



API REST AMB BASE DE DADES MYSQL

de

Islam El Mrabet Larhzaoui

Insitiucio Eductaiva: IES Serpis
Curs: 2 DAM
Data: 12/02/24

Índice

1. Instrucciones del proyecto.....	4
2. API Zapatos.....	5
2.1. Descripción API.....	5
2.2. Tecnología usada para la api.....	6
2.2. Dependencias de la api.....	7
2.2. Estructura de la api.....	8
2.2. Proyecto base de la api.....	9
2.2. Estructura del proyecto base de la api.....	10
2.2. Clases del proyecto base de la api.....	11
2.2. Pruebas de la api.....	16

ENLACE API

<http://localhost:9999/swagger-ui/index.html#/>

<http://localhost:9999/api/v1/shoes>

Instrucciones del proyecto

Desenvolupa una API per gestionar les operacions CRUD sobre un model que tu inventis.

Com sempre, haurà de ser completament funcional (compilable i executable) i seguir les instruccions donades a la documentació del mòdul (models, estructura, etc).

- *Caldrà implementar: inserció, modificació, llistat complet, llistat d'un registre, esborrat d'un registre i esborrat de tots els registres (6 punts)*
- *Hauràs de considerar restriccions a alguns atributs com per exemple: valors que no poden ser nuls, negatius, positius, etc. (1.5 punts)*
- *Inclusió d'enumeracions per donar valor a algun atribut i la validació d'aquest valor. Per exemple pel camp tipus_animal, el valor només pot ser: vertebrat/invertebrat (1 punt)*
- *Documentació amb OPEN API (1.5 punts)*

API DE ZAPATOS

Descripción API

Esta API está diseñada para obtener información sobre zapatos, lo cual permitirá a los usuarios que den uso de ella buscar información detallada sobre zapatos, como su talla, color, material, precio, etc.

Funcionalidades de la API:

1- Buscar zapatos por criterios específicos:

Los usuarios podrán buscar zapatos según el filtro establecido, es decir, se podría filtrar tanto por marca, talla, color, referencia, etc.

2- Agregar nuevos zapatos

Los usuarios serán capaces de añadir todos los zapatos que ellos deseen con talla, marca, referencia, color, material.

3- Modificar los zapatos

Los usuarios son capaces de modificar todos los zapatos usando todos sus atributos menos el id, ya que este es único para cada zapato.

4- Eliminar zapatos

Los usuarios tienen la posibilidad de eliminar los zapatos tanto de manera filtrada usando el **ID** del zapato o simplemente borrar todos los zapatos.

Tecnología usada para la API

La API de Gestión de Zapatos ha sido desarrollada utilizando un conjunto de tecnologías modernas y robustas para garantizar su funcionalidad, rendimiento y seguridad. A continuación, se detallan las principales tecnologías utilizadas en la implementación de la API:

1. Lenguaje de Programación:

- Se ha utilizado java y MySQL para escribir el código base de la API. Este lenguaje ofrece una combinación de eficiencia, flexibilidad y capacidad de escalamiento que es fundamental para satisfacer las demandas de nuestros usuarios.

2. Framework de Desarrollo:

- Se ha usado Spring Boot como el framework principal para el desarrollo de la API. Este framework proporciona una estructura sólida y herramientas integradas que simplifican el desarrollo, la prueba y el mantenimiento de la API.

3. Base de Datos:

- La persistencia de datos se gestiona a través de PHPmyAdmin, que ofrece un almacenamiento seguro. Se ha diseñado un esquema de base de datos optimizado para garantizar un acceso rápido y fiable a los datos.

2. Entorno de Desarrollo:

- Para el desarrollo de esta API se ha usado únicamente Apache NetBeans 20 junto a la versión jdk 21. Esta versión ofrece las últimas novedades de Apache NetBeans la cual nos permite sacarle la máxima funcionalidad a la API.

