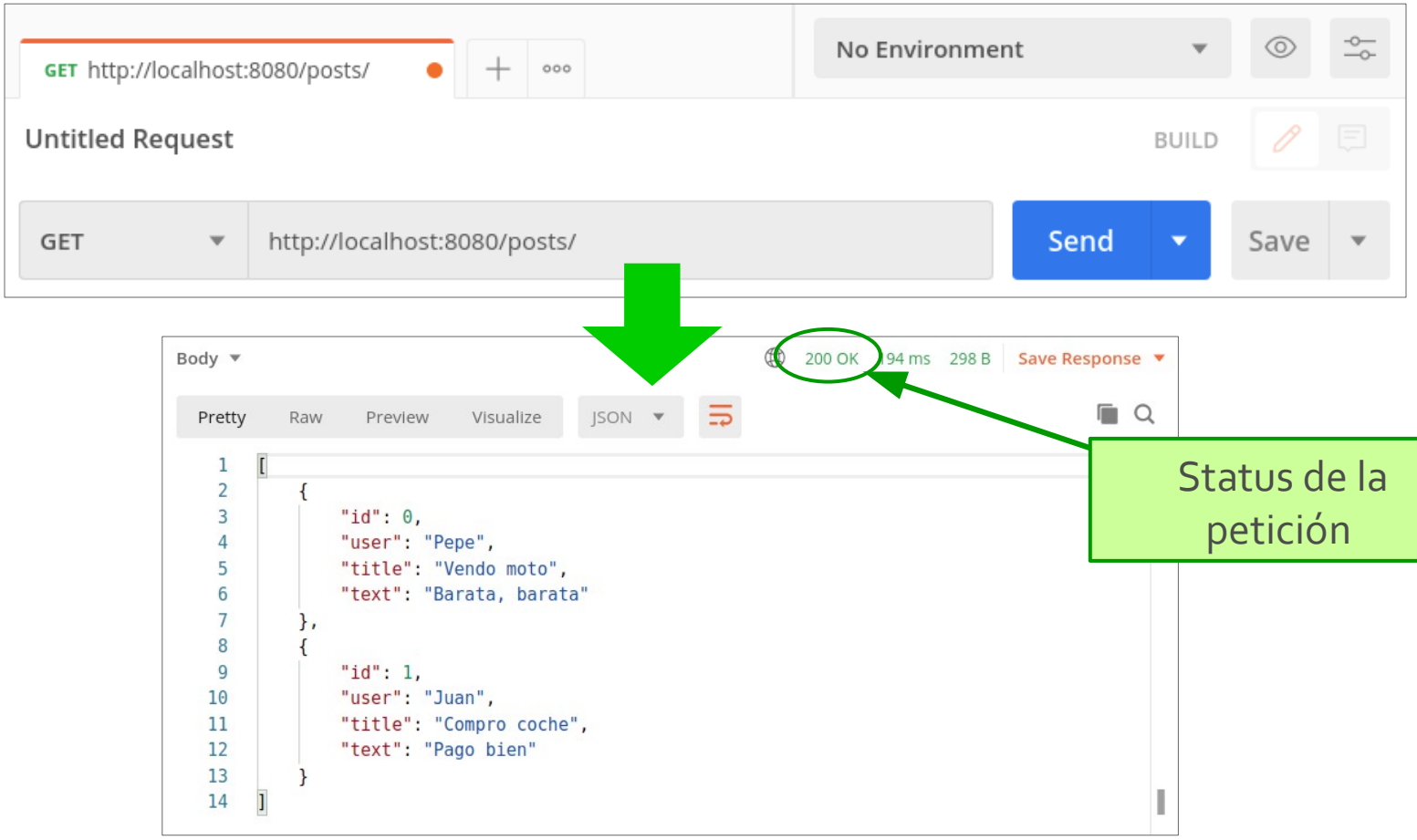
```
[
  { "id": 0, "user": "Pepe", "title": "Vendo moto", "text": "Barata, barata"},
  { "id": 1, "user": "Juan", "title": "Compro coche", "text": "Pago bien"}
]
```

    - Status code: 200 (OK)

Clientes de APIs REST

ejem1

- GET /posts/



Clientes de APIs REST

code URJC

ejem1

- **API REST Posts**
  - Consulta de un post concreto
    - Method: GET
    - URL: http://127.0.0.1:8080/posts/1
    - Result:

```
{ "id": 1, "user": "Pepe", "title": "Vendo moto", "text": "Barata, barata"}
```

    - Status code: 200 (OK)

Clientes de APIs REST

code URJC

ejem1

- **API REST Posts**
  - Creación de un post
    - Method: POST
    - URL: http://127.0.0.1:8080/posts/
    - Headers: Content-Type: application/json
    - Body:

```
{ "user": "Alberto", "title": "Cambio bici", "text": "MTB por carretera"}
```

    - Result:

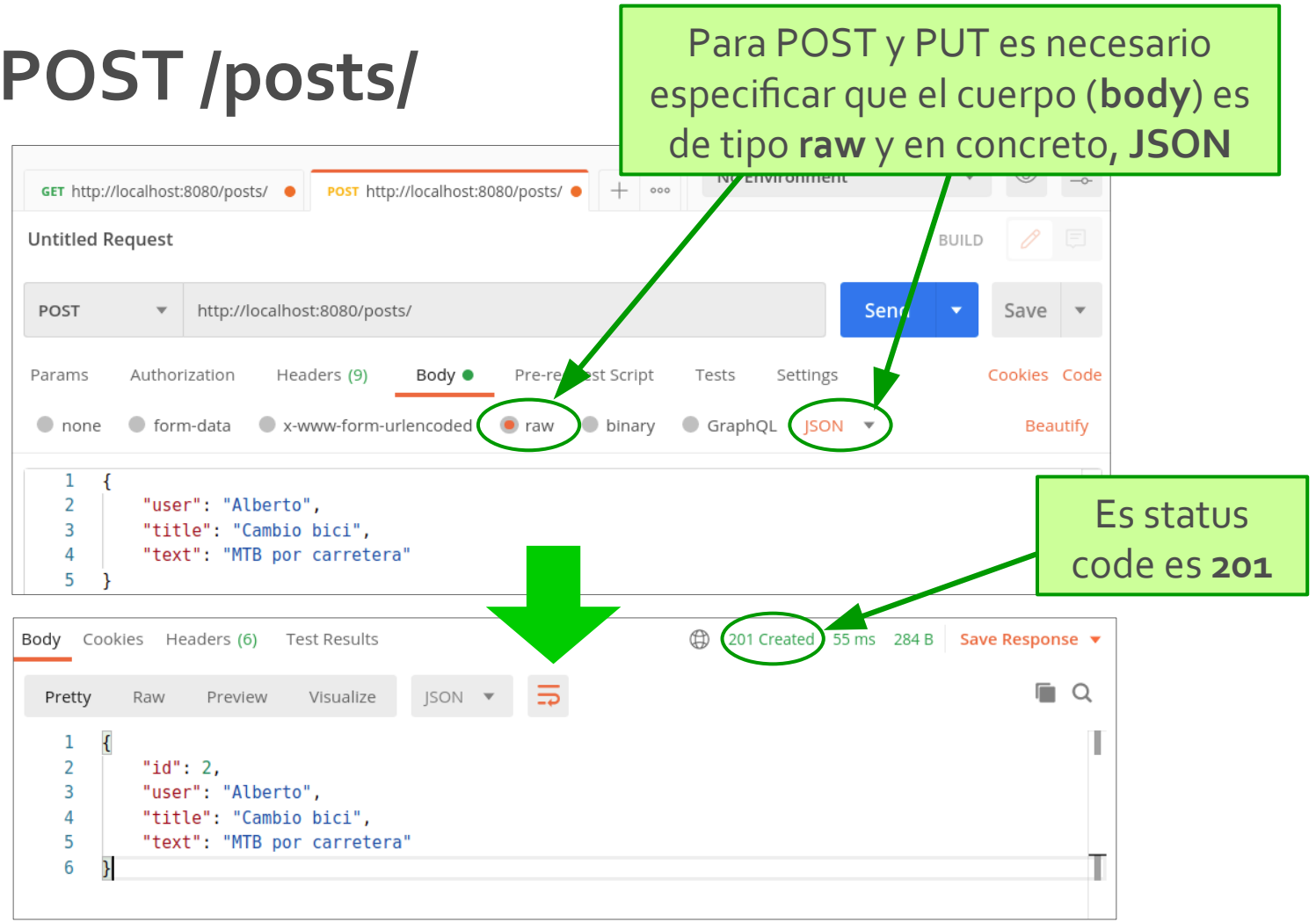
```
{"id":2, "user": "Alberto", "title": "Cambio bici", "text": "MTB por carretera"}
```

    - Status code: 201 (Created)
    - Headers: Location: http://127.0.0.1:8080/posts/2

Clientes de APIs REST

ejem1

- POST /posts/



Clientes de APIs REST

code URJC

ejem1

- **API REST Posts**
  - Reemplazo de posts
    - Method: PUT
    - URL: http://127.0.0.1:8080/posts/0
    - Headers: Content-Type: application/json
    - Body:

```
{ "id": 0, "user": "Pepe", "title": "Vendo moto", "text": "Regalada"}
```

    - Result:

```
{ "id": 0, "user": "Pepe", "title": "Vendo moto", "text": "Regalada"}
```

    - Status code: 200 (OK)

Clientes de APIs REST

code URJC

ejem1

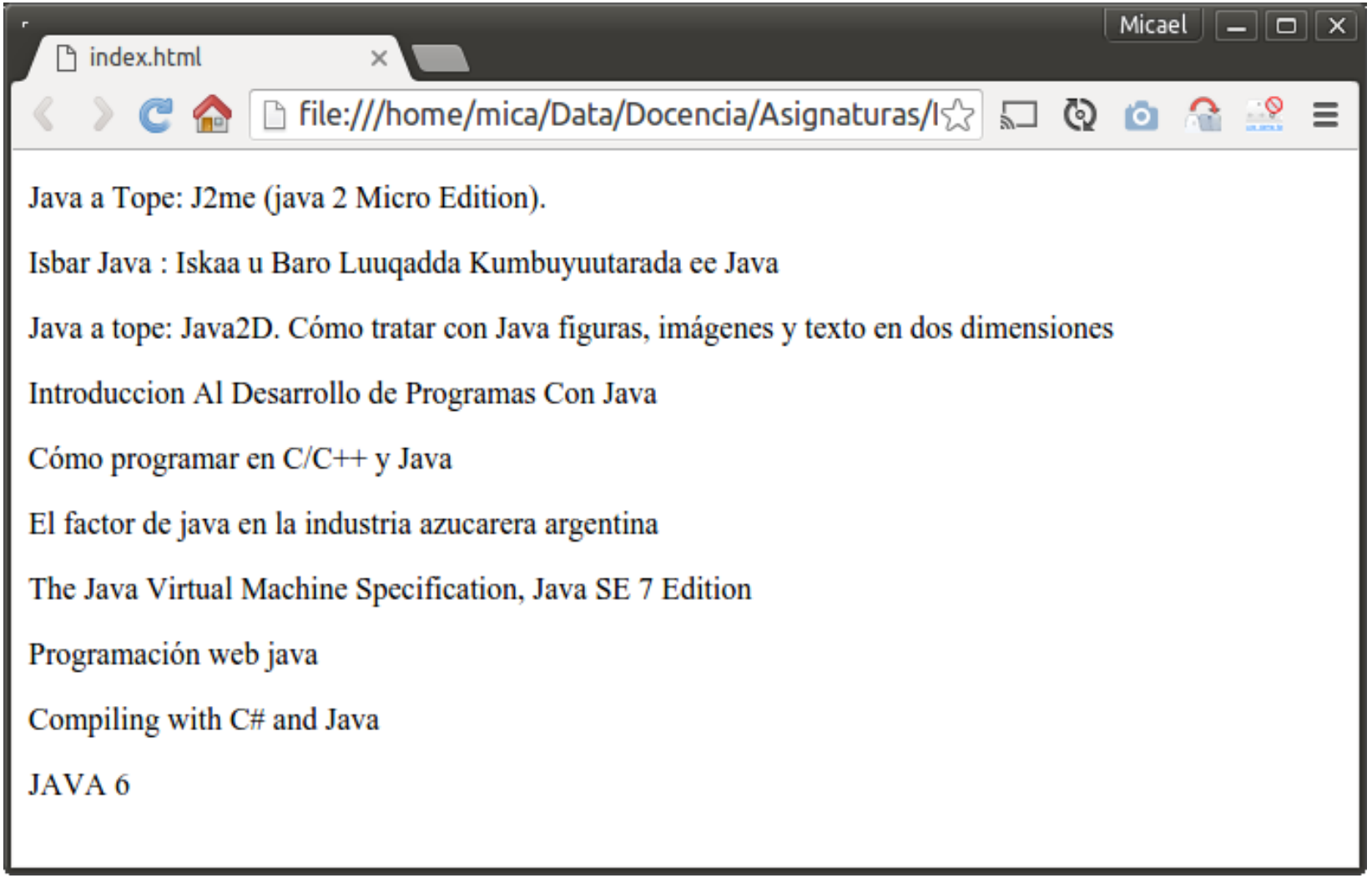
- **API REST Posts**
  - Borrado de posts
    - Method: DELETE
    - URL: http://127.0.0.1:8080/posts/1
    - Result:

```
{ "id": 0, "user": "Pepe", "title": "Vendo moto", "text": "Regalada"}
```

    - Status code: 200 (OK)

https://stackoverflow.com/questions/6439416/deleting-a-resource-using-http-delete

Clientes de APIs REST



Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- **Ejemplo 1 (APIs REST con Spring)**
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

APIs REST con Spring

code URJC

ejem1

- **Spring MVC** es una parte de Spring para la construcción de aplicaciones web
- Sigue la arquitectura **MVC (*Model View Controller*)**
- **Permite estructurar la aplicación en:**
  - **Model:** Modelos de datos (objetos Java)
  - **View:**
    - **Web**: Plantilla que genera la página HTML.
    - **REST**: Conversión del modelo a JSON (automático)
  - **Controller:** Controlador que atiende las peticiones HTTP que llegan del navegador

APIs REST con Spring

code URJC

ejem1

- **Ciclo de vida completa de un recurso**
  - **Endpoints REST**
    - Creación: POST
    - Consulta un recurso: GET
    - Consulta de varios recursos: GET
    - Borrado: DELETE
    - Actualización: PUT
  - **Gestión de recursos**
    - PostService: Mapa en memoria indexado por id

APIs REST con Spring

code URJC

ejem1

- **Aplicación principal**
  - La aplicación se ejecuta como una **app Java normal**
  - Botón derecho proyecto > Run as… > Java Application…

```
@SpringBootApplication
public class Application {

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

- **pom.xml**

Proyecto **padre** para aplicaciones **SpringBoot**

Dependencias necesarias para implementar aplicaciones web **Spring MVC** y **SpringBoot**

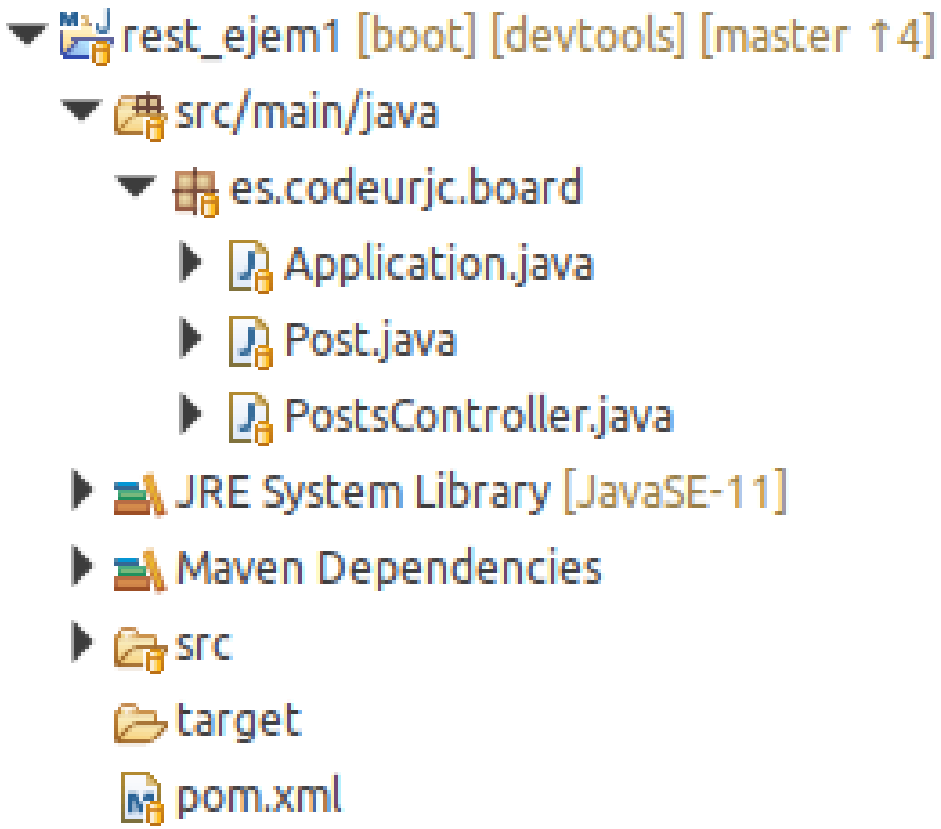
```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
      <modelVersion>4.0.0</modelVersion>

      <groupId>es.codeurjc</groupId>
      <artifactId>rest_ejer1</artifactId>
      <version>0.0.1-SNAPSHOT</version>
      <packaging>jar</packaging>

      <parent>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-parent</artifactId>
            <version>2.4.0</version>
            <relativePath/>
      </parent>

      <properties>
            <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
            <java.version>11</java.version>
      </properties>

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

</project>
```

APIs REST con Spring

code URJC

ejem1

- Estructura de la aplicación



APIs REST con Spring

code URJC

ejem1

- **Devolver un recurso concreto (GET)**
  - Con **@GetMapping** se indica que el método atiende peticiones **GET**
  - Se devuelve un array con todos los recursos
  - Si no hay recursos, el array se devuelve vacío

APIs REST con Spring

code URJC

ejem1

- **Devolver todos los recursos (GET)**

```
@RestController
public class PostController {

  @Autowired
  private PostService posts;

  @GetMapping("/posts/")
  public Collection<Post> getPosts() {
    return posts.findAll();
  }
  ...
}
```

APIs REST con Spring

code URJC

ejem1

- **Devolver un recurso concreto (GET)**
  - Con **@GetMapping** se indica que el método atiende peticiones **GET**
  - El id del recurso se condifica en la URL y se accede a él usando un **@PathVariable**
  - Si el recurso existe se devuelve, y si no, se devuelve **404 NOT FOUND**. Por eso el método devuelve un **ResponseEntity**

APIs REST con Spring

code URJC

ejem1

- Devolver un recurso concreto (GET)

El id está en la **URL** y se accede con **@PathVariable**

```
@GetMapping("/posts/{id}")
public ResponseEntity<Post> getPost(@PathVariable long id) {

    Post post = posts.findById(id);

    if (post != null) {
        return ResponseEntity.ok(post);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

Se devuelve el **objeto** o **NOT FOUND**

APIs REST con Spring

code URJC

ejem1

- **Nuevo recurso (POST)**
  - Con **@PostMapping** se indica que el método atiende peticiones **POST**
  - El cuerpo de la petición se obtiene con un parámetro anotado con **@RequestBody**
  - Se **devuelve el nuevo objeto** al cliente (con un id)

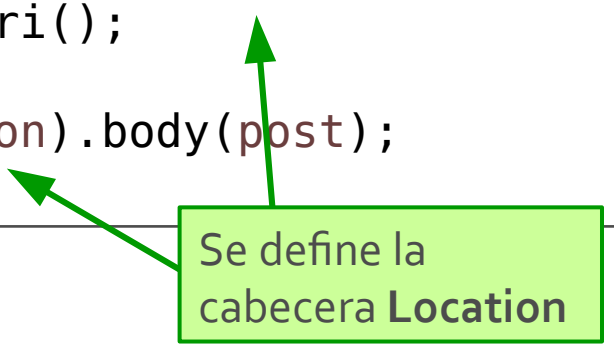
APIs REST con Spring

code URJC

ejem1

- Nuevo recurso (POST)

```
@PostMapping("/posts/")
public ResponseEntity<Post> createPost(@RequestBody Post post) {

  posts.save(post);

  URI location = fromCurrentRequest().path("/{id}")
     .buildAndExpand(post.getId()).toUri();

  return ResponseEntity.created(location).body(post);
}
```



APIs REST con Spring

code URJC

ejem1

- **Borrar un recurso (DELETE)**
  - Con **@DeleteMapping** se indica que el método atiende peticiones **DELETE**
  - El id del recurso se codifica en la URL y se accede a él usando un **@PathVariable**
  - Si el recurso existe se borra y opcionalmente se devuelve
  - Si no existe, se devuelve **404 NOT FOUND**. Por eso el método devuelve un **ResponseEntity**

APIs REST con Spring

code URJC

ejem1

- Borrar un recurso (DELETE)

```
@DeleteMapping("/posts/{id}")
public ResponseEntity<Post> deletePost(@PathVariable long id) {

    Post post = posts.findById(id);

    if (post != null) {
        posts.deleteById(id);
        return ResponseEntity.ok(post);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

APIs REST con Spring

code URJC

ejem1

- **Reemplazar un recurso (PUT)**
  - Con **@PutMapping** se indica que el método atiende peticiones **PUT**
  - El id del recurso se condifica en la URL y se accede a él usando un **@PathVariable**
  - El nuevo anuncio se envía en el body y se accede con **@RequestBody**
  - Si el recurso existe se actualiza y se devuelve de nuevo
  - Si no existe, se devuelve **404 NOT FOUND**. Por eso el método devuelve un **ResponseEntity**

APIs REST con Spring

code URJC

ejem1

- **Reemplazar un recurso (PUT)**

```
@PutMapping("/posts/{id}")
public ResponseEntity<Post> replacePost(@PathVariable long id,

    @RequestBody Post newPost) {

    Post post = posts.findById(id);

    if (post != null) {

        newPost.setId(id);
        posts.save(newPost);

        return ResponseEntity.ok(post);
    } else {
        return ResponseEntity.notFound().build();
    }
}
```

En caso de discrepancia prevalece el id de la URL

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- **Ejemplo 2 (APIs REST con Spring)**
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

APIs REST con Spring

code URJC

ejem2

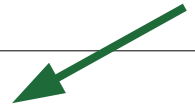
- **Factorizar URL mapping en el controller**
  - Cuando **todas las URLs** de un controlador empiezan de forma **similar**, se puede poner la anotación **@RequestMapping** a nivel de **clase** con la parte común
  - Cada **método** sólo tiene que incluir la **parte propia** (si existe)

APIs REST con Spring

code URJC

ejem2

- Factorizar URL mapping en el controller



```
@RestController
@RequestMapping("/posts")
public class PostController {

  @GetMapping("/")
  public Collection<Post> getPosts() {...}

  @GetMapping("/{id}")
  public ResponseEntity<Post> getPost(@PathVariable long id) {...}

  @PostMapping("/")
  public ResponseEntity<Post> createPost(@RequestBody Post post) {...}

  @PutMapping("/{id}")
  public ResponseEntity<Post> replacePost(@PathVariable long id, @RequestBody Post newPost) {...}

  @DeleteMapping("/{id}")
  public ResponseEntity<Post> deletePost(@PathVariable long id) {…}

}
```

APIs REST con Spring

code URJC

- **Acceso a los parámetros de la URL**
  - Filtros, definición de propiedades, términos de búsqueda, etc..
    - http://portal.com/anuncios?poblacion=Madrid

```
@GetMapping("/anuncios")
public List<Anuncio> anuncios(@RequestParam String poblacion) {
  ...
}
```

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- **Ejemplo 3 (Conversión entre objetos y JSON)**
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Conversión entre objetos y JSON

code URJC

ejem3

- Cuando se implementa una API REST es deseable controlar cómo se **convierten los objetos a JSON** (y viceversa)
- Spring utiliza la librería **Jackson** en modo **data binding** para hacer esta tarea
- Existen **diferentes formas** de controlar la conversión, pero la más sencilla es usando **anotaciones**

https://github.com/FasterXML/jackson

https://www.baeldung.com/jackson

Conversión entre objetos y JSON

code URJC

ejem3

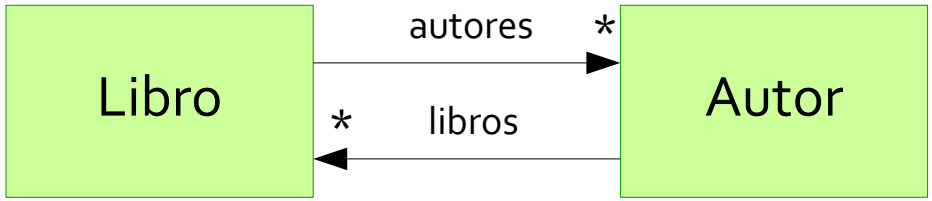
- **NO es una buena práctica** usar los objetos del modelo directamente en la API REST
- Es recomendable usar el patrón **DTO (*Data Transfer Object*)**
  - Hay veces que con una clase es suficiente **PostDTO**
  - Otras veces es necesario crear clases específicas por cada endpoint (**PostCreateRequest**, **PostReplaceRequest**, **PostResource**…)
- Estas estrategias se estudiarán más adelante

Conversión entre objetos y JSON

code URJC

ejem3

- Modelo para gestionar libros y autores




```
public class Libro {

  private long id;
  private String titulo;
  private int precio;

  private List<Autor> autores;
}
```

```
public class Autor {

  private long id;
  private String nombre;
  private String nacionalidad;

  private List<Libro> libros;
}
```

Conversión entre objetos y JSON

code URJC

ejem3

- ¿Qué ocurre si tenemos esta API REST?

```
@RestController
public class LibrosAutoresController {

    private List<Libro> libros = ...

    @GetMapping("/libros")
    public List<Libro> getLibros() {
        return libros;
    }
}
```

Conversión entre objetos y JSON

code URJC

ejem3

- ¿Qué ocurre si tenemos esta API REST?

```
@RestController
public class LibrosAutoresController {

    private List<Libro> libros = ...

    @GetMapping("/libros")
    public List<Libro> getLibros() {
        return libros;
    }
}
```

Al intentar convertir los objetos a JSON, se produce un **ERROR por StackOverflow** (o similar).
Como un libro tiene un autor y un autor tiene también un libro existe una referencia circular.

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- **Ejemplo 4 (Conversión entre objetos y JSON)**
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

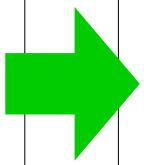
Conversión entre objetos y JSON

code URJC

ejem4

- **Ignorando atributos circulares**
  - Se pueden **ignorar del JSON** los atributos de la clases que generan la referencia circular

```
public class Libro {

  private long id;
  private String titulo;
  private int precio;

  @JsonIgnore
  private List<Autor> autores;
}
```



```
[
  {
    "id":0,
    "titulo":"Bambi",
    "precio":3
  },
  {
    "id":1,
    "titulo":"Batman",
    "precio":4
  }
]
```

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- **Ejemplo 5 (Conversión entre objetos y JSON)**
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Conversión entre objetos y JSON

code URJC

ejem5

- **Datos diferentes por URL**
  - El problema es que esta solución impide obtener la **lista de autores** asociada a un **libro**
  - Lo ideal sería tener **más o menos información** en función de si estamos accediendo a la **lista** de libros o a un libro **concreto**
    - **Lista de libros:** Sin autores
    - **Libro concreto:** Información del libro e información de sus autores (pero sin todos sus libros)

Conversión entre objetos y JSON

code URJC

ejem5

- **Datos diferentes por URL**
  - Creamos un nuevo **interfaz** Java
  - Anotamos algunos atributos con **@JsonView** pasando ese **interfaz** como parámetro
  - Anotamos el método de **@RestController** igual que los atributos (**@JsonView** con el interfaz como parámetro)
  - Los objetos que devuelva el método tendrán únicamente los **atributos con ese interfaz**

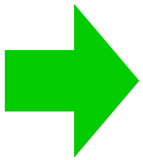
Conversión entre objetos y JSON

code URJC

ejem5

```
public class Autor {

    interface Basico {}

    @JsonView(Basico.class)
    private long id = -1;

    @JsonView(Basico.class)
    private String nombre;

    @JsonView(Basico.class)
    private String nacionalidad;

    private List<Libro> libros;
}
```

```
@JsonView(Autor.Basico.class)
@GetMapping("/autores")
public List<Autor> getAutores() {
  return autores;
}
```



```
[
  {
    "id":0,
    "titulo":"Bambi",
    "precio":3
  },
  {
    "id":1,
    "titulo":"Batman",
    "precio":4
  }
]
```

Índice de Ejemplos

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- **Ejemplo 6 (Conversión entre objetos y JSON)**

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Conversión entre objetos y JSON

code URJC

ejem6

- **Datos diferentes por URL**
  - Si queremos que en un **método** de la **API REST** se devuelvan atributos anotados con diferentes interfaces hay que **crear un nuevo interfaz**
  - Ese **nuevo interfaz** tiene que **heredar** de los interfaces usados por los atributos
  - Usamos ese interfaz en el **@JsonView** del método del **@RestController**

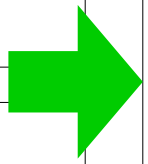
Conversión entre objetos y JSON

code URJC

ejem6

```java
public class Autor {

  interface Basico {}
  interface Libros {}

  @JsonView(Basico.class)
  private long id = -1;
  ...
  @JsonView(Libros.class)
  private List<Libro> libros;
}
```

```java
interface AutorDetalle
  extends Autor.Basico, Autor.Libros,
          Libro.Basico {}

@JsonView(AutorDetalle.class)
@GetMapping("/autores/{id}")
public Autor getAutor(@PathVariable int id){
  return autores.get(id);
}
```



```json
{
 "id":1,
 "nombre":"Gerard",
 "nacionalidad":"Frances",
 "libros":[
  {
   "id":1,
   "titulo":"Batman",
   "precio":4
  },
  {
   "id":2,
   "titulo":"Spiderman",
   "precio":2
  }
 ]
}
```

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- **Ejemplo 7 (Clientes REST en el Servidor)**
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Clientes REST en el servidor

code URJC

ejem7

- **RestTemplate de Spring**
  - Para hacer peticiones REST en **Spring** se usa un objeto de la clase **RestTemplate**
  - Se indica la **URL** y la **clase** de los objetos que devolverá la consulta

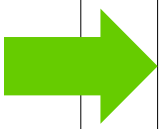
Clientes REST en el servidor

code URJC

ejem7

- **RestTemplate de Spring**
  - Para procesar la respuesta se indica la clase que estructura los datos

```
{
  "kind": "books#volumes",
  "totalItems": 579,
  "items": [
    {
      "kind": "books#volume",
      "volumeInfo": {
        "title": "Java a Tope: J2me...",
        "publisher": "Sergio Gálvez Rojas",
        ...
    },
    {...},
    {...}
   ]
   ...
}
```



Estas clases pueden tener métodos, atributos privados...

```
class BooksResponse {
  public List<Book> items;
}

class Book {
  public VolumeInfo volumeInfo;
}

class VolumeInfo {
  public String title;
}
```

Clientes REST en el servidor

code URJC

ejem7

- RestTemplate de Spring

```
RestTemplate restTemplate = new RestTemplate();

String url="https://www.googleapis.com/.../volumes?q=intitle:"+title;

BooksResponse data =
   restTemplate.getForObject(url, BooksResponse.class);

List<String> bookTitles = new ArrayList<String>();

for (Book book : data.items) {
  bookTitles.add(book.volumeInfo.title);
}

return bookTitles;
```

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- **Ejemplo 8 (Clientes REST en el Servidor)**
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Clientes REST en el servidor

code URJC

ejem8

- **RestTemplate: Cliente REST Spring**
  - Es posible acceder a los datos directamente **sin definir clases**
  - Para ello se usa la librería **Jackson** de procesamiento de **JSON** para Java
  - **Jackson** también se usa para convertir (mapear) el **JSON** a objetos (vistos del ejemplo anterior)

https://github.com/FasterXML/jackson

https://www.baeldung.com/jackson

Clientes REST en el servidor

code URJC

ejem8

- RestTemplate: Cliente REST Spring

```
RestTemplate restTemplate = new RestTemplate();

String url="https://www.googleapis.com/.../volumes?q=intitle:"+title;

ObjectNode data = restTemplate.getForObject(url, ObjectNode.class);

List<String> bookTitles = new ArrayList<String>();

ArrayNode items = (ArrayNode) data.get("items");

for (int i = 0; i < items.size(); i++) {
  JsonNode item = items.get(i);
  String bookTitle = item.get("volumeInfo").get("title").asText();
  bookTitles.add(bookTitle);
}
```

Clientes REST en el servidor

code URJC

ejem8

- **RestTemplate: Cliente REST Spring**

```
RestTemplate restTemplate = new RestTemplate();

String url="https://www.googleapis.com/.../volumes?q=intitle:"+title;

ObjectNode data = restTemplate.getForObject(url, ObjectNode.class);

List<String> bookTitles = new ArrayList<String>();

ArrayNode items = (ArrayNode) data.get("items");

for (int i = 0; i < items.size(); i++) {
  JsonNode item = items.get(i);
  String bookTitle = item.get("volumeInfo").get("title").asText();
  bookTitles.add(bookTitle);
}
```

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- **Ejemplo 9 (Clientes REST en el Servidor)**
- Ejemplo 10 (Documentación de APIs REST)
- Ejemplo 11 (Imágenes)

Clientes REST en el servidor

code URJC

- **Spring Feign**
  - Definición **declarativa** de una API REST que va a ser consumida
    - Se define un interfaz Java con un método por cada endpoint REST (con anotaciones)
    - Spring genera una implementación que realiza las consultas usando un cliente REST
    - Un componente puede inyectar el interfaz y al invocar los métodos se ejecutan las consultas
    - Se inyecta como una dependencia más

Clientes REST en el servidor

code URJC

ejem9

- Consumo de la API REST de libros

```
@FeignClient(name = "books", url = "https://www.googleapis.com/")
public interface BooksService {

     @GetMapping("books/v1/volumes")
     BookResponse getBooks(@RequestParam String q);
}
```

Anotamos la interfaz con la url del servidor rest

Definimos los métodos como en un controlador

```
@RestController
public class BooksController {
     @Autowired BooksService service;

     @GetMapping("/booktitles")
     public List<String> getBookTitles(@RequestParam String title) {

          BooksResponse data = service.getBooks("intitle:" + title);

          List<String> bookTitles = new ArrayList<String>();
          for  (Book book : data.items) {
               bookTitles.add(book.volumeInfo.title);
          }
          return bookTitles;
     }
}
```

Inyectamos la interfaz y la usamos para llamar al servidor rest

Clientes REST en el servidor

code URJC

ejem9

- Consumo de la API REST de anuncios

```
@SpringBootApplication
@EnableFeignClients
public class Application {

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

Anotamos la aplicación para que Spring genere automáticamente el cliente feign

Índice de Ejemplos

code URJC

- Ejemplo 1 Frontend (Clientes API REST)
- Ejemplo 2 Frontend (Clientes API REST)
- Ejemplo 1 (Clientes API REST)
- Ejemplo 1 (APIs REST con Spring)
- Ejemplo 2 (APIs REST con Spring)
- Ejemplo 3 (Conversión entre objetos y JSON)
- Ejemplo 4 (Conversión entre objetos y JSON)
- Ejemplo 5 (Conversión entre objetos y JSON)
- Ejemplo 6 (Conversión entre objetos y JSON)

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- **Ejemplo 10 (Documentación de APIs REST)**
- Ejemplo 11 (Imágenes)

Documentación de APIs REST

code URJC

ejem10

- El mecanismo más usado para documentar APIs REST es la especificación **OpenAPI**
- Es una evolución de **Swagger Specification**
- Tiene un ecosistema de herramientas: **generación de código**, webs interactivas, etc…

https://www.openapis.org/

Documentación de APIs REST

code URJC

ejem10

- **Generación de OpenAPI partiendo de código**
  - SpringDoc es un proyecto que genera la especificación OpenAPI partiendo de una API REST en SpringBoot

https://springdoc.org/

https://www.baeldung.com/spring-rest-openapi-documentation

Documentación de APIs REST

code URJC

ejem10

- **Generación de OpenAPI partiendo de código**
  - Si añadimos la dependencia Maven de SpringDoc

```
<dependency>
    <groupId>org.springdoc</groupId>
    <artifactId>springdoc-openapi-ui</artifactId>
    <version>1.5.0</version>
</dependency>
```

