code
URJC

Desarrollo de Aplicaciones Distribuidas

Introducción

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Michel Maes
michel.maes@urjc.es

Óscar Soto
oscar.soto@urjc.es

Universidad
Rey Juan Carlos

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Arquitectura por Capas

- Ejercicio 1
  - Se desea desarrollar una aplicación para monitorizar el estado de los bloques de disco y los ficheros
  - La aplicación debe mostrar por pantalla
    - Espacio usado y espacio libre
    - Espacio dañado
    - Lista de ficheros con sus tamaños
    - Lista de ficheros que tienen parte de su contenido en bloques dañados
  - La aplicación debe ser multiplataforma.

Arquitectura por Capas

code URJC

- Ejercicio 1
  - En concreto se soportan las siguientes plataformas y se indica para cada una los servicios que ofrece:
    - Linux
      - getTotalSizeInBytes()
      - getBlocksInUse() // List of block IDs
      - getBlockSizeInBytes()
      - getDamagedBlocks() // List of block IDs
      - getFilesInFolder(String) // List of Files
      - getBlocksForFile(File) // List of block IDs
      - getFileSizeInBytes(File)
    - Solaris
      - getTotalSizeInBytes()
      - getUsedSizeInBytes()
      - getFilesInFolder()
      - getBlocks() // List of block IDs
      - getFileInBlock(Block ID)
      - getFileSizeInBytes(File)

Arquitectura por Capas

- Ejercicio 1
  - Definir el número de capas y sus responsabilidades

01-2023

code URJC

Desarrollo de Aplicaciones Web

Tema 1.2 – Aplicaciones Web con Spring

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

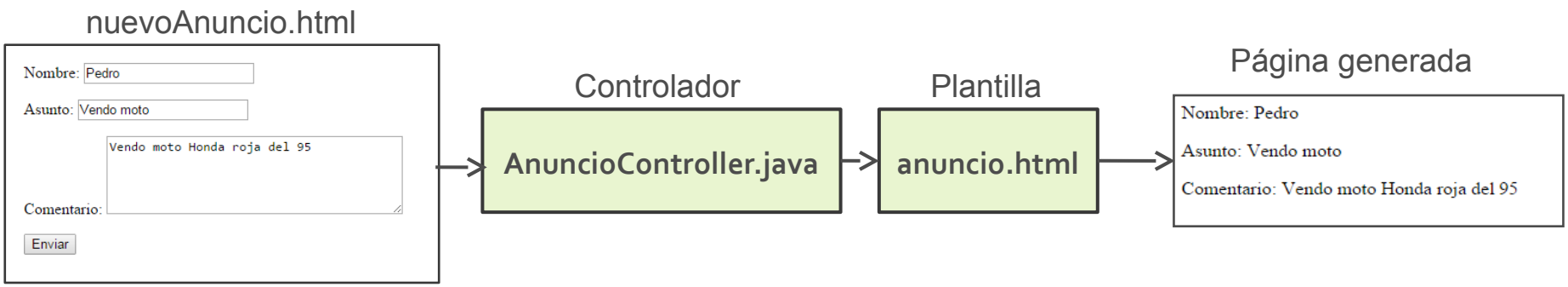
Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Ejercicio 1

code URJC

- 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

Ejercicio 1

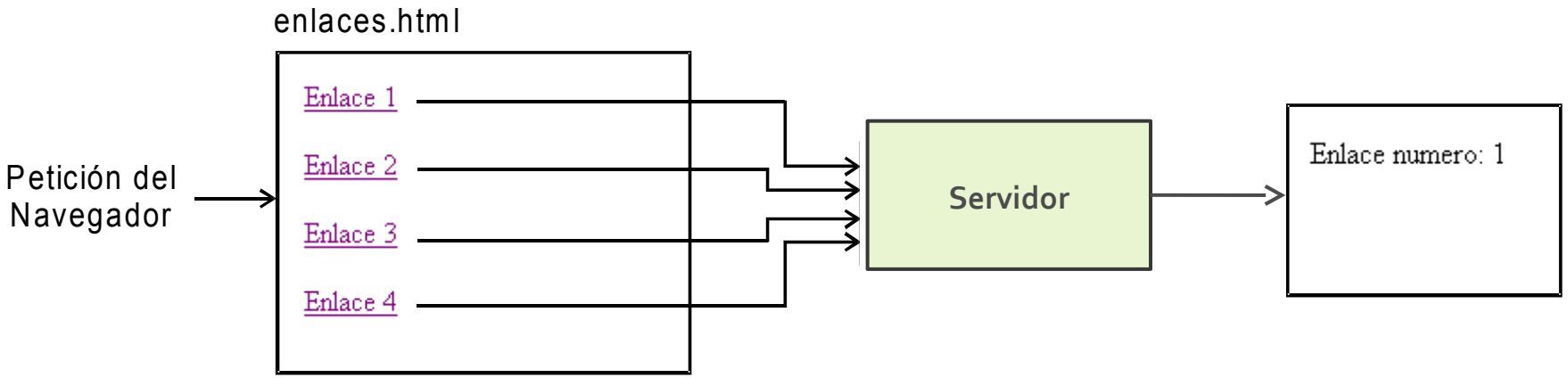
- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_1/web_ejer1