Dependencias de la API

```
<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <scope>runtime</scope>
    </dependency>
    <dependency>
        <groupId>org.projectlombok</groupId>
        <artifactId>lombok</artifactId>
        <optional>true</optional>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.springdoc</groupId>
        <artifactId>springdoc-openapi-starter-webmvc-ui</artifactId>
        <version>2.3.0</version>
    </dependency>
    <dependency>
        <groupId>jakarta.validation</groupId>
        <artifactId>jakarta.validation-api</artifactId>
    </dependency>
    <dependency>
        <groupId>org.hibernate</groupId>
        <artifactId>hibernate-validator</artifactId>
        <version>8.0.1.Final</version>
    </dependency>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-actuator</artifactId>
    </dependency>
</dependencies>
```

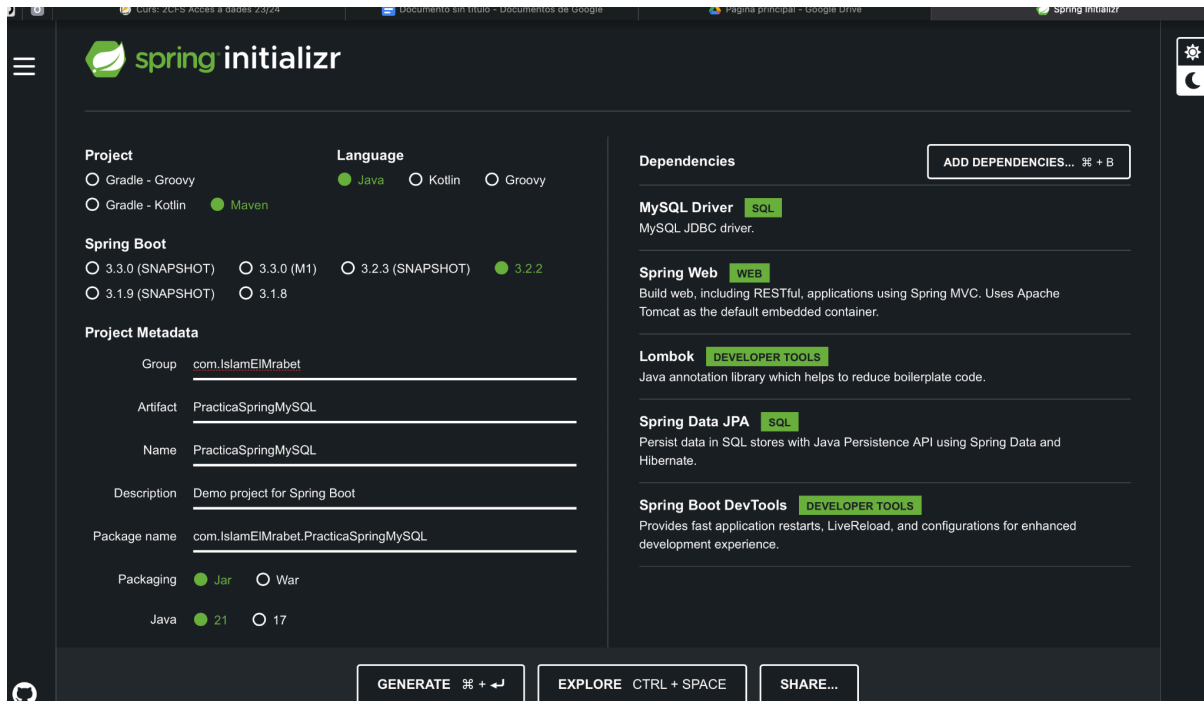
Estructura de la API

shoe-controller			^
POST	/api/v1/shoes/new	Registra un nuevo zapato	∨
POST	/api/v1/shoes/edit/{id}	Modifica un zapato	∨
GET	/api/v1/shoes	Obtiene el listado de zapatos	∨
GET	/api/v1/shoes/{id}	Obtiene un zapato determinado	∨
DELETE	/api/v1/shoes/{id}	Elimina un zapato	∨
DELETE	/api/v1/shoes/delete	Elimina todos los zapatos	∨

En la foto se puede observar la **estructura de la API** de Zapatos. Esta API consta de **6 métodos** de los cuales podemos ver la **creación** de un zapato, la **modificación** de un zapato, el **listado** de los zapatos, el listado de un solo zapato, el **borrado** de todos o de solo un zapato.