  - La especificación de la API se puede descargar de

http://localhost:8080/v3/api-docs/

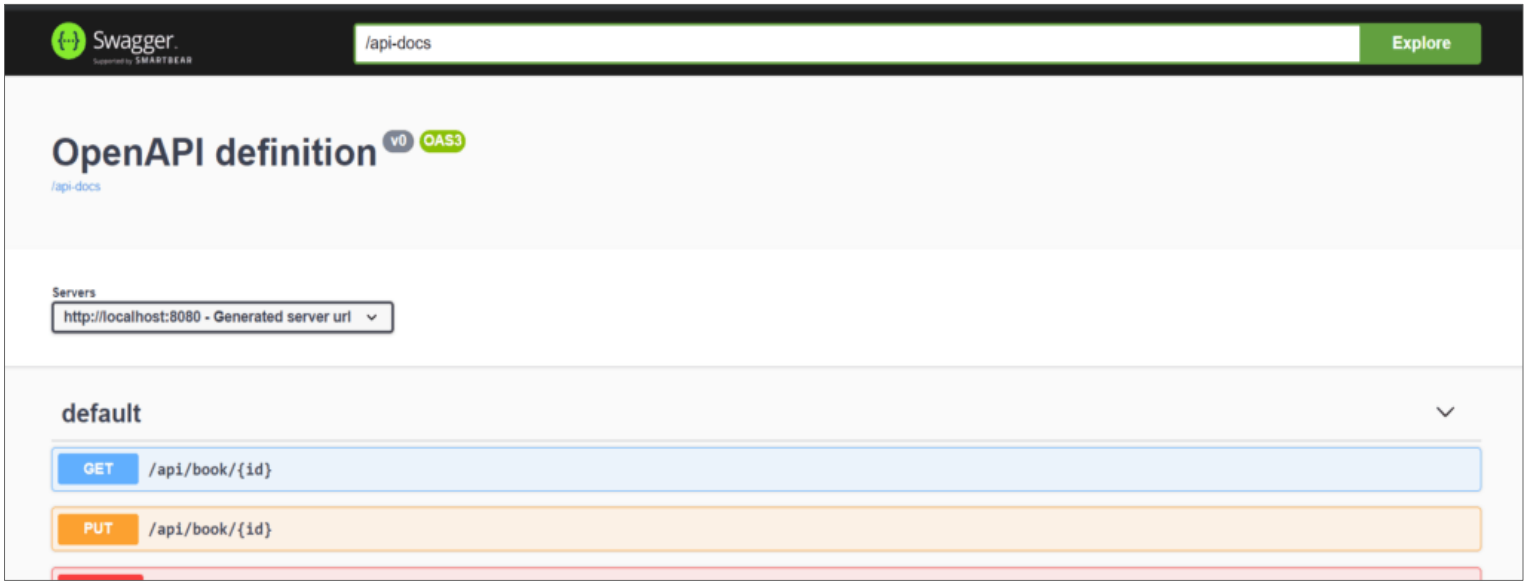
http://localhost:8080/v3/api-docs.yaml

Documentación de APIs REST

ejem10

- **Generación de OpenAPI partiendo de código**
  - Se publica un interfaz web interactivo que documenta la API y permite interactuar con ella

http://localhost:8080/swagger-ui.html



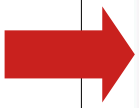
Documentación de APIs REST

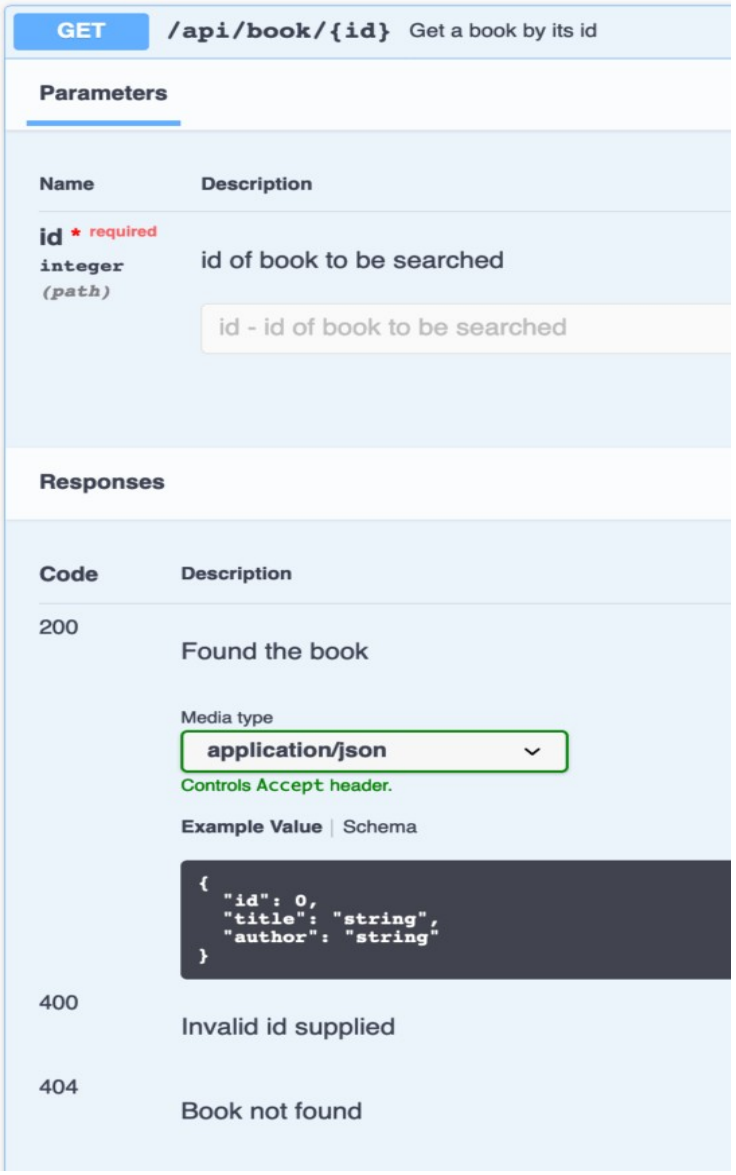
code URJC

ejem10

- **Generación de OpenAPI partiendo de código**
  - Se pueden añadir anotaciones específicas para documentar mejor la API
    - Usar la anotación **@ResponseStatus** en endpoints
    - Usar anotaciones **@Operation** y **@ApiResponses** proporcionadas por SpringDoc

```
@Operation(summary = "Get a book by its id")

@ApiResponses(value = {
  @ApiResponse(
    responseCode = "200",
    description = "Found the book",
    content = {@Content(
      mediaType = "application/json",
      schema = @Schema(implementation=Book.class)
    )}
  ),
  @ApiResponse(
    responseCode = "400",
    description = "Invalid id supplied",
    content = @Content
  ),
  @ApiResponse(
    responseCode = "404",
    description = "Book not found",
    content = @Content
  )
})

@GetMapping("/{id}")
public Book findById(
  @Parameter(description="id of book to be searched")
  @PathVariable long id) {

  return ...;
}
```





Índice de Ejemplos

code URJC

Índice de Ejemplos

- Ejemplo 7 (Clientes REST en el Servidor)
- Ejemplo 8 (Clientes REST en el Servidor)
- Ejemplo 9 (Clientes REST en el Servidor)
- Ejemplo 10 (Documentación de APIs REST)
- **Ejemplo 11 (Imágenes)**

Imágenes

code URJC

ejem11

- Existen **diversas estrategias** para gestionar imágenes en una API REST
- Vamos a considerar que cada entidad solo puede tener **una única imagen (opcional)**
- La imagen se podrán descargar en una URL que estará guardada como atributo del recurso

```
{
    "id": 1,
    "user": "Pepe",
    "title": "Vendo moto",
    "text": "Barata, barata",
    "image": "http://server/posts/1/image"
}
```

Imágenes

code URJC

ejem11

- Upload image

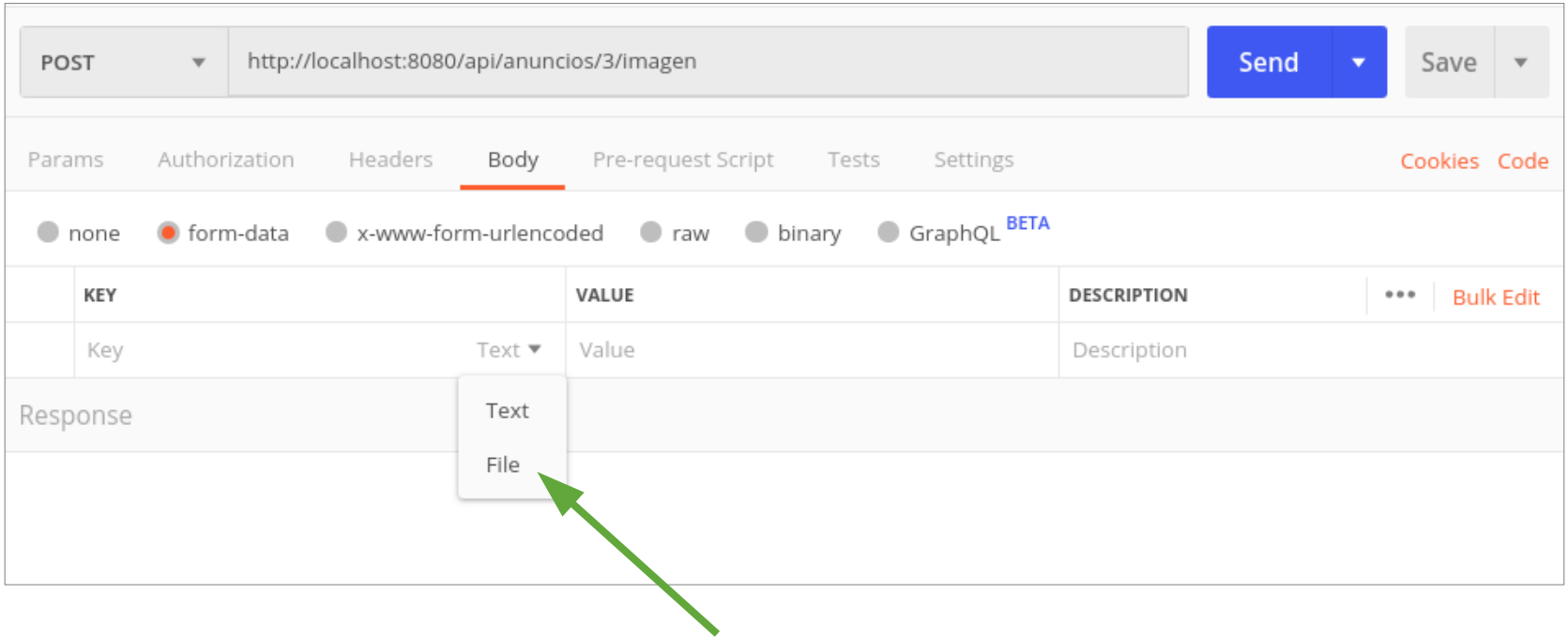
```
@PostMapping("/{id}/image")
public ResponseEntity<Object> uploadImage(@PathVariable long id,
  @RequestParam MultipartFile imageFile) throws IOException {

  Post post = posts.findById(id);

  if (post != null) {

    URI location = fromCurrentRequest().build().toUri();

    post.setImage(location.toString());
    posts.save(post);

    imgService.saveImage(POSTS_FOLDER, post.getId(), imageFile);

    return ResponseEntity.created(location).build();

  } else {
     return ResponseEntity.notFound().build();
  }
}
```

Imágenes

code URJC

ejem11

- Subir un fichero con Postman

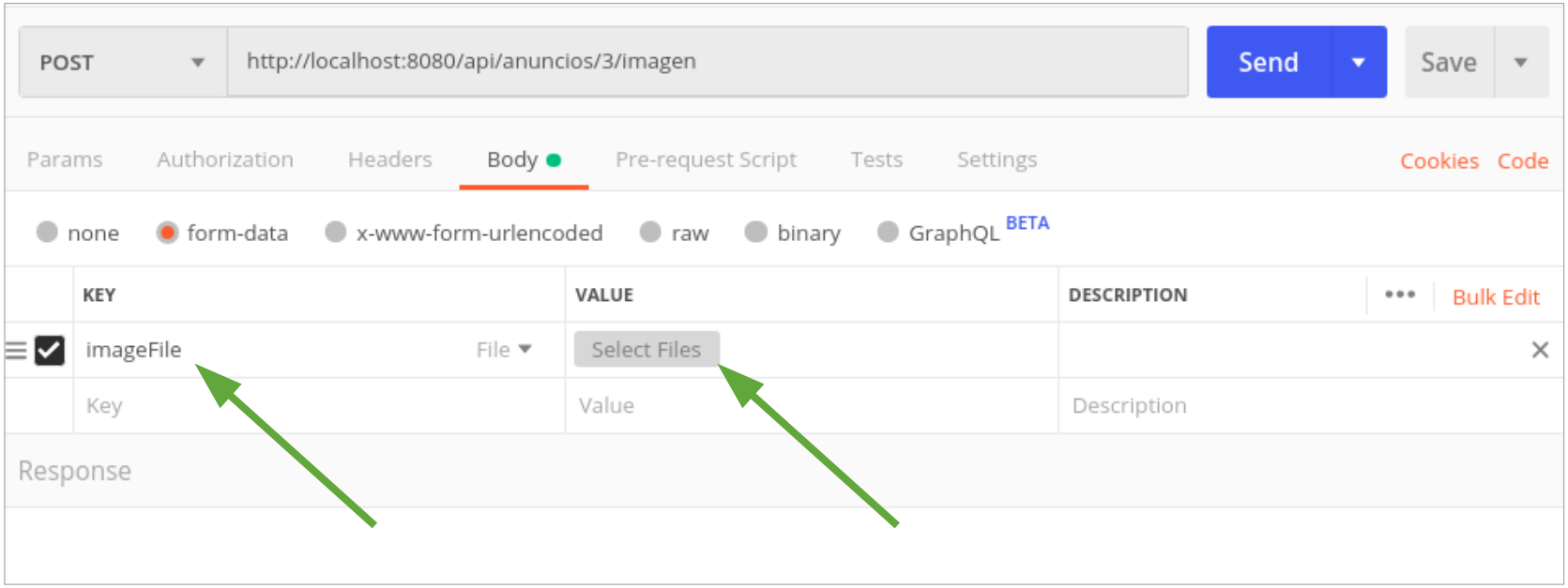


Imágenes

code URJC

ejem11

- Subir un fichero con Postman



Imágenes

code URJC

ejem11

- Download image

```
@GetMapping("/{id}/image")
public ResponseEntity<Object> downloadImage(@PathVariable long id)
  throws MalformedURLException {

  return this.imgService.createResponseFromImage(POSTS_FOLDER, id);
}
```

Imágenes

code URJC

ejem11

- Delete image

```
@DeleteMapping("/{id}/image")
public ResponseEntity<Object> deleteImage(@PathVariable long id)
  throws IOException {

  Post post = posts.findById(id);

  if(post != null) {

    post.setImage(null);
    posts.save(post);

    this.imgService.deleteImage(POSTS_FOLDER, id);

    return ResponseEntity.noContent().build();

  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 1

- Implementa una **API REST** en el servidor para gestionar **Items**
- Los items se gestionarán en **memoria** (como en el ejemplo de los posts)

Ejercicio 1

- **API REST Items**
  - Consulta de items
    - Method: GET
    - URL: http://127.0.0.1:8080/items/
    - Result:

```
[
  { "id": 1, "description": "Leche", "checked": false },
  { "id": 2, "description": "Pan", "checked": true }
]
```

    - Status code: 200 (OK)

Ejercicio 1

- **API REST Items**
  - Consulta de un item concreto
    - Method: GET
    - URL: http://127.0.0.1:8080/items/1
    - Result:

```
{ "id": 1, "description": "Leche", "checked": false }
```

    - Status code: 200 (OK)

Ejercicio 1

- **API REST Items**
  - Creación de un item
    - Method: POST
    - URL: http://127.0.0.1:8080/items/
    - Headers: Content-Type: application/json
    - Body:

```
{ "description" : "Galletas", "checked": true }
```

    - Result:

```
{ "id": 2, "description" : "Galletas", "checked": true }
```

    - Status code: 201 (Created)
    - Header: Location=http://127.0.0.1:8080/items/2

Ejercicio 1

- **API REST Items**
  - Reemplazo de un item
    - Method: PUT
    - URL: http://127.0.0.1:8080/items/1
    - Headers: Content-Type: application/json
    - Body:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

    - Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

    - Status code: 200 (OK)

Ejercicio 1

- **API REST Items**
  - Borrado de un item
    - Method: DELETE
    - URL: http://127.0.0.1:8080/items/1
    - Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

    - Status code: 200 (OK)

code
URJC

Desarrollo de Aplicaciones Web

Tema 3.2 – Bases de datos SQL en Spring

Índice de Ejemplos

Índice de Ejemplos

Índice de Ejemplos

Índice de Ejemplos

Bases de datos SQL en Spring

code URJC

ejem1

- El proyecto **Spring Data** ofrece mecanismos para el acceso a **Bases de datos relacionales y no relacionales**
  - **Creación del esquema** partiendo de las clases del código Java (o viceversa)
  - **Conversión automática** entre objetos Java y el formato propio de la base de datos
  - **Creación de consultas** en base a métodos en interfaces

http://projects.spring.io/spring-data/
http://projects.spring.io/spring-data-jpa/

Bases de datos SQL en Spring

code URJC

ejem1

- **Objeto de la base de datos (entidad)**

Anotaciones usadas para generar el esquema y para hacer la conversión entre objetos y filas

El atributo anotado con **@Id** es la clave primaria de la tabla. Lo habitual es que se genere de forma automática

```
@Entity
public class Customer {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;

  private String firstName;
  private String lastName;

  // Constructor necesario para la carga desde BBDD
  protected Customer() {}

  public Customer(String firstName, String lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
  }

  // Getter, Setters and toString
  ...
}
```

Bases de datos SQL en Spring

code URJC

ejem1

- **Repositorio**

El interfaz padre **JpaRepository** dispone de métodos para **consultar, guardar, borrar** y **modificar** objetos de la base de datos.

```
public interface CustomerRepository extends JpaRepository<Customer, Long> {

   List<Customer> findByLastName(String lastName);

   List<Customer> findByFirstName(String firstName);

}
```

Métodos que se traducen automáticamente en **consultas a la BBDD** en base al nombre y los parámetros.

Si no se necesita ningún tipo de consulta especial, el **interfaz puede quedar vacío**

Clase de la **entidad** y clase de su identificador (generalmente **Integer** o **Long**)

Bases de datos SQL en Spring

code URJC

ejem1

- **Uso de la base de datos**
  - Un componente **inyecta el repositorio** con @Autowired
  - Métodos disponibles:
    - Consultar todos (**findAll**)
    - Consulta por id (**findById**)
    - Guardar un nuevo objeto o actualizarlo (**save**)
    - Borrar un objeto (**delete**)
    - Consultas por cualquier criterio (métodos añadidos al interfaz)

Bases de datos SQL en Spring

code URJC

ejem1

- Uso de la BBDD

```
@Controller
public class DataBaseUsage implements CommandLineRunner {

  @Autowired
  private CustomerRepository repository;

  @Override
  public void run(String... args) throws Exception {

     repository.save(new Customer("Jack", "Bauer"));
     repository.save(new Customer("Chloe", "O'Brian"));

     List<Customer> bauers = repository.findByLastName("Bauer");
     for (Customer bauer : bauers) {
          System.out.println(bauer);
     }

     repository.delete(bauers.get(0));
  }
}
```

Controlador especial que se ejecuta cuando se **inicia la aplicación**. Puede leer los parámetros de la **línea de comandos**

Código de ejemplo que usa el **repositorio** para guardar, consultar y borrar datos de la **BBDD**

Bases de datos SQL en Spring

code URJC

ejem1

- pom.xml

```
<groupId>es.codeurjc</groupId>
<artifactId>bbdd_ejem1</artifactId>
<version>0.1.0</version>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.4.0</version>
</parent>
...
<dependencies>
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
 <dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
  </dependency>
</dependencies>
```

Dependencia a **Spring Data** para BBDD **relacionales**

Dependencia a la **BBDD en memoria H2**. En las bases de datos en memoria, el esquema se genera de forma **automática** al iniciar la app

Índice de Ejemplos

Índice de Ejemplos

Bases de datos SQL en Spring

code URJC

ejem2

- Una aplicación web puede gestionar la información en una BBDD
  - Se añaden las dependencias de JPA y la BBDD en el **pom.xml**
  - En vez de usar una estructura en memoria se usa un **repositorio**
- Como ejemplo vamos a transformar la API REST de gestión de posts

Bases de datos SQL en Spring

code URJC

ejem2

- pom.xml

```
...
<dependencies>

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

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

</dependencies>
```

Dependencias **Web**

Dependencia a la **BBDD H2**

Bases de datos SQL en Spring

code URJC

ejem2

- Objeto de la base de datos (entidad)

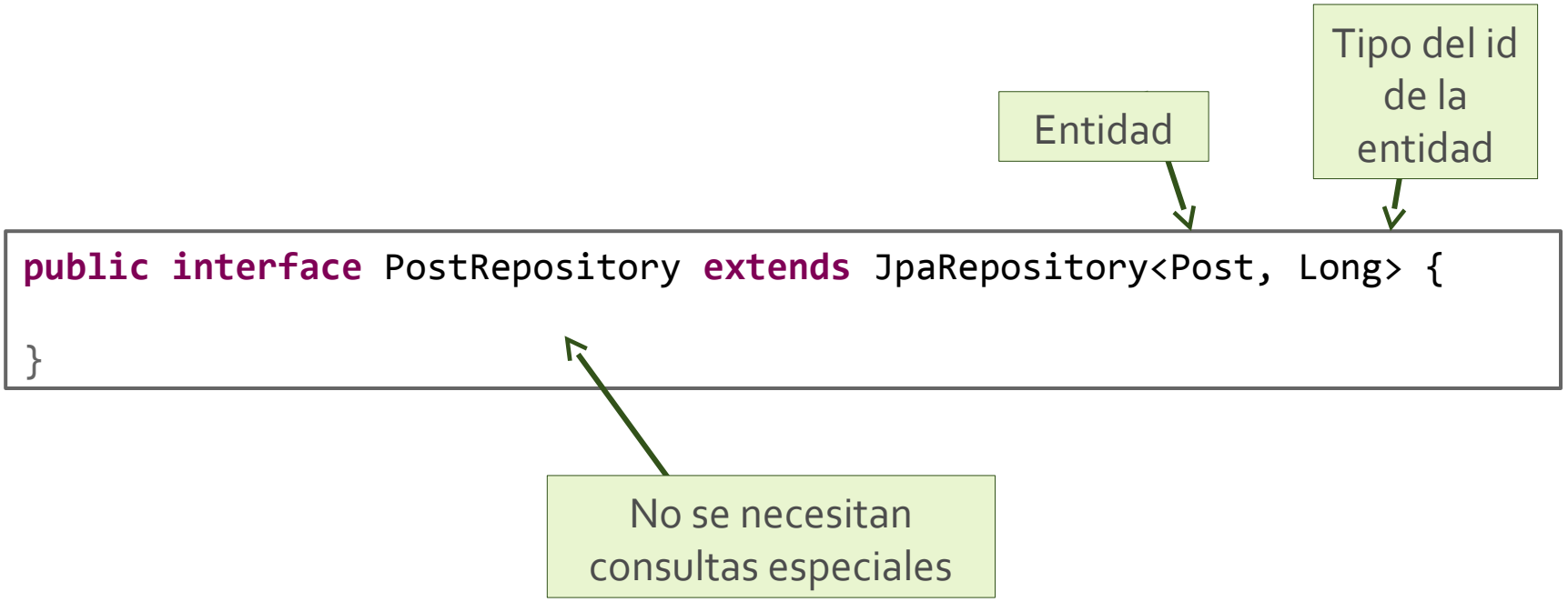
```
@Entity
public class Post {

	@Id
	@GeneratedValue(strategy = GenerationType.AUTO)
	private Long id;

	private String username;
	private String title;
	private String text;

	public Post() { }

	public Post(String username, String title, String text) {
		super();
		this.username = username;
		this.title = title;
		this.text = text;
	}
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- Repositorio



Bases de datos SQL en Spring

code URJC

ejem2

- Controlador REST

Se pueden inyectar tantos repositorios como sea necesario

Un método **@PostConstruct** se ejecutará después de haber inyectado las dependencias

```
@RestController
@RequestMapping("/posts")
public class PostController {

  @Autowired
  private PostRepository posts;

  @PostConstruct
  public void init() {
    posts.save(new Post("Pepe", "Vendo moto", "Barata, barata"));
    posts.save(new Post("Juan", "Compro coche", "Pago bien"));
  }

  @GetMapping("/")
  public Collection<Post> getPosts() {
    return posts.findAll();
  }

...
```

Desde los métodos se usa el repositorio

Bases de datos SQL en Spring

code URJC

ejem2

- **Consulta de un recurso (GET)**
  - Gestión de *Optional* con *findById*

```
@GetMapping("/{id}")
public ResponseEntity<Post> getPost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {
    return ResponseEntity.ok(post.get());
  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- **Consulta de un recurso (GET)**
  - Gestión de *Optional* con *findById*

```
@GetMapping("/{id}")
public ResponseEntity<Post> getPost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {
    return ResponseEntity.ok(post.get());
  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Si hay valor (**isPresent**) lo obtenemos (**get**) para devolverlo

En un repositorio el método **findById** devuelve *Optional<Post>*

Bases de datos SQL en Spring

code URJC

ejem2

- Creación de recurso (POST)

```
@PostMapping("/")
public ResponseEntity<Post> createPost(@RequestBody Post post) {

  posts.save(post);

  URI location = fromCurrentRequest().path("/{id}")
    .buildAndExpand(post.getId()).toUri();

  return ResponseEntity.created(location).body(post);
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- Reemplazo de recurso (PUT)

```
@PutMapping("/{id}")
public ResponseEntity<Post> replacePost(@PathVariable long id,
  @RequestBody Post newPost) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {

    newPost.setId(id);
    posts.save(newPost);

    return ResponseEntity.ok(newPost);
  } else {
     return ResponseEntity.notFound().build();
  }
}
```

Bases de datos SQL en Spring

code URJC

ejem2

- Borrado de recurso (DELETE)

```
@DeleteMapping("/{id}")
public ResponseEntity<Post> deletePost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  if (post.isPresent()) {
    posts.deleteById(id);
    return ResponseEntity.ok(post.get());
  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Bases de datos SQL en Spring

code URJC

ejem2

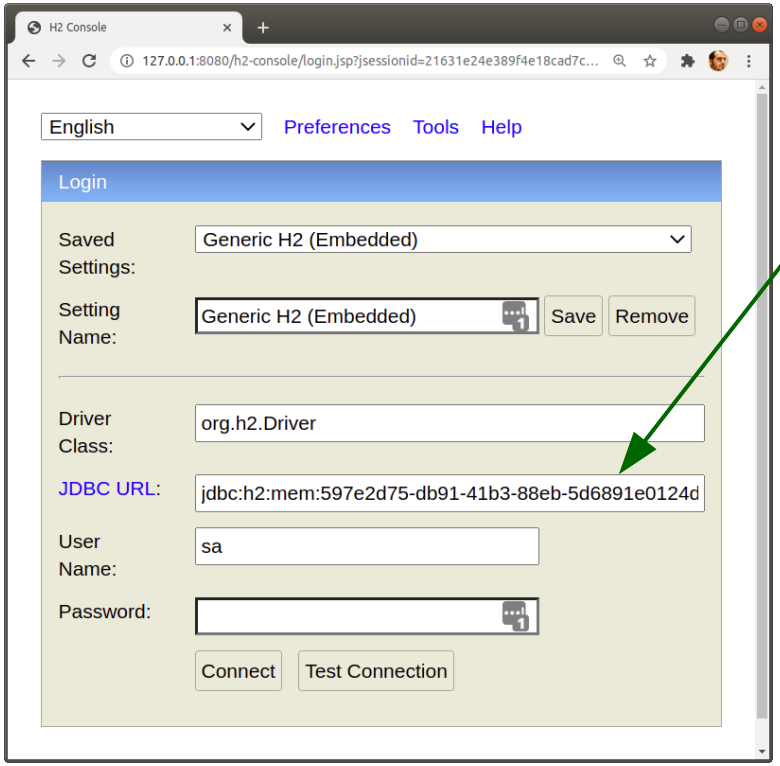
- **Consola web de base de datos H2**
  - Si desarrollamos una **aplicación web** que use una **base de datos H2** podemos acceder a una consola web de la propia BBDD
  - La consola es accesible en
    http://127.0.0.1:8080/h2-console
  - Para activar esa consola:
    - Usar las **devtools** en ese proyecto
    - O especificar en el fichero **application.properties**

```
spring.h2.console.enabled=true
```

Bases de datos SQL en Spring

ejem2

- Consola web de base de datos H2



Hay que poner como JDBC URL la ruta que aparece en el log

```
H2 console available at '/h2-
console'. Database available at
'jdbc:h2:mem:597e2d75-db91-41b3-
88eb-5d6891e0124d'
```

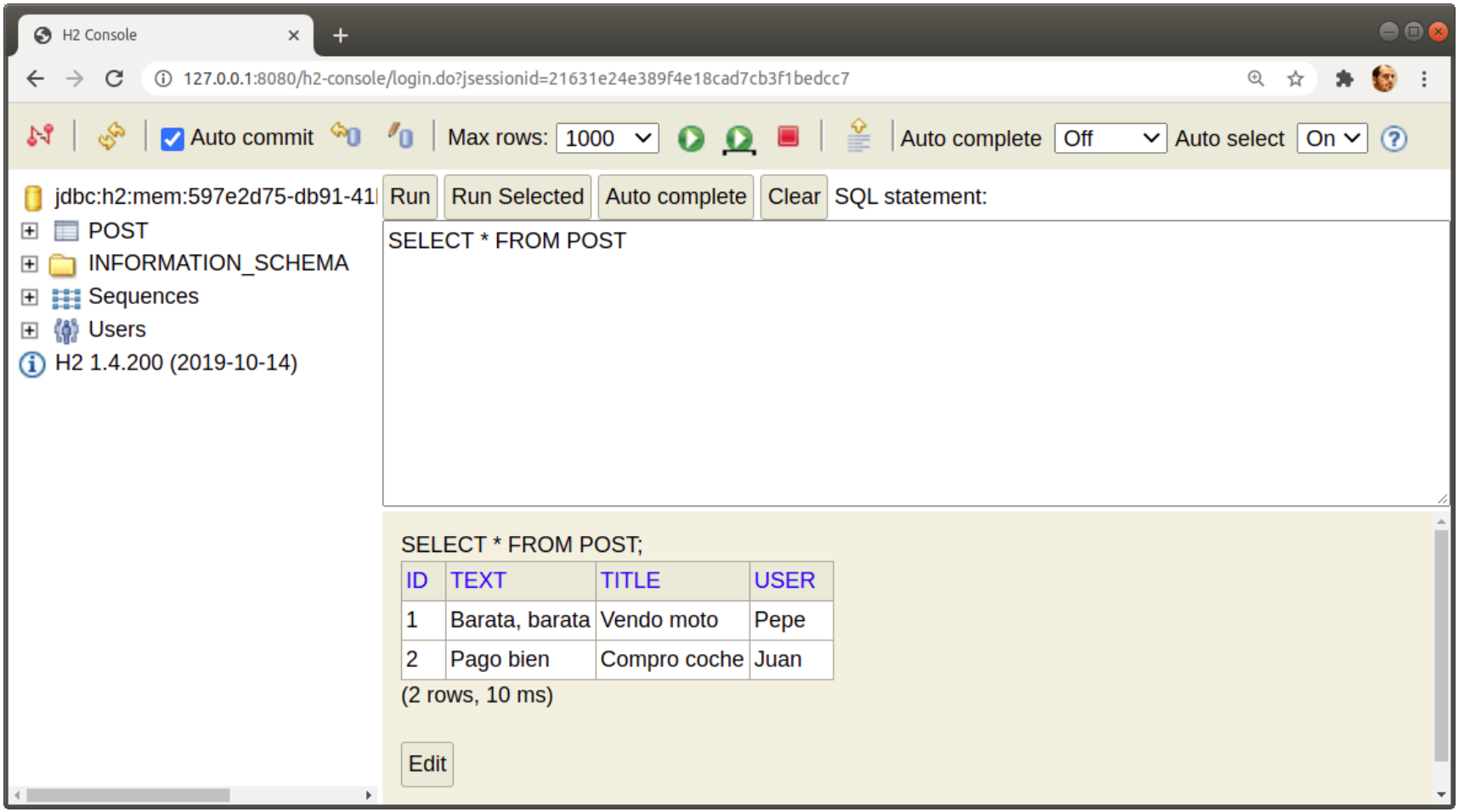
Para simplificar podemos configurar una URL fija:

```
spring.datasource.url=jdbc:h2:mem:testdb
```

Bases de datos SQL en Spring

code URJC

ejem2



Índice de Ejemplos

Índice de Ejemplos

Bases de datos SQL en Spring

code URJC

ejem3

- **Manipulación de Optional**
  - Históricamente en Java cuando un método de consulta indica la **ausencia de valor** devolviendo **null**
  - Usar null es propenso a errores y puede provocar **NullPointerException**
  - **Optional** representa en el sistema de tipos que el valor es opcional y obliga al desarrollador a considerar cómo quiere gestionar la ausencia de valor

https://blogs.oracle.com/javamagazine/12-recipes-for-using-the-optional-class-as-its-meant-to-be-used

Bases de datos SQL en Spring

code URJC

ejem3

- **Manipulación de Optional**
  - SpringBoot ofrece dos estrategias para gestionar mejor el Optional en una API REST:
    - **Optional.orElseThrow():** Se eleva una excepción NoSuchElementException si el recurso no existe. Esa excepción se puede convertir en un 404
    - **ResponseEntity.of(Optional resource):** Devuelve 404 si el recurso no existe.

Bases de datos SQL en Spring

code URJC

ejem3

- **Manipulación de Optional**
  - Configuración de Spring para que las excepciones **NoSuchElementException** generadas en un controlador se conviertan en un 404

```
@ControllerAdvice
class NoSuchElementExceptionControllerAdvice {

  @ResponseStatus(HttpStatus.NOT_FOUND)
  @ExceptionHandler(NoSuchElementException.class)
  public void handleNoTFound() {
  }
}
```

Bases de datos SQL en Spring

code URJC

- ## Manipulación de Optional

Elevando la excepción

ejem3

```
@GetMapping("/{id}")
public Post getPost(@PathVariable long id) {

  return posts.findById(id).orElseThrow();
}
```

Si no hay post se eleva **NoSuchElementException**

ResponseEntity.of(...)

ejem4

```
@GetMapping("/{id}")
public ResponseEntity<Post> getPost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);

  return ResponseEntity.of(post);
}
```

Método **ResponseEntity.of()** devuelve 404 si el post no está presente

Bases de datos SQL en Spring

code URJC

- ## Manipulación de Optional

ejem3

Elevando la excepción

```
@DeleteMapping("/{id}")
public Post deletePost(@PathVariable long id) {

  Post post = posts.findById(id).orElseThrow();
  posts.deleteById(id);
  return post;
}
```

Si no hay post se eleva **NoSuchElementException**

ejem4

ResponseEntity.of(...)

```
@DeleteMapping("/{id}")
public ResponseEntity<Post> deletePost(@PathVariable long id) {

  Optional<Post> post = posts.findById(id);
  post.ifPresent(p -> posts.deleteById(id));
  return ResponseEntity.of(post);
}
```

Sólo se ejecuta la operación de borrado si hay post: **ifPresent()**

Bases de datos SQL en Spring

code URJC

- ## Manipulación de Optional

ejem3

Elevando la excepción

```
@PutMapping("/{id}")
public Post replacePost(@PathVariable long id,
  @RequestBody Post newPost) {

  posts.findById(id).orElseThrow();

  newPost.setId(id);
  posts.save(newPost);

  return newPost;
}
```

Índice de Ejemplos

Índice de Ejemplos

Bases de datos SQL en Spring

code URJC

- **Manipulación de Optional**

ejem4

ResponseEntity.of(...)

```
@PutMapping("/{id}")
public ResponseEntity<Post> replacePost(@PathVariable long id,
@RequestBody Post newPost) {

  Optional<Post> post = posts.findById(id);

  return ResponseEntity.of(post.map(p -> {

    newPost.setId(id);
    posts.save(newPost);

    return newPost;
  }));
}
```

El método **map(...)** permite guardar el newPost y devolverlo si hay post

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem4

- **Mostrar las sentencias SQL**
  - Para aprender JPA y depurar los posibles errores es muy útil poder ver qué sentencias se ejecutan en la BBDD

application.properties

```
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type.descriptor.sql.BasicBinder=TRACE
```

Generación de tablas desde entidades

code URJC

ejem4

- **Entidad con atributos básicos**
  - Se genera una tabla por cada entidad
  - Por cada atributo de la clase de un tipo simple (entero, float, String, boolean…), se crea un campo en la tabla

```
@Entity
public class Post {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private long id;

  private String username;
  private String title;
  private String text;
}
```

```
create table post (
  id bigint not null,
  text varchar(255),
  title varchar(255),
  username varchar(255),
  primary key (id)
)
```

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem5

- **Relaciones entre entidades**
  - **1:1**
    - Unidireccional
    - Bidireccional
  - **1:N**
    - Unidireccional (1 → N y N → 1)
    - Bidireccional
  - **N:M**
    - Bidireccional

Generación de tablas desde entidades

code URJC

ejem5

- **Relación 1:1 unidireccional**
  - Una entidad tiene una referencia a otra entidad
  - El atributo que apunta a otro es anotado con **@OneToOne**
  - Cada objeto tiene que ser guardado en la base de datos de forma **independiente**
  - Al **cargar un objeto** de la BBDD podemos acceder al objeto **relacionado sin cargarlo explícitamente**

Generación de tablas desde entidades

code URJC

ejem5

- Relación 1:1 unidireccional

```
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int year;

  @OneToOne
  private Project project;

}
```

```
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;
}
```

Generación de tablas desde entidades

ejem5

- Relación 1:1 unidireccional

```
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int year;

  @OneToOne
  private Project project;

}
```

```
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;
}
```

Relación 1:1 unidireccional

code URJC

ejem5

```java
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int year;

  @OneToOne
  private Project project;
}
```

```java
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;
}
```

Generación de tablas

```
create table project (
  id bigint not null,
  calification integer not null,
  title varchar(255),
  primary key (id)
)

create table student (
  id bigint not null,
  name varchar(255),
  year integer not null,
  project_id bigint,
  primary key (id)
)

alter table student
  add constraint Fkr...
  foreign key (project_id)
  references project
```

Generación de tablas desde entidades

code URJC

ejem5

- Relación 1:1 unidireccional

```
Project p1 = new Project("TFG1", 8);
projectRepository.save(p1);

Student s1 = new Student("Pepe", 2010);
s1.setProject(p1);

Student s2 = new Student("Juan", 2011);

studentRepository.save(s1);
studentRepository.save(s2);
```

Cada objeto se tiene que guardar en su repositorio

Inicialización de datos

Generación de tablas desde entidades

code URJC

ejem5

- Relación 1:1 unidireccional

Consulta de datos

```
studentRepository.findAll();
```

```
[
 {
  "id" : 1,
  "name" : "Pepe",
  "year" : 2010,
  "project" : {
    "id" : 1,
    "title" : "TFG1",
    "calification" : 8
  }
 },
 {
  "id" : 2,
  "name" : "Juan",
  "year" : 2011
 }
]
```

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem6

- **Operaciones en cascada**
  - Hay veces que un objeto **siempre está asociado** a otro objeto (relación de composición)
  - Por ejemplo, un proyecto siempre está asociado a un estudiante
  - Cuando se **crea el estudiante**, **se crea el proyecto**, cuando se borra el estudiante, se borra el proyecto

Generación de tablas desde entidades

code URJC

ejem6

- **Operaciones en cascada**
  - Si la anotación @OneToOne se configura con **cascade = CascadeType.ALL** entonces ambos objetos de la relación tienen el mismo ciclo de vida
  - Al **guardar** el objeto principal, se **guarda** el asociado
  - Al **borrar** el objeto principal, se **borra** el asociado

Generación de tablas desde entidades

code URJC

ejem6

- Operaciones en cascada

```
//Deleting a student delete her associated project
@DeleteMapping("/students/{id}")
public Student deleteStudent(@PathVariable Long id) {
    Student student = studentRepository.findById(id).get();
    studentRepository.deleteById(id);
    return student;
}

//A project only can be deleted if it has no associated student.
@DeleteMapping("/projects/{id}")
public Project deleteProject(@PathVariable Long id) {
    Project project = projectRepository.findById(id).get();
    projectRepository.deleteById(id);
    return project;
}
```

Generación de tablas desde entidades

code URJC

ejem6

- Operaciones en cascada

```
@Entity
public class Student {

    @Id
    @GeneratedValue(...)
    private long id;

    private String name;
    private int year;

    @OneToOne(cascade=CascadeType.ALL)
    private Project project;
}
```

Como la relación en **cascade=ALL**, no es necesario guardar el objeto project explícitamente.