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

Ejercicio 2

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_1/web_ejer2

Ejercicio 3

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

Ejercicio 3

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_1/web_ejer3

Ejercicio 4 – Tablón de mensajes

code URJC

- 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 – Tablón de mensajes

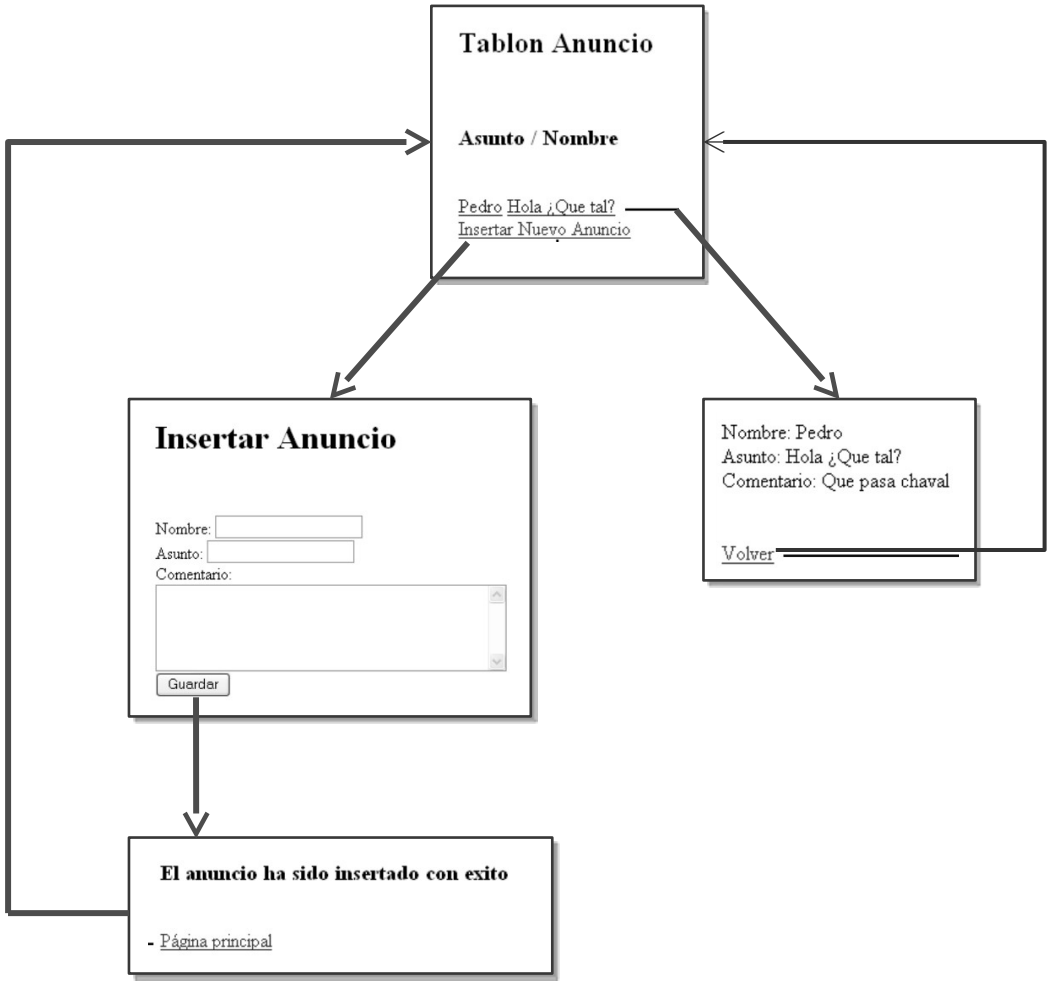
code URJC

- 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 – Tablón de mensajes

- **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 – Tablón de mensajes



Ejercicio 4

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_1/web_ejer4

Ejercicio 5

- 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)

Ejercicio 5

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_1/web_ejer5

Ejercicio 6 – Tablón mejorado

code URJC

- 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 – Tablón mejorado

- 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

Ejercicio 6 – Tablón mejorado

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_1/web_ejer6

Ejercicio 7 – Tablón con imágenes

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

Ejercicio 7 – Tablón con imágenes

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_1/web_ejer6

01-2023

code URJC

Desarrollo de Aplicaciones Web

Tema 2 – APIs REST con Spring

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

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

code URJC

- **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)