Proyecto base de la API

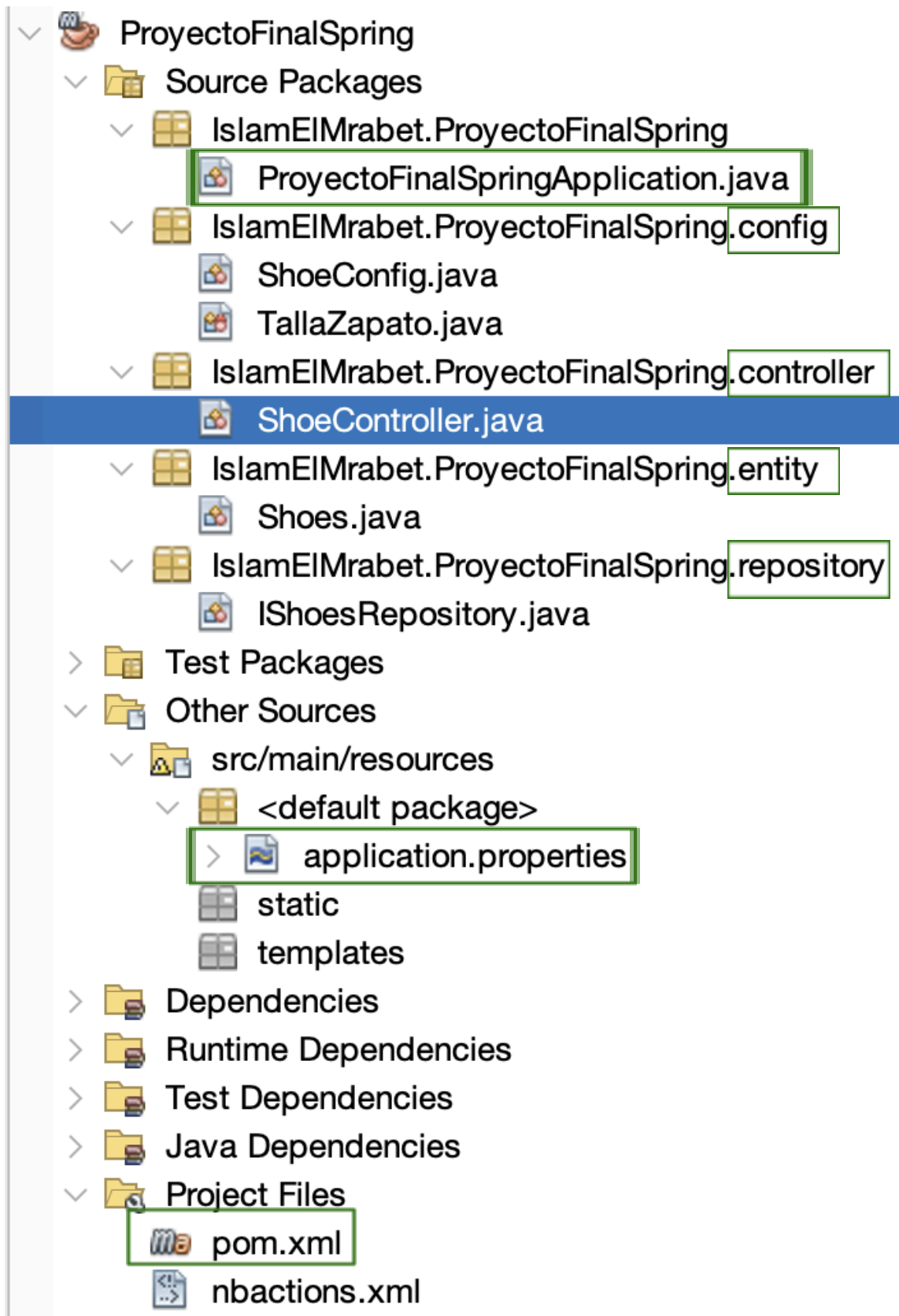
En primer lugar se generará un programa maven con todas las dependencias que cubrirán nuestras necesidades para generar la **API**.