El **project** se guarda cuando el **student** se guarda

```
Student s1 = new Student("Pepe", 2010);
s1.setProject(new Project("TFG1", 8));
studentRepository.save(s1);
```

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem7

- **Relación 1:1 bidireccional**
  - Para facilitar el desarrollo, los **dos objetos de una relación** pueden tener un atributo apuntando al otro objeto
  - Ambas entidades tienen que anotarse con **@OneToOne**
  - Aunque en memoria ambos objetos se apunten entre sí, en la base de datos la relación se guarda en la **tabla de una entidad**

Generación de tablas desde entidades

code URJC

ejem7

- **Relación 1:1 bidireccional**
  - La entidad principal cuya tabla guarda la información anota el atributo con **@OneToOne**
  - El atributo debe apuntar al otro objeto de la otra entidad cuando se **guardar en la BBDD**
  - La otra entidad se anota con **@OneToOne(mappedby="attr")** indicando el nombre del atributo de la otra principal
  - Este atributo sólo se usa en lecturas

Generación de tablas desde entidades

ejem7

- Relación 1:1 bidireccional

```java
@Entity
public class Student {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int year;

  @OneToOne(cascade=CascadeType.ALL)
  private Project project;
}
```

```java
@Entity
public class Project {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private int calification;

  @OneToOne(mappedBy="project")
  private Student student;

}
```

```java
Student s1 = new Student("Pepe", 2010);
s1.setProject(new Project("TFG1", 8));
studentRepository.save(s1);
```

Al guardar, la entidad principal debe apuntar a la otra entidad

Generación de tablas desde entidades

code URJC

ejem7

- Relación 1:1 bidireccional

```
create table project (
   id bigint not null,
   calification integer not null,
   title varchar(255),
   primary key (id)
)

create table student (
   id bigint not null,
   name varchar(255),
   year integer not null,
   project_id bigint,
   primary key (id)
)

alter table student
   add constraint FKr6av6arpoy7wpbqipg41id1mn
   foreign key (project_id)
   references project
```

Como la entidad principal sigue siendo Student, las tablas son las mismas que con relación unidireccional

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem8

- **Relaciones entre entidades**
  - **1:1**
    - Unidireccional
    - Bidireccional
  - **1:N**
    - Unidireccional (1 → N y N → 1)
    - Bidireccional
  - **N:M**
    - Bidireccional

Generación de tablas desde entidades

code URJC

ejem8

- **Relación 1:N**
  - Cuando existe una relación **1:N** entre entidades se usan las anotaciones **@OneToMany** y **@ManyToOne**
  - Puede ser **unidireccional** o **bidireccional**
  - Cualquier sentido de la relación puede ser **la entidad principal** (la que se usa para guardar los datos)
  - También se puede usar **cascade**

Generación de tablas desde entidades

code URJC

ejem8

- Relación 1:N unidireccional

```
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int ranking;

  @OneToMany
  private List<Player> players;

}
```

```
@Entity
public class Player {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int goals;

}
```

Generación de tablas desde entidades

ejem8

- Relación 1:N unidireccional

```
create table player (
  id bigint not null,
  goals integer not null,
  name varchar(255),
  primary key (id)
)

create table team (
  id bigint not null,
  name varchar(255),
  ranking integer not null,
  primary key (id)
)

create table team_players (
  team_id bigint not null,
  players_id bigint not null
)
```

Como es @OneToMany, un team sólo puede estar asociado a un equipo

```
alter table team_players
  add constraint UK... unique (players_id)

alter table team_players
  add constraint FK...
  foreign key (players_id)
  references player

alter table team_players
  add constraint FK...
  foreign key (team_id)
  references team
```

Tabla que relaciona Teams y Players

Generación de tablas desde entidades

code URJC

ejem8

- Relación 1:N unidireccional

```
Player p1 = new Player("Torres", 10);
Player p2 = new Player("Iniesta", 10);

playerRepository.save(p1);
playerRepository.save(p2);

Team team = new Team("Selección", 1);

team.getPlayers().add(p1);
team.getPlayers().add(p2);

teamRepository.save(team);
```

Generación de tablas desde entidades

code URJC

ejem8

- **Relación 1:N unidireccional**
  - Sin cascase los objetos tienen ciclos de vida independientes (Un player puede estar sin team)

```
//Deleting a team doesn't delete its associated players
@DeleteMapping("/teams/{id}")
public Team deleteTeam(@PathVariable Long id) {
  Team team = teamRepository.findById(id).get();
  //Force loading players from database to be returned as JSON
  Hibernate.initialize(team.getPlayers());
  teamRepository.deleteById(id);
  return team;
}
```

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem9

- Relación 1:N unidireccional (cascade)

```
@Entity
public class Blog {

  @Id
  @GeneratedValue(...)
  private long id;

  private String title;
  private String text;

  @OneToMany(cascade=CascadeType.ALL)
  private List<Comment> comments;
}
```

```
@Entity
public class Comment {

  @Id
  @GeneratedValue(...)
  private long id;

  private String author;
  private String message;
}
```

```
Blog blog = new Blog("New", "My new product");
blog.getComments().add(new Comment("Cool", "Pepe"));
blog.getComments().add(new Comment("Very cool", "Juan"));
repository.save(blog);
```

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem10

- Relación 1:N bidireccional

```java
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int ranking;

  @OneToMany(mappedBy="team")
  private List<Player> players;

}
```

```java
@Entity
public class Player {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int goals;

  @ManyToOne
  private Team team;
}
```

Generación de tablas desde entidades

code URJC

ejem10

- Relación 1:N bidireccional

```
create table player (
  id bigint not null,
  goals integer not null,
  name varchar(255),
  team_id bigint,
  primary key (id)
)

create table team (
  id bigint not null,
  name varchar(255),
  ranking integer not null,
  primary key (id)
)

alter table player
  add constraint FK...
  foreign key (team_id)
  references team
```

Ahora la entidad principal es **Player**, por eso tiene el atributo **team_id** y no hay tabla de relación

Generación de tablas desde entidades

code URJC

ejem10

- Relación 1:N bidireccional

```
Team team = new Team("Selección", 1);
teamRepository.save(team);


Player p1 = new Player("Torres", 10);
Player p2 = new Player("Iniesta", 10);

p1.setTeam(team);
p2.setTeam(team);

playerRepository.save(p1);
playerRepository.save(p2);
```

La **entidad principal** ahora es **Player**, por eso se configura el Team en el objeto Player antes de guardar (y no al revés)

Un Player puede estar sin asociar a un Team

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem11

- **Relación 1:N bidireccional (cascade)**
  - Cuando se configura una relación de composición en la BBDD se suele configurar **cascade** y **orphanRemoval**
  - Se suele configurar **cascade=ALL** para que no haya que guardar los objetos de forma independiente
  - Se suele configurar **orphanRemoval=true** para que no pueda haber una parte sin si todo

https://vladmihalcea.com/orphanremoval-jpa-hibernate/

Generación de tablas desde entidades

code URJC

ejem11

- Relación 1:N bidireccional (cascade)

```
@Entity
public class Post {

      @Id
      @GeneratedValue(...)
      private long id;

      private String username;
      private String title;
      private String text;

      @OneToMany(mappedBy="post", cascade=CascadeType.ALL, orphanRemoval=true)
      private List<Comment> comments = new ArrayList<>();

      public void addComment(Comment comment) {
          comments.add(comment);
          comment.setPost(this);
      }

      public void removeComment(Comment comment) {
          comments.remove(comment);
          comment.setPost(null);
      }
}
```

Mantiene los atributos sincronizados en ambos extremos (Post y Comment)

Generación de tablas desde entidades

code URJC

ejem11

- Relación 1:N bidireccional (cascade)

```
@Entity
public class Comment {

    @Id
    @GeneratedValue(...)
    private long id;

    private String username;
    private String comment;

    @ManyToOne
    @JsonIgnore
    private Post post;

}
```

Generación de tablas desde entidades

code URJC

ejem11

- Relación 1:N bidireccional (cascade)

```
create table comment (
  id bigint not null,
  comment varchar(255),
  username varchar(255),
  post_id bigint,
  primary key (id)
)

create table post (
  id bigint not null,
  text varchar(255),
  title varchar(255),
  username varchar(255),
  primary key (id)
)

alter table comment
  add constraint FK..
  foreign key (post_id)
  references post
```

Generación de tablas desde entidades

code URJC

ejem11

- **Relación 1:N bidireccional (cascade)**
  - **API REST**
    - Al obtener un post individual o en una colección (GET) se devolverán los comentarios
    - Los comentarios de un post se podrán consultar, borrar, actualizar y añadir de forma independiente
      - http://server/posts/6/comments/3
    - Al reemplazar un post (PUT) no se verán afectados sus comentarios

Índice de Ejemplos

Índice de Ejemplos

Generación de tablas desde entidades

code URJC

ejem12

- **Relaciones entre entidades**
  - **1:1**
    - Unidireccional
    - Bidireccional
  - **1:N**
    - Unidireccional (1 → N y N → 1)
    - Bidireccional
  - **N:M**
    - Bidireccional

Generación de tablas desde entidades

code URJC

ejem12

- **Relación M:N**
  - Cuando existe una relación M:N entre entidades se usa la anotación **@ManyToMany**
  - Puede ser **unidireccional** o **bidireccional**
  - Cualquier sentido de la relación puede ser **la entidad principal** (la que se usa para guardar los datos)
  - También se puede usar **cascade**

Generación de tablas desde entidades

code URJC

ejem12

- Relación M:N bidireccional

```
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int ranking;

  @ManyToMany(mappedBy="team")
  private List<Player> players;

}
```

```
@Entity
public class Player {

  @Id
  @GeneratedValue(...)
  private long id;

  private String name;
  private int goals;

  @ManyToMany
  private List<Team> teams;

}
```

Generación de tablas desde entidades

ejem12

- Relación M:N bidireccional

```
create table player (
  id bigint not null,
  goals integer not null,
  name varchar(255),
  primary key (id)
)

create table team (
  id bigint not null,
  name varchar(255),
  ranking integer not null,
  primary key (id)
)

create table player_teams (
  players_id bigint not null,
  teams_id bigint not null
)
```

Como es @ManyToMany ya no está la restricción de que un player sólo pueda estar asociado a un team

```
alter table player_teams
  add constraint FK...
  foreign key (teams_id)
  references team

alter table player_teams
  add constraint FK...
  foreign key (players_id)
  references player
```

Generación de tablas desde entidades

code URJC

ejem12

- Relación M:N bidireccional

```
Team team1 = new Team("Selección", 1);
Team team2 = new Team("FC Barcelona", 1);
Team team3 = new Team("Atlético de Madrid", 2);

teamRepository.save(team1);
teamRepository.save(team2);
teamRepository.save(team3);

Player p1 = new Player("Torres", 10);
Player p2 = new Player("Iniesta", 10);

p1.getTeams().add(team1);
p1.getTeams().add(team3);

p2.getTeams().add(team1);
p2.getTeams().add(team2);

playerRepository.save(p1);
playerRepository.save(p2);
```

Índice de Ejemplos

Índice de Ejemplos

Consultas

code URJC

ejem13

- Estrategias para especificación de consultas con SpringData:
  - **Métodos en los repositorios**
  - Java Persistence Query Language (JPQL)
  - QueryDSL

Consultas

code URJC

- **Métodos en los repositorios**

ejem13

  - En base al nombre del método y el tipo de retorno se construye la consulta a la BBDD

```
public interface PostRepository extends JpaRepository<Post, Long> {

    List<Post> findByUsername(String username);

    List<Post> findByTitle(String title);
}
```

Consultas con un único campo como criterio

http://docs.spring.io/spring-data/jpa/docs/current/reference/html/#repositories.query-methods.details

Consultas

- **Métodos en los repositorios**

Consultas combinando varios campos

```
public interface PersonRepository extends Repository<Person, Long> {

	List<Person> findByEmailAddressAndLastname(EmailAddress emailAddress, String lastname);

	// Enables the distinct flag for the query
	List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname, String firstname);
	List<Person> findPeopleDistinctByLastnameOrFirstname(String lastname, String firstname);

	// Enabling ignoring case for an individual property
	List<Person> findByLastnameIgnoreCase(String lastname);

	// Enabling ignoring case for all suitable properties
	List<Person> findByLastnameAndFirstnameAllIgnoreCase(String lastname, String firstname);

	// Enabling static ORDER BY for a query
	List<Person> findByLastnameOrderByFirstnameAsc(String lastname);
	List<Person> findByLastnameOrderByFirstnameDesc(String lastname);

}
```

Ordenación, IgnoreCase, Order...

Consultas

- **Métodos en los repositorios**
  - **Combinación lógica:** And, Or
  - **Comparadores:** Between, LessThan, GreatherThan
  - **Modificadores:** IgnoreCase
  - **Ordenación:** OrderBy…Asc / OrderBy…Desc

Consultas

- **Métodos en los repositorios**
  - **Consulta**
    - List<Elem> find…By…(…)
    - List<Elem> read…By…(…)
    - List<Elem> query…By…(…)
    - List<Elem> get…By…(…)
  - **Número de elementos**
    - int count…By…(…)

Consultas

- **Métodos en los repositorios**
  - Propiedades de los objetos relacionados
    - No sólo podemos filtrar por una propiedad de la entidad, también podemos filtrar por un **atributo de otra entidad** con la que esté relacionada
      - **Person** con un atributo **address**
      - **Address** tiene un atributo **zipCode**

```
List<Person> findByAddressZipCode(ZipCode zipCode);
```

```
List<Person> findByAddress_ZipCode(ZipCode zipCode);
```

Consultas

- **Métodos en los repositorios**
  - Ordenación
    - Podemos pasar un parámetro Sort que controla la ordenación
    - Existe el método findAll(Sort sort)

```
repository.findAll(Sort.by("nombre")));
repository.findAll(Sort.by(Order.asc("nombre"))));
```

    - Podemos añadir métodos personalizados

```
List<User> findByLastname(String lastname, Sort sort);
```

Consultas

- **Métodos en los repositorios**
  - Ordenación
    - El objeto Sort se puede crear desde la URL

```
http://server/anuncios/?sort=nombre,desc
```

```
@GetMapping("/posts/")
public List<Post> getPosts(Sort sort) {

  return posts.findAll(sort);

}
```

Índice de Ejemplos

Índice de Ejemplos

Consultas

code URJC

ejem14

- Estrategias para especificación de consultas con SpringData:
  - Métodos en los repositorios
  - **Java Persistence Query Language (JPQL)**
  - QueryDSL

Consultas

- **Java Persistence Query Language (JPQL)**
  - JPQL es un lenguaje similar a SQL pero referencia conceptos de las entidades JPA

ejem14

```
public interface TeamRepository extends JpaRepository<Team, Long> {

    @Query("select t from Team t where t.name = ?1")
    List<Team> findByName(String name);
}
```

NOTA: En este ejemplo la query JPQL no sería necesaria porque el nombre del método define justo esa consulta

Consultas

- **Java Persistence Query Language (JPQL)**
  - Estructura de sentencia SELECT

```
SELECT ... FROM ...
[WHERE ...]
[GROUP BY ... [HAVING ...]]
[ORDER BY ...]
```

  - Actualización y borrado

```
DELETE FROM ... [WHERE ...]

UPDATE ... SET ... [WHERE ...]
```

Consultas

- **Java Persistence Query Language (JPQL)**
  - Ejemplos

```
SELECT e from Employee e
WHERE e.salary BETWEEN 30000 AND 40000
```

```
SELECT e from Employee e WHERE e.name LIKE 'M%'
```

```
SELECT DISTINCT b FROM Blog b JOIN b.comments c
WHERE c.author=?1
```

Índice de Ejemplos

Índice de Ejemplos

Consultas

- Java Persistence Query Language (JPQL)

ejem15

```
public interface PostRepository extends JpaRepository<Post, Long> {

  @Query("SELECT DISTINCT p FROM Post p JOIN p.comments c WHERE c.username=?1")
  List<Post> findByCommentsUser(String user);

}
```

NOTA: La query JPQL no sería necesaria porque el método del repositorio ejecuta justo esa consulta

Consultas

- **Structured Query Language (SQL)**
  - Incluso podemos usar SQL nativo directamente

```
public interface UserRepository extends JpaRepository<User, Long> {

  @Query(
     value = "SELECT * FROM USERS WHERE EMAIL_ADDRESS = ?1",
     nativeQuery = true)
  User findByEmailAddress(String emailAddress);

}
```

Índice de Ejemplos

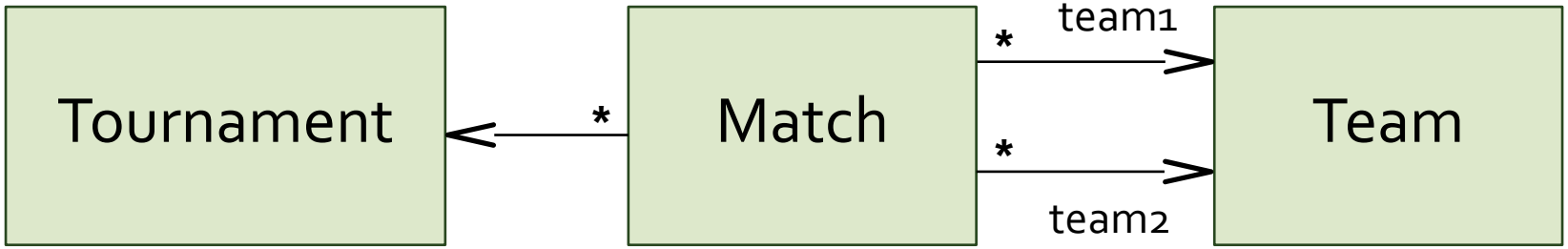
Índice de Ejemplos

Consultas

code URJC

ejem16

- Las consultas permiten relacionar varias entidades sin necesidad de tener atributos directos
  - **Modelo**



  - ¿Qué equipos juegan en un torneo?
  - ¿En qué torneos juega un equipo?

Consultas

code URJC

ejem16

```java
@Entity
public class Tournament {

  @Id
  @GeneratedValue(...)
  private long id;

  private String data;

}
```

```java
@Entity
public class Team {

  @Id
  @GeneratedValue(...)
  private long id;

  private String data;

}
```

```java
@Entity
public class Match {

  @Id
  @GeneratedValue(...)
  private long id;

  private String data;

  @ManyToOne
  private Team team1;

  @ManyToOne
  private Team team2;

  @ManyToOne
  Tournament tournament;
}
```

Consultas

code URJC

ejem16

```
public interface MatchRepository extends JpaRepository<Match, Long> {

  @Query("SELECT m FROM Match m WHERE m.tournament = :t")
  public List<Match> getMatches(Tournament t);

}
```

```
public interface TeamRepository extends JpaRepository<Team, Long> {

  @Query("SELECT distinct team FROM Match m, Team team "
    + "WHERE (m.team1 = team OR m.team2 = team) AND  m.tournament = :t")
  public List<Team> getTeams(Tournament t);

}
```

```
public interface TournamentRepository extends JpaRepository<Tournament, Long> {

  @Query("SELECT distinct t FROM Match m JOIN m.tournament t "
    + "WHERE m.team1 = :team OR m.team2 = :team")
  public List<Tournament> getTournaments(Team team);

}
```

Índice de Ejemplos

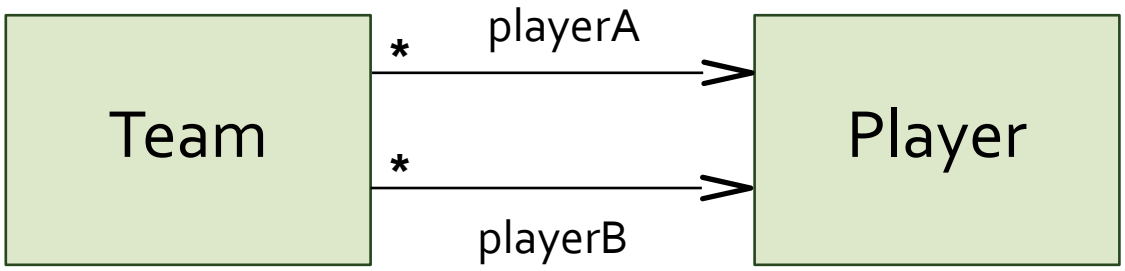
Índice de Ejemplos

Consultas

code URJC

ejem16b

- Ejemplo de consultas avanzadas
  - ¿Qué jugadores juegan conmigo?




```
public interface PlayerRepository extends JpaRepository<Player, Long> {

  @Query("""
    select distinct u from Player u, Team t
    where (t.playerA = u and t.playerB = :player) or (t.playerB = u and t.playerA = :player)
  """)
  List<Player> findPairsOf(Player player);
}
```

Índice de Ejemplos

Índice de Ejemplos

Consultas

code URJC

ejem17

- Estrategias para especificación de consultas con SpringData:
  - Métodos en los repositorios
  - Java Persistence Query Language (JPQL)
  - **QueryDSL**

Consultas

code URJC

- **QueryDSL** **ejem17**
  - Permite implementar las consultas usando código Java
  - A diferencia de JPQL, el **compilador** avisa de **errores** en la consulta
  - Favorece la **refactorización** y el **autocompletado**
  - Se puede usar con **SpringData**

http://www.querydsl.com/

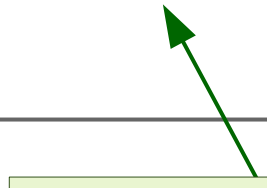
Consultas

- **QueryDSL**

ejem17

  - Partiendo de las entidades se genera código que sirve para hacer las consultas
  - Uso en SpringData

```
public interface TodoRepository extends
    Repository<Todo, Long>, QuerydslPredicateExecutor<Todo> {

}
```



Añadimos otro interfaz padre al repositorio

Consultas

code URJC

- **QueryDSL**

ejem17

  - La consulta se define Partiendo de la clase "Q" generada automáticamente
  - Se define el predicado (filtro)
  - Se usa el método findAll(...) del repositorio

```
@GetMapping("/")
public Iterable<Todo> todos() {

    Predicate q = QTodo.todo.title.eq("Foo");

    return repository.findAll(page);

}
```

Consultas

code URJC

- **QueryDSL**

ejem17

Con un static import de Qtodo.* las sentencias son más concisas

```
todo.title.eq("Foo").and(todo.description.eq("Bar"));
```

```
todo.title.eq("Foo").or(todo.title.eq("Bar"));
```

```
todo.title.eq("Foo").and(todo.description.eq("Bar").not());
```

```
todo.description.containsIgnoreCase(searchTerm)
    .or(todo.title.containsIgnoreCase(searchTerm));
```

Consultas

- QueryDSL

ejem17

```
@GetMapping("/")
public Iterable<Post> getPosts(
  @RequestParam(required = false) String commentsUser) {

  if(commentsUser == null) {
    return posts.findAll();
  } else {
    return posts.findAll(post.comments.any().username.eq(commentsUser));
  }
}
```

Consultas

code URJC

- **QueryDSL**
  - Dependencias

ejem17

```
<dependencies>
  ...
  <dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-core</artifactId>
  </dependency>
  <dependency>
    <groupId>com.querydsl</groupId>
    <artifactId>querydsl-jpa</artifactId>
  </dependency>
</dependencies>
```

Consultas

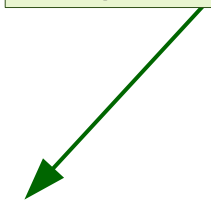
code URJC

- QueryDSL

ejem17

```
<plugin>
  <groupId>com.mysema.maven</groupId>
  <artifactId>apt-maven-plugin</artifactId>
  <version>1.1.3</version>
  <dependencies>
    <dependency>
      <groupId>com.querydsl</groupId>
      <artifactId>querydsl-apt</artifactId>
      <version>4.4.0</version>
    </dependency>
  </dependencies>
  <executions>
    <execution>
      <goals>
        <goal>process</goal>
      </goals>
     <configuration>
        <outputDirectory>target/generated-sources</outputDirectory>
        <processor>com.querydsl.apt.jpa.JPAAnnotationProcessor</processor>
      </configuration>
    </execution>
  </executions>
</plugin>
```

Tarea para generar las clases partiendo de las entidades



Índice de Ejemplos

Índice de Ejemplos

Paginación

code URJC

ejem18

- Con SpringData es muy sencillo que los resultados de las consultas se devuelvan **paginados**
- Basta poner el parámetro **Pageable** en el método del controlador y pasar ese parámetro al método del repositorio
- Cambiar el result de **List**<Element> a **Page**<Element>
- Una Page contiene la información de una **página** (info, nº pág, nº total elementos)
- Se integra con Spring MVC para las **webs** y **APIs REST**

Paginación

ejem18

Devolvemos un objeto **Page<Anuncio>**

En nuestros métodos podemos añadir un parámetro **Pageable**

```
public interface PostRepository extends JpaRepository<Post, Long> {

    Page<Post> findByUsername(String username, Pageable page);

    Page<Post> findByTitle(String title, Pageable page);
}
```

```
Page<Post> a = repository.findByUsername("pepe", PageRequest.of(0, 50));

Page<Post> a = repository.findAll(PageRequest.of(3, 20));
```

El método findAll también tiene versión **Paginable**

Creamos un objeto **PageRequest** para indicar en num página y tamaño

Paginación

code URJC

ejem18

- En el controlador, podemos crear directamente el objeto Pageable como parámetro

```
@RestController
@RequestMapping("/posts")
public class PostController {

  @Autowired
  private PostRepository posts;

  @GetMapping("/")
  public Page<Post> getPosts(
    @RequestParam(required = false) String user, Pageable page) {

    if (user != null) {
      return posts.findByUsername(user, page);
    } else {
      return posts.findAll(page);
    }
  }
}
```

Paginación

code URJC

ejem18

- Con la URL habitual se devuelve la **primera página** con **20 elementos**

```
http://server/posts/
```

- En las peticiones se pueden incluir **parámetros** para solicitar cualquier página

```
http://server/posts/?page=1&size=3
```

- Válido para **webs** y para **APIs REST**

Paginación

- En las APIs REST, el JSON de respuesta incluye información de paginación


```
{
    "content": [
        {
            "id": 1,
            "username": "Pepe",
            "title": "Hola caracola",
            "text": "XXXX"
        },
        {
            "id": 2,
            "username": "Juan",
            "title": "Hola caracola",
            "text": "XXXX"
        }
    ],
    "pageable": {
        "sort": {
            "sorted": false,
            "unsorted": true,
            "empty": true
        },
        "offset": 0,
        "pageNumber": 0,
        "pageSize": 20,
        "paged": true,
        "unpaged": false
    },
    "totalPages": 1,
    "totalElements": 2,
    "last": true,
    "size": 20,
    "number": 0,
    "sort": {
        "sorted": false,
        "unsorted": true,
        "empty": true
    },
    "first": true,
    "numberOfElements": 2,
    "empty": false
}
```

Paginación

- La serialización por defecto en SpringBoot > 1 no es muy adecuada, se puede implementar un conversión a JSON personalizada

```
{
    "content": [
        {
            "id": 1,
            "username": "Pepe",
            "title": "Hola caracola",
            "text": "XXXX"
        },
        {
            "id": 2,
            "username": "Juan",
            "title": "Hola caracola",
            "text": "XXXX"
        }
    ],
    "first": true,
    "last": true,
    "totalPages": 1,
    "totalElements": 2,
    "numberOfElements": 2,
    "size": 20,
    "number": 0,
    "sort": []
}
```

Índice de Ejemplos

Índice de Ejemplos

code URJC

ejem19

```
@JsonComponent
public class PageImplJacksonSerializer extends JsonSerializer<PageImpl<?>> {

	@SuppressWarnings("rawtypes")
	@Override
	public void serialize(PageImpl page, JsonGenerator jsonGenerator, SerializerProvider serializerProvider)
			throws IOException {

		jsonGenerator.writeStartObject();

		jsonGenerator.writeFieldName("content");
		serializerProvider.defaultSerializeValue(page.getContent(), jsonGenerator);

		jsonGenerator.writeBooleanField("first", page.isFirst());
		jsonGenerator.writeBooleanField("last", page.isLast());
		jsonGenerator.writeNumberField("totalPages", page.getTotalPages());
		jsonGenerator.writeNumberField("totalElements", page.getTotalElements());
		jsonGenerator.writeNumberField("numberOfElements", page.getNumberOfElements());

		jsonGenerator.writeNumberField("size", page.getSize());
		jsonGenerator.writeNumberField("number", page.getNumber());

		Sort sort = page.getSort();

		jsonGenerator.writeArrayFieldStart("sort");

		for (Sort.Order order : sort) {
			jsonGenerator.writeStartObject();
			jsonGenerator.writeStringField("property", order.getProperty());
			jsonGenerator.writeStringField("direction", order.getDirection().name());
			jsonGenerator.writeBooleanField("ignoreCase", order.isIgnoreCase());
			jsonGenerator.writeStringField("nullHandling", order.getNullHandling().name());
			jsonGenerator.writeEndObject();
		}

		jsonGenerator.writeEndArray();
		jsonGenerator.writeEndObject();
	}
}
```

https://www.wimdeblauwe.com/blog/2018/2018-06-10-pageimpl-json-serialization-with-spring-boot-2/

Consultas

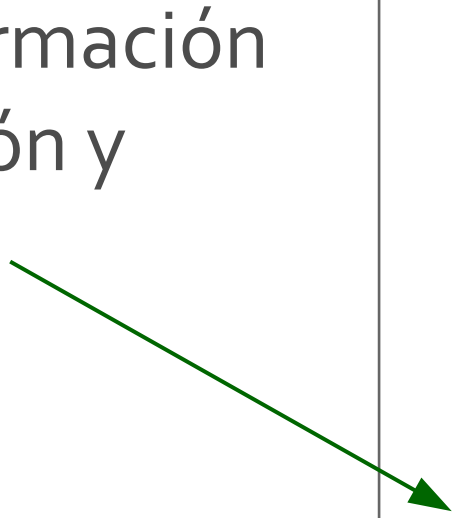
- **Métodos en los repositorios**
  - Ordenación
    - El objeto Pageable incluye la información de ordenación de la URL

```
http://server/posts/?page=1&size=3&sort=username,desc
```

```
@GetMapping("/posts/")
public Page<Post> getPost(Pageable page) {

  return posts.findAll(page);

}
```

Consultas

- En las APIs REST, el JSON de respuesta incluye información de paginación y ordenación



```
{
    "content": [
        {
            "id": 1,
            "username": "Pepe",
            "title": "Hola caracola",
            "text": "XXXX"
        },
        {
            "id": 2,
            "username": "Juan",
            "title": "Hola caracola",
            "text": "XXXX"
        }
    ],
    "first": true,
    "last": true,
    "totalPages": 1,
    "totalElements": 2,
    "numberOfElements": 2,
    "size": 3,
    "number": 0,
    "sort": [
        {
            "property": "username",
            "direction": "DESC",
            "ignoreCase": false,
            "nullHandling": "NATIVE"
        }
    ]
}
```

Índice de Ejemplos

Índice de Ejemplos

Gestión del esquema

code URJC

ejem20

- En las **Bases de datos SQL** es necesario **crear el esquema** antes de insertar los datos (crear tablas, relaciones, etc…)
- En las bases de **datos en memoria** (H2, Derby, HSQL…), el esquema siempre se construye de forma **automática** al iniciar la aplicación
- Cuando se usa una base de datos en producción los datos tienen que guardarse en disco y gestionar adecuadamente la **creación y actualización del esquema**

MySQL

code URJC

ejem20

- Instalación en ubuntu
- Servidor

```
$ sudo apt-get install mysql-server
```

- Herramienta interactiva

```
$ sudo apt-get install mysql-workbench
```

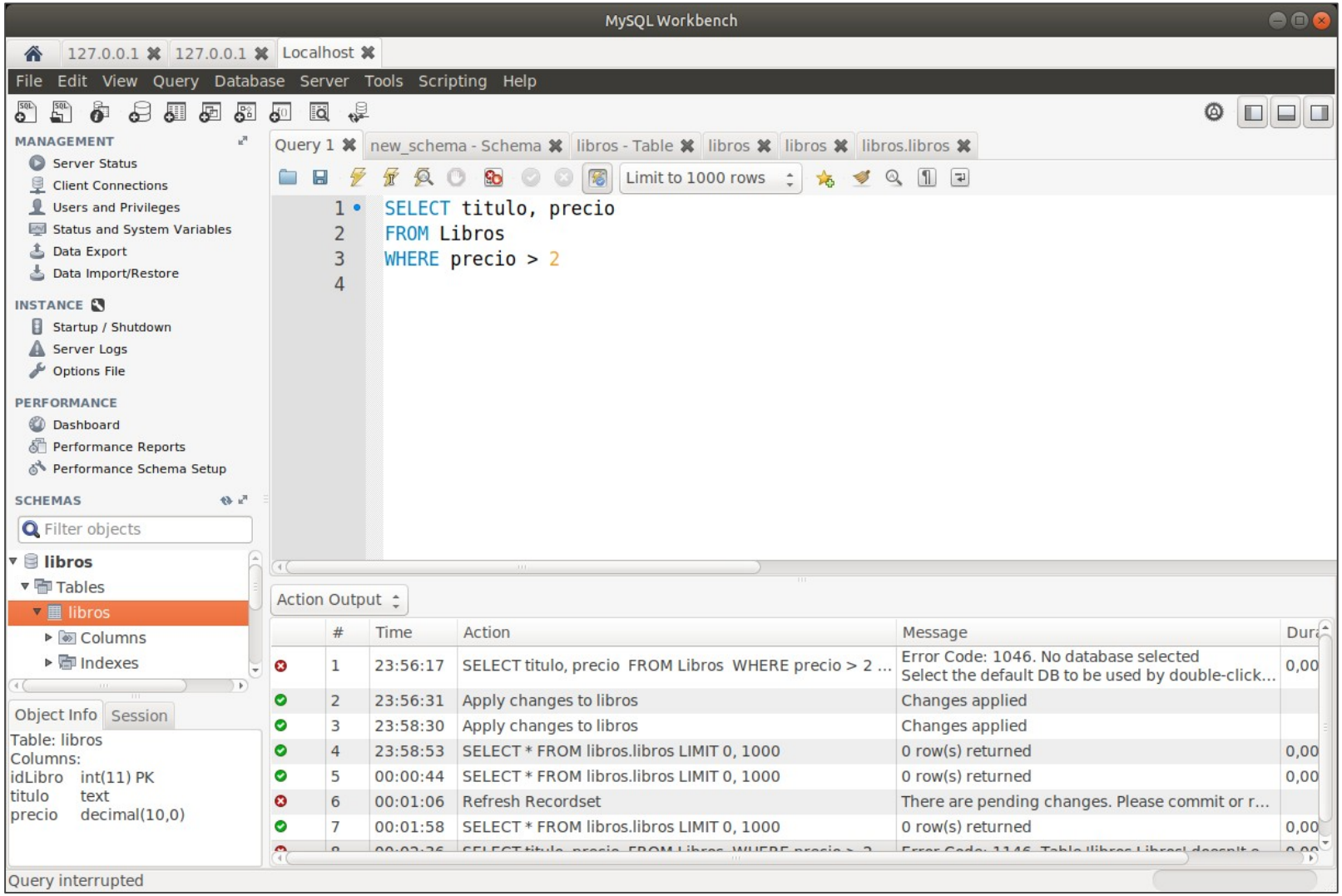
- Docker

```
$ docker run --rm -e MYSQL_ROOT_PASSWORD=password \
  -e  MYSQL_DATABASE=posts -p 3306:3306 -d mysql:8.0.22
```

http://dev.mysql.com/downloads/

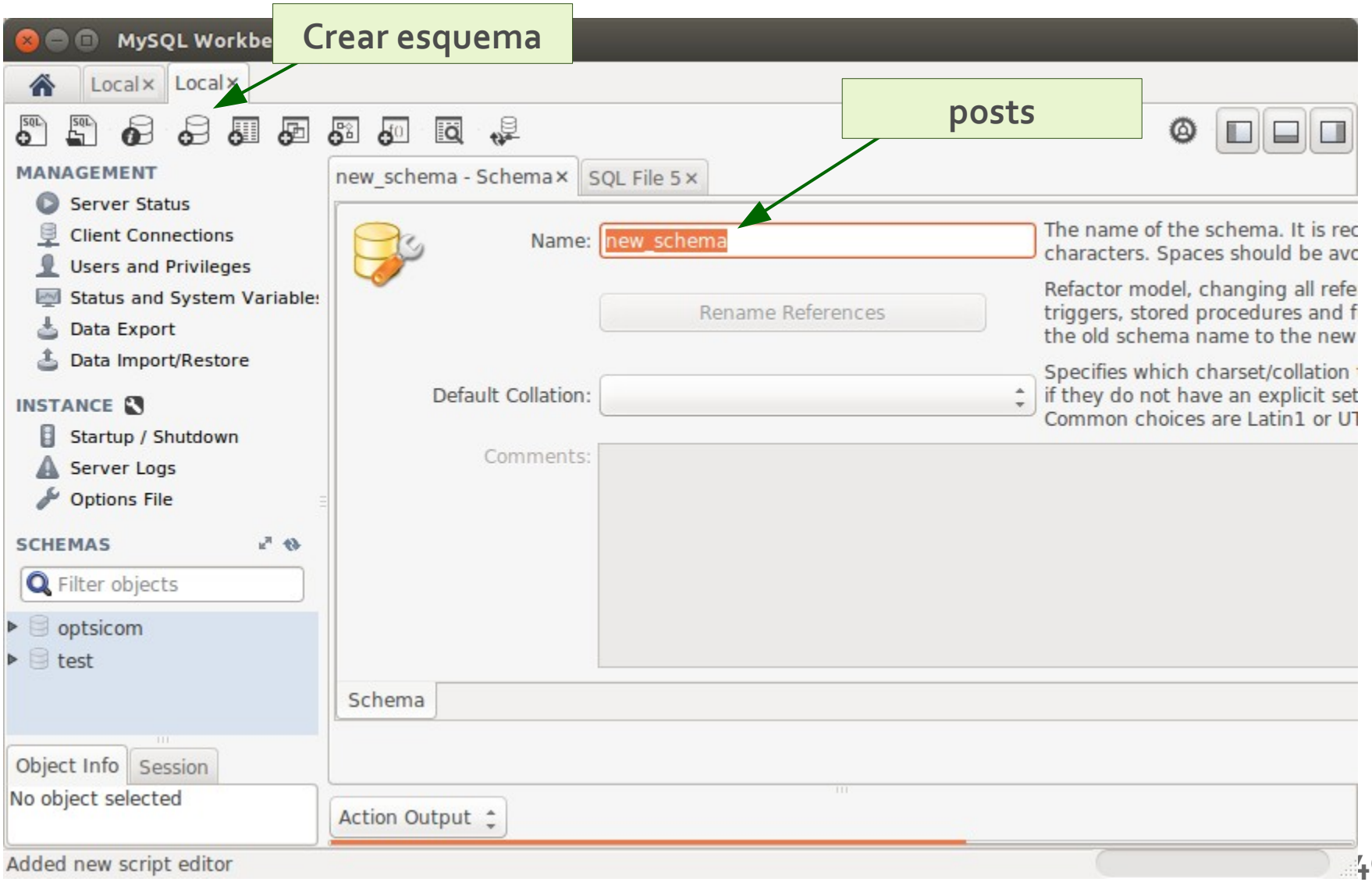
MySQL

code URJC



MySQL

code URJC



MySQL

ejem20

- pom.xml

```
<groupId>es.codeurjc</groupId>
<artifactId>bbdd_ejem16</artifactId>
<version>0.1.0</version>

<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>2.4.0</version>
</parent>
...
<dependencies>
  ...
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
  </dependency>
  <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
  </dependency>
</dependencies>
```

Dependencia a **Spring Data** para BBDD relacionales

Dependencia al driver **MySQL**

MySQL

code URJC

ejem20

- Datos de conexión a la BBDD

Este fichero tiene que estar en el fichero **src/main/resources**

Host en el que está alojado el servidor MySQL

Nombre del **esquema**. Este esquema tiene que ser creado manualmente por el desarrollador usando herramientas de MySQL

application.properties

```
spring.datasource.url=jdbc:mysql://localhost/posts
spring.datasource.username=root
spring.datasource.password=pass
spring.jpa.hibernate.ddl-auto=...
```

Usuario y contraseña. En algunos SO estos datos se configuran al instalar MySQL

Gestión del esquema

code URJC

ejem20

- **Propiedad de generación del esquema**

```
spring.jpa.hibernate.ddl-auto=...
```

- **create-drop:** Crea el esquema al iniciar la aplicación y le borra al finalizar (ideal para programar como si la BBDD estuviera en memoria)
- **create:** Crea el esquema al iniciar la aplicación
- **update:** Añade al esquema actual las tablas y atributos necesarios para hacer el esquema compatible con las clases Java (no borra ningún elemento). Si el esquema está vacío, se genera completo.
- **validate:** Verifica que el esquema de la BBDD es compatible con las entidades de la aplicación y si no lo es genera un error.
- **none:** No hace nada con el esquema y asume que es correcto.