Ejercicio 1

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_2_rest/rest_ejer1

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

Anotamos la aplicación para que Spring genere automáticamente el cliente feign

```
@SpringBootApplication
@EnableFeignClients
public class Application {

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

APIs REST con Spring

code URJC

- Introducción
- Clientes de APIs REST
- APIs REST con Spring
- Conversión entre objetos y JSON
- Cliente REST en el servidor
- **Documentación de APIs REST**
- Imágenes

Documentación de APIs REST

code URJC

- 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…

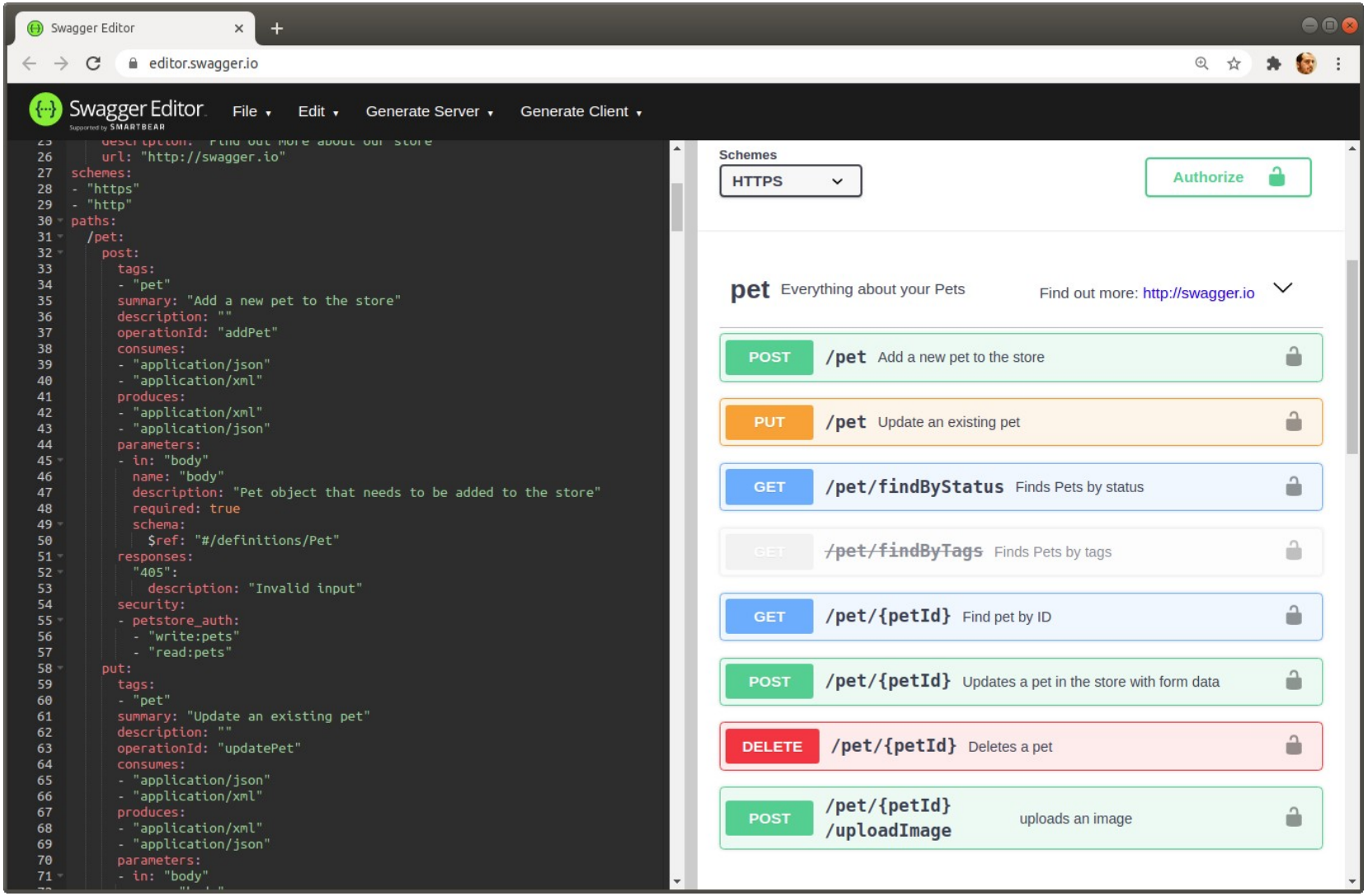
https://www.openapis.org/

Documentación de APIs REST

code URJC

- **Especificación OpenAPI**
  - Se puede especificar en YAML o en JSON
  - Describe cada operación (*endpoint*)
    - Formato de URL
    - Formato requerido en el cuerpo de la petición
    - Formato de los resultados
  - Se puede editar usando el **Swagger Editor**, un editor interactivo con "previsualización en tiempo real"

https://editor.swagger.io/



https://editor.swagger.io/

Documentación de APIs REST

- **Formato OpenAPI 3**

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Sample API
  description: A sample API to illustrate OpenAPI concepts
paths:
  /list:
    get:
      description: Returns a list of stuff
      responses:
        '200':
          description: Successful response
```

https://app.swaggerhub.com/help/tutorials/openapi-3-tutorial

Documentación de APIs REST

code URJC

- **Formato OpenAPI 3**
  - Los ficheros se dividen en 3 partes
    - **Meta Information**
    - **Path Items (endpoints)**
      - Parameters, Request bodies, Responses
    - **Reusable Components**
      - Schemas (data models), Parameters, Responses, Other components

Documentación de APIs REST

code URJC

- **Formato OpenAPI 3**
  - Meta Information

```yaml
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple Artist API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

# Basic authentication
components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

paths: {}
```

Documentación de APIs REST

code URJC

- **Formato OpenAPI 3**
  - Path Items

```
openapi: 3.0.0
info:
  version: 1.0.0
  title: Simple API
  description: A simple API to illustrate OpenAPI concepts

servers:
  - url: https://example.io/v1

components:
  securitySchemes:
    BasicAuth:
      type: http
      scheme: basic
security:
  - BasicAuth: []

#  ----- Added lines  ----------------------------------------
paths:
  /artists:
    get:
     description: Returns a list of artists
#  ---- /Added lines  ----------------------------------------
```

```
openapi: 3.0.0
info:
...

paths:
  /artists:
    get:
      description: Returns a list of artists
      #  ----- Added lines  ----------------------------------------
      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
                  properties:
                    artist_name:
                      type: string
                    artist_genre:
                      type: string
                    albums_recorded:
                      type: integer
                    username:
                      type: string

        '400':
          description: Invalid request
          content:
            application/json:
              schema:
                type: object
                properties:
                  message:
                    type: string
      #  ---- /Added lines  ----------------------------------------
```

```yaml
openapi: 3.0.0
info:
...

paths:
  /artists:
    get:
      description: Returns a list of artists
      #  ----- Added lines  ----------------------------------------
      parameters:
        - name: limit
          in: query
          description: Limits the number of items on a page
          schema:
            type: integer
        - name: offset
          in: query
          description: Specifies the page number of the artists to be displayed
          schema:
            type: integer
      #  ---- /Added lines  ----------------------------------------

      responses:
        '200':
          description: Successfully returned a list of artists
          content:
            application/json:
              schema:
                type: array
                items:
                  type: object
                  required:
                    - username
                  properties:
                    artist_name:
                      type: string
                    artist_genre:
                      type: string
                    albums_recorded:
                      type: integer
                    username:
                      type: string

    ...
```

Documentación de APIs REST

- **Formato OpenAPI 3**
  - Reusable Components
    - Schemas (data models)
    - Parameters
    - Request bodies
    - Responses
    - Response headers
    - Examples
    - Links
    - Callbacks