Estas son las dependencias que ha requerido este proyecto:

- MySQL Driver
- Spring Web
- Lombok
- Spring Data JPA
- Spring Boot DevTools

Estructura del proyecto base de la API



Clases del proyecto base de la API

Las clases del proyecto base de la API incluyen:

1. Controladores:

Manejan las solicitudes HTTP y coordinan las respuestas.

```
/**
 * Controlador para zapatos
 * @author Islam El Mrabet Larhzaoui
 * @version Curso 2024-2025
 */
@RestController
@RequestMapping("/api/v1")
public class ShoeController {

    @Autowired
    private IShoesRepository iShoeRepository;

    @Operation(summary = "Obtiene el listado de zapatos")
    @ApiResponses(value = {
        @ApiResponse(
            responseCode = "200",
            description = "Listado de zapatos",
            content = @Content(array = @ArraySchema(
                schema = @Schema(implementation = Shoes.class)))
        ),})
    @GetMapping("/shoes")
    public List<Shoes> getAllShoes() {
        return iShoeRepository.findAll();
    }

    @Operation(summary = "Registra un nuevo zapato")
    @ApiResponses(value = {
        @ApiResponse(
            responseCode = "201",
            description = "Se registra el zapato",
            content = @Content(schema = @Schema(implementation = Shoes.class)))
    })
    @PostMapping("/shoes/new")
    public Shoes addShoe (@RequestBody Shoes newShoe) {
        return iShoeRepository.save(entity: newShoe);
    }

    @Operation(summary = "Modifica un zapato")
    @ApiResponses(value = {
        @ApiResponse(
            responseCode = "200",
            description = "Se modifica el zapato",
            content = @Content(schema = @Schema(implementation = Shoes.class)))
    })
    @PostMapping("/shoes/edit/{id}")
    public Shoes editShoe(@PathVariable Integer id, @RequestBody Shoes shoe) {
```

2. Modelos:

Representan las entidades de datos que la API maneja.

```

/*
 * @author islam
 */

@Data
@Entity
public class Shoes {
    @Id
    @GeneratedValue (strategy = AUTO)
    private Integer id;

    @Schema(description = "Referencia del zapato", example = "Nike Air Force 1")
    @NotNull
    @Size(min = 5, message = "Referencia minima de 5 caracteres ")
    private String reference;

    @Schema(description = "Talla del zapato", example = "TALLA_40")
    @NotNull
    private TallaZapato talla;

    @Schema(description = "Material del zapato", example = "Nike Air Force 1")
    @Size(min = 5, max = 10, message = "Material minimo de 5 caracteres ")
    private String material;

    @Schema(description = "Color del zapato", example = "Blanco")
    @Size(min = 3, message = "Color minima de 3 caracteres ")
    private String color;

    @Schema(description = "Marca del zapato", example = "Nike")
    @NotNull
    private String marca;

    @Schema(description = "Precio unitario", example = "120.99", defaultValue="0")
    @Min(value = 0)
    private float precio;
}

```

En esta clase se define la entidad la cual es usada por la api, en la que se especifica si será nula o no, la documentación para una mejor interpretación y el tamaño de los atributos.

```

/*
 * Click nbfs://nbhost/System
 * Click nbfs://nbhost/System
 */
package IslamElMrabet.Proyec

/**
 *
 * @author islamt|
 */
public enum TallaZapato {
    TALLA_35,
    TALLA_36,
    TALLA_37,
    TALLA_38,
    TALLA_39,
    TALLA_40,
    TALLA_41,
    TALLA_42,
    TALLA_43,
    TALLA_44,
    TALLA_45,
    TALLA_46;
}