Índice de Ejemplos

Índice de Ejemplos

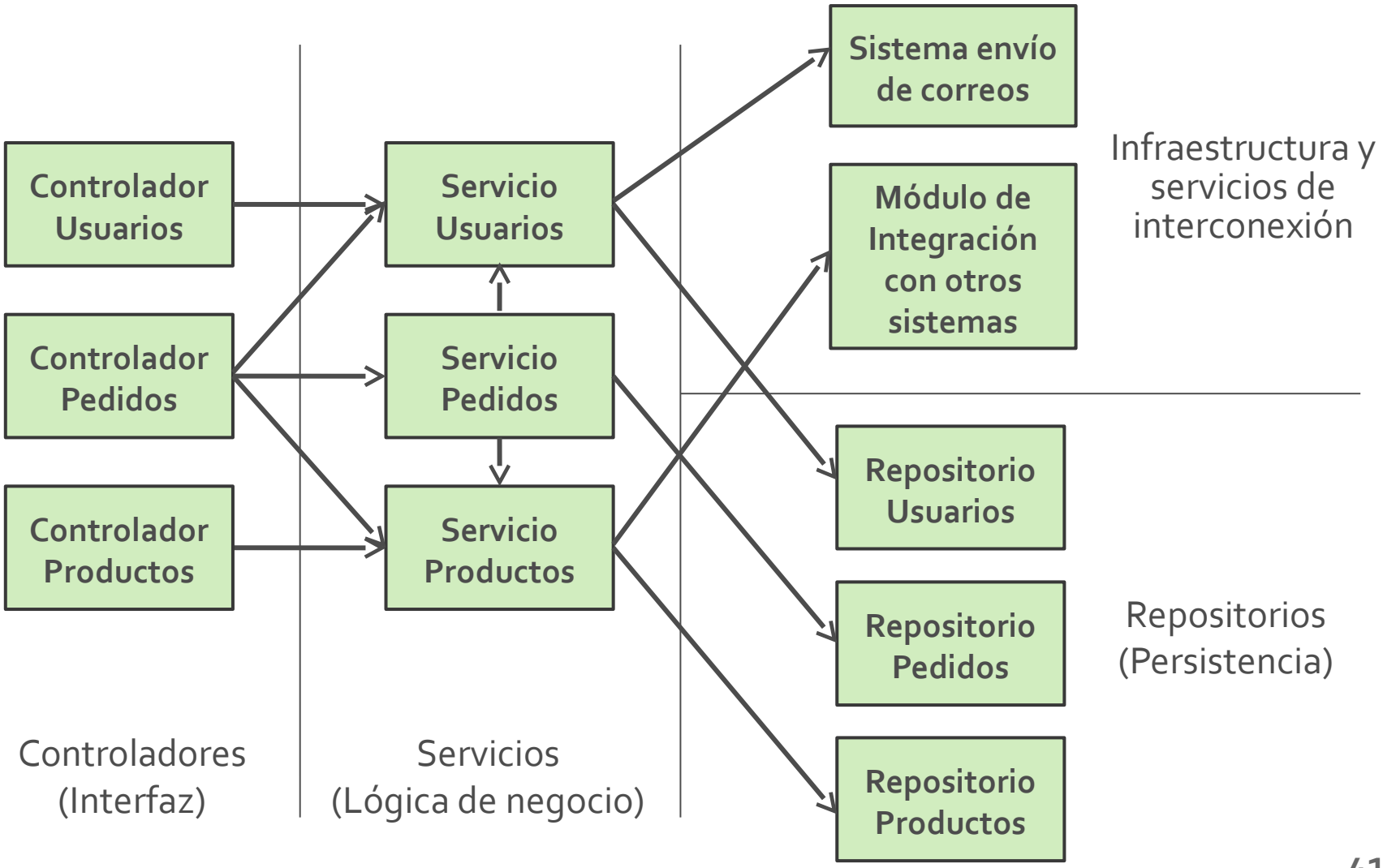
Arquitectura

code URJC

ejem21

- La **arquitectura de software** define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación entre ellos.
- Existen diferentes **estilos arquitectónicos** que pueden ser adecuados para diferentes tipos de aplicaciones

Arquitectura



Arquitectura

code URJC

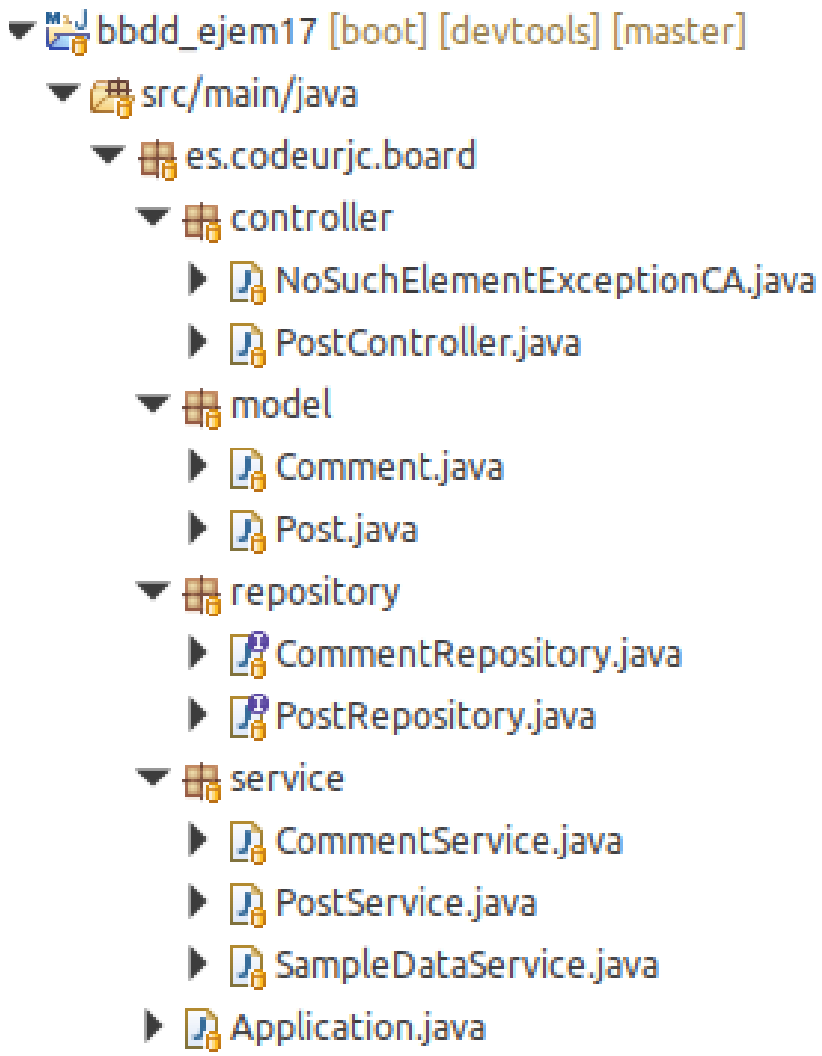
ejem21

- **Ventajas de esta arquitectura:**
  - Los **servicios** pueden testearse de forma unitaria (con dobles de repositorios y otros servicios)
  - Los **controladores** pueden testearse de forma unitaria (con doble del servidor web y de los servicios)
  - La **lógica de negocio** se puede reutilizar entre varios tipos de controladores (web, REST, línea de comandos…)

Arquitectura

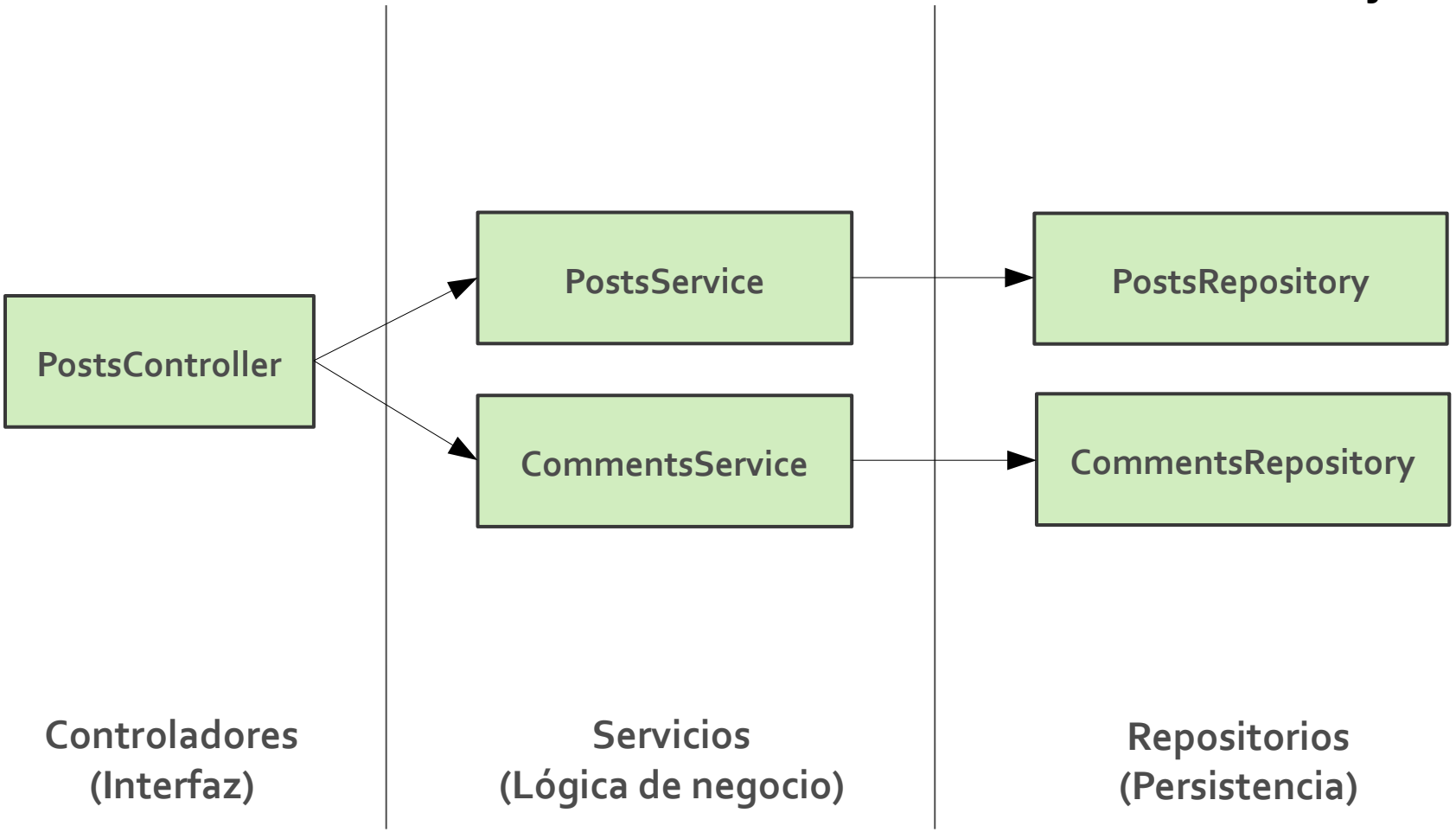
code URJC

ejem21



Arquitectura

ejem21



Índice de Ejemplos

Índice de Ejemplos

Imágenes

code URJC

ejem22

- En vez de guardar las **imágenes** en disco, se pueden **guardar en la BD**
- Eso favorece la **escalabilidad** porque las aplicaciones son ***stateless*** (no tienen estado) y se pueden replicar en **diferentes máquinas**
- Simplifica el despliegue en entornos en los que **no se tiene acceso al disco duro** de forma persistente (Heroku, Kubernetes…)
- Otra opción es usar **servicios de persistencia de ficheros** (S3, Minio…)

Imágenes

code URJC

ejem22

- Atributo Blob (*Binary Large Object*)

```
@Entity
public class Post {

    @Id
    @GeneratedValue(...)
    private long id;

    private String username;
    private String title;
    private String text;
    private String image;

    @Lob
    @JsonIgnore
    private Blob imageFile;

    ...
}
```

Imágenes

code URJC

ejem22

- Upload Image

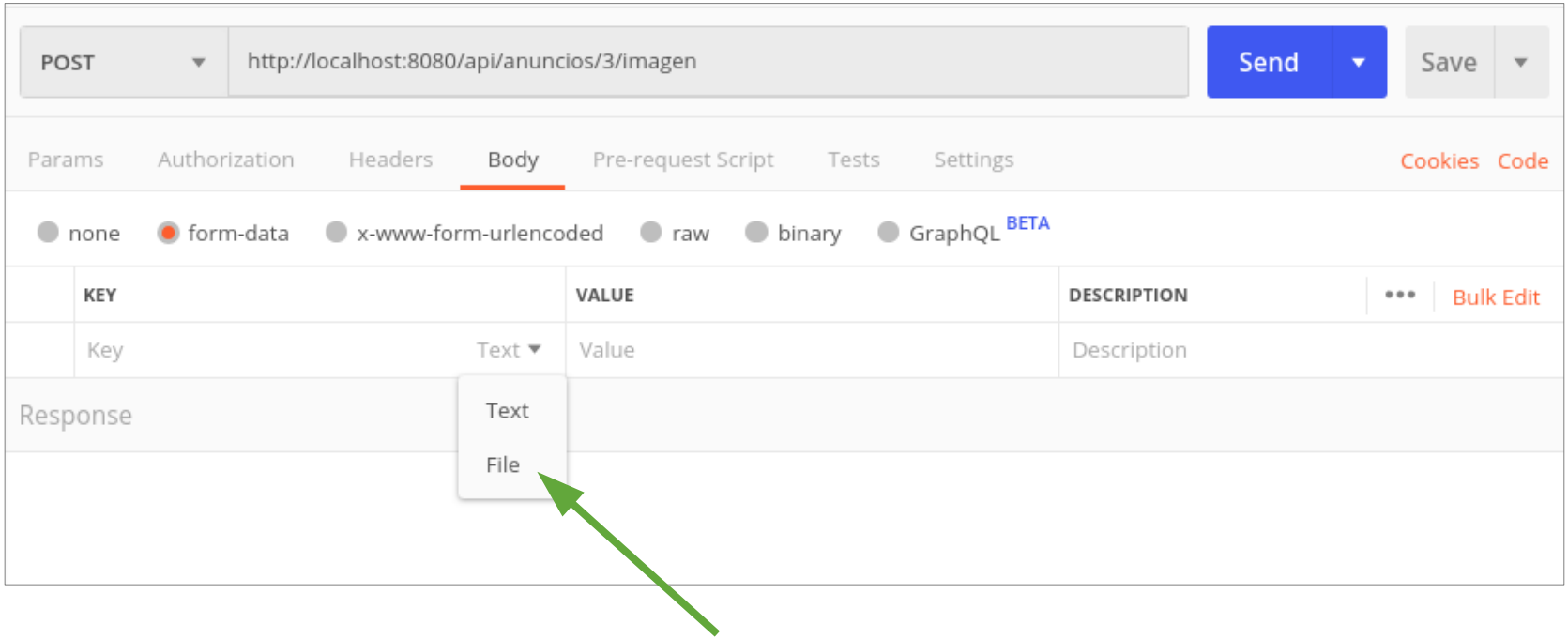
```
@PostMapping("/{id}/image")
public ResponseEntity<Object> uploadImage(@PathVariable long id,
  @RequestParam MultipartFile imageFile) throws IOException {

  Post post = posts.findById(id).orElseThrow();

  URI location = fromCurrentRequest().build().toUri();

  post.setImage(location.toString());

  post.setImageFile(BlobProxy.generateProxy(
    imageFile.getInputStream(), imageFile.getSize()));

  posts.save(post);

  return ResponseEntity.created(location).build();
}
```

Imágenes

code URJC

ejem22

- Subir un fichero con Postman

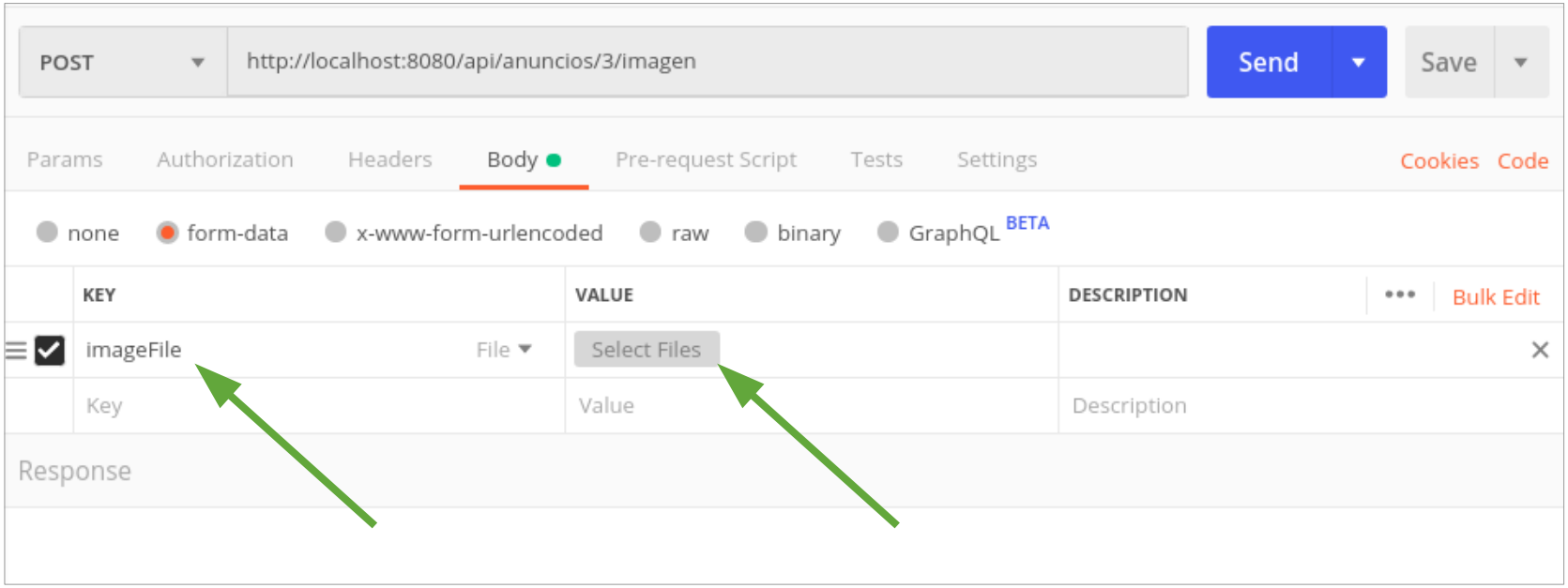


Imágenes

code URJC

ejem22

- Subir un fichero con Postman



Imágenes

code URJC

ejem22

- Download Image

```
@GetMapping("/{id}/image")
public ResponseEntity<Object> downloadImage(@PathVariable long id)
  throws SQLException {

  Post post = posts.findById(id).orElseThrow();

  if (post.getImageFile() != null) {

    Resource file = new InputStreamResource(
      post.getImageFile().getBinaryStream());

    return ResponseEntity.ok()
      .header(HttpHeaders.CONTENT_TYPE, "image/jpeg")
      .contentLength(post.getImageFile().length())
      .body(file);

  } else {
    return ResponseEntity.notFound().build();
  }
}
```

Imágenes

code URJC

ejem22

- Delete Image

```
@DeleteMapping("/{id}/image")
public ResponseEntity<Object> deleteImage(@PathVariable long id)
  throws IOException {

  Post post = posts.findById(id).orElseThrow();

  post.setImageFile(null);
  post.setImage(null);

  posts.save(post);

  return ResponseEntity.noContent().build();
}
```

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 1

- Transforma el ejercicio de la **gestión de Items** para que utilice una base de datos en vez de guardar la información en memoria

Índice de Ejercicios

Ejercicio 2

- Añade paginación a la página web de gestión de **posts**, en la página principal

01-2023

code
URJC

Desarrollo de Aplicaciones Web

Tema 4 – Construcción y despliegue en Spring

Índice de Ejemplos

- Ejemplo 1 (Construcción en Spring Boot)

Índice de Ejemplos

- Ejemplo 1 (Construcción en Spring Boot)

Construcción en Spring Boot

code URJC

- Actualmente se recomienda usar **.jar**
- Incluye un servidor web integrado (**Tomcat**, **Jetty**, **Undertow**, **Netty**…)
- Facilita la evolución de la aplicación (al no estar limitada por el servidor en el que se despliegan los .war)

Apache Tomcat

Jar con Spring Boot

code URJC

- **Añadir el plugin de Maven de Spring Boot encargado de empaquetar el .jar**
  - Crea un fichero .jar con todas las librerías de la aplicación y el servidor web en la carpeta **target**

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
</build>
```

Jar con Spring Boot

- **¿Cómo generar el .jar?**
  - Desde eclipse
    - Botón derecho proyecto > Run as…> Maven install
  - Desde la línea de comandos
    - (en la raíz del proyecto) mvn package
- **Resultado**
  - Fichero <nombreproyecto>_<version>.jar en la carpeta target

```
despliegue_ejem1-0.0.1.jar
```

Jar con Spring Boot

- **Ejecutar la app web con el .jar**
  - Es requisito tener instalado un Java JRE
  - Ejecutamos el comando

```
$ java -jar despliegue_ejem1-0.0.1.jar
```

  - Para finalizar la aplicación ejecutar **Ctrl+C** en la consola (se envía una señal de apagado **SIGTERM**)
  - También se puede habilitar una **URL REST** para **apagar en remoto** (protegida por contraseña)

Jar con Spring Boot

- **Configuración de la aplicación**
  - Al ejecutar la aplicación se pueden sobreescribir las propiedades de configuración del fichero **application.properties** con:
    - 1) Parámetros de la línea de comandos
    - 2) Variables de entorno
    - 3) Fichero application.properties junto al fichero .jar

http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-external-config.html

Jar con Spring Boot

code URJC

- **Configurar la app web con el .jar**
  - Configuración de puerto

```
$ java -jar despliegue_ejem1-0.0.1.jar --server.port=8081
```

  - Configuración de base de datos

```
$ java -jar despliegue_ejem2-0.0.1.jar \
  --spring.datasource.url=jdbc:mysql://localhost/test \
  --spring.datasource.username=root \
  --spring.datasource.password=pass \
  --spring.jpa.hibernate.ddl-auto=create-drop
```

Jar con Spring Boot

- **Perfiles**
  - Es habitual que tengamos valores de propiedades diferentes en varios entornos (desarrollo y producción)
  - Podemos tener diferentes ficheros de configuración
    - application-prod.properties
    - application-dev.properties
  - Al ejecutar la aplicación se selecciona el perfil

```
$ java -jar -Dspring.profiles.active=prod despliegue_ejem2-0.0.1.jar
```

http://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-profiles.html

code
URJC

Desarrollo de Aplicaciones Web

Tema 5 – Seguridad en Spring

Índice de Ejemplos

Índice de Ejemplos

1) Comunicación cifrada con https

code URJC

ejem1

- No es necesario **spring-security** porque esta funcionalidad la ofrece el servidor web **Tomcat (incluido en nuestra aplicación)**
- Usaremos un **certificado autofirmado** que generará un aviso de seguridad en el navegador

1) Comunicación cifrada con https

code URJC

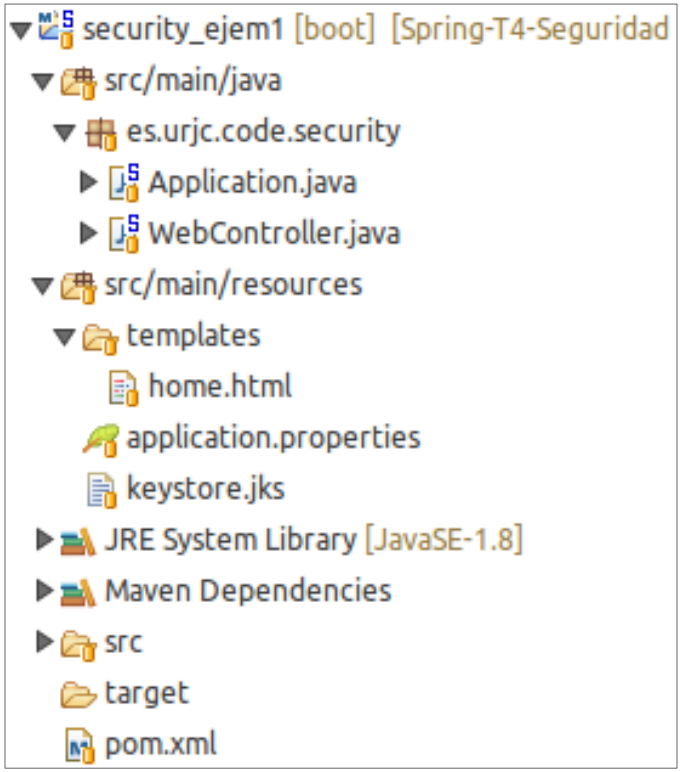
ejem1

- Usando un **certificado autofirmado** podríamos sufrir un ataque y nuestros datos podrían ser descifrados
- Si usamos un **certificado de una CA,** nuestros datos no podrán ser descifrados ni alterados, pero necesitaríamos un **dominio** y un **servidor**

1) Comunicación cifrada con https

ejem1

- No necesitamos dependencias especiales



pom.xml

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

1) Comunicación cifrada con https

ejem1

- **Generamos un certificado con keytool**

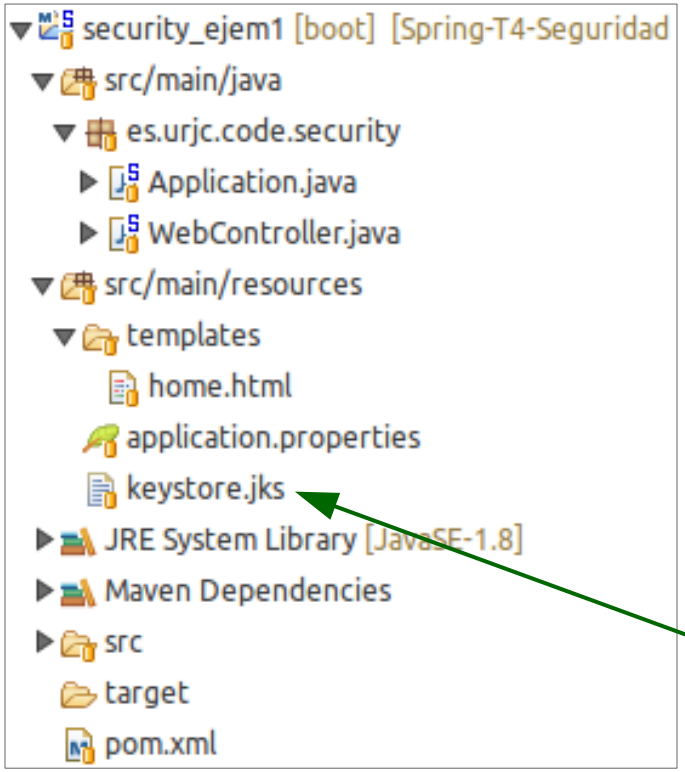
Se puede contestar cualquier valor. La contraseña debe ser **secret**

```
$ cd $JAVA_HOME/bin
$ keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass password
-validity 360 -keysize 2048
¿Cuáles son su nombre y su apellido?
  [Unknown]:  Cualquier nombre
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  Cualquier unidad
¿Cuál es el nombre de su organización?
  [Unknown]:  URJC
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  Madrid
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:  Madrid
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  ES
¿Es correcto CN=Micael Gallego, OU=Code, O=URJC, L=Madrid, ST=Madrid, C=ES?
  [no]:  si
Introduzca la contraseña de clave para <selfsigned>
        (INTRO si es la misma contraseña que la del almacén de claves): secret
Volver a escribir la contraseña nueva: secret
```

1) Comunicación cifrada con https

ejem1

- Configuramos puerto y certificado



Para indicar que es accesible por https en **desarrollo** se usa el puerto **8443**. En **producción** se usa el puerto **443**

application.properties

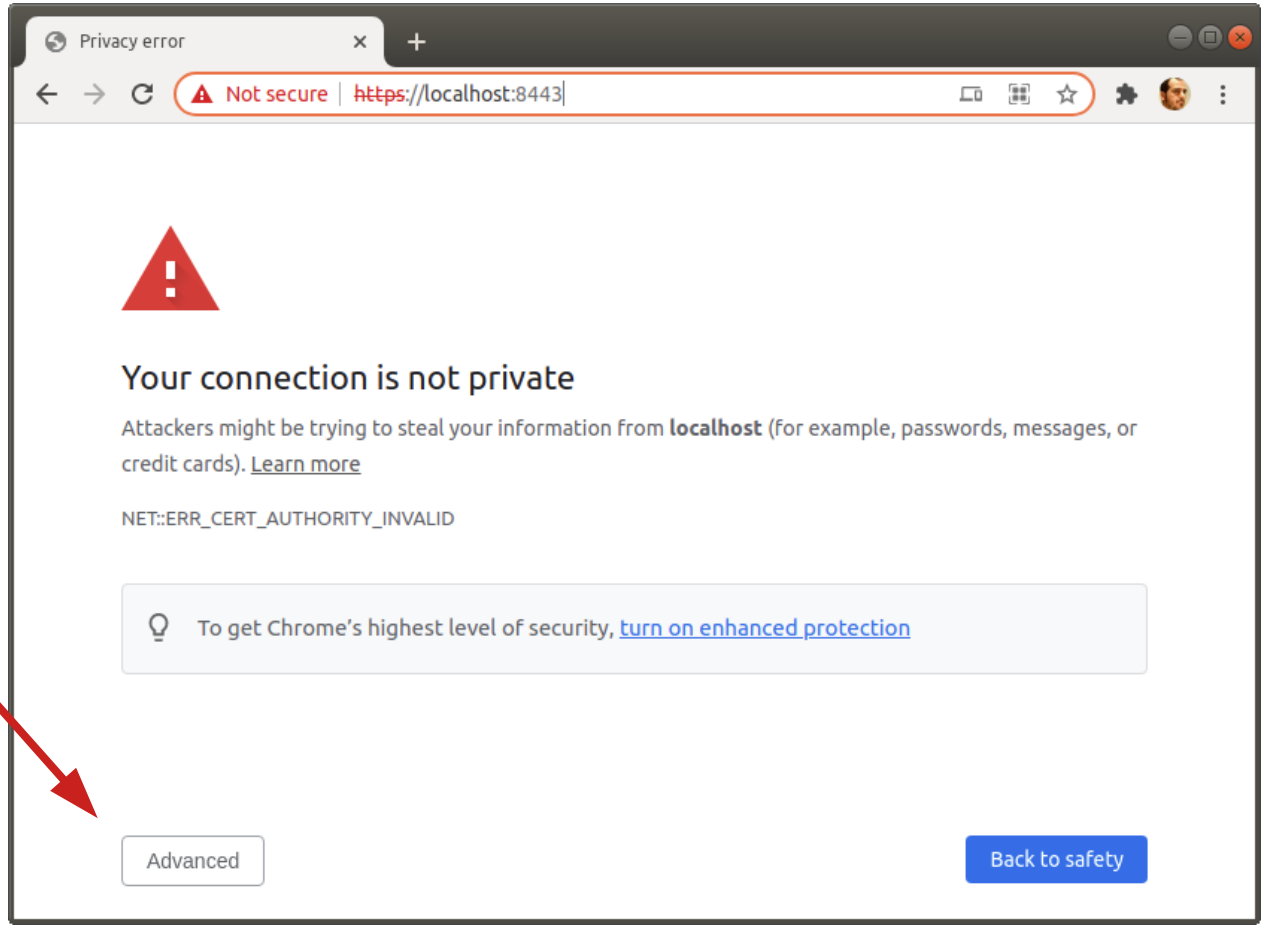
```
server.port = 8443
server.ssl.key-store = classpath:keystore.jks
server.ssl.key-store-password = password
server.ssl.key-password = secret
```

Fichero que contiene el certificado autofirmado generado con la herramienta del JDK **keytool**

1) Comunicación cifrada con https

code URJC

ejem1

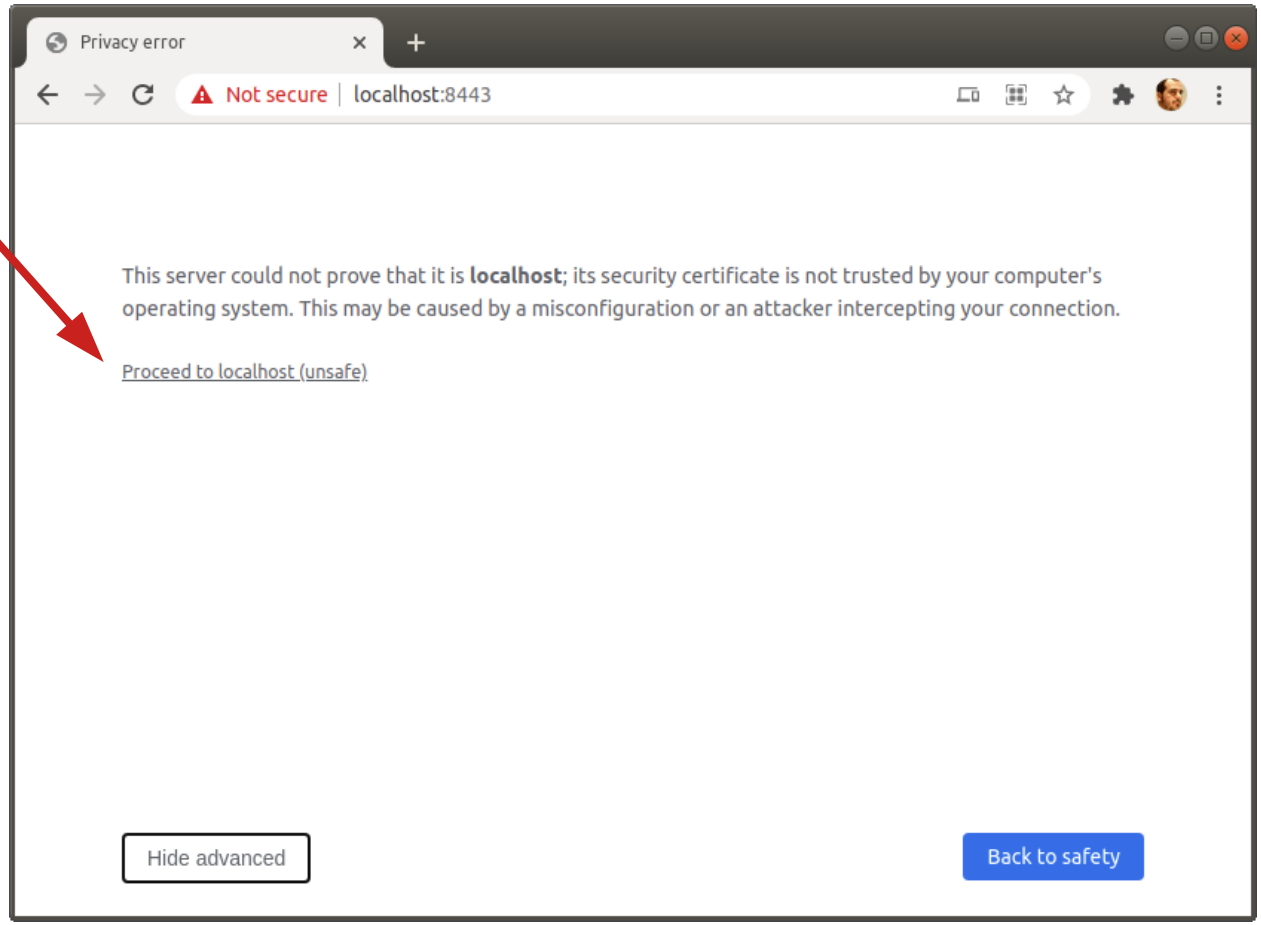


https://localhost:8443

1) Comunicación cifrada con https

code URJC

ejem1

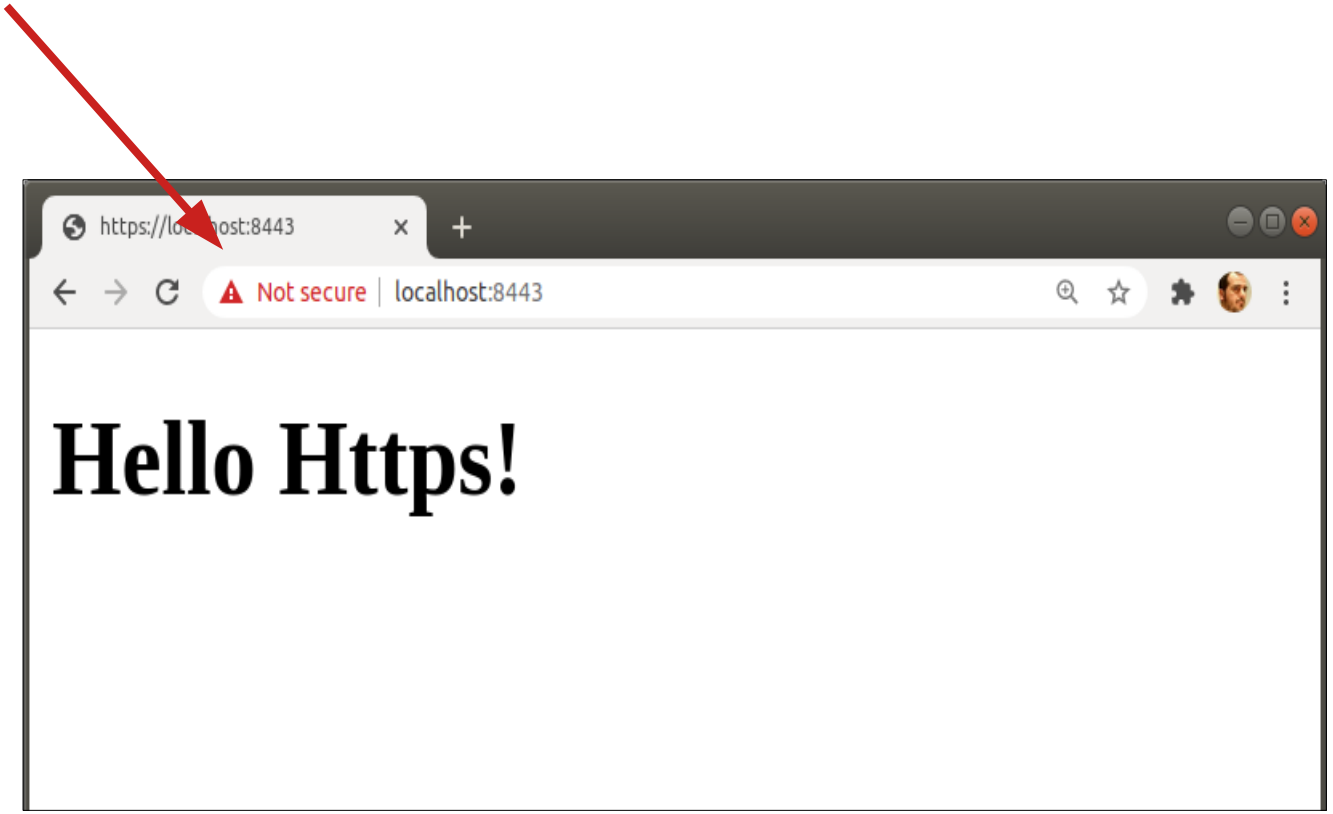


https://localhost:8443

1) Comunicación cifrada con https

code URJC

ejem1



https://localhost:8443

Índice de Ejemplos

2) Usuario con credenciales en código

code URJC

ejem2

- **Spring-security** permite a las aplicaciones web y a las APIs REST gestionar **usuarios**
  - **Autenticación:** Permite autenticar a los usuarios con contraseña u otros mecanismos
  - **Autorización:** Se puede configurar a qué URLs tienen acceso qué tipos de usuarios (**cualquier** usuario, sólo **autenticados** o sólo los que tengan un **rol** determinado (profesor, alumno…)

2) Usuario con credenciales en código

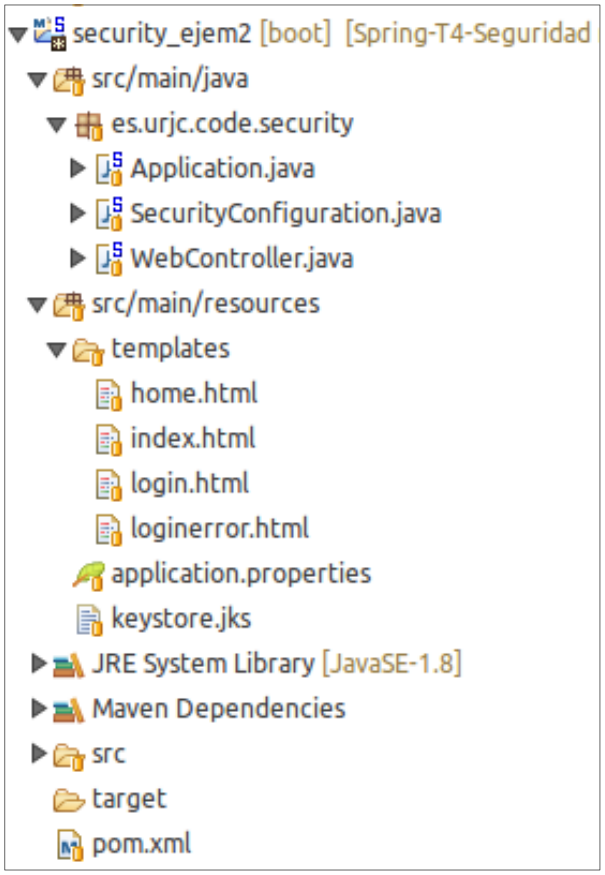
code URJC

ejem2

- Existen diversas formas de **gestionar las credenciales** de los usuarios (contraseña):
  - **Un único usuario:** Con la contraseña en código o en un fichero de configuración
  - **Varios usuarios:** Con la contraseña en la base de datos

2) Usuario con credenciales en código

ejem2



pom.xml

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

Dependencia spring-security

2) Usuario con credenciales en código

WebController.java

ejem2