Documentación de APIs REST

code URJC

- **Enfoque *API First***
  - Es muy importante que una API sea entendible por sus clientes
  - Al igual que se diseñan interfaces gráficas de usuario con los propios usuarios antes de su implementación, también es conveniente diseñar las APIs con los futuros usuarios
  - OpenAPI se utiliza para diseñar APIs

https://swagger.io/resources/articles/adopting-an-api-first-approach/

Documentación de APIs REST

- **Benificios del enfoque *API First***
  - Es mucho más sencillo de obtener feedback de los clientes
  - Es más fácil que sea coherente y cumpla reglas de estilo (al evitar detalles de implementación)
  - Equipos pueden trabajar en paralelo (front y back) una vez definida la API
  - Se pueden usar herramientas de generación de código

https://swagger.io/resources/articles/adopting-an-api-first-approach/

Documentación de APIs REST

- **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/

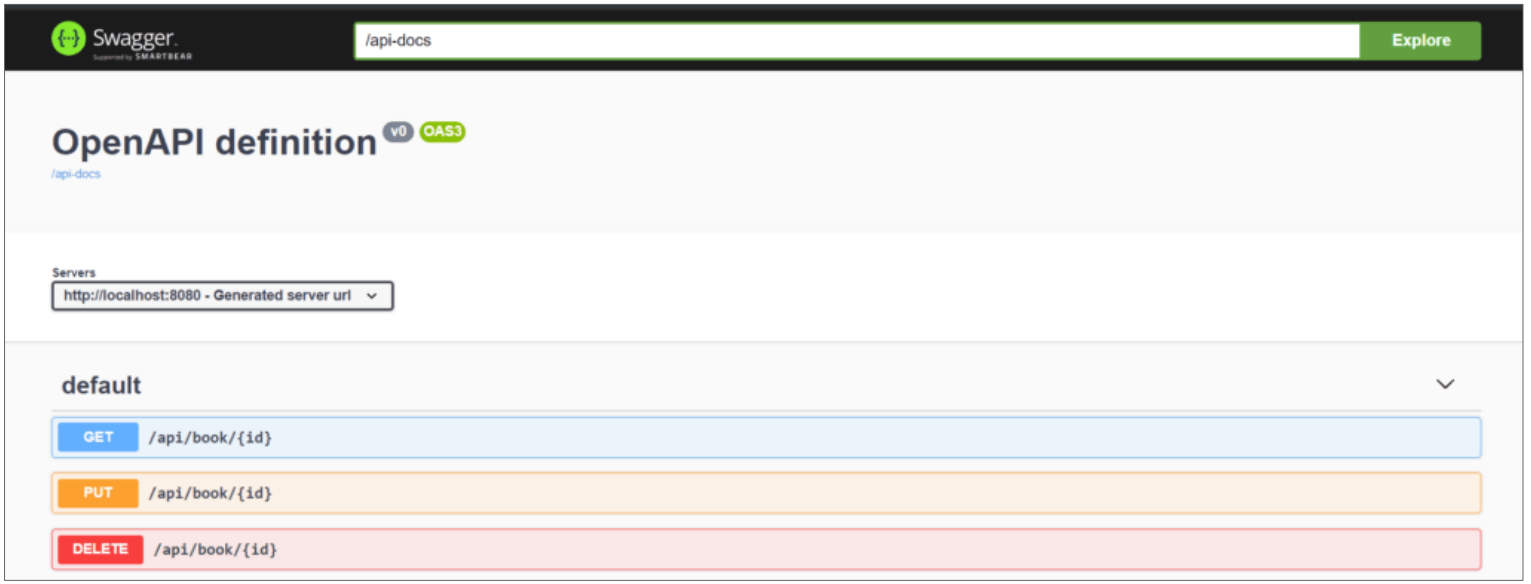
http://localhost:8080/v3/api-docs.yaml

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



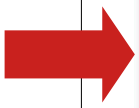
Documentación de APIs REST

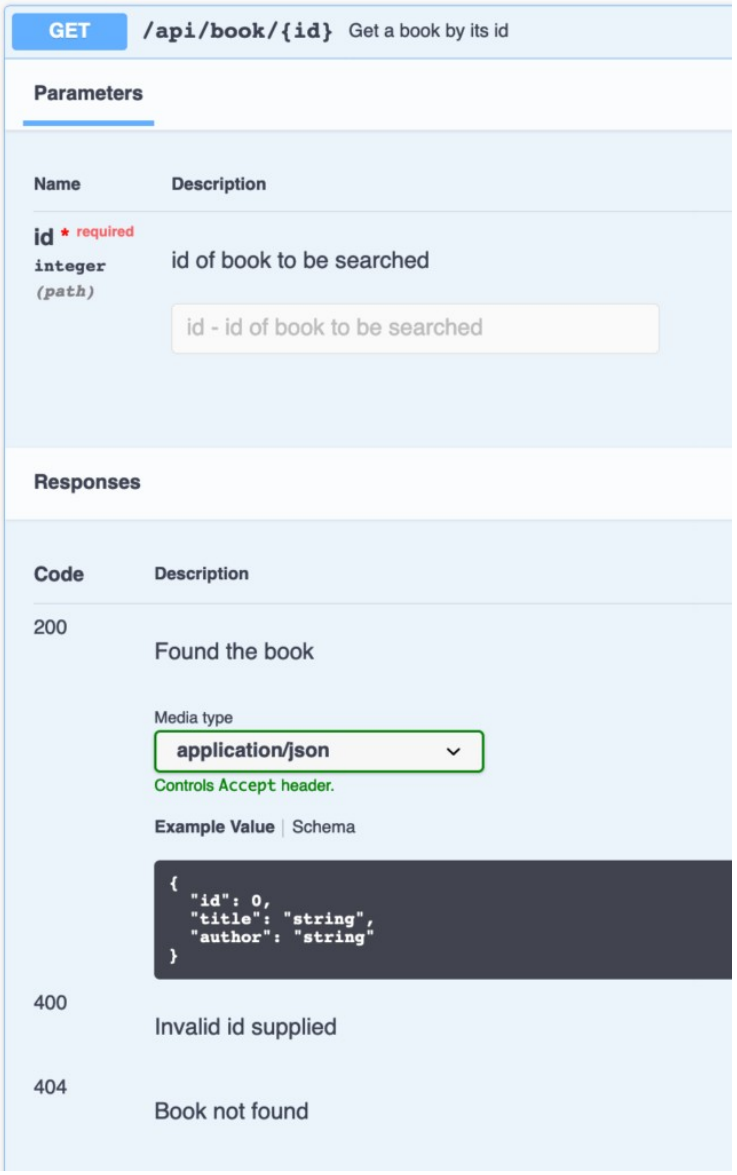
code URJC

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 ...;
}
```





APIs REST con Spring

code URJC

- Introducción
- Clientes de APIs REST
- APIs REST con Spring
- Conversión entre objetos y JSON
- Cliente REST en el servidor
- Documentación de APIs REST
- **Imágenes**

Imágenes

- 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

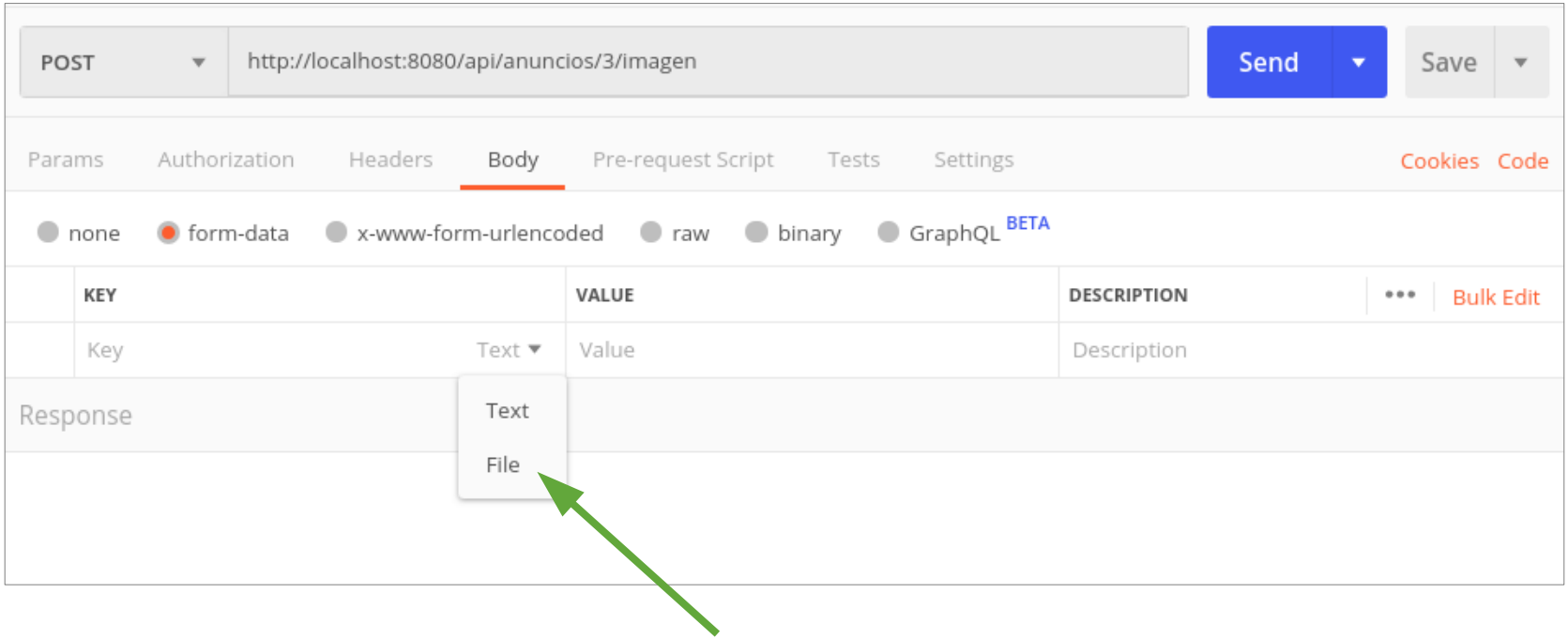
```
@PostMapping("/{id}/image")
public ResponseEntity<Object> uploadImage(@PathVariable long id,
  @RequestParam MultipartFile imageFile) throws IOException {