```

3. Repositorio:

Representan el repositorio de datos de JPA que la API maneja.

```

/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package IslamElMrabet.ProyectoFinalSpring.repository;

import IslamElMrabet.ProyectoFinalSpring.entity.Shoes;
import org.springframework.data.jpa.repository.JpaRepository;

/**
 *
 * @author islam
 */
public interface IShoesRepository extends JpaRepository <Shoes, Integer> {

}

```

4. Configuración:

Representan la configuración de la API, que es tanto la documentación personal de la api como las herramientas que se usan.

```

package IslamElMrabet.ProyectoFinalSpring.config;

import io.swagger.v3.oas.models.Components;
import io.swagger.v3.oas.models.OpenAPI;
import io.swagger.v3.oas.models.info.Contact;
import io.swagger.v3.oas.models.info.Info;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

/**
 *
 * @author islam
 */
@Configuration
public class ShoeConfig {

    @Bean
    public OpenAPI customOpenAPI() {
        return new OpenAPI()
            .components(new Components())
            .info(new Info().title(title: "Zapato API")
                .description(description: "Informacion de Zapatos")
                .contact(new Contact()
                    .name(name: "Islam El Mrabet Larhzaoui")
                    .email(email: "islemmlar@alu.edu.gva")
                    .url(url: "http://islamsportfolio.000webhostapp.com/")))
            .version(version: "1.0");
    }
}

```

Este archivo es muy importante ya que es donde definiremos las propiedades de la API. Aquí se definirán parámetros como la url de conexión a la base de datos, el puerto y el usuario de la base de datos.

```
spring.datasource.url = jdbc:mysql://localhost:3306/shoes_db

spring.datasource.username = root
spring.datasource.password =

spring.jpa.hibernate.ddl-auto = update

server.port = 9999

management.endpoints.web.exposure.include=mappings
```

En mi caso se ha modificado el puerto de acceso para ofrecer más seguridad y no hacer un bloqueo de puertos.

6. Manejo de Errores

En esta API se ha creado un manejo de errores personalizado para hacerlo más bonito visualmente y ofrecer una mejor experiencia a usuarios tanto avanzados como nuevos.

```
/**
 *
 * @author islam
 */
public class validation {

    public static boolean validateEmptyParameters(Shoes newShoe){
        if("".equals(newShoe.getReference()) || newShoe.getReference() == null){
            throw new RuntimeException(code: "P-555", message: "La referencia es un campo obligatorio");
        }
        if("".equals(newShoe.getMarca()) || newShoe.getMarca() == null){
            throw new RuntimeException(code: "P-555", message: "La marca es un campo obligatorio");
        }
        if (newShoe.getTalla() == null || !isValidTalla(talla: newShoe.getTalla())) {
            throw new RuntimeException(code: "P-555", message: "La talla es un campo obligatorio y debe cumplir este formato");
        }
        return true;
    }

    private static boolean isValidTalla(String talla) {
        try {
            TallaZapato.valueOf(string: talla.toUpperCase());
            return true;
        } catch (IllegalArgumentException e) {
            return false;
        }
    }
}
```

```

/**
 *
 * @author islam
 */

@RestControllerAdvice
public class ControllerAdvice {

    @ExceptionHandler(value = RuntimeException.class)
    public ResponseEntity<ErrorDTO> runtimeExceptionHandler(RuntimeException ex){
        ErrorDTO error = ErrorDTO.builder().code(code: "P-500").message(message:ex.getMessage()).build();
        return new ResponseEntity<>(body: error, status: HttpStatus.BAD_REQUEST);
    }

    @ExceptionHandler(value = RequestException.class)
    public ResponseEntity<ErrorDTO> requestExceptionHandler(RequestException ex){
        ErrorDTO error = ErrorDTO.builder().code(code: ex.getCode()).message(message:ex.getMessage()).build();
        return new ResponseEntity<>(body: error, status: HttpStatus.BAD_REQUEST);
    }

}