```
@Controller
public class WebController {

    @GetMapping("/")
    public String index() {
        return "index";
    }

    @GetMapping("/login")
    public String login() {
     return "login";
    }

    @GetMapping("/loginerror")
    public String loginerror() {
     return "loginerror";
    }

    @GetMapping("/private")
    public String privatePage() {
        return "private";
    }
}
```

Controlador de ejemplo que asocia cada URLs a una plantilla diferente

2) Usuario con credenciales en código

code URJC

ejem2

- Configuración del usuario en código

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

  ...

  @Override
  protected void configure(AuthenticationManagerBuilder auth) throws Exception {

    PasswordEncoder encoder =
      PasswordEncoderFactories.createDelegatingPasswordEncoder();

    String encodedPassword = encoder.encode("pass");

    auth.inMemoryAuthentication()
      .withUser("user").password(encodedPassword).roles("USER");
  }
}
```

Se cifra la password por seguridad (se verá más adelante)

2) Usuario con credenciales en código

code URJC

ejem2

- Configuraciones de seguridad

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

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

      // Security configuration

      // - Public and private pages
      // - Login and logout
      // - Other security configurations
    }
    ...
}
```

2) Usuario con credenciales en código

ejem2

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

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

      // Public pages
      http.authorizeRequests().antMatchers("/").permitAll();
      http.authorizeRequests().antMatchers("/login").permitAll();
      http.authorizeRequests().antMatchers("/loginerror").permitAll();
      http.authorizeRequests().antMatchers("/logout").permitAll();

      // Private pages (all other pages)
      http.authorizeRequests().anyRequest().authenticated();
      ...

    }
    ...
}
```

Las demás URLs son privadas (sólo para usuarios autenticados)

URLs públicas (permitidas a todos los usuarios, autenticados o no)

2) Usuario con credenciales en código

code URJC

ejem2

- **Formulario de autenticación (login)**
  - El desarrollador crea el **formulario** de autenticación (**usuario** y **contraseña**)
  - También crea la **página de error** en caso de que las credenciales sean incorrectas
  - Spring proporciona el controlador web que **verifica usuario y contraseña** (no tiene que implementarse)

2) Usuario con credenciales en código

ejem2

- Formulario de autenticación (login)

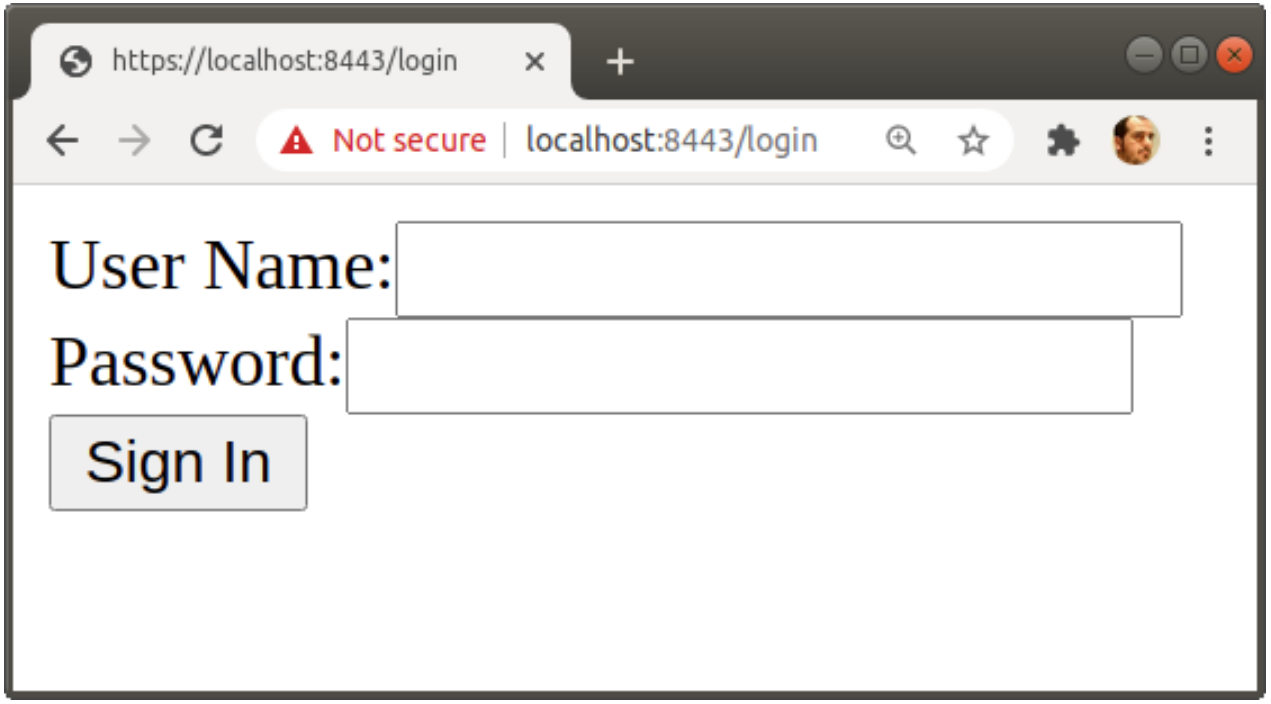
```
<!DOCTYPE html>
<html>
<body>
<form action="/login" method="post">
  <label>User Name:</label>
  <input type="text" name="username" /><br/>
  <label>Password:</label>
  <input type="password" name="password" /><br/>
  <input type="submit" value="Sign In" /><br/>
</form>
</body>
</html>
```

2) Usuario con credenciales en código

code URJC

ejem2

- Formulario de autenticación (login)



2) Usuario con credenciales en código

code URJC

ejem2

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

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


      ...

      // Login form
      http.formLogin().loginPage("/login");
      http.formLogin().usernameParameter("username");
      http.formLogin().passwordParameter("password");
      http.formLogin().defaultSuccessUrl("/private");
      http.formLogin().failureUrl("/loginerror");

      ...

    }
    ...
}
```

URLs del formulario de login

Nombres de los campos del formulario

URL a la que navegar si se autentica correctamente

URL a la que navegar si hay error en la autenticación

2) Usuario con credenciales en código

ejem2

- **Acceso a URLs privadas**
  - Si un usuario no autenticado **navega a una URL que requiere autenticación**, Spring le lleva de forma automática al formulario de login
  - Si se **autentica** correctamente, navega automáticamente a la **URL solicitada**
  - Si no, **navega a la página de error**

2) Usuario con credenciales en código

code URJC

ejem2

2) Usuario con credenciales en código

ejem2

- **Elimiar autenticación (logout)**
  - Se puede crear un formulario apuntando a una URL especial para cerrar la sesión del usuario

```
<!DOCTYPE html>
<html>
<body>
    <p>This is a private page</p>
    <form action="/logout" method="post">
        <input type="submit" value="Sign Out" />
    </form>
</body>
</html>
```

2) Usuario con credenciales en código

code URJC

ejem2

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

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

      ...

      // Logout
      http.logout().logoutUrl("/logout");
      http.logout().logoutSuccessUrl("/");


      // Disable CSRF at the moment
      http.csrf().disable();

      ...

    }
    ...
}
```

URL para hacer logout

URL a la que navegar cuando se hace logout

Desactivar protección CSRF (se verá más adelante)

2) Usuario con credenciales en código

code URJC

ejem2

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

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

        // Public pages
        http.authorizeRequests().antMatchers("/").permitAll();
        http.authorizeRequests().antMatchers("/login").permitAll();
        http.authorizeRequests().antMatchers("/loginerror").permitAll();
        http.authorizeRequests().antMatchers("/logout").permitAll();

        // Private pages (all other pages)
        http.authorizeRequests().anyRequest().authenticated();

        // Login form
        http.formLogin().loginPage("/login");
        http.formLogin().usernameParameter("username");
        http.formLogin().passwordParameter("password");
        http.formLogin().defaultSuccessUrl("/private");
        http.formLogin().failureUrl("/loginerror");

        // Logout
        http.logout().logoutUrl("/logout");
        http.logout().logoutSuccessUrl("/");

        // Disable CSRF at the moment
        http.csrf().disable();
    }
    ...
}
```

URLs públicas

URLs privadas (las demás)

Configuración del formulario de login

Configuración de la página de logout

Deshabilitamos CSRF

2) Usuario con credenciales en código

ejem2

- **URLs que tienen que ser públicas:**
  - **URL de login:** Si no lo es, al navegar de forma automática se obtendrá un error de acceso
  - **URL de error en login:** Si no lo es, se obtendrá un error de acceso cuando las credenciales no sean correctas

2) Usuario con credenciales en código

code URJC

ejem2

- **¿Cómo se identifica al usuario después de hacer login?**
  - El navegador **recibe una cookie** (JSESSIONID) con un identificador de sesión
  - En cada petición posterior se **envía esa cookie**
  - El servidor **guarda en memoria** las cookies de los usuarios con la sesión activa

Set-Cookie: JSESSIONID=666E4BBCA9049796E74636DDCC849C91; Path=/; Secure; HttpOnly

Índice de Ejemplos

3) Usuario con credenciales en fichero

code URJC

ejem3

- **Password cifrada con BCrypt**
  - Se puede generar en varios sitios web

    https://bcryptgenerator.com/
  - Se configura en application.properties
  - Cuidado con los espacios al final

application.properties

```
...

security.user=user
security.encodedPassword=$2y$10$Ig4HQf20NDxKwyyU5pAzpuNHqcK5...
```

3) Usuario con credenciales en fichero

code URJC

ejem3

- Se obtiene credenciales del fichero

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Value("${security.user}")
    private String user;

    @Value("${security.encodedPassword}")
    private String encodedPassword;

    ...

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {

     PasswordEncoderFactories.createDelegatingPasswordEncoder();

          auth.inMemoryAuthentication()
               .withUser(user).password("{bcrypt}"+encodedPassword).roles("USER");
    }
}
```

Se cargan en atributos los valores del fichero de configuración

Índice de Ejemplos

4) Diferentes tipos de usuarios (roles)

code URJC

ejem4

- Damos de alta dos usuarios, uno con rol USER y otro ADMIN

```
@Override
protected void configure(AuthenticationManagerBuilder auth)
  throws Exception {

  PasswordEncoder encoder =
    PasswordEncoderFactories.createDelegatingPasswordEncoder();

  auth.inMemoryAuthentication()
    .withUser("user").password(encoder.encode("pass"))
    .roles("USER");

  auth.inMemoryAuthentication()
    .withUser("admin").password(encoder.encode("adminpass"))
    .roles("USER", "ADMIN");

}
```

4) Diferentes tipos de usuarios (roles)

code URJC

ejem4

- Configuramos las URLs que puede ver cada tipo de usuario

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

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

      ...
      http.authorizeRequests().antMatchers("/private").hasAnyRole("USER");
      http.authorizeRequests().antMatchers("/admin").hasAnyRole("ADMIN");
      ...
   }
}
```

4) Diferentes tipos de usuarios (roles)

code URJC

ejem4

- En el controlador podemos saber el nombre y el rol del usuario que se ha logueado

```
@GetMapping("/private")
public String privatePage(Model model, HttpServletRequest request) {

  model.addAttribute("username", request.getUserPrincipal().getName());
  model.addAttribute("admin", request.isUserInRole("ADMIN"));

  return "private";
}
```

```
<p>Hello {{username}}. This is a private page</p>

{{#admin}}
<a href="/admin">Admin Page</a>
{{/admin}}
```

4) Diferentes tipos de usuarios (roles)

code URJC

ejem4

- Si un usuario intenta acceder a una URL para la que no tiene permisos se genera un error

Índice de Ejemplos

code URJC

5) Usuarios en BD

code URJC

ejem5

- Habitualmente los **usuarios** están en la **BD** y se pueden **añadir, borrar, modificar**, etc.
- Se tratan como una entidad más:
  - Se crea la **entidad User** de JPA
  - Se implementa un **UserRepository** para acceder a la base de datos

5) Usuarios en BD

code URJC

ejem5

- **Autenticación de usuarios**
  - Se proporciona un **proveedor de autenticación basado en BD**
  - Cada vez que un usuario se quiera autenticar, comprobamos **si está en la BD** y si la **contraseña es la correcta**

5) Usuarios en BD

code URJC

ejem5

- Acceso a los usuarios en la BD

```
@Entity
public class User {

  @Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  private Long id;

  private String name;
  private String encodedPassword;

  @ElementCollection(fetch = FetchType.EAGER)
  private List<String> roles;

  //Constructor, getters and setters
}
```

Roles del usuario (ADMIN y/o USER)

```
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findByName(String name);
}
```

5) Usuarios en BD

code URJC

ejem5

- Cargador de datos de ejemplo

```
@Component
public class DatabaseUsersLoader {

    @Autowired
    private UserRepository userRepository;

    @Autowired
    private PasswordEncoder passwordEncoder;

    @PostConstruct
    private void initDatabase() {

        userRepository.save(new User("user",
            passwordEncoder.encode("pass"), "USER"));

        userRepository.save(new User("admin",
            passwordEncoder.encode("adminpass"), "USER", "ADMIN"));
    }
}
```

Guardamos en la BBDD usuarios de ejemplo. La password se cifra antes de guardar

5) Usuarios en BD

code URJC

ejem5

- UserDetailService basado en BD

```
@Service
public class RepositoryUserDetailsService implements UserDetailsService {

  @Autowired
  private UserRepository userRepository;

  @Override
  public UserDetails loadUserByUsername(String username)
          throws UsernameNotFoundException {

    User user = userRepository.findByName(username)
      .orElseThrow(() -> new UsernameNotFoundException("User not found"));

    List<GrantedAuthority> roles = new ArrayList<>();
    for (String role : user.getRoles()) {
      roles.add(new SimpleGrantedAuthority("ROLE_" + role));
    }

    return new org.springframework.security.core.userdetails.User(user.getName(),
      user.getEncodedPassword(), roles);
  }
}
```

Cargamos el usuario de la BBDD

Cargamos los roles en la estructura de datos requerida

Devolvemos un User de Spring

5) Usuarios en BD

code URJC

ejem5

- Configuramos el UserDetailService

```
@Configuration
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Autowired
    public RepositoryUserDetailsService userDetailService;


    @Bean
    public PasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder(10, new SecureRandom());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth)
        throws Exception {

        auth.userDetailsService(userDetailsService)
            .passwordEncoder(passwordEncoder());
    }
    ...
}
```

Creamos un cifrador de passwords

Los usuarios se gestionan en RepositoryUserDetailService

5) Usuarios en BD

code URJC

ejem5

- Se accede al usuario con el repositorio

```
@Controller
public class WebController {

    @Autowired
    private UserRepository userRepository;
    ...

    @GetMapping("/private")
    public String privatePage(Model model, HttpServletRequest request) {

        String name = request.getUserPrincipal().getName();

        User user = userRepository.findByName(name).orElseThrow();

        model.addAttribute("username", user.getName());
        model.addAttribute("admin", request.isUserInRole("ADMIN"));

        return "private";
    }
}
```

Con el nombre del usuario podemos acceder a la base de datos

Índice de Ejemplos

6) Protección con CSRF

ejem6

- Cada formulario en una web con protección CSRF es ser similar a este

```
<!DOCTYPE html>
<html>
<body>
  <p>This is a private page</p>

  <form method="post" action="/logout">
    <input type="submit" value="Sign Out" />
    <input type="hidden" name="_csrf"
      value="c54a70a7-1586-4dc3-8e64-4fac09625ce2" />
  </form>
</body>
</html>
```

6) Protección con CSRF

code URJC

ejem6

- Para usar CSRF con Mustache tenemos que cargar en el modelo el token desde la request

```
@GetMapping("/login")
public String login(Model model, HttpServletRequest request) {

    CsrfToken token = (CsrfToken) request.getAttribute("_csrf");
    model.addAttribute("token", token.getToken());

    return "login";
}
```

6) Protección con CSRF

code URJC

ejem6

- Y generar el formulario usando ese token

```
<!DOCTYPE html>
<html>
<body>
  ...
  <p>This is a private page</p>
  ...
  <form action="/logout" method="post">
    <input type="submit" value="Sign Out" />
    <input type="hidden" name="_csrf" value="{{token}}"/>
  </form>
</body>
</html>
```

Índice de Ejemplos

6) Protección con CSRF

code URJC

ejem6b

- Pasar el token al modelo en cada método del controlador es **repetitivo**
- Para evitarlo, implementamos un **Handler** que se ejecutará después de cualquier método de un controlador web

6) Protección con CSRF

code URJC

ejem6b

```
@Configuration
public class CSRFHandlerConfiguration extends WebMvcConfigurerAdapter {

  @Override
  public void addInterceptors(InterceptorRegistry registry) {
    registry.addInterceptor(new CSRFHandlerInterceptor());
  }
}

class CSRFHandlerInterceptor implements HandlerInterceptor {

  @Override
  public void postHandle(final HttpServletRequest request,
       final HttpServletResponse response, final Object handler
       final ModelAndView modelAndView) throws Exception {

    if (modelAndView != null) {
      CsrfToken token = (CsrfToken) request.getAttribute("_csrf");
      if (token != null) {
        modelAndView.addObject("token", token.getToken());
      }
    }
  }
}
```

Activación del Handler

Handler

Índice de Ejemplos

Aplicaciones web seguras

code URJC

ejem7

- Aplicación web de gestión de libros con con base de datos con **diferentes permisos**:
  - **Usuario anónimo:** Acceso de lectura a los libros
  - **Usuario registrado:** Creación y modificación de libros
  - **Administrador:** Borrado de libros

Aplicaciones web seguras

code URJC

ejem7

- Control de acceso por URL

```
@Configuration
public class WebSecurityConfig extends WebSecurityConfigurerAdapter {

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

    // Public pages
    http.authorizeRequests().antMatchers("/").permitAll();
    http.authorizeRequests().antMatchers("/login").permitAll();
    http.authorizeRequests().antMatchers("/loginerror").permitAll();
    http.authorizeRequests().antMatchers("/logout").permitAll();

    // Private pages
    http.authorizeRequests().antMatchers("/newbook").hasAnyRole("USER");
    http.authorizeRequests().antMatchers("/editbook/*").hasAnyRole("USER");
    http.authorizeRequests().antMatchers("/removebook/*").hasAnyRole("ADMIN");

    ...
  }
}
```

Aplicaciones web seguras

code URJC

ejem7

- Mostrar el usuario en todas las páginas

```
@Controller
public class BookWebController {
  ...
  @ModelAttribute
  public void addAttributes(Model model, HttpServletRequest request) {

    Principal principal = request.getUserPrincipal();

    if(principal != null) {

      model.addAttribute("logged", true);
      model.addAttribute("userName", principal.getName());
      model.addAttribute("admin", request.isUserInRole("ADMIN"));

    } else {
      model.addAttribute("logged", false);
    }
  }
  ...
}
```

Se anota el método como @ModelAttribute

Se crean atributos en base al usuario

Índice de Ejemplos

Autenticación básica

code URJC

ejem8

- Las credenciales (login y password) se envían en la cabecera **Authorization**
- El valor se calcula como:

```
'Basic '+base64(login+':'+password)
```

- Ejemplo para login "Aladdin" y password "OpenSesame"

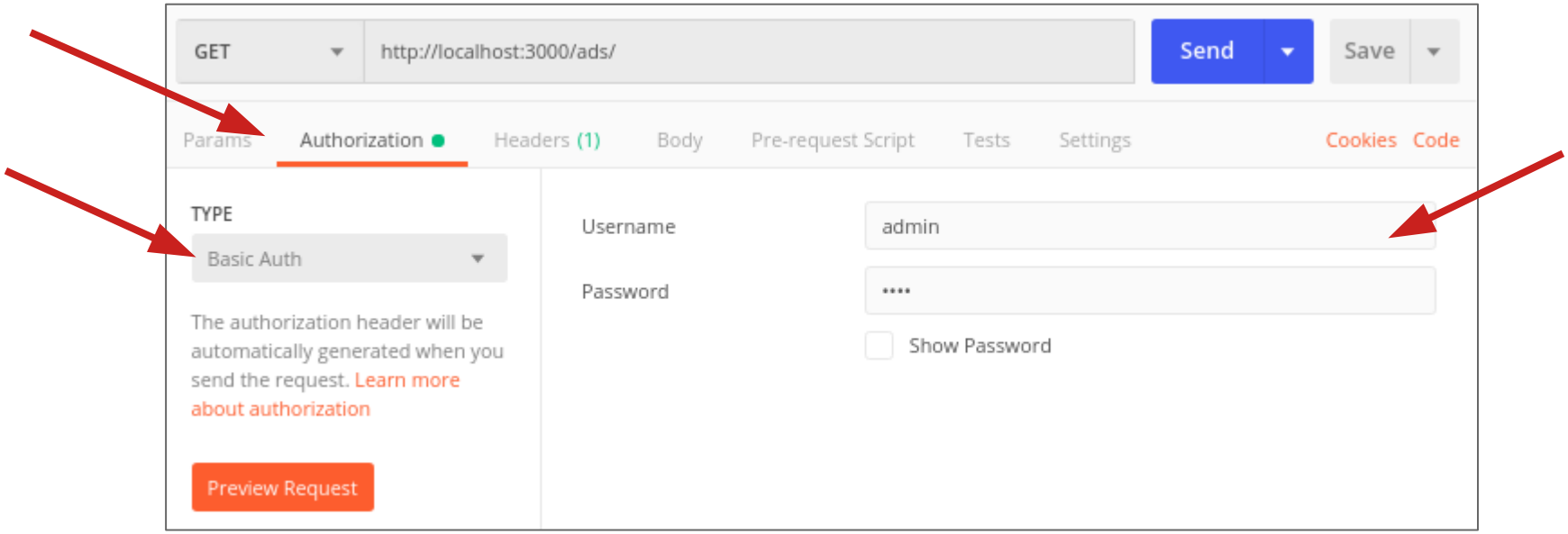
```
Authorization: Basic QWxhZGRpbjpPcGVuU2VzYW1l
```

Autenticación básica

code URJC

ejem8

- En Postman se especifica el tipo de Autorización "**Basic Auth**", el login y la password y él genera la cabecera



Autenticación básica

code URJC

```
@Configuration
public class RestSecurityConfig extends WebSecurityConfigurerAdapter {

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

    // Private endpoints
    http.authorizeRequests().antMatchers(HttpMethod.POST, "/api/books/**").hasRole("USER");
    http.authorizeRequests().antMatchers(HttpMethod.PUT, "/api/books/**").hasRole("USER");
    http.authorizeRequests().antMatchers(HttpMethod.DELETE, "/api/books/**").hasRole("ADMIN");

    // Other endpoints are public
    http.authorizeRequests().anyRequest().permitAll();

    // Disable CSRF protection (it is difficult to implement in REST APIs)
    http.csrf().disable();

    // Enable Basic Authentication
    http.httpBasic();

    // Disable Form login Authentication
    http.formLogin().disable();

    // Avoid creating session (because every request has credentials)
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

  }
}
```

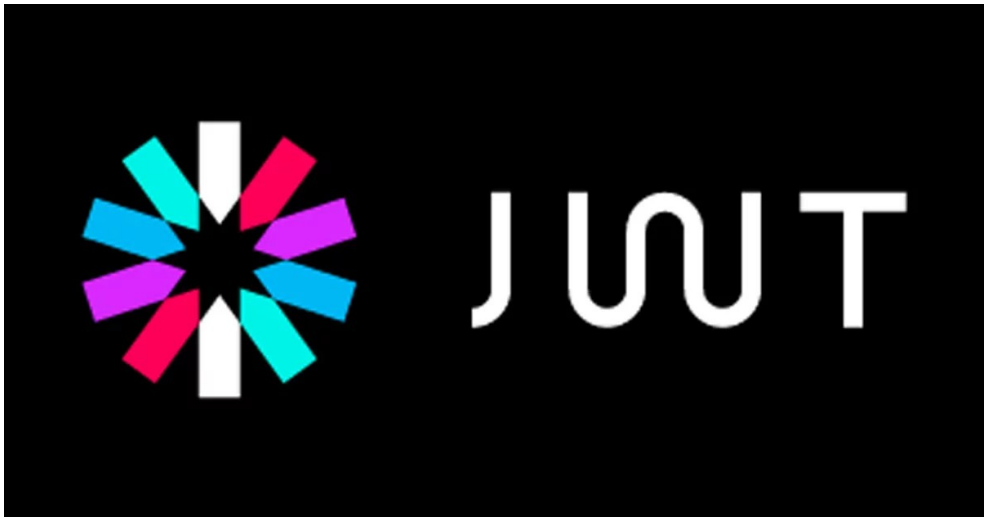
Índice de Ejemplos

Autenticación con tokens

code URJC

ejem9

- JSON Web Tokens



JSON Web Tokens are an open, industry standard RFC 7519 method for representing claims securely between two parties.

https://jwt.io/

JWT (JSON Web Tokens)

code URJC

ejem9

- **Implementación en Spring**
  - Se usa una librería externa para gestión de Token JWT

```
<dependency>
  <groupId>io.jsonwebtoken</groupId>
  <artifactId>jjwt</artifactId>
  <version>0.7.0</version>
</dependency>
```

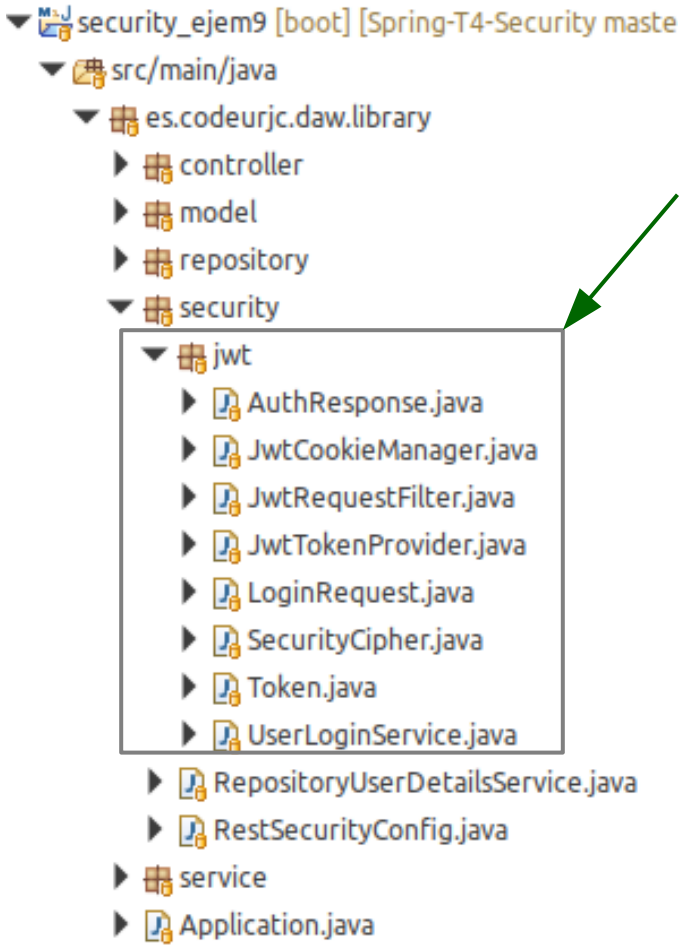
https://github.com/jwtk/jjwt

  - Requiere un procesamiento manual de los tokens y de las cookies

JWT (JSON Web Tokens)

- **Implementación en Spring**
  - Se proporciona código de gestión de tokens que se usa tal cual (no necesita ser adaptado para cada aplicación)

ejem9



JWT (JSON Web Tokens)

code URJC

ejem9

- Endpoints REST para login

```
@RestController
@RequestMapping("/api/auth")
public class LoginController {

  @Autowired
  private UserLoginService userService;

  @PostMapping("/login")
  public ResponseEntity<AuthResponse> login(
      @CookieValue(name = "accessToken", required = false) String accessToken,
      @CookieValue(name = "refreshToken", required = false) String refreshToken,
      @RequestBody LoginRequest loginRequest) {

    return userService.login(loginRequest, accessToken, refreshToken);
  }

  @PostMapping("/refresh")
  public ResponseEntity<AuthResponse> refreshToken(
      @CookieValue(name = "refreshToken", required = false) String refreshToken) {

    return userService.refresh(refreshToken);
  }

  @PostMapping("/logout")
  public ResponseEntity<AuthResponse> logOut(HttpServletRequest request, HttpServletResponse response) {
    return ResponseEntity.ok(new AuthResponse(Status.SUCCESS, userService.logout(request, response)));
  }
}
```

Se obtienen las cookies y las credenciales del cliente y se envían al UserLoginService

JWT (JSON Web Tokens)

code URJC

ejem9

- Cambios al SecurityConfig

```
@Configuration
public class RestSecurityConfig extends WebSecurityConfigurerAdapter {

  ...

  //Expose AuthenticationManager as a Bean to be used in other services

  @Bean
  @Override
  public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
  }

  ...
}
```

Es necesario publicar como un Bean el AuthenticationManager para que el código de gestión de tokens JWT lo pueda usar

JWT (JSON Web Tokens)

code URJC

ejem9

```java
@Configuration
public class RestSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private JwtRequestFilter jwtRequestFilter;

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

    // URLs that need authentication to access to it
    // Other URLs can be accessed without authentication
    ...

    // Disable CSRF protection (it is difficult to implement in REST APIs)
    http.csrf().disable();

    // Disable Http Basic Authentication
    http.httpBasic().disable();

    // Disable Form login Authentication
    http.formLogin().disable();

    // Avoid creating session
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    // Add JWT Token filter
    http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

  }
}
```

Se añade un filtro de JWT

JWT (JSON Web Tokens)

code URJC

ejem9

```java
@Configuration
public class RestSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private JwtRequestFilter jwtRequestFilter;

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

    // URLs that need authentication to access to it
    // Other URLs can be accessed without authentication
    ...

    // Disable CSRF protection (it is difficult to implement in REST APIs)
    http.csrf().disable();

    // Disable Http Basic Authentication
    http.httpBasic().disable();

    // Disable Form login Authentication
    http.formLogin().disable();

    // Avoid creating session
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    // Add JWT Token filter
    http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

  }
}
```

Se deshabilita tanto basic auth como form login

JWT (JSON Web Tokens)

code URJC

ejem9

```
@Configuration
public class RestSecurityConfig extends WebSecurityConfigurerAdapter {

  @Autowired
  private JwtRequestFilter jwtRequestFilter;

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

    // URLs that need authentication to access to it
    // Other URLs can be accessed without authentication
    ...

    // Disable CSRF protection (it is difficult to implement in REST
    http.csrf().disable();

    // Disable Http Basic Authentication
    http.httpBasic().disable();

    // Disable Form login Authentication
    http.formLogin().disable();

    // Avoid creating session
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

    // Add JWT Token filter
    http.addFilterBefore(jwtRequestFilter, UsernamePasswordAuthenticationFilter.class);

  }
}
```

Se desactiva la gestión de sesión (cada petición lleva credenciales)

JWT (JSON Web Tokens)

code URJC

ejem9

- Configuración de las URLs

```
@Configuration
public class RestSecurityConfig extends WebSecurityConfigurerAdapter {

  ...