  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

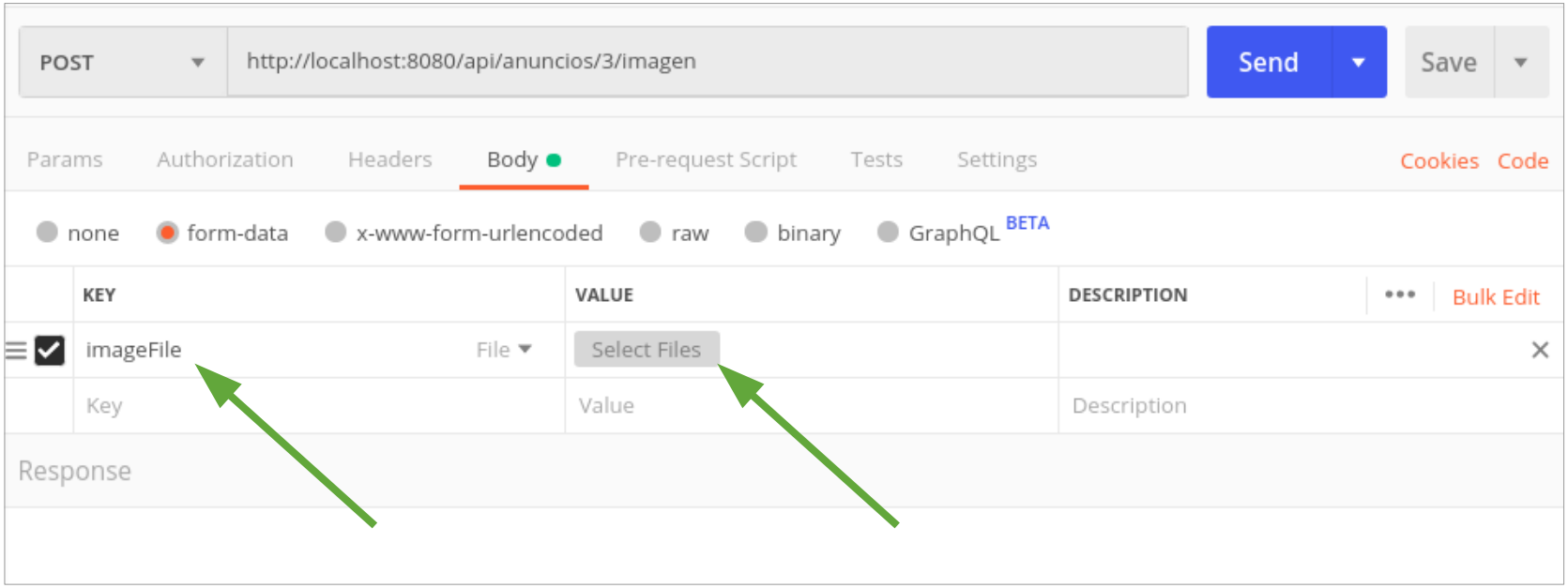


Imágenes

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();
  }
}
```

01-2023

code URJC

Desarrollo de Aplicaciones Web

Tema 3.2 – Bases de datos SQL en Spring

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

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

Ejercicio 1

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_3_db/bd_ejer1

Ejercicio 2

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

Ejercicio 2

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_2/tema_3_db/bd_ejer2

2.1 - Tecnologías de Servicios Web

Tema 7.2 – Tecnologías de comunicación con Mensajes

Universidad Rey Juan Carlos

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Óscar Soto
oscar.soto@urjc.es

Michel Maes
michel.maes@urjc.es

code URJC

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

WebSocket

- Ejemplos
  - Java
    - Servidor Spring WebSocket con cliente en Browser (comunicacion-ws-ejem1)

https://github.com/codeurjc/dad/tree/main/parte_3/tema_1_websockets/comunicacion-ws-ejem1

AMQP

- **Ejemplos**
  - Spring/comunicacion-rabbitmq-ejem1-producer
  - Spring/comunicacion-rabbitmq-ejem1-consumer

https://github.com/codeurjc/dad/tree/main/parte_3/tema_2_colas

Kafka

- **Ejemplos en Java**
  - Spring/comunicacion-kafka-ejem1-publisher
  - Spring/comunicacion-kafka-ejem1-subscriber

https://github.com/codeurjc/dad/tree/main/parte_3/tema_2_colas

Spring Cloud Stream

code URJC

- **Ejemplo 5**
  - Préstamos
  - Dos sercicios
    - Un servicio emite eventos de préstamos solicitados
      - loansource
    - Un segundo servicio determina si esos préstamos se autorizan o no
      - loancheck
  - Utiliza Spring Cloud Stream para desacoplar completamente lógica de negocio de la lógica de gestión de las colas
  - https://github.com/benwilcock/spring-cloud-stream-demo

CloudApps

MASTER

2.1 - Tecnologías de Servicios Web

Tema 7 – Tecnologías de comunicación

Universidad Rey Juan Carlos

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Óscar Soto
oscar.soto@urjc.es

Michel Maes
michel.maes@urjc.es

code URJC

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

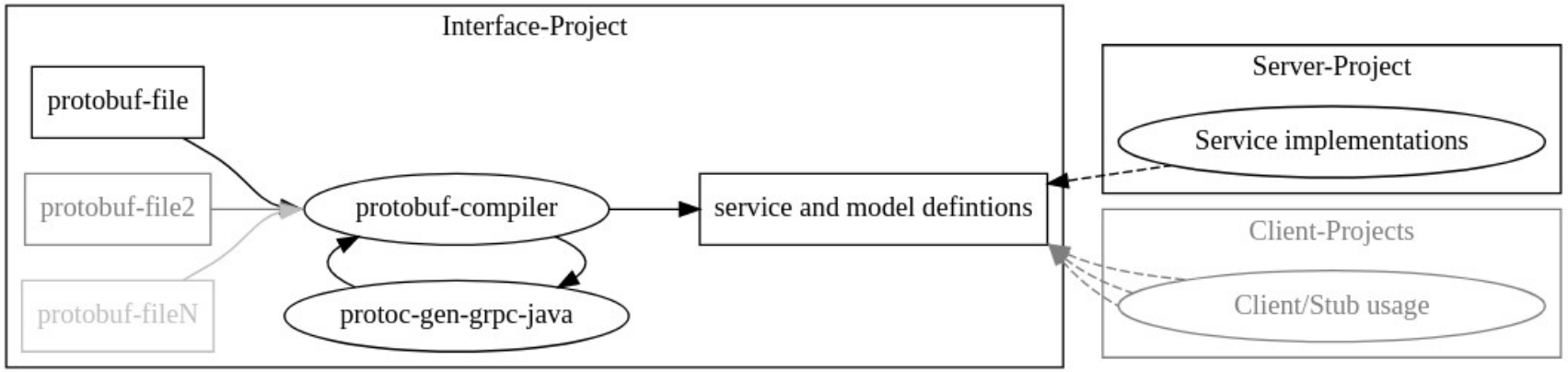
gRPC

- **Ejemplos Java**
  - Java plano (sin Spring)
    - grpc-ejem1
  - SpringBoot
    - grpc-ejem2-client
    - grpc-ejem2-server
    - grpc-ejem2-interface (código generado desde .proto)

https://github.com/codeurjc/dad/tree/main/parte_3/tema_3_protocolos

gRPC

- **Ejemplos Java**
  - SpringBoot



01-2023

code URJC

Desarrollo de Aplicaciones Web

Tema 2 – Docker

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

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

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)

https://github.com/codeurjc/dad/tree/main/parte_4/tema_2/aplicacion-java-enunciado

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

https://github.com/codeurjc/dad/tree/main/parte_4/tema_2/aplicacion-java-enunciado

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

Ejercicio 4

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_4/tema_2/ejer4

Ejercicio 6

- Crea un Multistage Docker file optimizando las capas para no descargar las librerías en cada construcción

https://github.com/codeurjc/dad/tree/main/parte_4/tema_2/aplicacion-java-enunciado

Ejercicio 6

ejemplo-2

- Solución

cache-multistage.Dockerfile