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

    // URLs that need authentication to access to it
    http.authorizeRequests().antMatchers(HttpMethod.POST, "/api/books/**").hasRole("USER");
    http.authorizeRequests().antMatchers(HttpMethod.PUT, "/api/books/**").hasRole("USER");
    http.authorizeRequests().antMatchers(HttpMethod.DELETE, "/api/books/**").hasRole("ADMIN");

    // Other URLs can be accessed without authentication
    http.authorizeRequests().anyRequest().permitAll();

    ...
  }
}
```

Se define el acceso a las URLs de la misma forma

Índice de Ejemplos

Aplicaciones web y REST

code URJC

ejem10

- Es posible que una misma aplicación ofrezca un **interfaz web y una API REST**
- Para ello, se usan las dos configuraciones:
  - **WebSecurityConfig**
  - **RestSecurityConfig**
- Hay que hacer ciertos ajustes en **RestSecurityConfig** para que puedan coordinarse

Aplicaciones web y REST

code URJC

ejem10

La configuración para REST se carga la primera

Se usa el bean del WebSecurityConfig en vez de crear uno

```
@Configuration
@Order(1)
public class RestSecurityConfig extends WebSecurityC

 ...
 @Autowired
 private PasswordEncoder passwordEncoder;

 @Override
 protected void configure(AuthenticationManagerBuilder auth) throws Exception {

   auth.userDetailsService(userDetailsService).passwordEncoder(passwordEncoder);
 }

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

   http.antMatcher("/api/**");
   ...
  }
}
```

Para evitar colisiones, la configuración de este fichero sólo se aplica a las URLs que empiezan por /api

code
URJC

Desarrollo de Aplicaciones Web

Parte 3 – Despliegues de aplicaciones web

code
URJC

Desarrollo de Aplicaciones Web

Tema 1
Virtualización con Vagrant

Índice de Ejemplos

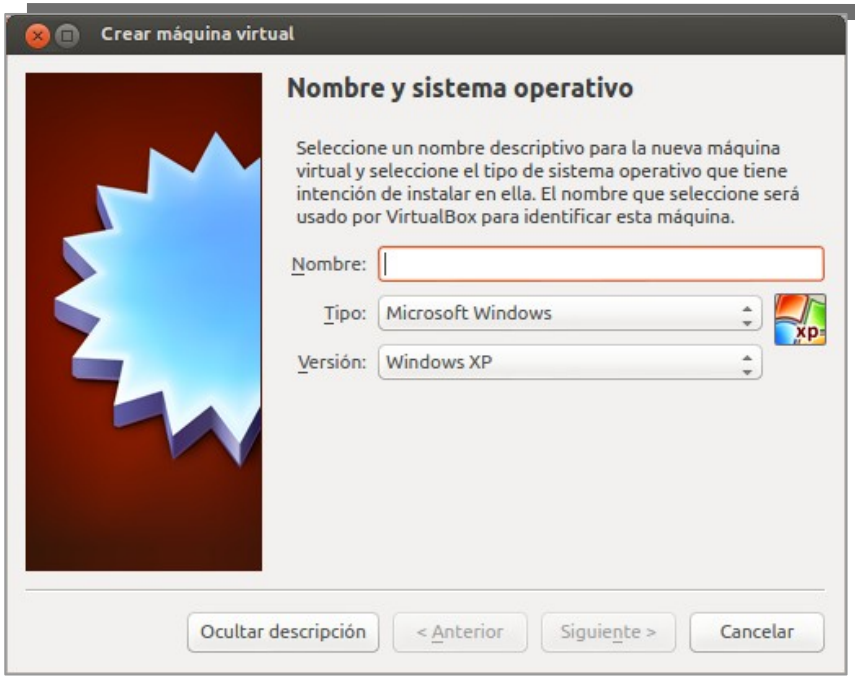
- Ejemplo 1 (VirtualBox)
- Ejemplo 2 (Vagrant)

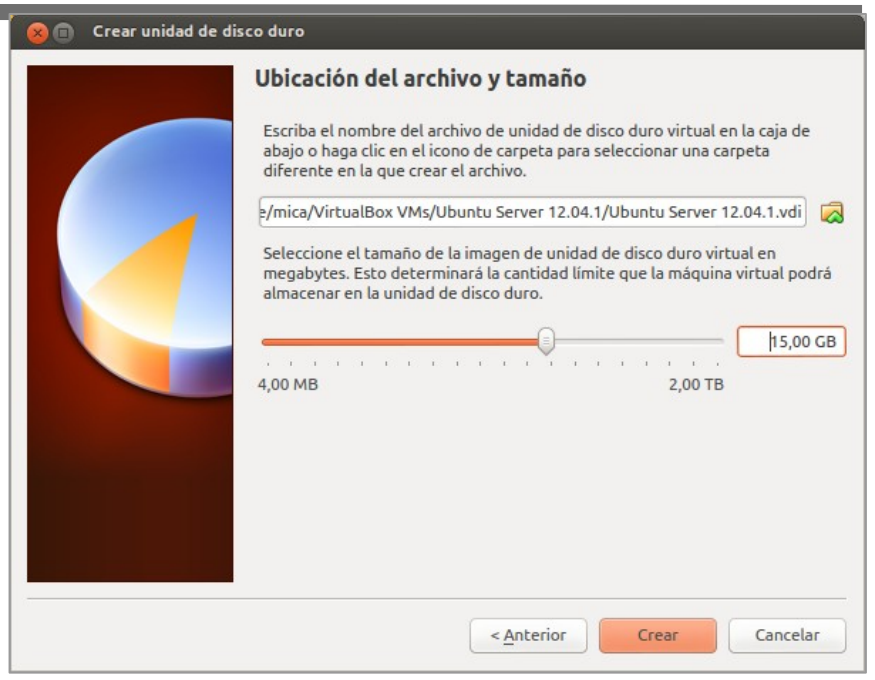
Índice de Ejemplos

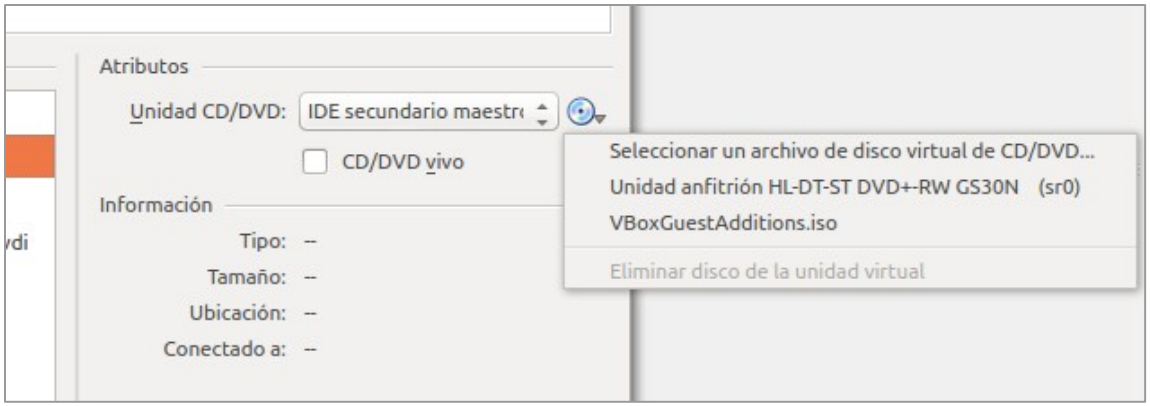
VirtualBox

- **Configuración interactiva**
  - Pasos:
    - Crear una máquina virtual limpia
    - Conecta una imagen ISO (simulando un CD real)
    - Instala un sistema operativo completo
  - Consume tiempo y no es sencillo compartir la configuración de las las máquinas virtuales

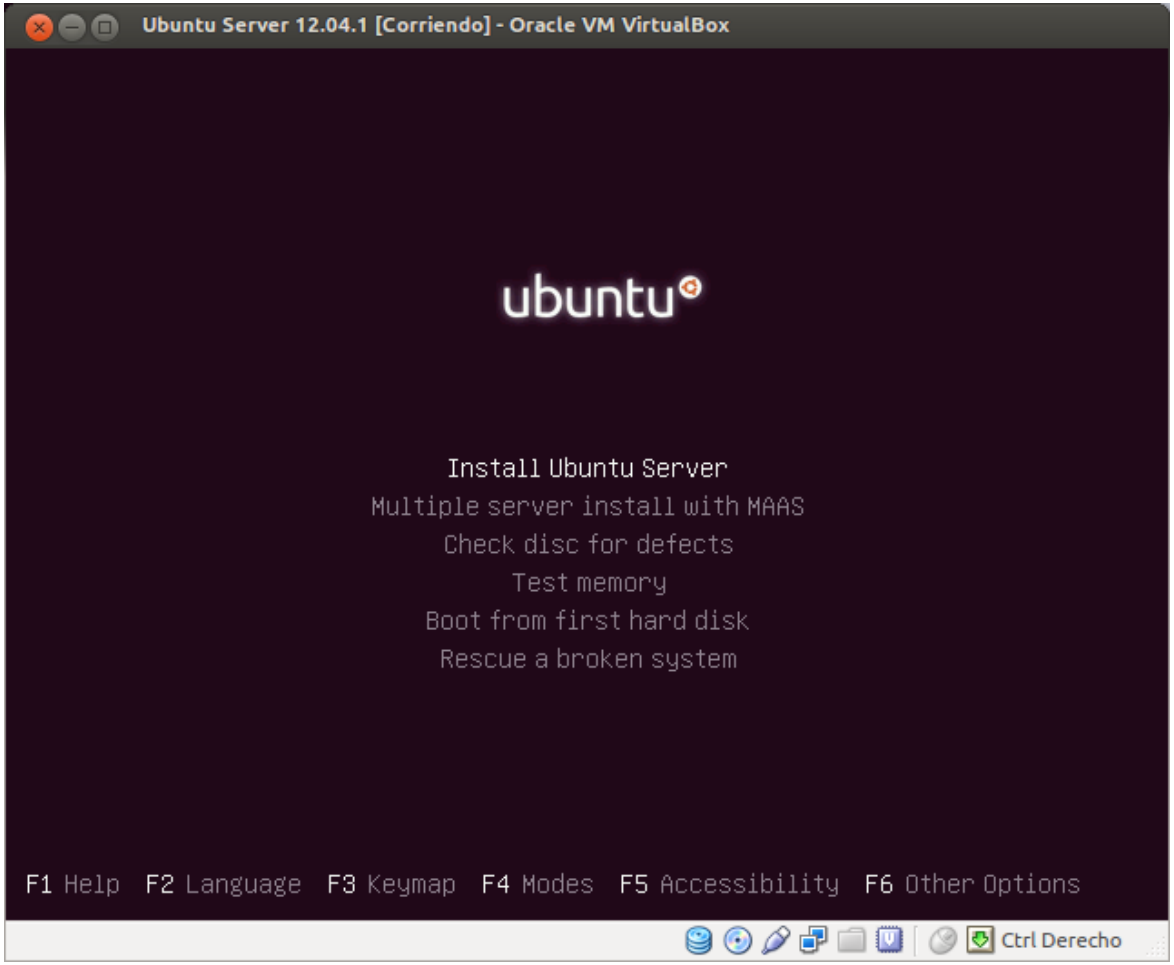
VirtualBox


Crear máquina virtual
Nombre y sistema operativo
Seleccione un nombre descriptivo para la nueva máquina virtual y seleccione el tipo de sistema operativo que tiene intención de instalar en ella. El nombre que seleccione será usado por VirtualBox para identificar esta máquina.
Nombre:
Tipo:
Microsoft Windows
Versión:
Windows XP
Ocultar descripción
< Anterior
Siguiente >
Cancelar

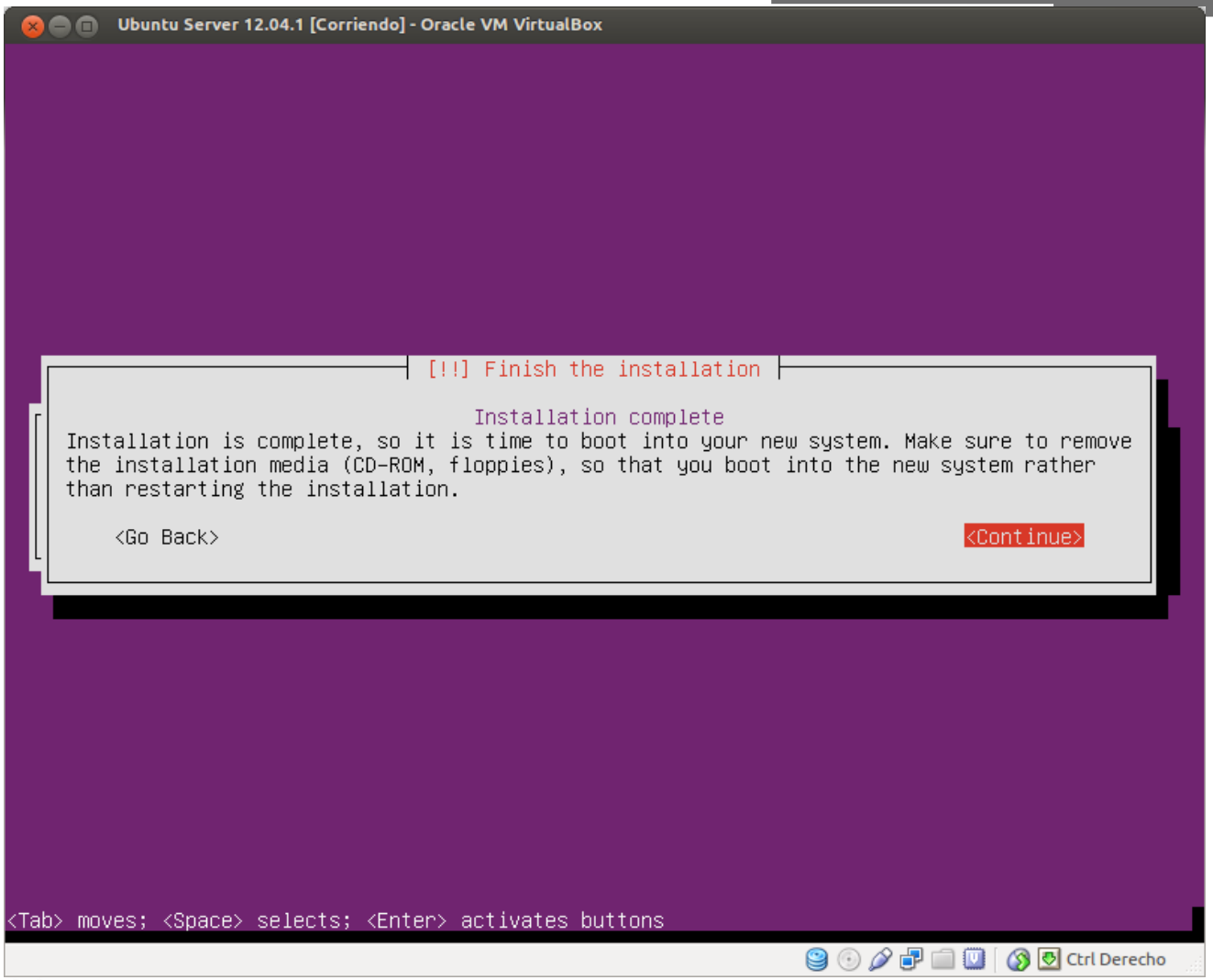

Crear unidad de disco duro
Ubicación del archivo y tamaño
Escriba el nombre del archivo de unidad de disco duro virtual en la caja de abajo o haga clic en el icono de carpeta para seleccionar una carpeta diferente en la que crear el archivo.
e/mica/VirtualBox VMs/Ubuntu Server 12.04.1/Ubuntu Server 12.04.1.vdi
Seleccione el tamaño de la imagen de unidad de disco duro virtual en megabytes. Esto determinará la cantidad límite que la máquina virtual podrá almacenar en la unidad de disco duro.
15,00 GB
4,00 MB
2,00 TB
< Anterior
Crear
Cancelar


Atributos
Unidad CD/DVD:
IDE secundario maestr
CD/DVD vivo
Información
Tipo: --
Tamaño: --
Ubicación: --
Conectado a: --
Seleccionar un archivo de disco virtual de CD/DVD...
Unidad anfitrión HL-DT-ST DVD+-RW GS30N (sr0)
VBoxGuestAdditions.iso
Eliminar disco de la unidad virtual

VirtualBox

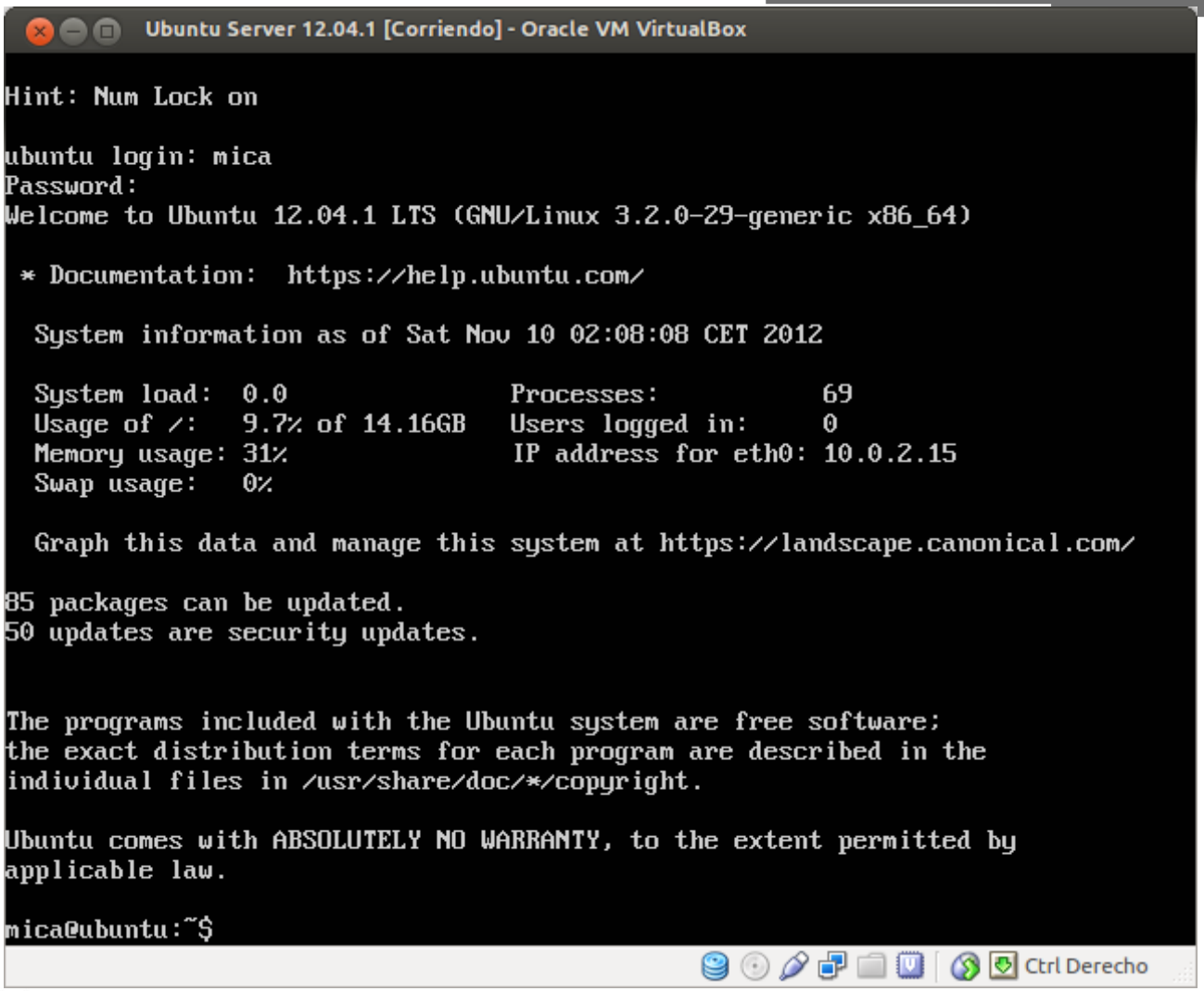

Ubuntu Server 12.04.1 [Corriendo] - Oracle VM VirtualBox
ubuntu
Install Ubuntu Server
Multiple server install with MAAS
Check disc for defects
Test memory
Boot from first hard disk
Rescue a broken system
F1 Help F2 Language F3 Keymap F4 Modes F5 Accessibility F6 Other Options
Ctrl Derecho

VirtualBox



Universidad Rey Juan Carlos

VirtualBox


Ubuntu Server 12.04.1 [Corriendo] - Oracle VM VirtualBox
Hint: Num Lock on
ubuntu login: mica
Password:
Welcome to Ubuntu 12.04.1 LTS (GNU/Linux 3.2.0-29-generic x86_64)
* Documentation: https://help.ubuntu.com/
System information as of Sat Nov 10 02:08:08 CET 2012
System load: 0.0 Processes: 69
Usage of /: 9.7% of 14.16GB Users logged in: 0
Memory usage: 31% IP address for eth0: 10.0.2.15
Swap usage: 0%
Graph this data and manage this system at https://landscape.canonical.com/
85 packages can be updated.
50 updates are security updates.
The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.
mica@ubuntu:~$
Ctrl Derecho

VirtualBox

- **Para parar una VM:**
  - Simulando el apagado de la máquina física
  - Enviado la señal de apagado
  - Haciendo que el sistema se pare a sí mismo
    - `sudo poweroff`
    - `sudo halt`
    - `sudo shutdown -h now`
    - `sudo reboot (reiniciar)`
  - Pausando para reanudar después

Índice de Ejemplos

- Ejemplo 1 (VirtualBox)
- **Ejemplo 2 (Vagrant)**

Vagrant

- **Instalar Vagrant**
  - Tener instalador VirtualBox (o cualquiera de los soportados
  - Instalar Vagrant
    - https://www.vagrantup.com/docs/installation/

Vagrant

- Crear una máquina virtual

```
$ mkdir project
$ cd project
$ vagrant init bento/ubuntu-16.04
$ vagrant up
```

- Conectarse a la VM por ssh

```
$ vagrant ssh
```

Vagrant

- **Crear una máquina virtual**

```
$ vagrant init bento/ubuntu-16.04
```

  - Se genera un fichero **Vagrantfile** que describe la máquina virtual basada en la **"box"** de ubuntu xenial de 64bits publicada en el repositorio

    https://atlas.hashicorp.com/bento/boxes/ubuntu-16.04

  - Cualquiera puede crear una cuenta y subir sus propias **boxes** con las configuraciones necesarias

Vagrant

- **Manejar la nueva máquina virtual**
  - Las máquinas se gestionan enteramente desde la **línea de comandos** (arrancar, parar, reanudar…)
  - **Arrancar** la máquina virtual

```
$ vagrant up
```

  - Puede tardar bastante tiempo:
    - Es posible que tenga que **descargar el binario** del box si no está disponible en la máquina
    - Las VMs pueden tardar **minutos en arrancar**

Vagrant

- **Manejar la nueva máquina virtual**
  - Detener la ejecución de la máquina virtual pero mantener el estado (disco duro)

```
$ vagrant halt
```

  - Destruir todos los ficheros de la máquina virtual

```
$ vagrant destroy
```

  - Pausar la VM (mantiene la memoria):

```
$ vagrant pause
```

  - Reanudar la VM pausada

```
$ vagrant resume
```

Vagrant

- **Manejar la nueva máquina virtual**
  - La máquinas **no tienen interfaz gráfico**, sólo pueden usarse mediante una conexión **ssh** (lo habitual en el cloud).

```
$ vagrant ssh
```

  - La conexión ssh se realiza con una **clave privada** (en vez de con contraseña). En la imagen de ubuntu oficial se genera una clave de forma **automática** para conectar
  - Para cerrar la conexión ssh y volver a la shell del SO host:

```
ubuntu@ubuntu-xenial:~$ exit
```

Vagrant

- **Configuración de red en la máquina virtual**
  - Para acceder a la máquina virtual por red se descomenta la siguiente línea de Vagrantfile

```
# config.vm.network "private_network", ip: "192.168.33.10”
```

  - Verificar que la máquina arranca con ip 192.168.33.10 y tiene conexión a Internet

```
$ vagrant up
$ ping 192.168.33.10
$ vagrant ssh
ubuntu> ping www.google.es
ubuntu> exit
$ vagrant destroy -f
```

Vagrant

- **Ejecutar aplicaciones en la máquina virtual**
  - La carpeta en la que se encuentra el Vagrantfile es accesible directamente desde la máquina virtual en la ruta **/vagrant**
  - Un flujo de desarrollo puede ser copiar el binario de la aplicación en la carpeta del host para que esté accesible desde la máquina virtual

Vagrant

- **Ejecutar una app web dentro de una VM**
  - Copiar webapp.jar en la carpeta del fichero Vagrantfile
  - Iniciar la VM y arrancar la app

```
$ vagrant up
$ vagrant ssh
Ubuntu> sudo add-apt-repository ppa:webupd8team/java
ubuntu> sudo apt-get update
ubuntu> sudo apt-get install oracle-java8-installer
ubuntu> cd /vagrant
ubuntu> java -jar webapp.jar
```

  - Abrir http://192.168.33.10:8080 en un browser
  - Para la app y la VM

```
ubuntu> Ctrl+C
ubuntu> exit
$ vagrant destroy -f
```

01-2023

code
URJC

Desarrollo de Aplicaciones Web

Tema 2 -Docker

Índice de Ejemplos

- Ejemplo 1 (Dockerizar una aplicación)
- Ejemplo 2 (Dockerizar una aplicación compilada)
- Ejemplo 3 (ENTRYPOINT y CMD)

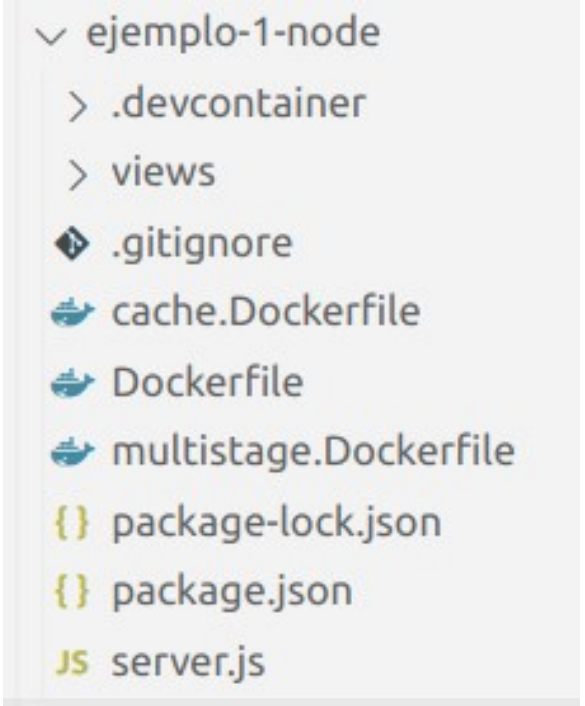
Índice de Ejemplos

- **Ejemplo 1 (Dockerizar una aplicación)**
- Ejemplo 2 (Dockerizar una aplicación compilada)
- Ejemplo 3 (ENTRYPOINT y CMD)

Dockerizar una aplicación

ejemplo-1-node

- Para dockerizar una aplicación hay que crear una imagen docker de la aplicación
- Crearemos una imagen con una **aplicación web** implementada en **Node**



Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Contenido de la imagen docker:**
  - Código fuente de la aplicación
  - Node
  - Librerías necesarias (express y mustache-express)
- Una vez creada la imagen, se puede **ejecutar la aplicación dockerizada**
- También se puede **publicar en DockerHub** (o cualquier otro registro) para compartirla

Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Dockerfile**
  - Fichero usado para describir el contenido de una imagen docker
  - Contenido:
    - Imagen en la que se basará la nueva imagen
    - Comandos que añaden el software necesario a la imagen base
    - Ficheros de la aplicación para incluir en la imagen
    - Puertos abiertos para poder bindearlos al host
    - Comando por defecto a ejecutar al arrancar el contenedor

```
# Selecciona la imagen base
FROM node:lts-alpine

# Especificamos esta variable para la correcta ejecución
de las librerías en modo de producción
ENV NODE_ENV production

# Definimos el directorio de trabajo en /usr/src/app/
WORKDIR /usr/src/app/

# Copiamos los ficheros de la aplicación
COPY server.js /usr/src/app/
COPY views/index.html /usr/src/app/views/
COPY package.json /usr/src/app/

# Instalamos las dependencias que necesita la app
RUN npm install --only=production

# Indica el puerto que expone el contenedor
EXPOSE 5000

# Comando que se ejecuta cuando se arranque el contenedor
CMD ["node", "server.js"]
```

Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Dockerfile**
  - **FROM:** Imagen base
  - **RUN:** Ejecuta comandos para instalar y configurar el software de la imagen
  - **COPY:** Copy ficheros desde la carpeta del Dockerfile
  - **EXPOSE:** Define los puertos públicos
  - **CMD:** Comando por defecto que se ejecuta al arrancar el contenedor

https://docs.docker.com/engine/userguide/eng-image/dockerfile_best-practices/

Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Construir la imagen**
  - Se puede crear una imagen para que sea usada **únicamente en la máquina** que se ha creado
  - Lo más habitual es crear una imagen para **subirla a un registro** de imágenes (como DockerHub)
    - Creamos una **cuenta en DockerHub**
    - **Conectamos** nuestra máquina a DockerHub

```
$ docker login
```

Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Construir la imagen**
  - En la carpeta del **Dockerfile** se ejecuta

```
$ docker build -t miusuario/webgatos .
```

    - **miusuario** corresponde al usuario creado en DockerHub
    - **webgatos** es el nombre del repositorio al que subir la imagen
    - **.** es la ruta del Dockerfile

Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Construir la imagen**
  - Acciones ejecutadas:
    - Se ejecuta un nuevo contenedor partiendo de la imagen base
    - Se ejecutan los comandos (RUN y COPY) del Dockerfile en ese contenedor
    - El resultado se empaqueta en el nuevo contenedor

Dockerizar una aplicación

code URJC

ejemplo-1-node

- Ejecutar la nueva imagen

```
$ docker run -p 9000:5000
miusuario/webgatos
 * Running on http://localhost:5000/
   (Press CTRL+C to quit)
```

- Abrir http://localhost:5000/ en el navegador web

Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Publicar la imagen**

```
$ docker push miusuario/webgatos
```

- La imagen se sube a DockerHub y se hace pública
- Cualquiera puede ejecutar un contenedor partiendo de esa imagen
- Se pueden instalar registros privados en una organización

Dockerizar una aplicación

code URJC

ejemplo-1-node

- **Caché de construcción por capas**
  - Cambia la plantilla de la web en **`views\index.html`**
  - Construye la imagen de nuevo

```
$ docker build -t miusuario/webgatos .
```

  - Los pasos del Dockerfile que no han cambiado **NO se vuelven a ejecutar** (se reutilizan de la ejecución previa)
  - Cada paso está en una **capa independiente**
  - La nueva imagen se crea muy rápidamente

Índice de Ejemplos

- Ejemplo 1 (Dockerizar una aplicación)
- **Ejemplo 2 (Dockerizar una aplicación compilada)**
- Ejemplo 3 (ENTRYPOINT y CMD)

Dockerizar una aplicación compilada

code URJC

ejemplo-2

- Dockerizar una aplicación con lenguaje de script es bastante sencillo, porque el **código fuente** se puede **ejecutar directamente**
- Cuando la aplicación está implementada con un lenguaje compilado, se realizan dos pasos:
  - 1) **Compilar** la aplicación (preferiblemente en un contenedor)
  - 2) **Empaquetar** la aplicación en un contenedor

Dockerizar una aplicación compilada

ejemplo-2

- **Compilar una aplicación Java en un contenedor**
  - Descargar proyecto de ejemplo

```
$ git clone https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores
$ cd 3.2.Contenedores-y-orquestadores/docker/ejemplo-2
```

  - Compilar y generar el fichero .jar

```
$ docker run --rm -v "$PWD":/data -w /data \
    maven mvn package
```

  - "mvn package" es el comando de compilación y empaquetado
  - -w configura el directorio de trabajo

Dockerizar una aplicación compilada

code URJC

ejemplo-2

- **Compilar una aplicación Java en un contenedor**
  - La aplicación compilada y empaquetada es un fichero .jar que se encuentra en la carpeta **target**
  - Para ejecutar ese fichero es necesario el **Java Runtime Environment (JRE)**, pero no es necesario un compilador ni otras herramientas de construcción como Maven
  - Se ejecuta con el comando

```
$ java -jar ./target/java_webapp-0.0.1.jar
```

Dockerizar una aplicación compilada

code URJC

ejemplo-2

- **Dockerizar la aplicación Java**
  - Hay que crear un nuevo contenedor con Java para poder ejecutar el .jar (No se necesita maven)
  - Hay que copiar el fichero .jar recién creado
  - Al arrancar el contenedor, se ejecuta

```
java -jar java-webapp-0.0.1.jar
```

Dockerizar una aplicación compilada

code URJC

ejemplo-2

- **Dockerizar la aplicación Java**
  - jar.Dockerfile

```
FROM openjdk:8-jre
COPY target/*.jar /usr/app/
WORKDIR /usr/app
CMD [ "java", "-jar", "java-webapp-0.0.1.jar" ]
```

  - Construir el contenedor

```
$ docker build -f jar.Dockerfile -t miusuario/java-webapp .
```

  - Ejecutar el contenedor

```
$ docker run -p 5000:8080 miusuario/java-webapp
```

Dockerizar una aplicación compilada

- **Multistage Dockerfile**
  - Se han realizado dos pasos para dockerizar la aplicación
    - **Paso 1:** Compilar el código fuente y generar el binario usando un contenedor
    - **Paso 2:** Crear un contenedor con el binario generado
  - Los **Multistage Dockerfiles** son ficheros Dockerfile que permiten definir varios pasos.
  - Cada paso se ejecuta en su propio contenedor

https://docs.docker.com/develop/develop-images/multistage-build/

Dockerizar una aplicación compilada

code URJC

ejemplo-2

- **Multistage Dockerfile**

multistage.Dockerfile

```
FROM maven as builder
COPY . /code/
WORKDIR /code
RUN mvn package

FROM openjdk:8-jre
COPY --from=builder /code/target/*.jar
/usr/app/
WORKDIR /usr/app
CMD [ "java", "-jar", "java-webapp-
0.0.1.jar" ]
```

```
$ docker build -f multistage.Dockerfile -t miusuario/java-webapp2 .
```

Índice de Ejemplos

- Ejemplo 1 (Dockerizar una aplicación)
- Ejemplo 2 (Dockerizar una aplicación compilada)
- **Ejemplo 3 (ENTRYPOINT y CMD)**

Diferencias ENTRYPOINT y CMD

ejemplo-3

- **CMD**
  - Define un comando predeterminado que se ejecuta al iniciar un contenedor
  - Este comando se puede sobreescribir al arrancar la imagen con el comando **"docker run image [OPTIONS]"**

```
FROM alpine:3.12
CMD ["echo", "Hello from CMD"]
```

```
$ docker container run my-image
  Hello from CMD
```

```
$ docker container run my-image echo "Hello from the CLI"
  Hello from the CLI
```

https://dominikbraun.io/blog/docker/dockerfile-run-vs-cmd-vs-entrypoint/

Diferencias ENTRYPOINT y CMD

code URJC

ejemplo-3

- **ENTRYPOINT**
  - También define un comando por defecto que se ejecuta al iniciar un contenedor
  - El **ENTRYPOINT** es ideal para las imágenes que siempre ejecutan el mismo **comando**
  - Por defecto **ENTRYPOINT** ejecuta el comando **"/bin/sh"**
  - A diferencia de **CMD** este comando no se puede reemplazar con el comando **"docker run image [OPTIONS]"**
  - Sino que todos los comandos que pasemos al **CMD** se enviarán como parámetros al **ENTRYPOINT**

https://dominikbraun.io/blog/docker/dockerfile-run-vs-cmd-vs-entrypoint/

Diferencias ENTRYPOINT y CMD

code URJC

ejemplo-3

- ENTRYPOINT

```
FROM alpine:3.12
ENTRYPOINT ["echo"]
CMD ["Hello from CMD"]
```

```
$ docker container run my-image
  Hello from CMD
```

```
$ docker container run my-image "Hello from the CLI"
  Hello from the CLI
```

https://dominikbraun.io/blog/docker/dockerfile-run-vs-cmd-vs-entrypoint/

Ejemplo ENTRYPOINT y CMD

code URJC

ejemplo-3

- Aplicación de línea de comandos Java

Application.java

```
public class Application {
    public static void main(String[] args) {
        System.out.println("============================");
        System.out.println("Command line application");
        System.out.println("============================");

        System.out.println("=> Read arguments:");
        Arrays.stream(args)
                .forEach(arg -> System.out.println("\t" + arg));
    }
}
```

Dockerfile

```
FROM openjdk:8-jre-slim
WORKDIR /usr/src/app/
COPY --from=builder /project/target/app-jar-with-dependencies.jar
/usr/src/app/
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app-jar-with-dependencies.jar"]
CMD ["--server"]
```

Ejemplo ENTRYPOINT y CMD

ejemplo-3

```
$ git clone
https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestado
res
$ cd 3.2.Contenedores-y-orquestadores/docker/ejemplo-3
$ docker build -t my-image .
```

```
$ docker run my-image
============================
Command line application
============================
=> Read arguments:
    --server
```