```
FROM maven:3.6.3-openjdk-8 as builder
WORKDIR /project
COPY pom.xml /project/
RUN mvn clean verify
COPY /src /project/src
RUN mvn package -o

FROM openjdk:8-jre-slim
WORKDIR /usr/src/app/
COPY --from=builder /project/target/*.jar
/usr/src/app/
EXPOSE 8080
CMD [ "java", "-jar", "java-webapp-0.0.1.jar" ]
```

Ejercicio 7

- Crea la imagen de la aplicación Java del Ejercicio 6 con jib

https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores/tree/master/docker/aplicacion-java-enunciado

Ejercicio 8

- Crea la imagen de la aplicación Java del Ejercicio 6 con Buildpacks

https://github.com/MasterCloudApps/3.2.Contenedores-y-orquestadores/tree/master/docker/aplicacion-java-enunciado

01-2023

code URJC

Desarrollo de Aplicaciones Web

Tema 3 – Docker Compose

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

01-2023

code
URJC

Ejercicio 1

- **Dockerizar una aplicación SpringBoot**
  - Se usará docker-compose
  - La aplicación necesita una BBDD MySQL
    - Password de root: pass
  - Utiliza una de las estrategias de espera vistas
  - En una aplicación SpringBoot se puede configurar la ruta de la BBDD y el esquema con la variable de entorno:
    - SPRING_DATASOURCE_URL=jdbc:mysql://<host>/<database>
  - Desactivar los tests al construir la app Maven: -DskipTests=true

https://github.com/codeurjc/dad/tree/main/parte_4/tema_3/ejercicio-1-enunciado

Ejercicio 1

- Solución:
  - https://github.com/codeurjc/dad/tree/main/parte_4/tema_3/ejercicio-1

code URJC

Desarrollo de Aplicaciones Distribuidas

Tema 3

Escalabilidad y tolerancia a fallos

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Michel Maes
michel.maes@urjc.es

Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Separación de servidor y bbdd

- Ejemplo: ejemplobbdd2-mysql
  - El servidor estará en una máquina virtual
  - La base de datos en otra
  - Código:
    - https://github.com/codeurjc/dad/tree/main/parte_5/ejem1-mysql

Separación de servidor y bbdd

- En la máquina database:
  - Por defecto mysql **sólo** escucha en **127.0.0.1**
  - Pero necesitamos que sea accesible desde la máquina web
  - Hay que hacer que escuche en la **IP privada** de la máquina database
  - Configuración del servicio mysql para que escuche en una IP concreta:
    - Comprobar la **ip privada** de la máquina con ifconfig (o en Azure)
    - Editar el fichero */etc/mysql/my.cnf*, cambiar el valor de *bind_address* por la ip de la vm (guardar con Ctrl+O, salir con Ctrl+X)

      ```
      sudo pico /etc/mysql/my.cnf
      ```

    - Reiniciar el servicio con

      ```
      sudo service mysql restart
      ```

Separación de servidor y bbdd

- En la máquina database:
  - Configuración de la base de datos
    - Acceder como root con

```
mysql -u root -p
mysql> create user 'anuncio'@'<ip vm
 java>' identified by 'anuncios';
mysql> create database anuncios;
mysql> grant all privileges on
 anuncios.* to 'anuncio'@'<ip_vm_java>';
mysql> flush privileges;
mysql> exit
```

Separación de servidor y bbdd

- En la máquina web:
  - Desplegar la aplicación
    - Cambiar valor de propiedad *spring.jpa.hibernate.ddl-auto* a *none*
    - mvn package
    - Subir el jar a la vm

```
scp -i /home/<user>/.ssh/azure.key -P <puerto
  ssh vm> ejemplobbdd2-mysql-0.0.1-SNAPSHOT.jar
  azureuser@<ip vm>:/home/azureuser/
  ejemplobbdd2-mysql-0.0.1-SNAPSHOT.jar
```

Separación de servidor y bbdd

- En la máquina web:
  - Acceder a la máquina web por ssh
  - Arrancar la aplicación con los siguientes parámetros:

```
java -jar ejemplobbdd2-mysql-0.0.1-SNAPSHOT.jar --
 server.address="<ip vm>" --
 spring.datasource.url="jdbc:mysql://<ip interna
 mysql>:3306/anuncios" --
 spring.datasource.username="anuncio" --
 spring.datasource.password="anuncios" --
 spring.jpa.hibernate.ddl-auto="create"
```

  - Realizar una peticion a http://<cloud service name>/anuncios
  - Esto creara el esquema de datos en la base de datos
  - Finalizar la aplicacion (Ctrl+C)

Separación de servidor y bbdd

- En la máquina web:
  - Ahora ya podemos arrancar el servidor con:

```
java -jar ejemplobbdd2-mysql-0.0.1-SNAPSHOT.jar --
  server.address="<ip vm>" --
  spring.datasource.url="jdbc:mysql://<ip interna
  mysql>:3306/anuncios" --
  spring.datasource.username="anuncio" --
  spring.datasource.password="anuncios"
```

code URJC

Desarrollo de Aplicaciones Distribuidas

Tema 3

Escalabilidad y tolerancia a fallos

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Michel Maes
michel.maes@urjc.es

Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Servidor con caché

- Ejemplo: Ejemplobbdd2-cache
  - https://github.com/codeurjc/dad/tree/main/parte_5/ejem2-cache

  - Tenemos los siguientes recursos:
    - GET /anuncios
    - POST /anuncios
    - GET /anuncios/{asunto}
  - ¿Cuál cachear?

Servidor con caché

- Ejemplo: Ejemplobbdd2-cache
  - Tenemos los siguientes recursos:
    - GET /anuncios
    - POST /anuncios
    - GET /anuncios/{asunto}
  - ¿Cuál cachear?
    - Candidatos:
      - GET /anuncios
      - GET /anuncios/{asunto}

Servidor con caché

- Ejemplo: Ejemplobbdd2-cache
  - Tenemos los siguientes recursos:
    - GET /anuncios
    - POST /anuncios
    - GET /anuncios/{asunto}
  - ¿Dónde invalidar la caché?

Servidor con caché

- Ejemplo: Ejemplobbdd2-cache
  - Tenemos los siguientes recursos:
    - GET /anuncios
    - POST /anuncios
    - GET /anuncios/{asunto}
  - ¿Dónde invalidar la caché?
    - POST /anuncios
      - Insertar un nuevo anuncio invalida siempre GET /anuncios
      - Insertar un nuevo anuncio podría invalidar GET /anuncios/{asunto} si el asunto es el mismo

Servidor con caché

- Ejercicio
  - Construir la aplicación Ejemplobbdd2-cache
  - Desplegarla en Azure
    - El modelo de datos es el mismo que Ejemplobbdd2-mysql, no hay que cambiar nada en la base de datos
  - Utilizar jmeter con el fichero ejemplobbdd2-cache.jmx para comprobar la reducción del tiempo en las peticiones una vez cacheadas
    - Abrir el fichero ejemplobbdd2-cache.jmx
    - Ajustar la ip y el puerto
    - Comprobar los tiempos

code URJC

Desarrollo de Aplicaciones Distribuidas

Tema 3

Escalabilidad y tolerancia a fallos

Francisco Gortázar
francisco.gortazar@urjc.es
@fgortazar

Micael Gallego
micael.gallego@urjc.es
@micael_gallego

Michel Maes
michel.maes@urjc.es

Óscar Soto
oscar.soto@urjc.es

code
URJC

Micael Gallego, Francisco Gortázar, Michel Maes, Óscar Soto

Balanceadores de carga

- Ejemplo: Ejemplobbdd2-cache
  - Código: https://github.com/codeurjc/dad/tree/main/parte_5/ejem2-cache

  - Crear una vm para el balanceador
    - Asociar a esta vm el puerto 80 del servicio: 80 público -> 80 privado
  - Instalar haproxy
    - sudo apt-get update
    - sudo apt-get -y install haproxy
  - Editar el fichero **/etc/default/haproxy** y arrancar el servicio
    - Cambiar ENABLED=0 a ENABLED=1
    - sudo service haproxy start

Balanceadores de carga

- Ejemplo: Ejemplobbdd2-cache
  - Configurar haproxy editando/creando la siguiente sección al final del fichero **/etc/haproxy/haproxy.cfg**:

```
listen anuncios 0.0.0.0:80
 mode http
 stats enable
 stats uri /haproxy?stats
 balance roundrobin
 option httpclose
 option forwardfor
 server web1 <ip server web 1>:8080 check
 server web2 <ip server web 2>:8080 check
```

Balanceadores de carga

- Ejemplo: Ejemplobbdd2-cache
  - Arrancaremos varios servidores web en vms diferentes
    - Crear dos vms
    - Subir el jar a ambas
    - Arrancar el servidor en ambas como se ha visto en ejemplos anteriores

Balanceadores de carga

- Ejemplo: Ejemplobbdd2-cache
  - La url <dominio>/haproxy?stats debería devolver algo como:

HAProxy version 1.4.24, released 2013/06/17

Statistics Report for pid 2354

> General process information

pid = 2354 (process #1, nbproc = 1)
uptime = 2d 22h22m54s
system limits: memmax = unlimited; ulimit-n = 4013
maxsock = 4013; **maxconn =** 2000; **maxpipes =** 0
current conns = 1; current pipes = 0/0
Running tasks: 1/3

active UP | backup UP
active UP, going down | backup UP, going down
active DOWN, going up | backup DOWN, going up
active or backup DOWN | not checked
active or backup DOWN for maintenance (MAINT)
Note: UP with load-balancing disabled is reported as "NOLB".

Display option:
- Hide 'DOWN' servers
- Refresh now
- CSV export

External resources:
- Primary site
- Updates (v1.4)
- Online manual

anuncios

	Queue			Session rate			Sessions					Bytes		Denied		Errors			Warnings		Server								
	Cur	Max	Limit	Cur	Max	Limit	Cur	Max	Limit	Total	LbTot	In	Out	Req	Resp	Req	Conn	Resp	Retr	Redis	Status	LastChk	Wght	Act	Bck	Chk	Dwn	Dwntme	Thrtle
Frontend				1	2	-	1	1	2 000	38		8 479	50 831	0	0	1					OPEN								
web1	0	0	-	0	1		0	1	-	4	4	1 182	1 083		0		0	0	0	0	2d6h DOWN	L4CON in 0ms	1	Y	-	4	3	2d21h	-
web2	0	0	-	0	0		0	0	-	0	0	0	0		0		0	0	0	0	2d6h DOWN	L4CON in 0ms	1	Y	-	2	2	2d21h	-
Backend	0	0		0	2		0	1	2 000	31	4	8 479	50 831	0	0		27	0	0	0	2d6h DOWN		0	0	0		3	2d21h	