```
$ docker run my-image --override-argument1 --override-argument2
============================
Command line application
============================
=> Read arguments:
    --override-argument1
    --override-argument2
```

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 1

- **Ejecuta una web con Drupal en un contenedor docker**
  - Revisa la documentación de la página de DockerHub de Drupal
  - Accede al drupal desde un navegador web

Índice de Ejercicios

Ejercicio 2

- **Genera el .jar de una aplicación con un contenedor docker**
  - Utiliza la aplicación **"application-java-enunciado"**
  - Busca una imagen adecuada en Docker Hub (tiene que tener Maven y un JDK de Java)
  - Monta las carpetas adecuadas (para que el compilador pueda acceder al fuente y para que pueda generar el binario)

Índice de Ejercicios

Ejercicio 3

- **Usa Maven dockerizado del ejercicio 2 como si fuera una mini máquina virtual**
  - Ejecuta una shell en el contenedor
  - Ejecuta los comandos de compilación dentro de la shell cada vez que quieras compilar

Índice de Ejercicios

Ejercicio 4

- **Crea una imagen docker con una aplicación web Java**
  - Utiliza la aplicación **"applicacion-java-enunciado"**
  - Basada en una imagen con Maven para poder compilar la aplicación en el proceso de construcción de la imagen

Existen **estrategias más convenientes** de empaquetar una aplicación Java en un contenedor Docker que veremos más adelante

Índice de Ejercicios

Ejercicio 6

- Crea un Multistage Docker file optimizando las capas para no descargar las librerías en cada construcción

Índice de Ejercicios

Ejercicio 7

- Crea la imagen de la aplicación Java del Ejercicio 6 con jib

Índice de Ejercicios

Ejercicio 8

- Crea la imagen de la aplicación Java del Ejercicio 6 con Buildpacks

code
URJC

Desarrollo de Aplicaciones Web

Parte 4. Tecnologías web avanzadas de cliente

code
URJC

Desarrollo de Aplicaciones Web

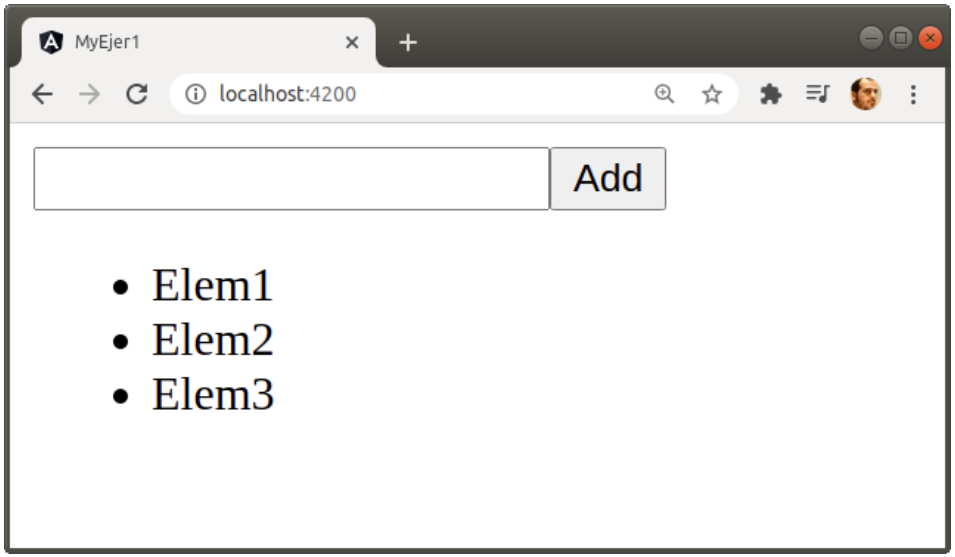
2. Componentes

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 1

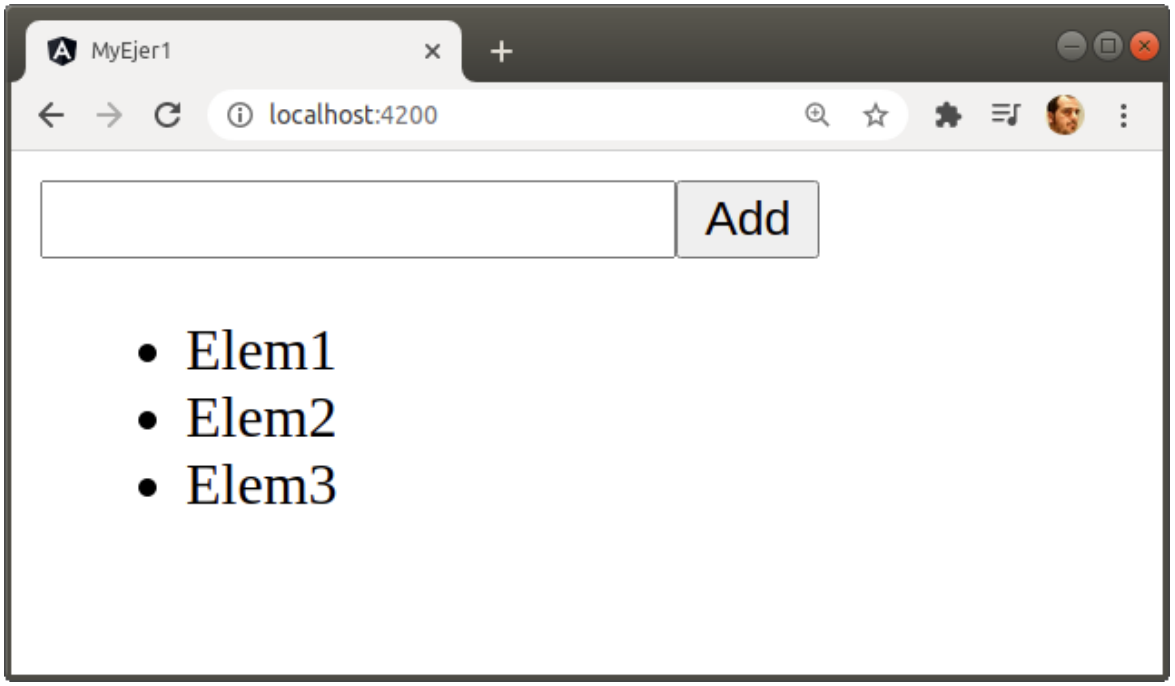
- Implementa una aplicación con un **campo de texto** y un **botón de Añadir**
- Cada vez que se pulse el **botón**, el **contenido** del campo de texto se **añadirá al documento**



Índice de Ejercicios

Ejercicio 1b

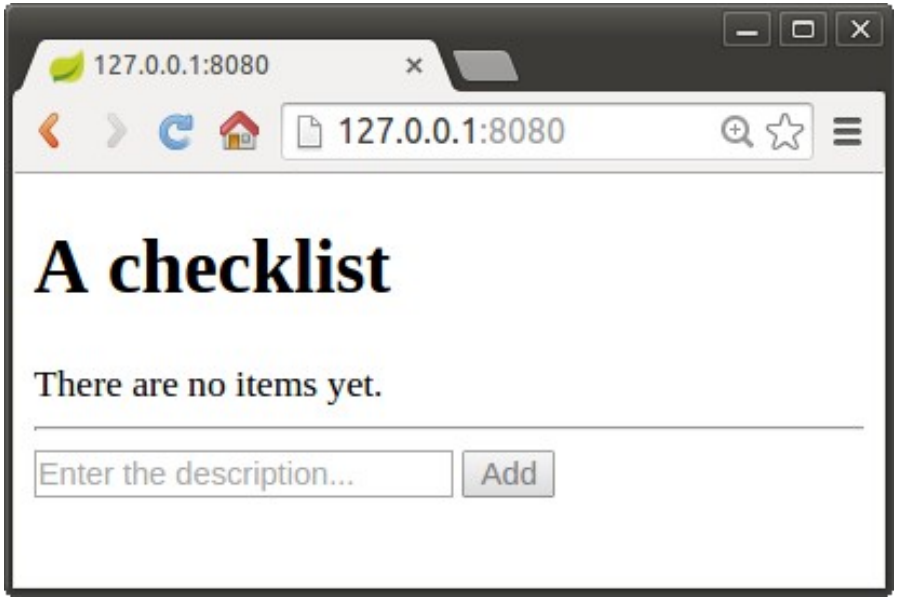
- Implementa el ejercicio 1 usando ***Template Reference Variables***

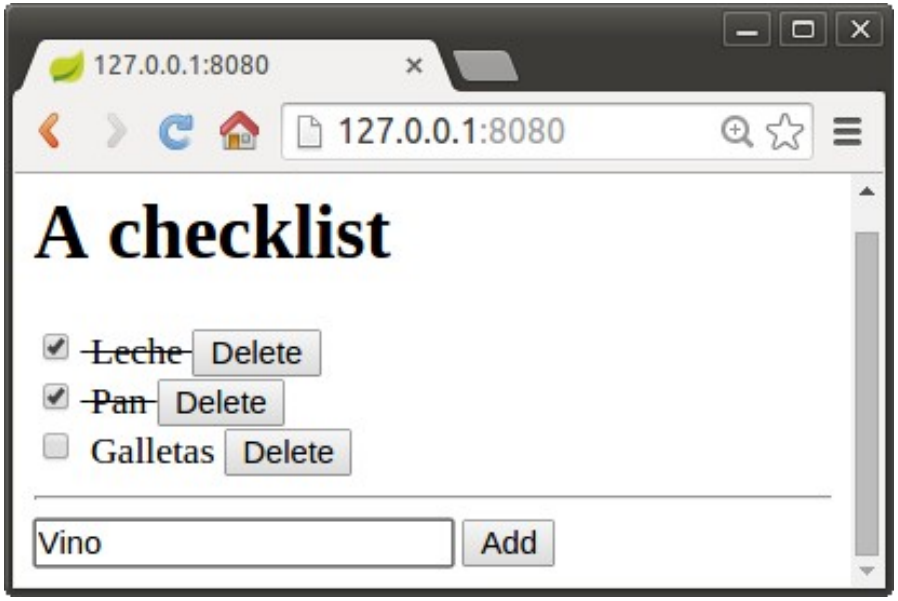


Índice de Ejercicios

Ejercicio 2

- Implementa una aplicación de **gestión de tareas**
- Las tareas se mantendrán en **memoria**

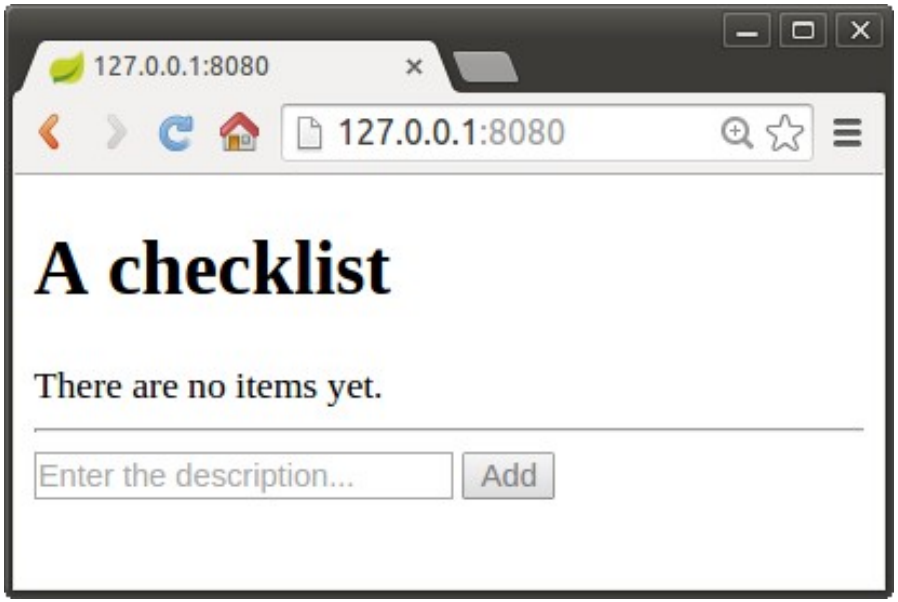


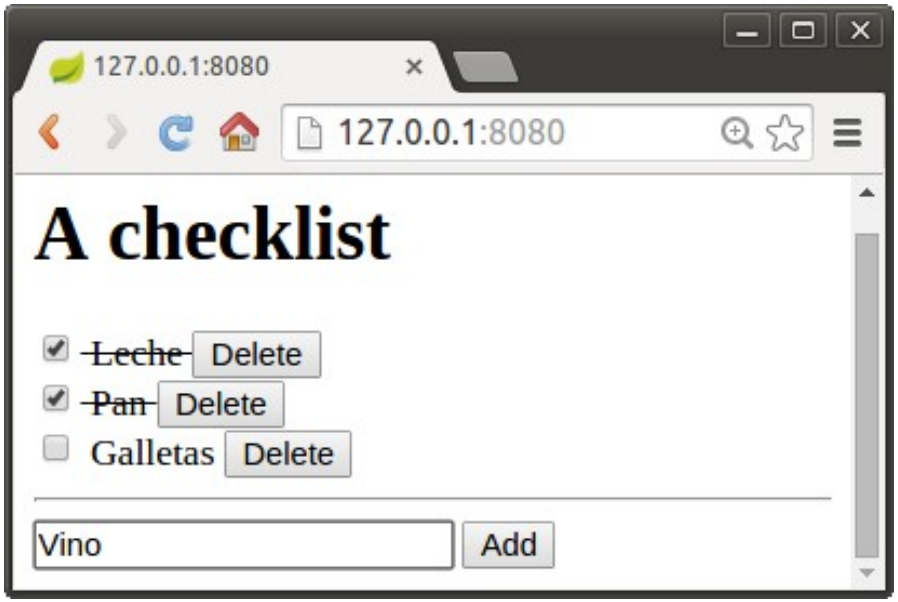


Índice de Ejercicios

Ejercicio 3

- Refactoriza la aplicación de **gestión de tareas** para que cada tarea se visualice en su propio componente





code
URJC

Desarrollo de Aplicaciones Web

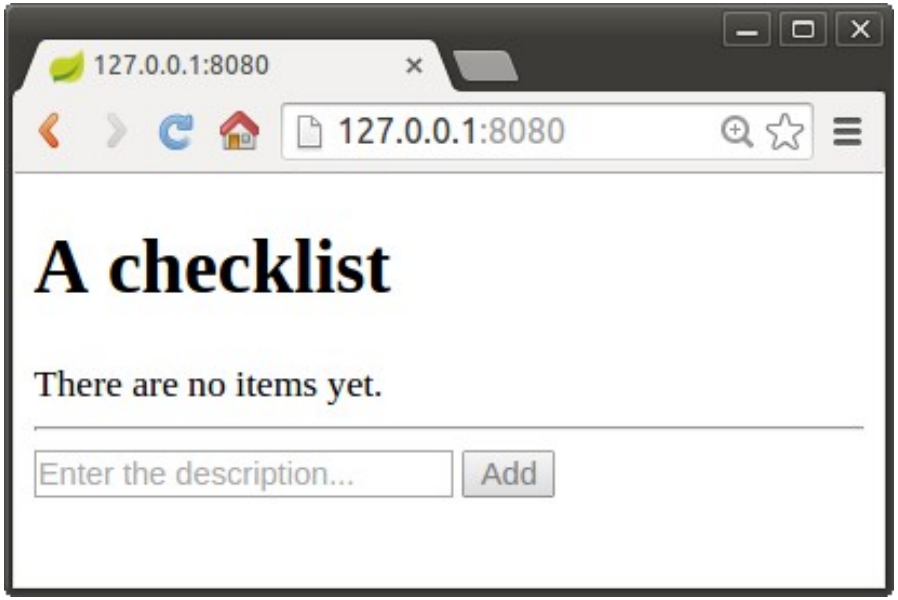
3. APIs REST y Servicios

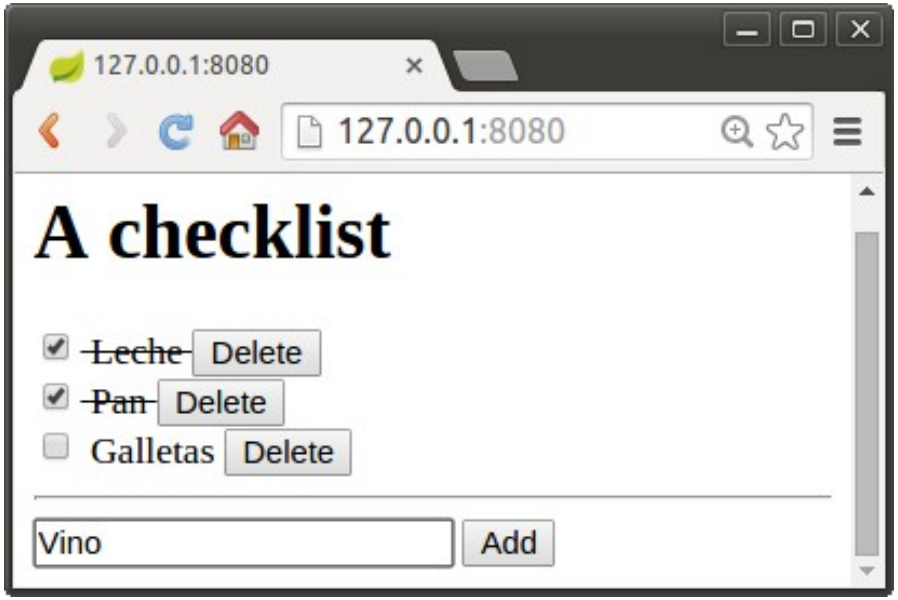
Índice de Ejercicios

Índice de Ejercicios

Ejercicio 4

- Amplía el **servicio de gestión de items** para que utilice una **API REST** para gestionar los items





Ejercicio 4

- Se proporciona un servicio web que exporta una API REST

Java

```
$ cd ejer4_backend/java

$ java -jar items-service.jar
```

Node

```
$ cd ejer4_backend/node

$ node src/app.js
```


```
Items.postman_collection.json
```

Ejercicio 4

code URJC

- **API REST Items**
  - Creación de items
    - Method: POST
    - URL: http://127.0.0.1:8080/items/
    - Headers: Content-Type: application/json
    - Body:

```
{ "description" : "Leche", "checked": false }
```

    - Result:

```
{ "id": 1, "description" : "Leche", "checked": false }
```

    - Status code: 201 (Created)

Ejercicio 4

- **API REST Items**
  - Consulta de items
    - Method: GET
    - URL: http://127.0.0.1:8080/items/
    - Result:

```
[
  { "id": 1, "description": "Leche", "checked":
false },
  { "id": 2, "description": "Pan", "checked": true }
]
```

    - Status code: 200 (OK)

Ejercicio 4

- **API REST Items**
  - Modificación de items
    - Method: PUT
    - URL: http://127.0.0.1:8080/items/1
    - Headers: Content-Type: application/json
    - Body:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

    - Result:

```
{ "id": 1, "description" : "Leche", "checked": true }
```

    - Status code: 200 (OK) / 404 (Not Found)

Ejercicio 4

code URJC

- **API REST Items**
  - Modificación de items
    - Method: DELETE
    - URL: http://127.0.0.1:8080/items/1
    - Result:

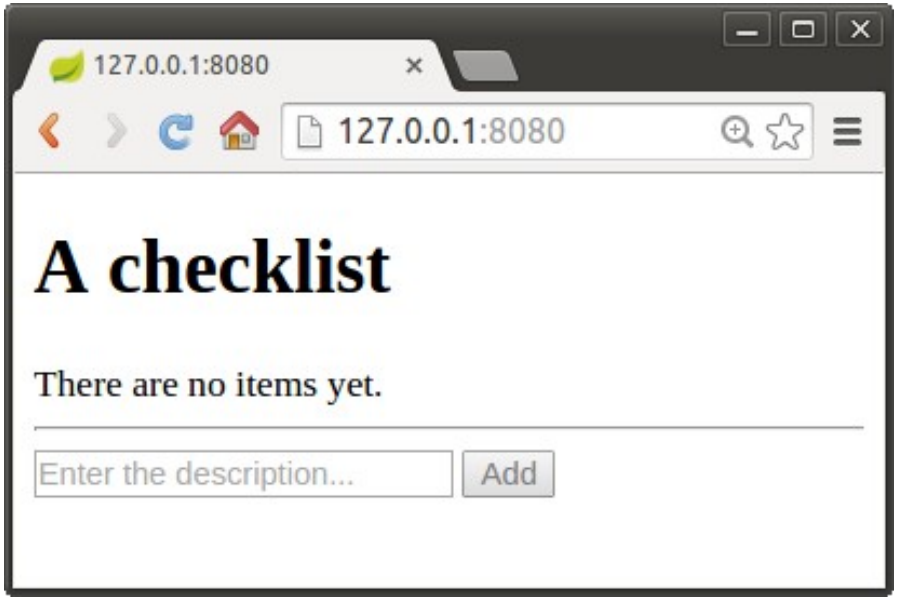
```
{ "id": 1, "description" : "Leche", "checked": true }
```

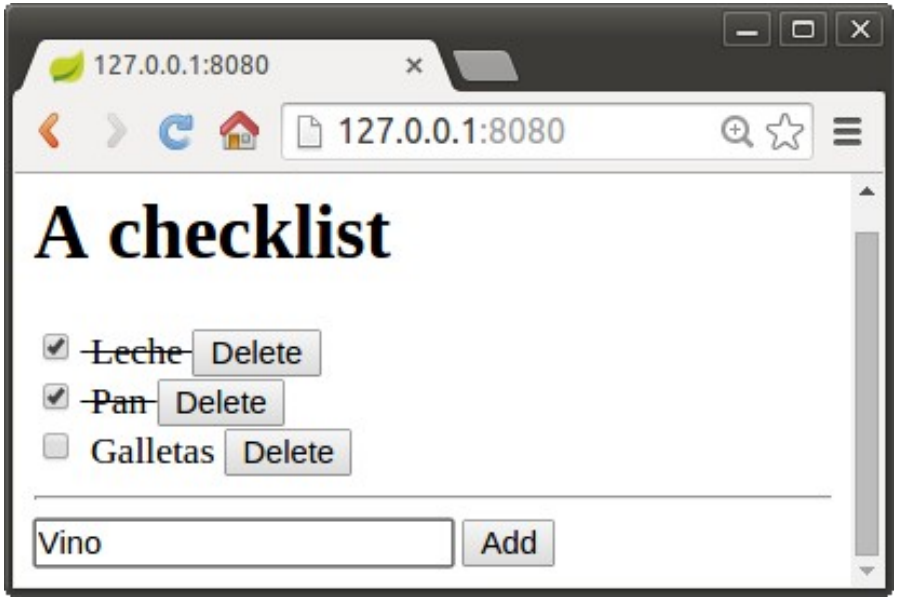
    - Status code: 200 (OK) / 404 (Not Found)

Índice de Ejercicios

Ejercicio 5

- Refactoriza el **Ejercicio 4** para que las llamadas a la API REST estén en un servicio stateless **ItemsService**

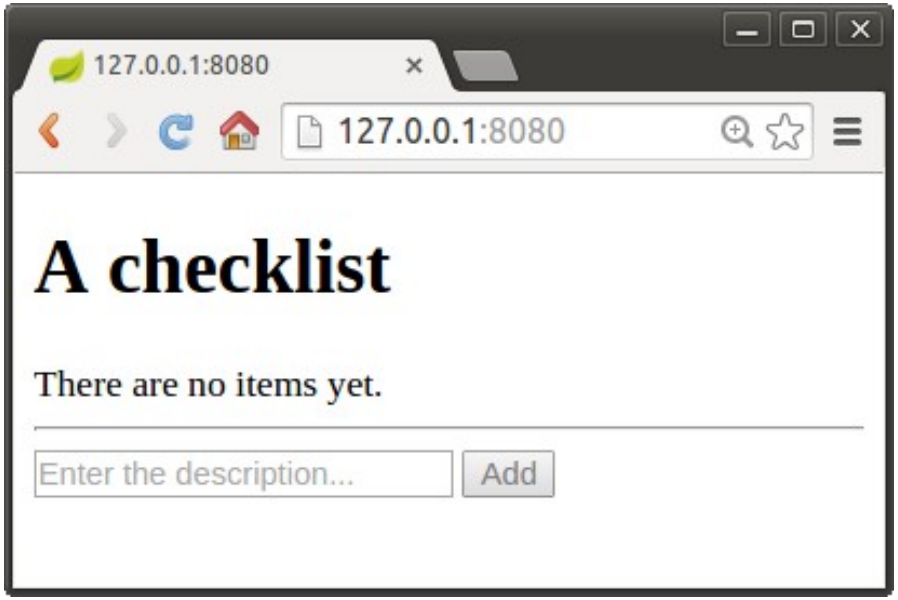


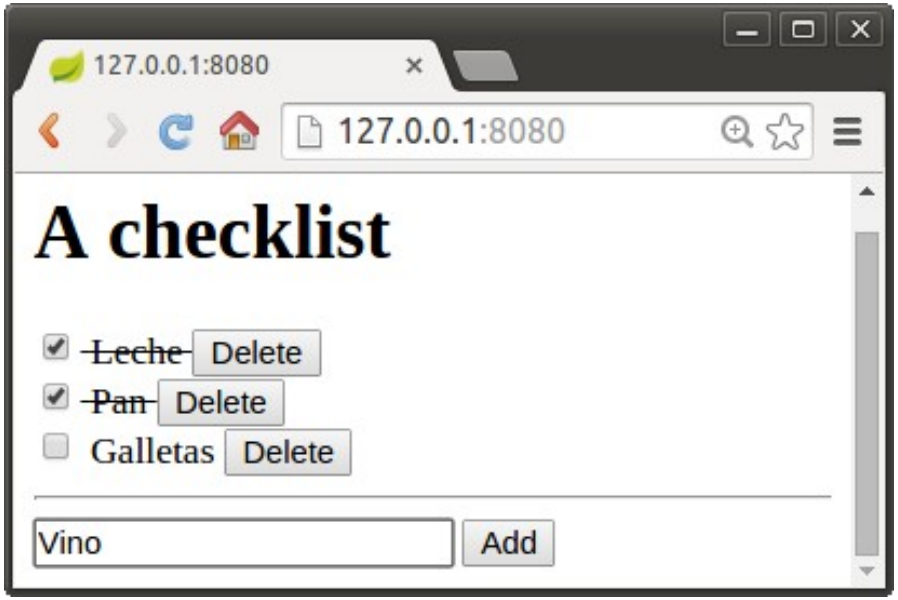


Índice de Ejercicios

Ejercicio 5b

- Refactoriza el ejercicio anterior para usar **async pipe**





01-2023

code
URJC

Desarrollo de Aplicaciones Web

4. Aplicaciones Multipágina - Router

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 6

- Implementa una **aplicación CRUD** de gestión de **libros**
- Funcionalidades (**varias pantallas**)
  - Listado de todos los libros (títulos)
  - Formulario de nuevo libro
  - Vista de detalle de un libro
  - Modificación de libro
  - Borrado de un libro
- Se proporciona una **API REST** (similar a la de los items pero para books).
- Cada libro tiene las propiedades: id, title, description

code
URJC

Desarrollo de Aplicaciones Web

5. Librerías de Componentes

Índice de Ejercicios

Índice de Ejercicios

Ejercicio 7

- **Selecciona una librería de componentes o un template**
- **Mejora el aspecto de la aplicación de gestión de libros**
  - Haz que los formularios tengan "estilo"
  - Que los cuadros de diálogo no sean los nativos del navegador
  - En el listado principal incluye más información además del título del libro (forma de tabla)
  - Pon una gráfica de "descargas" (con datos simulados)

code URJC

Desarrollo de Aplicaciones Web

Proyecto Final

Proyecto

Diseña e implementa una aplicación web

NOTA: El contenido de este enunciado puede actualizarse y completarse a medida que avanza el curso. Cualquier actualización será notificada en los foros de la asignatura en el aula virtual.

Objetivo

Que el alumno implemente una aplicación web de la forma más parecida a como lo haría a nivel profesional. Algunas de las características de la práctica que simulan un entorno real son las siguientes:

- La práctica se desarrollará en un equipo formado por 4 o 5 alumnos. De esa forma el equipo será similar al que se puede encontrar en un desarrollo web profesional.
- Se utilizarán herramientas profesionales, tanto para el desarrollo en sí como para compartir el código entre los miembros del equipo (Eclipse, Visual Studio Code, GitHub, Docker...)
- La temática de la web podrá ser elegida libremente por los alumnos de cada equipo. De esta forma, los alumnos estarán más motivados y podrán incluir en la aplicación aquellas funcionalidades que deseen (dentro de unos límites que se definirán más adelante en este enunciado)

Temática de la aplicación web

La temática de la aplicación web que hay que diseñar e implementar será elegida libremente por los miembros de cada equipo. Cada equipo puede decidir qué funcionalidades ofrece al usuario, el aspecto gráfico, el esquema de navegación, etc.

A modo de ejemplo, se presentan algunos tipos de aplicaciones web que se podrían implementar, pero el equipo podrá elegir cualquier temática (aunque no esté aquí listada):

- Web de compra/venta de objetos usados (listado, proceso de compra, registro de compras, comunicación entre comprador y vendedor).
- Web de selección/contratación de cuidadores de niños por horas (listado, valoraciones/reputación, filtrado por disponibilidad, características...)
- Web de un gimnasio con seguimiento de entrenamiento (clases colectivas, horarios, registro, información pública...)

- Web de una liga de fútbol / torneo de pádel (equipos, jugadores, partidos, clasificación, calendario...)
- Web de gestión docente de una universidad (horarios, fechas de exámenes, asignación de profesores a asignaturas, fichas de asignaturas, alumnos..)
- Web de un ayuntamiento (noticias, votaciones, registrarse en actividades, calendario de actividades...)
- Web para diseñar viajes turísticos (lugares de interés, planificación, mapas, horarios...)
- Web de una academia de formación (cursos ofertados, activos, reserva, matrícula...)
- Web de venta online (productos, carrito de la compra, stock, pedido, análisis de ventas...)
- Web de reserva de aulas de informática en la universidad (registro de software por aula, calendario, conflictos, gestión manual…)
- Web para formación (tipo Moodle) (asignaturas/cursos por alumno, material trabajo, entrega de trabajos, foro..)
- Web para seguimiento y evaluación de una práctica por fases: (equipos, notas del equipo y de cada integrante, lista de items de corrección, análisis de datos, fechas...)

Es posible copiar alguna web que esté disponible en Internet, por ejemplo una web de noticias como menéame, un gestor de blogs como tumblr, la página web de consulta de horarios de la universidad, etc. En caso de que copie una web existente, se podrán copiar también su diseño gráfico, logotipos, iconos, etc.

Metodología de desarrollo

El proyecto deberá realizarse siguiendo la siguiente metodología de desarrollo.

Desarrollo colaborativo

La aplicación web se desarrollará usando un repositorio de la plataforma GitHub[1]. El código fuente de la aplicación tendrá la licencia Apache 2. El repositorio será creado por el profesor de la asignatura y se le darán permisos de edición a todos los miembros del mismo.

Durante el desarrollo de la aplicación se irá subiendo el código al repositorio a medida que se vaya desarrollando. Es importante que se sigan las buenas prácticas y los commits no sean muy grandes. Es decir, no se considerará adecuado que una aplicación se implemente con un commit que añada decenas de ficheros nuevos. El repositorio no sólo se utilizará para entregar la aplicación, si no que debe usarse para desarrollar la aplicación. Idealmente deberá realizarse un commit del menor tamaño posible que deje la aplicación en un estado estable (que compile). Como mucho, habrá un commit por funcionalidad.

Los mensajes de commit también tienen que ser adecuados y describir correctamente el objetivo del commit. Se recomienda que cada mensaje tenga en su primera línea un resumen del commit de no más de 50 caracteres y opcionalmente una descripción más extensa a partir de la tercera línea

1 https://github.com

(dejando una línea en blanco entre el resumen y la descripción). Existen muchas páginas web con recomendaciones sobre cómo hacer buenos commits en git[2] [3].

Para gestionar el repositorio de github se puede usar cualquier cliente de git.

Los miembros del equipo de desarrollo pueden coordinarse entre sí como consideren conveniente, aunque es recomendable que utilicen un tablero Trello[4] o el servicio de GitHub Projects[5] para gestionar el estado de las tareas y su responsable.

Desarrollo por fases

El proyecto será desarrollado en diferentes fases. De esa forma, los alumnos realizarán un trabajo continuo a lo largo del curso y el profesor podrá hacer un mejor seguimiento del mismo.

Las fases en las que se divide el desarrollo de la aplicación web son:

- **Fase 0:** Formación del equipo y definición de las funcionalidades de la web.
- **Fase 1:** Maquetación de páginas web con HTML y CSS.
- **Fase 2:** Web con HTML generado en servidor y AJAX.
- **Fase 3:** API REST a la aplicación web y despliegue con docker.
- **Fase 4:** Web con arquitectura SPA.

El contenido de cada una de estas fases se describe en detalle más adelante en el enunciado del proyecto.

El código de cada una de las fases deberá estar en el repositorio de código en las siguientes fechas antes de que comience la clase:

- **Fase 0:** 6 Febrero 2022 – 2 semanas
- **Fase 1:** 13 Febrero 2022 – 1 semana
- **Fase 2:** 6 Marzo 2022 – 3 semanas
- **Fase 3:** 28 Marzo 2022 – 3 semanas
- **Fase 4:** 25 Abril 2022 – 4 semanas

Cuando el código del repositorio esté listo para la entrega de una fase, será necesario hacer un tag en el repositorio con el nombre de la fase que se vaya a entregar. Por ejemplo, al entregar la Fase 2 se deberá crear un tag llamado “fase2”, que será el que será evaluado. Los tags se pueden hacer desde la interfaz web de GitHub al crear una “release”.

IMPORTANTE: No se evaluará el código de la rama main. Si no se ha creado el tag se considera que la fase no se ha entregado.

2 https://github.com/trein/dev-best-practices/wiki/Git-Commit-Best-Practices
3 https://github.com/erlang/otp/wiki/writing-good-commit-messages
4 https://trello.com
5 https://github.com/blog/2256-a-whole-new-github-universe-announcing-new-tools-forums-and-features

Documentación

La documentación del proyecto se incluirá en el fichero README.md en la raíz del repositorio de código. Este fichero se editará usando adecuadamente el formato Markdown (títulos, negritas/cursivas, tablas, texto en formato código, etc.). Se debe verificar que el documento README.md se visualiza correctamente desde la web de GitHub.

En la descripción de cada una de las fases se indica de forma detallada el contenido de la documentación solicitada.

Evaluación

Cada una de las fases serán evaluadas mediante defensa presencial en horario de clase. Todos los alumnos del equipo deben estar presentes en la defensa al tratarse de una actividad de evaluación, aunque tengan dispensa académica.

Presentación de la fase 0 (no evaluable)

La fase 0 no lleva asociada calificación porque consiste únicamente en definir la funcionalidad de la web y la creación del equipo.

En la presentación de esta fase el profesor verificará que la funcionalidad de la web definida por los alumnos cumple con los requisitos exigidos en el enunciado. Si se identifican carencias serán indicadas a los alumnos para que las solventen antes de comenzar con la siguiente fase.

Presentación de la fase 1 (no evaluable)

La fase 1 no lleva asociada calificación porque se asume que los alumnos ya han adquirido los conocimientos necesarios para desarrollar esta fase en la asignatura “Fundamentos para la web” de 2º curso.

En la presentación de esta fase el profesor verificará que el diseño de las pantallas y la navegación entre ellas cumple con los requisitos exigidos en el enunciado. Si se identifican carencias serán indicadas a los alumnos para que las solventen antes de comenzar con la siguiente fase.

Evaluación de las fases 3, 4 y 5 (prácticas 1, 2 y 3)

La evaluación de las fases 3, 4 y 5 del proyecto llevará asociada una calificación que corresponderá con las notas de la práctica 1, 2 y 3 respectivamente.

La evaluación de cada fase se realizará en dos partes:

- **Parte 1: Cumplimentar formulario de auto-evaluación:** Un único miembro del equipo cumplimentará un cuestionario de auto-evaluación sobre el estado de la práctica (sólo se completará un cuestionario por cada equipo). Este cuestionario se podrá actualizar todas las veces que sea necesario hasta el momento límite de presentación de la fase. En él se realizan preguntas detalladas sobre el estado de las funcionalidades, la calidad del código y el estado de la documentación. Asociado a cada uno de los elementos del cuestionario se indica la

penalización que tiene para la calificación no completar ese elemento.

IMPORTANTE: Si se detecta cualquier intento de completar el cuestionario con información no veraz el profesor calificará la práctica como **suspensa** hasta la convocatoria extraordinaria.

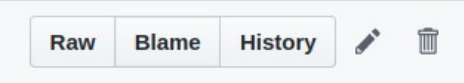
- **Parte 2: Evaluación por el profesor:** El profesor evaluará cada fase de la siguiente forma:
  - **Demostración:** El profesor pedirá que se le haga una demostración de la aplicación usándola como si fuera un usuario. Los miembros del equipo realizarán un recorrido por la aplicación mostrando sus funcionalidades más importantes con los diferentes tipos de usuarios: anónimo, registrado y administrador.

    IMPORTANTE: La demostración deberá estar preparada de antemano, de forma que los alumnos la puedan realizar de forma rápida. Por ejemplo, si hay que subir una imagen, el alumno que haga la demostración deberá tener una imagen adecuada preparada en su disco.
  - **Revisión de la auto-evaluación:** El profesor revisará cada una de las preguntas del cuestionario de de auto-evaluación durante la defensa. Para cada pregunta, pedirá que los miembros del equipo realicen una demostración del funcionamiento de la aplicación web o pedirá que le muestren la documentación o el código fuente. Por ejemplo, si un diagrama se indica como "completado", se podrá pedir al equipo que muestre ese diagrama durante la defensa.

Participación de los miembros del equipo

Todos los miembros del equipo tienen que participar activamente en el desarrollo de cada una de las fases de la práctica. Para poder justificar su participación, cada uno de los miembros del equipo debe incluir en el README.md:

- Un párrafo describiendo de forma textual las tareas realizadas en esa fase.
- Listado de los 5 commits más significativos durante la fase. Cada uno de esos commits deberá ser un link que lleve a este commit en la interfaz web de GitHub.
- Listado de los 5 ficheros en los que más haya participado el miembro. Se podrá demostrar su participación al usar la opción "blame" de GitHub al visualizar ese fichero.



Si el profesor detecta una baja participación de un alumno concreto, podrá reducirse la nota de ese alumno y podría llegar a **suspender** dicha fase.

Aunque el trabajo es colaborativo y es importante el reparto de tareas, todos los miembros del equipo deberán conocer el código completo de la aplicación (aunque algunas partes hayan sido desarrolladas por otro compañero).

Todos los miembros del equipo deberán estar presentes en la defensa porque el profesor podrá preguntar individualmente a cualquier alumno si lo considera necesario. La ausencia no justificada

llevará aparejada una penalización de **2 puntos** en la calificación.

Calidad del código fuente

Se exigirá una mínima calidad del código fuente. Al menos deberán tenerse en cuenta los siguientes aspectos:

- El código deberá estar formateado correctamente y usar la misma reglas de estilo en todos los ficheros (se recomienda configurar el entorno de desarrollo para formatear al guardar y así evitar problemas de este tipo).
- El código y los comentarios del mismo deberán estar escritos completamente en inglés. Sólo podrán existir términos en castellano en el código cuando vayan a mostrarse en el interfaz de usuario. La existencias de texto en castellano en el código llevará aparejada una penalización en la calificación.
- Las variables, parámetros, atributos, clases e interfaces deberán tener nombres adecuados que describan su objetivo.
- Se evitará el código duplicado. Cuando dos fragmentos de código sean similares, se utilizarán las técnicas adecuadas para fomentar la reutilización (herencia, composición, subprogramación…)
- Los métodos no serán muy largos y no tendrán mucha complejidad ciclomática.
- El código deberá ser razonablemente eficiente. Por ejemplo, no se considerará válido hacer una consulta a la base de datos para obtener una lista de elementos y luego filtrar esa lista en memoria usando código Java.
- Si se quiere mostrar texto de depuración en la consola del proceso Java, se usará la librería de logs en vez de usar System.out.println().

Si se incumple alguna de estos buenas prácticas la calificación podrá verse penalizada hasta en **2 puntos** dependiendo de su grado.

Calificación

Al evaluar cada una de las fases se tendrán en cuenta los siguientes aspectos:

- Las fases 3, 4 y 5 tendrán una calificación entre 0 y 10.
- Si la fase no cumple con los mínimos exigidos (falta de funcionalidad, incompleta) o no funciona correctamente (tiene errores) se considerará suspensa. El equipo que suspenda una entrega podrá realizar las entregas de las sucesivas fases (subsanando los errores de la entrega suspensa) y recuperar la fase suspensa en la convocatoria extraordinaria.
- Si el profesor detecta carencias menores durante la evaluación de fase podrá poner una nota condicionada a que esos aspectos sean subsanados en un plazo de 1 semana. Si no son subsanados, la nota final para esa fase sería inferior a la nota indicada.
- La nota de cada alumno del equipo es independiente. Un alumno puede obtener más o menos nota que el resto de sus compañeros. La nota individual de cada alumno se determina en base a:

- El conocimiento de la aplicación durante la defensa, ya que todos los alumnos tienen que conocer el código de toda la aplicación aunque ellos no hayan desarrollado esa parte.
- La contribución de cada miembro en esa fase. Se puede llegar al extremo de suspender la práctica a un miembro del equipo si no ha colaborado de forma mínima en la fase.

Convocatoria extraordinaria

Aquellas fases aprobadas en la convocatoria ordinaria se guardan para la convocatoria extraordinaria. Es decir, en la convocatoria extraordinaria los alumnos sólo tendrán que realizar aquellas fases suspensas o no entregadas.

La nota final de la práctica se calculará con los mismos porcentajes que en la convocatoria ordinaria.

En la convocatoria extraordinaria si no se cumplen con las funcionalidades o la aplicación tiene fallos, la práctica (y por tanto la asignatura) se considerará suspensa.

Modificación de miembros del equipo

Aunque se debería evitar en la medida de lo posible, los equipos pueden tener modificaciones en sus miembros debido a las siguientes causas:

- **División del equipo en dos debido a conflictos entre los miembros:** En ese caso, el equipo se puede dividir en 2 y cada nuevo equipo podrá seguir el desarrollo de la práctica partiendo del código en el momento de la división. Hay que evitar esta situación por el incremento de trabajo que supondría para los dos nuevos equipos.
- **Alumno abandona un equipo por conflictos:** Se podrá unir a otro equipo siempre y cuando tenga menos de 4 alumnos y tenga aprobadas las mismas fases que el alumno que quiere incorporarse. Es decir, un alumno no se puede ir a un equipo que tenga aprobadas las fases 2 y 3 si él no tiene aprobada la fase 2 en su equipo original.
- **Alumno deja la asignatura:** Se considera que el alumno ya no forma parte del equipo.

Fase 0: Formación del equipo y definición de las funcionalidades de la web

En la fase 0 se deben realizar las siguientes tareas:

- Formar el equipo de desarrollo.
- Definir las funcionalidades de la aplicación web.

Esta fase no es evaluable.

Formación del equipo

Los alumnos registrarán su equipo en el siguiente documento compartido antes del 30 de Enero a las 14:00.

https://urjc-my.sharepoint.com/:x:/g/personal/micael_gallego_urjc_es/EQh_OkIhGEZKqIOl6nB3g7EBnlL636X23om9P4F96RIy1A?e=ETwTOm

Los equipos serán de 4 o 5 integrantes. Aquellos alumnos que no hayan podido formar equipo en la fecha prevista serán asignados a un equipo por el profesor.

Aquellos equipos de 4 alumnos podrán incorporar un nuevo alumno no asignado si el profesor lo considera oportuno.

Requisitos de la aplicación web

Para que la complejidad sea similar para todos los equipos, la aplicación web se deberán diseñar de forma que cumpla con los siguientes requisitos:

- **Entidades:** Una entidad representa un concepto que la aplicación web guarda en la base de datos. Son las clases de dominio, las tablas de la base de datos. La aplicación web deberá gestionar 4 entidades. Una de ellas será la entidad Usuario (para guardar la información de los usuarios que acceden a la web). Las otras 3 entidades dependen de la temática de la web. Es importante que las entidades estén relacionadas entre sí. Por ejemplo:
  - **Web de gestión de torneos o ligas:** Entidades usuario, equipo, torneo y partido. La relación se tiene porque un equipo juega en partidos que forman parte de un torneo.
  - **Web de gestión de cursos:** Entidades usuario, curso, material, mensaje. La relación se tiene porque un alumno está en un curso, que tiene materiales y mensajes en el foro.
  - **Web de cuidado de niños:** Entidades usuario (progenitor o cuidador), solicitud, jornada de cuidado. La relación se tiene porque una jornada de trabajo ha sido realizada a un progenitor por un cuidador. Además, el progenitor puede poner una solicitud, que se puede convertir en una jornada realizada por un cuidador.
  - **Web de gestión de exámenes:** usuario (profesor o alumno), examen, asignatura, realización de examen. Las entidades están relacionadas porque un profesor imparte una asignatura con alumnos. Esos alumnos tendrán que realizar el examen de la asignatura.
- **Tipos de usuarios:** La aplicación web deberá considerar tres tipos de usuarios:
  - **Usuario anónimo:** Aquél usuario que visita la web y no introduce ningún tipo de credenciales para consultar contenido y realizar búsquedas. Salvo que esté justificado por el tipo de aplicación, este usuario sólo podrá consultar información de la web, pero no podrá crearla ni modificarla.
  - **Usuario registrado:** Aquél usuario que tiene que usar sus credenciales para acceder a la web. Este usuario tendrá datos personalizados como su nombre, una imagen, el histórico de acciones realizadas en la web (mensajes en foros, compras, etc..). La web deberá permitir el registro de nuevos usuarios.
  - **Usuario administrador:** Aquél usuario que tiene control total sobre la información de la web. Por ejemplo el alta de productos, la creación de los campeonatos, registro de información o actividades, etc. La web tendrá un único usuario administrador con una contraseña especificada en un fichero de configuración (cifrada).
- **Permisos de los usuarios:** La web tiene que estar diseñada para que los usuarios registrados

puedan ser dueños de ciertos datos. Por ejemplo, si se trata de una tienda, los pedidos previos que ha realizado. Si es una web de contratación de cuidadores, el histórico de cuidados. Si es una web de un gimnasio, los comentarios que pone a cada clase. Esta funcionalidad es importante porque la web debe implementar los mecanismos de seguridad adecuados para que sólo el usuario que ha creado un elemento (su dueño) pueda borrarlo o editarlo.

- **Imágenes:** La web tiene que permitir la subida de imágenes desde el navegador web. Por ejemplo como avatar de los usuarios, fotos de productos, etc.
- **Gráficos:** Se deberán usar gráficos (*charts*) para mostrar algún tipo de información en la web. Por ejemplo:
  - **Web de gestión de torneos o ligas:** Gráfica de líneas con la puntuación/clasificación de los equipos a lo largo del tiempo.
  - **Web de gestión de cursos:** Número de alumnos registrados en cada uno de los cursos.
  - **Web de cuidado de niños:** Gráfica de líneas con las jornadas realizadas por mes para un cuidador.
  - **Web de gestión de exámenes:** Gráfica de barras con el número de alumnos por asignatura.
- **Tecnología complementaria:** Se deberá hacer uso de alguna funcionalidad que se implemente con alguna tecnología o librería que no se haya visto en el material de la asignatura. Por ejemplo:
  - Envío de correos a los usuarios.
  - Generación de PDFs. (facturas, entradas, etc.)
  - Uso de websockets para implementar edición o avisos en tiempo real (chat, pizarra compartida, etc.)
  - Uso de mapas de GoogleMaps / OpenStreetMap para posicionar elementos de la aplicación web (restaurantes, pedidos, etc.)
  - Uso de una API REST de algún servicio externo (Información meteorológica, libros, películas, etc...).
- **Algoritmo o consulta avanzada:** La aplicación web deberá ofrecer alguna funcionalidad que requiera la implementación de un algoritmo o un tratamiento avanzado sobre los datos que gestiona. No basta con que las entidades se puedan crear, modificar, actualizar y borrar. Por ejemplo:
  - Si se opta por una web que permita hacer el seguimiento de una liga de fútbol, la clasificación se deberá calcular de forma automática a medida que se vayan registrando los resultados los partidos.
  - Si se opta por una web de contratación de cuidadores para niños, se implementará un sistema de valoraciones, de forma que la búsqueda tenga en cuenta las valoraciones (mejor valorados primero), o una búsqueda basada en la distancia a la que puede ir el cuidador, etc.
  - Si es una página web de venta de productos, se deberá implementar un sistema de ofertas personalizadas en base a los productos que haya comprado previamente el usuario (por ejemplo mostrar como recomendado un producto de la categoría que más haya comprado el usuario en el pasado).

- Si se diseña una página de gestión de exámenes, se deberán poder analizar las posibles restricciones que existan (que un mismo alumno no tenga dos exámenes el mismo día, que un profesor no tenga exámenes en el mismo momento, etc).

Documentación

Se creará un fichero README.md usando el formato Markdown en la raíz del repositorio de GitHub. Este fichero se visualizará al entrar en la web de GitHub de ese repositorio y tendrá que estar correctamente formateado usando secciones, subsecciones, tablas, código fuente como texto en formato monoespaciado, imágenes, etc.

La documentación se podrá escribir en castellano, aunque se recomienza que se haga en inglés.

El contenido del fichero README.md deberá contener la siguiente información:

- Nombre de la aplicación web.
- Integrantes del equipo de desarrollo: Nombre, Apellidos, correo oficial de la universidad y cuenta en GitHub.
- Si se utiliza trello o cualquier otra herramienta para la coordinación del equipo, deberá ser pública y se incluirá el link de la misma.
- Una sección describiendo cada uno de los aspectos principales de la aplicación web:
  - **Entidades:** Es necesario indicar las entidades principales que gestionará la aplicación. No es necesario definir sus atributos, aunque si las relaciones que existan entre ellas.
  - **Permisos de los usuarios:** Describir someramente los permisos de cada uno de los tipos de usuario. Es importante indicar de qué entidades es dueño el usuario.
  - **Imágenes:** Indicar qué entidades tendrán asociadas uno o varias imágenes por cada objeto/registro.
  - **Gráficos:** Qué información se mostrará usando gráficos. De qué tipo serán los gráficos (líneas, barras, tarta, etc).
  - **Tecnología complementaria:** Qué tecnología complementaria se empleará.
  - **Algoritmo o consulta avanzada:** Indicar cuál será el algoritmo o la consulta avanzada que se implementará.

Fase 1: Maquetación de páginas con HTML y CSS

En la fase 1 se debe definir el esquema de navegación de la web y la maquetación (HTML y CSS) de las páginas más importantes de la web. Se deben incluir las páginas de los administradores (que tienen permisos para editar toda la información de la web).

Para realizar el diseño, se utilizarán datos de ejemplo. Por ejemplo, si la aplicación web es una tienda virtual, para diseñar la página principal, se utilizarán productos de ejemplo con sus imágenes, descripciones, etc. Esta información de ejemplo puede copiarse de cualquier aplicación web de temática similar.

Para el diseño web se utilizará un framework CSS (Bootstrap[6] usando una plantilla gratuita o comercial o Material Design[7]). El objetivo es que la aplicación web tenga un aspecto profesional.

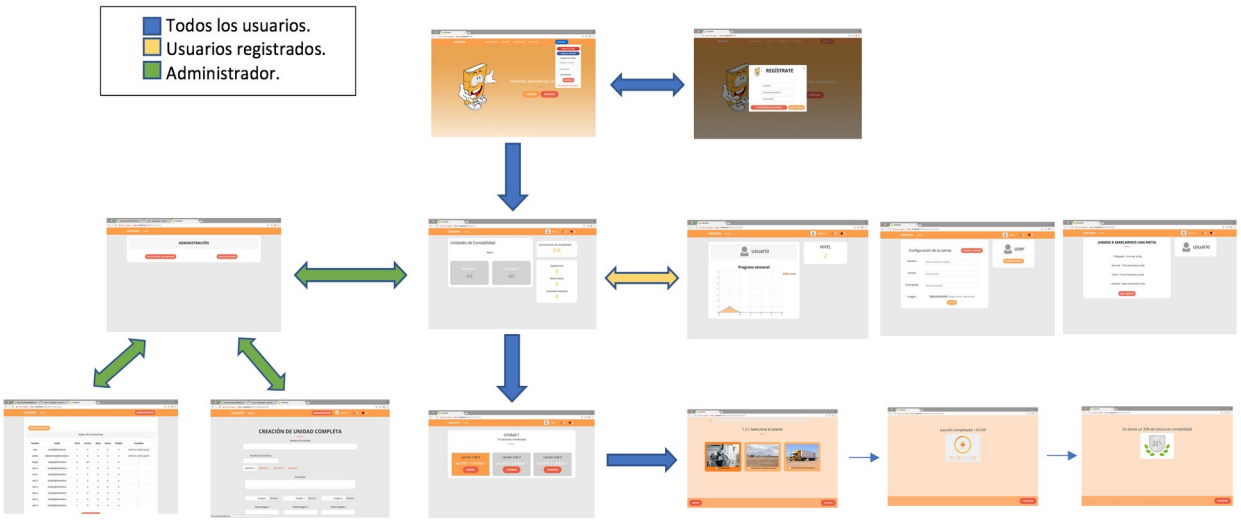
En los casos en los que sea posible, algunos enlaces de las pantallas permitirán simular la navegación real por la página. Por ejemplo, si la aplicación es una tienda, cuando se pulse el enlace para ver la información del producto en detalle, se debería navegar a una página de detalle de un producto (no tiene por qué ser el producto concreto). No obstante, en esta fase los formularios no funcionarán.

Esta fase no es evaluable.

Documentación

Se añadirá a la información de la fase 0 en el README del repositorio la siguiente documentación:

- **Capturas de las pantallas:** Se incluirán capturas de pantalla de cada una de las páginas principales que hayan sido maquetadas. Se acompañarán con una breve descripción en cada una de ellas (un párrafo como mucho).
- **Diagrama de navegación**: Para mostrar la navegación se creará un diagrama en el que se indicará desde qué página se puede navegar hasta otras páginas. Para ello, las páginas del diagrama pueden ser capturas de pantalla en miniatura de las maquetaciones que se hayan realizado. A continuación se muestra un ejemplo de diagrama de navegación:



6 https://getbootstrap.com/
7 https://github.com/material-components/material-components-web/

Fase 2: Web con HTML generado en servidor y AJAX (Práctica 1)

En la fase 2 se debe implementar la aplicación web completamente funcional.

Funcionalidades de la aplicación web

La aplicación deberá tener las siguientes características:

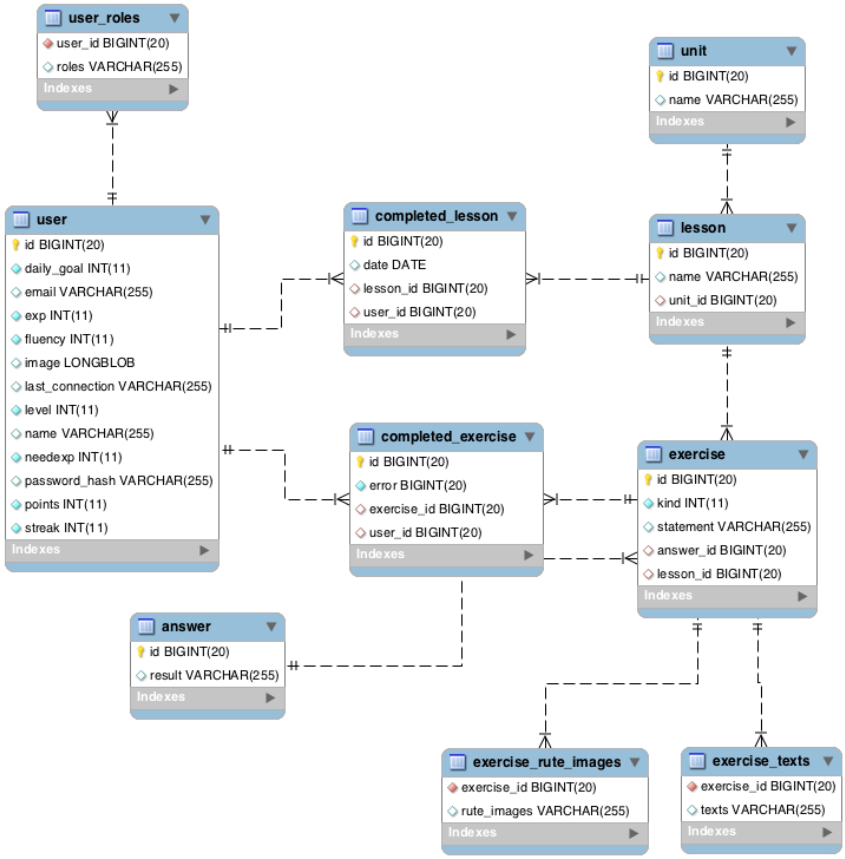
- **Aplicación completa:** La aplicación web se implementará con SpringBoot y base de datos PostgreSQL. Es muy importante que la funcionalidad de la aplicación web sea completa y no queden cosas sin implementar. Es preferible tener menos funcionalidades de las inicialmente previstas a tener funcionalidades a medio implementar. Por ejemplo, no tiene sentido tener correctamente implementado el acceso a la base de datos y tener sin completar la mitad de las páginas. Es preferible tener funcionalidades implementadas completamente (aunque sean menos de las inicialmente previstas) que tener funcionalidades a medias e incompletas. Obviamente, el profesor valorará si el recorte en funcionalidades es razonable o la fase tiene que recuperarse.
- **Datos de ejemplo:** Se cargarán datos de ejemplo en la base de datos al ejecutar la aplicación. Estos datos de ejemplo estarán especificados en el código Java. Los datos e imágenes deberán ser representativos del tipo de aplicación web que se esté desarrollando: libros, restaurantes, productos, etc. Esta información se puede obtener de otras webs.
- **Páginas de error:** Cuando se intente acceder a una URL inexistente o se produzca un error en el servidor se deberá generar una página de error que tenga el mismo estilo gráfico que el resto de páginas de la aplicación.
- **Paginación:** Todas las páginas que potencialmente vayan a mostrar más de 10 elementos deberán mostrar únicamente los 10 primeros y se permitará al usuario cargar más. Para la carga de más elementos, se mostrará un botón o enlace de "Más resultados". Al pulsar este botón, un código JavaScript realizará una llamada AJAX al servidor y cargará los siguientes 10 elementos, que se añadirán en la página actual a los elementos ya existentes. Pulsar el botón o enlace de "Más resultados" no debe recargar la página completa en el navegador web. El servidor web devolverá el HTML que será incrustado usando JavaScript en la página web previamente cargada. Durante el tiempo que dure la carga de elementos, en la página deberá aparecer una animación de espera (spinner). Se puede ver un ejemplo de este funcionamiento en la página de Google Images.
- **Usuarios:** La aplicación gestionará los usuarios usando Spring Security. Si un usuario intenta acceder a una URL sobre la que no tiene permisos, se debería mostrar un error de acceso. Por ejemplo, si un usuario intenta editar un registro del que no es dueño, se deberá mostrar un error. El administrador tendrá una contraseña especificada en el fichero de propiedades (cifrada con BCrypt). Existirán dos usuarios registrados precargados en la base de datos. Se deberá ofrecer un mecanismo para que un usuario pueda registrarse en la aplicación web con un formulario de registro.
- **Código fuente:** Todo el código fuente del proyecto SpringBoot se guardará en una carpeta "backend" del proyecto. Es decir, el fichero pom.xml estará dentro de la carpeta "backend", no en la raíz del repositorio.

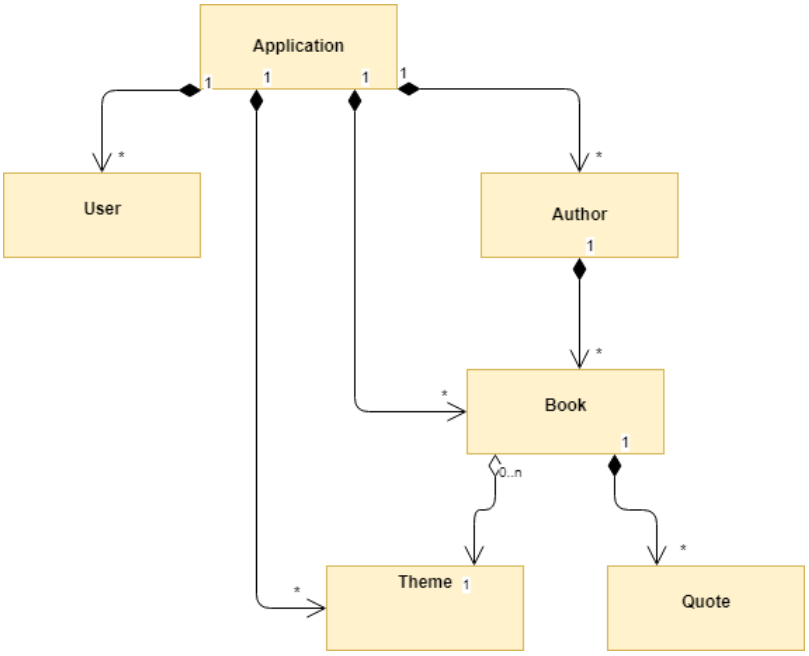
- **Seguridad:** La aplicación web deberá servirse por https en el puerto 8443.
- **Imágenes en base de datos:** Para facilitar el despliegue de la aplicación en entornos restringidos, las imágenes se guardarán en la base de datos en vez de en el sistema de ficheros.

Documentación

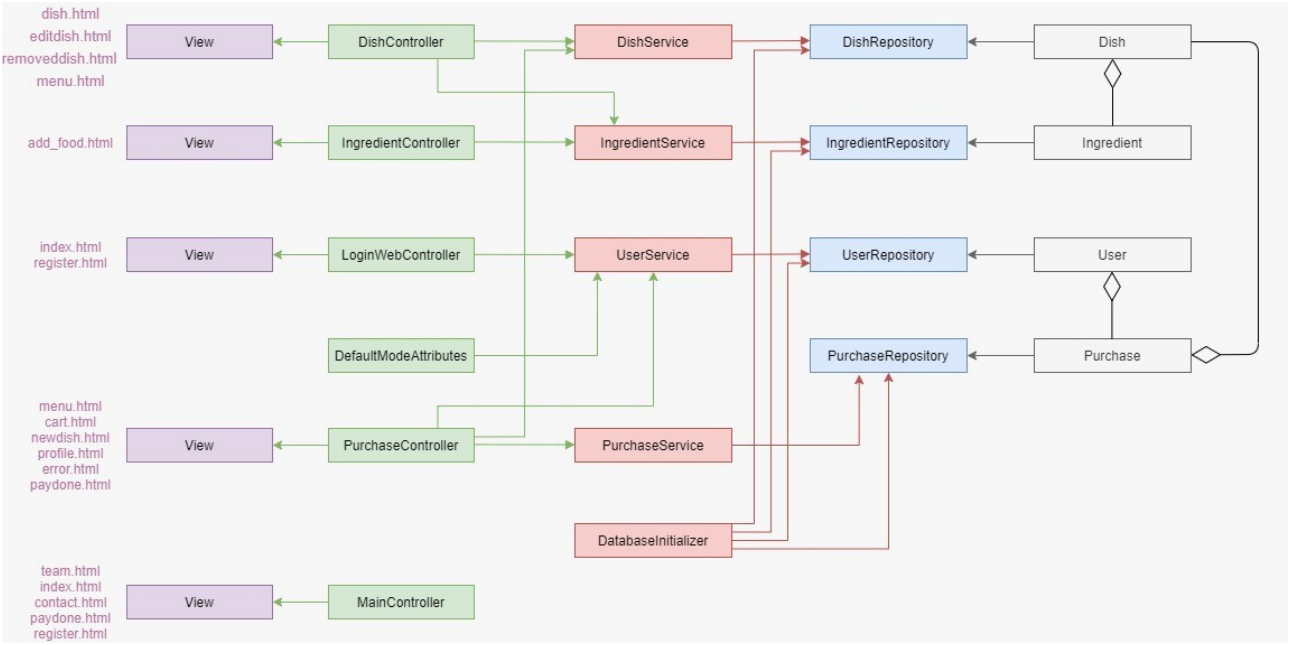
En el README se deberá incluir la siguiente información:

- **Navegación:** Se actualizarán las capturas de pantalla de las páginas principales de la aplicación que se hicieron en la fase 1. En caso de que haya cambiado la navegación, se deberá actualizar también el diagrama de navegación.
- **Instrucciones de ejecución:** Se indicará qué pasos deben seguirse para poder descargar el código del repositorio, construir y ejecutar la aplicación. También se debe especificar cuales son los requisitos (versión de Java, versión de PostgreSQL, maven, etc.). Las instrucciones se especificarán preferentemente como comandos a ejecutar por la línea de comandos. En caso de que no sea posible, se indicará cómo instalar/configurar aplicaciones de forma interactiva.
- **Diagrama con las entidades de la base de datos:** Se incluirá un diagrama con las entidades de la base de datos, sus campos y las relaciones entre ellas. Se podrá incluir un diagrama entidad-relación de la base de datos o un diagrama UML de las clases Java. A continuación se muestran algunos ejemplos de diagramas:





- **Diagrama de clases y templates:** Se creará un diagrama de clases de la aplicación. No se incluirán ni atributos ni métodos en las clases. Se mostrarán las relaciones entre las clases (asociación, composición y herencia) y se diferenciará claramente qué clases son @Controller, @Service, Repository, clases del dominio (entidades) u otro tipo de clases. Para ello se puede usar un código de colores, una distribución de las clases por partes u otro mecanismo. En este diagrama también se incluirán los ficheros que contienen los templates y se indicará con qué @Controller se relacionan. A continuación se muestra un diagrama de ejemplo que usa los colores para indicar el tipo de elemento.



- **Participación de miembros:** Se debe incluir una sección en el README en la que cada miembro del equipo indique su participación en esa fase con:
  - Descripción textual de las tareas realizadas en la fase.
  - Listado de los 5 commits más significativos durante la fase (enlazados a dichos commits en GitHub).
  - Listado de los 5 ficheros en los que más haya participado (enlazados a dichos ficheros en GitHub).

Defensa

Los miembros del equipo deberán tener preparada una demostración de las funcionalidades de la aplicación. Por ejemplo, deberán diseñar un guión con los pasos a seguir y deberán tener preparados datos de ejemplo (imágenes, otros datos, etc.). En esa demostración se mostrará el comportamiento con diferentes tipos de usuarios: no autenticado, autenticado y administrador. Para facilitar la demostración, uno de los integrantes del equipo ya tendrá la aplicación lista para ser usada en su ordenador. Puede utilizar varios navegadores o el modo de incógnito para simular varios usuarios.

Fase 3: Incorporación de una API REST a la aplicación web y despliegue con Docker (Práctica 2)

En la fase 3 se deberá implementar una API REST para la aplicación y se deberá empaquetar y distribuir usando la tecnología Docker.

Requisitos de la API REST

La implementación de la API REST se tendrá que realizar teniendo en cuenta las siguientes cuestiones:

- La aplicación web ofrecerá a la misma vez la interfaz web y la API REST. Esto permite que la web siga disponible y que clientes móviles u otros servicios puedan acceder a la funcionalidad proporcionada por la web.
- Todas las URLs de la API REST comenzarán con “/api” para diferenciarse de las rutas con las que se accede a páginas de la web.
- La API REST debe diseñarse teniendo en cuenta las buenas prácticas:
  - Los métodos http GET, PUT, POST, DELETE deben usarse de forma adecuada para las operaciones de consulta, modificación, creación y borrado.
  - Las URLs deben identificar los recursos. Cada tipo de recurso se identificará en inglés y en plural.
  - Los códigos de estado de respuesta deben usarse de forma adecuada.
  - Las operaciones de creación deberán devolver la header “Location” cuyo valor es la

URL con la que se puede obtener la representación del recurso recién creado.

- Los operaciones de la API REST que permitan filtrar o buscar elementos deberán tener los criterios de filtrado o búsqueda como parámeotros de la URL (después del ?).

- Toda funcionalidad que se pueda realizar desde el interfaz web debe poder realizarse desde la API REST. Es decir, si se implementara una aplicación SPA o una aplicación móvil podría realizar cualquier operación disponible en la web.
- Los listados que estén paginados en la aplicación web también lo estarán en la API REST. Los parámetros usados para configurar la paginación serán los mismos que los usados en la aplicación web (?page=1).
- Para evitar la duplicación de código y para que la lógica de negocio no esté acoplada a los controladores, se deberá implementar un @Service que será usado tanto por el @Controller y por el @RestController. Ese @Service será el que utilice los repositorios. Un controlador nunca podrá acceder a un repositorio directamente.
- Se deberá incluir en la raíz del repositorio de código una colección de peticiones Postman (api.postman_collection.json) con ejemplos de peticiones a todos los endpoints de la API REST que deberá ser completamente funcional con los datos de ejemplo de la aplicación. De esta forma, será muy sencillo verificar el comportamiento de la API REST desde Postman.
- La API REST deberá estar documentada con OpenAPI generado mediante SpringDoc.

Empaquetado con Docker

El empaquetado con Docker deberá implementarse con las siguientes características:

- **Aplicación web y API REST:**
  - Todas las funcionalidades de la web deberán estar disponibles cuando se ejecute en un contenedor Docker. Hay que prestar especial cuidado en la subida de imágenes, ya que deberá implementarse correctamente para que funcione en este entorno.
  - La aplicación deberá estar disponible en el puerto 8443 por https.
  - Para facilitar la implementación, se desactivará la protección contra CSRF en la API REST (en la web seguirá estando disponible).
- **Docker-compose**
  - Se usará docker compose para ejecutar la aplicación completa formada por el contenedor de la base de datos y el contenedor de la aplicación web.
  - Se usará el contenedor de PostgreSQL estándar de DockerHub. No hay que crear una nueva imagen de base de datos.
  - Se creará un nuevo contenedor que contendrá la aplicación SpringBoot (aplicación web y API REST). Este contenedor se debe publicar en DockerHub en una cuenta creada por alguno de los miembros del equipo (puede ser su cuenta personal).
  - Para que la aplicación web se inicie correctamente una vez que la base de datos ha arrancado y es accesible se implementará algún mecanismo de espera o reintentos. Por

ejemplo usando la política `restart:always`[8].

- El fichero docker-compose.yml configurará el nombre del esquema de la base de datos, la ruta de la base de datos y la password de root de la base de datos en el contenedor de la web y en el de la BBDD. Para ello, usará la capacidad de especificar variables de entorno. Conviene revisar la documentación sobre las variables de entorno de la imagen Docker de PostgreSQL y las variables de entorno SPRING_DATASOURC_URL, SPRING_DATASOURCE_USERNAME y SPRING_DATASOURCE_PASSWORD que permiten configurar la base de datos en una aplicación implementada con SpringBoot.

- **Código fuente:** En el repositorio de GitHub se deberá crear una carpeta /docker que tendrá los siguientes ficheros:
  - **Dockerfile:** Fichero usado para crear la imagen docker de la web
  - **create_image.sh/.bat/ps1:** Script de bash, bat o power shell que construirá la imagen docker de la aplicación desde el código fuente. El script se ejecutará una vez clonado el repositorio de código. El único requisito para que el script funcione correctamente debería ser que el sistema tenga instalado docker. No se deberá usar el JDK del sistema. Es decir, el script deberá compilar la aplicación usando JDK y Maven dockerizado. Hay que prestar especial atención a las buenas prácticas para crear el contenedor de forma que tenga exclusivamente el software necesario para ejecutar la aplicación.
  - **docker_compose.yml:** Fichero que al ejecutarse con el comando "docker-compose up" se descargue la imagen de la aplicación y de la base de datos de DockerHub y ejecute la aplicación en el puerto 8443 local de forma que sea accesible en https://localhost:8443/.

Despliegue en Railway

La aplicación web deberá desplegarse en Railway usando la misma imagen Docker usada en el docker-compose.

El nombre del proyecto deberá tener el formato: `codeurjc-daw-2021-22-webappX` (sustituyendo X por el número del equipo).

La base de datos MySQL la proporciona Railway, no se ejecutará con Docker.

La defensa de la práctica se realizará con la aplicación web desplegada en Railway.

Documentación

El README se deberá ampliar con:

- **La URL de la aplicación desplegada en Railway:** Se deberá incluir en el README la URL de Railway donde se ha alojado la aplicación. También se deberán incluir las credenciales de los usuarios de ejemplo (incluyendo el administrador).
- **Documentación de la API REST:** La documentación de la API REST se alojará en la carpeta /api-docs de la raíz del repositorio y estará formada por la especificación OpenAPI (en un fichero api-docs.yaml) y por un fichero HTML generado a partir de esa

8 https://docs.docker.com/compose/compose-file/compose-file-v3/#restart

documentación (api-docs.html).

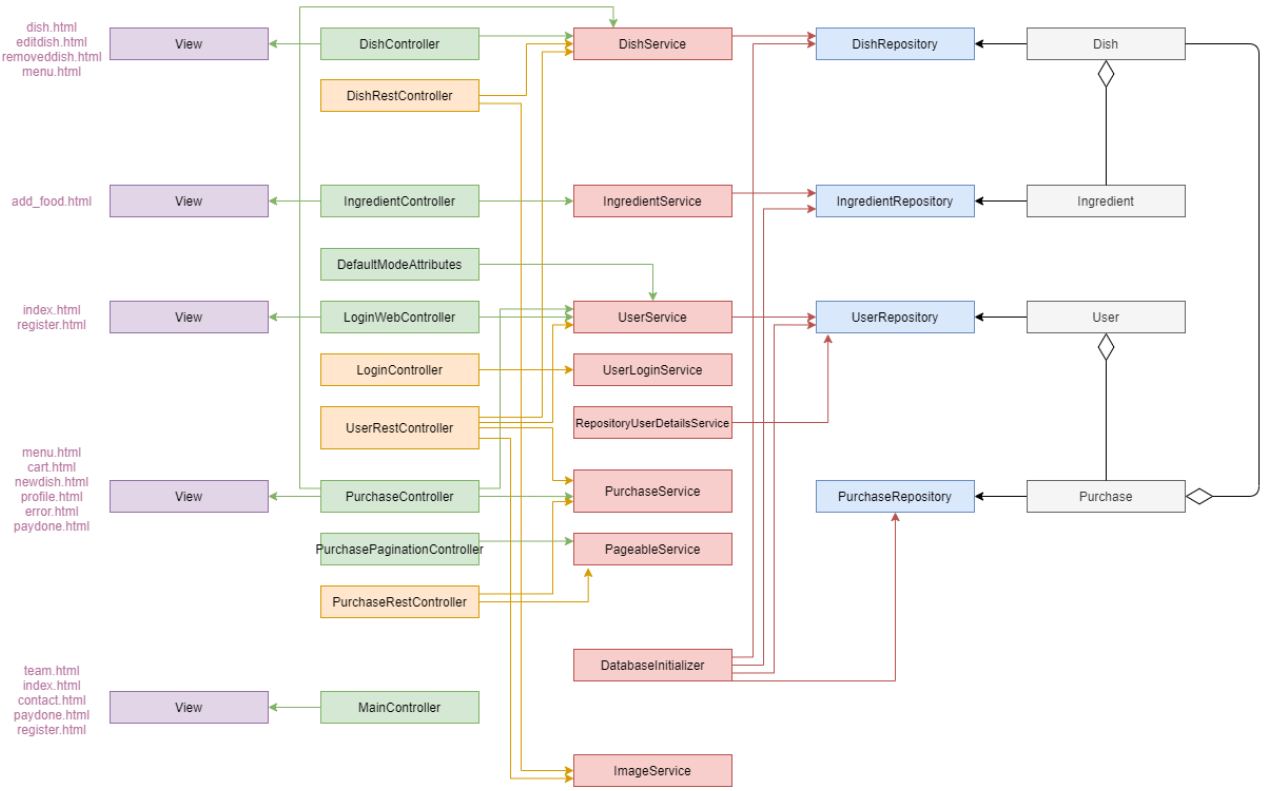
El proyecto de ejemplo de la fase 3[9] contiene la configuración de Maven (en el pom.xml) que permite generar ambos ficheros de forma automática cuando se construye el proyecto con el comando:

```
$ mvn verify
```

Se debe definir la documentación de forma adecuada en el código Java con SpringDoc de forma que las descripciones del documento HTML sean comprensibles.

En el README se debe poner un link a la especificación OpenAPI (fichero /api-docs/api-docs.yaml). También se debe poner un link a la documentación en HTML, pero para que se visualice correctamente en el browser se deberá generar el link con el servicio https://raw.githack.com. Este servicio permite acceder a ficheros HTML de GitHub que se visualizan correctamente en un navegador web.

- **Actualización de diagrama de clases:** Se actualizará el diagrama de clases y templates incluyendo las nuevas clases @RestController. Se prestará especial atención a que los @Service compartidos entre los @Controller y los @RestController aparezcan correctamente reflejados en el diagrama. A continuación se muestra un ejemplo de este tipo de diagrama:



9 https://github.com/codeurjc/daw/tree/main/sample-project-phase3

- **Instrucciones de ejecución de la aplicación dockerizada:** Instrucciones de ejecución usando el docker-compose.yml. Se deberán indicar los requisitos para que se pueda ejecutar, el comando necesario y cómo acceder a la página cuando esté lista para ser usada.
- **Documentación para construcción de la imagen docker:** Se indicará el requisito necesario para construir la imagen dockerizada y cómo ejecutar el script necesario para construir y publicar la imagen Docker.
- **Documentación para desplegar en Railway:** Se deberán describir los requisitos y comandos para poder desplegar la aplicación en Railway.
- **Participación de miembros:** Se debe incluir una sección en el README en la que cada miembro del equipo indique su participación en esa fase con:
  - Descripción textual de las tareas realizadas en la fase.
  - Listado de los 5 commits más significativos durante la fase (enlazados a dichos commits en GitHub).
  - Listado de los 5 ficheros en los que más haya participado (enlazados a dichos ficheros en GitHub).

Defensa

La aplicación deberá estar correctamente desplegada en Railway al empezar la defensa.

Para la demostración de la API REST se deberá tener cargada la colección de peticiones Postman que serán ejecutadas a petición del profesor. Las peticiones deberán estar parametrizadas de forma que se puedan ejecutar contra el despliegue en local o contra el despliegue en Railway.

Para probar la funcionalidad del empaquetado docker, se deberá arrancar la aplicación usando "docker compose up". Los contenedores deberán estar ya descargados en la máquina para no esperar el tiempo de descarga durante la defensa.

Fase 4: Implementación de la web con arquitectura SPA (Práctica 3)

En la fase 4 se debe implementar la aplicación web como cliente SPA con Angular.

Funcionalidades de la aplicación SPA

Esta fase se implementará teniendo en cuenta los siguientes aspectos:

- Este cliente debe tener exactamente la misma funcionalidad que la web implementada en la fase 2. Aunque no es necesario que la interfaz gráfica sea exactamente igual mientras se puedan realizar las mismas operaciones. No obstante, se tendrá que seguir manteniendo el mismo estilo general.
- La aplicación web se podrá usar desde el interfaz original (implementado en la fase 2) o desde el cliente SPA de esta fase. Si se editan datos desde la interfaz original, serán visibles desde la aplicación Angular y viceversa.
- La web con arquitectura tradicional (MVC) será accesible desde la URL principal

(https://localhost:8443/) y la web SPA desde la URL https://localhost:8443/new/

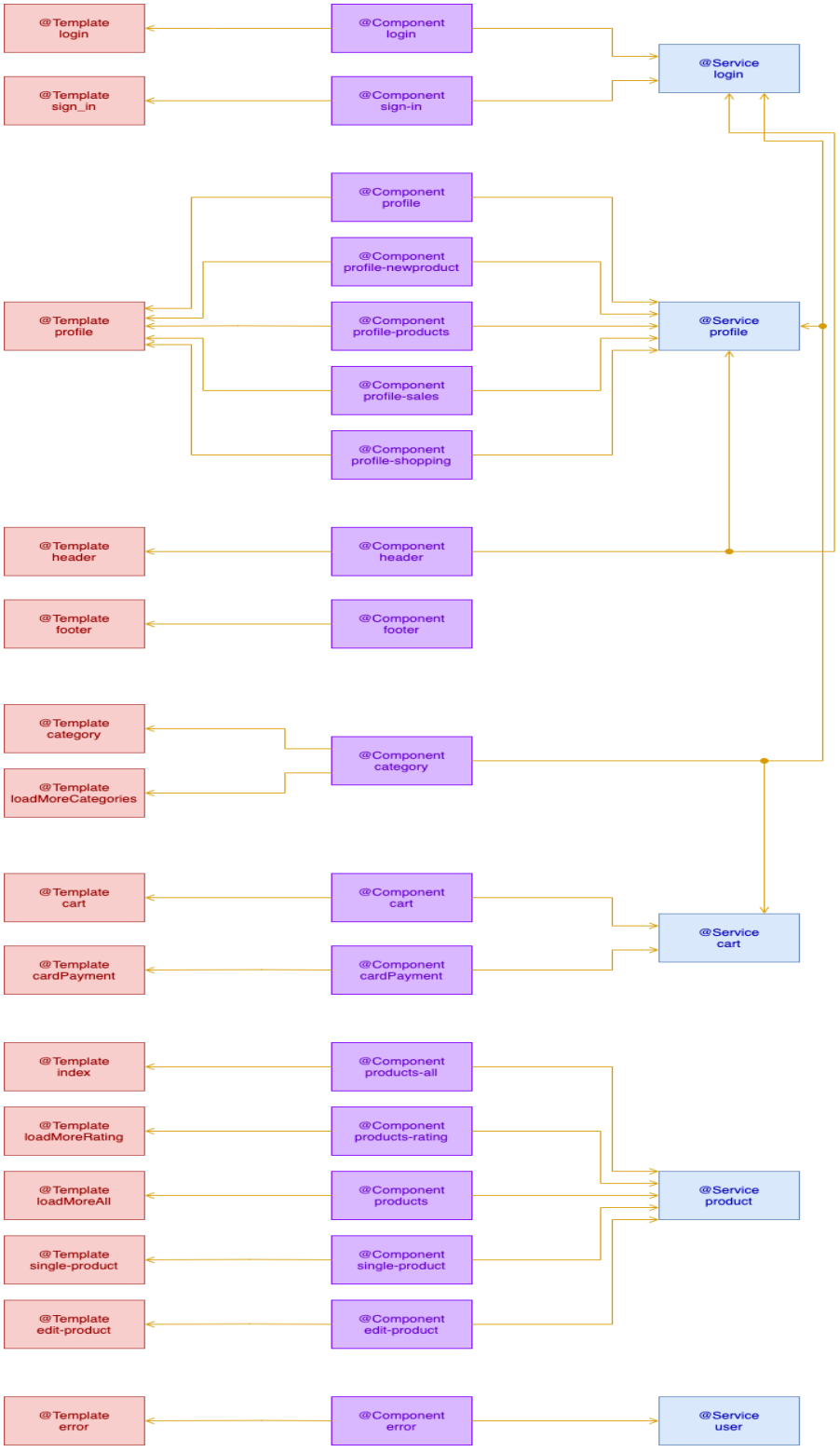
- Las pantallas se implementarán usando las librerías gráficas ng-bootstrap[10] o angular-material[11] dependiendo del estilo gráfico de la web de la fase 2.
- Todo el código JavaScript implementado en la fase 2 tendrá que volver a implementarse con TypeScript dentro del proyecto Angular. En concreto, se deberá implementar el mismo mecanismo de paginación de la fase 2 basado en AJAX, pero usando la API REST y mostrando las páginas con Angular.
- Se deberán seguir las buenas prácticas de diseño Angular, separando la lógica de gestión del interfaz en componentes y la lógica de conexión con el backend en servicios.
- Publicación:
  - La aplicación Angular se publicará como otro recurso estático más de la aplicación web SpringBoot. Es decir, el resultado de construir la aplicación Angular con `ng build` que está en la carpeta `dist` deberá copiarse a la carpeta `src/main/resources/public/new`. De esa forma, se podrá acceder a la aplicación cuando se acceda a la ruta https://localhost:8443/new/.
  - Como la construcción de la aplicación deberá estar completamente dockerizado, deberá ampliarse el script `create_image` de la fase 3 para que construya también la aplicación Angular. Para ello, puede usarse la imagen oficial de Node.js, en la que se podrá ejecutar el comando "npm install" antes del "ng build".
- **Código fuente:** Todo el código fuente del proyecto Angular se guardará en una carpeta "frontend" del repositorio. Es decir, el fichero angular.cli estará dentro de la carpeta frontend, no en la raíz del repositorio.

Documentación

El README se deberá ampliar incluyendo la información sobre el cliente SPA:

- **Preparación del entorno de desarrollo:** Se añadirá a las instrucciones cómo instalar y configurar el entorno de desarrollo para poder compilar y ejecutar la aplicación SPA con Angular.
- **Diagrama de clases y templates de la SPA:** Reflejará las clases y templates del código Angular. El diagrama deberá diferenciar claramente qué elementos son componentes y cómo se relacionan entre sí (relaciones padre-hijo). También se deberán incluir los servicios y el resto de clases auxiliares. A continuación se muestra un diagrama de este tipo:

10 https://ng-bootstrap.github.io/
11 https://material.angular.io/



- **Participación de miembros:** Se debe incluir una sección en el README en la que cada miembro del equipo indique su participación en esa fase con:

- Descripción textual de las tareas realizadas en la fase.
- Listado de los 5 commits más significativos durante la fase (enlazados a dichos commits en GitHub).
- Listado de los 5 ficheros en los que más haya participado (enlazados a dichos ficheros en GitHub).

- **Vídeo:** En esta fase además se deberá crear un vídeo, subir ese vídeo a YouTube y enlazar el vídeo en el inicio del README. Este vídeo tendrá una duración de entre 2 y 3 minutos y mostrará las principales funcionalidades de la aplicación web usada desde la interfaz Angular. Uno de los alumnos deberá grabarse la voz describiendo qué está haciendo según vaya interactuando con la aplicación web. El vídeo deberá mostrar, como mínimo, los aspectos más relevantes de la aplicación web:
  - Acceso de un usuario anónimo (sin hacer login).
  - Hacer login con un usuario existente y mostrar las funcionalidades adicionales.
  - Si es necesario se usarán varios usuarios (usando varios navegadores web) para que se observe cómo las acciones de un usuario afectan a otro (mensajes, puntos, comentarios, solicitudes de amistad, etc.)
  - Modificación de los datos de una entidad de la aplicación web (cita, autor, libro, entrega, etc...)
  - Subida de imágenes.
  - Login como administrador y mostrar las funcionalidades propias del administrador.