```

Esta clase es el controlador de errores, el cual se puede personalizar y agregar tantos errores como se necesite haciéndolo más atractivo e intuitivos visualmente.

```

/**
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package IslamElMrabet.ProyectoFinalSpring.config;

import lombok.Builder;
import lombok.Data;

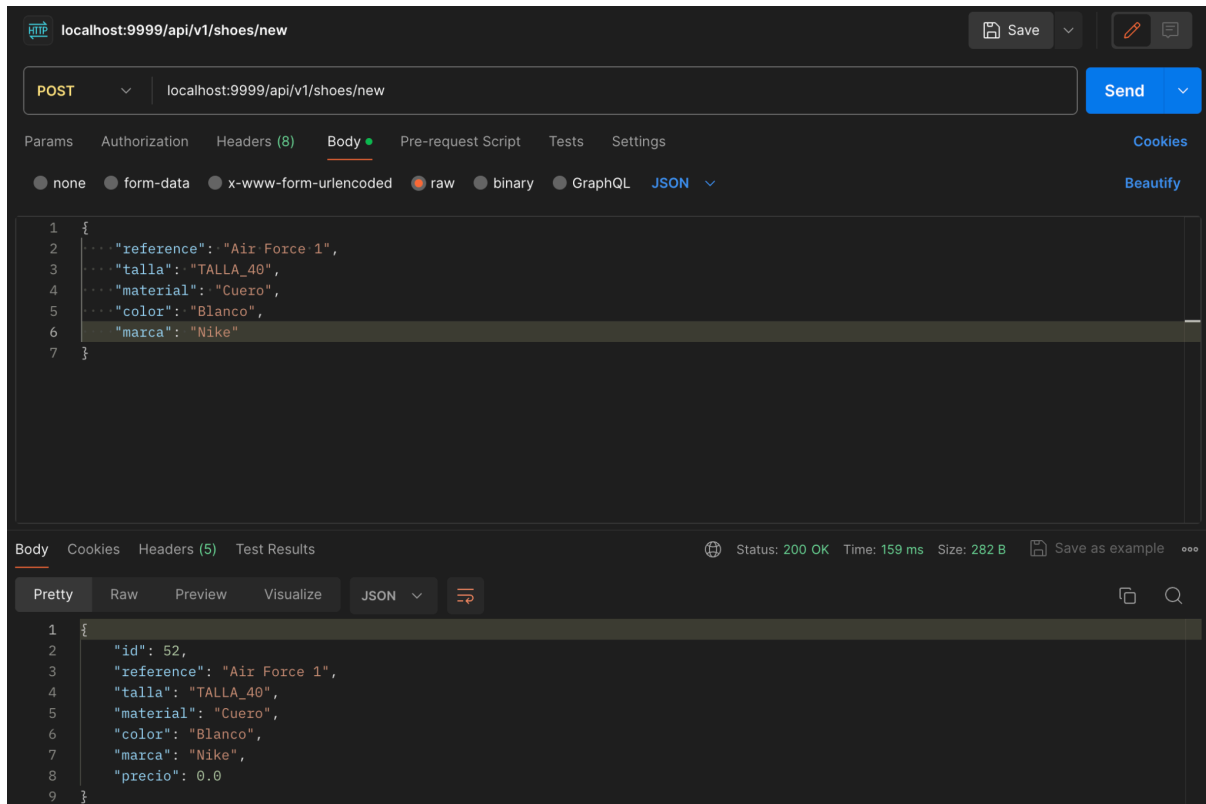
/**
 *
 * @author islam
 */
@Data
@Builder
public class ErrorDTO {
    private String code;
    private String message;
}

```

Clase Error con los atributos que usaremos, para definir la estructura del error.

Pruebas de la API

- POST

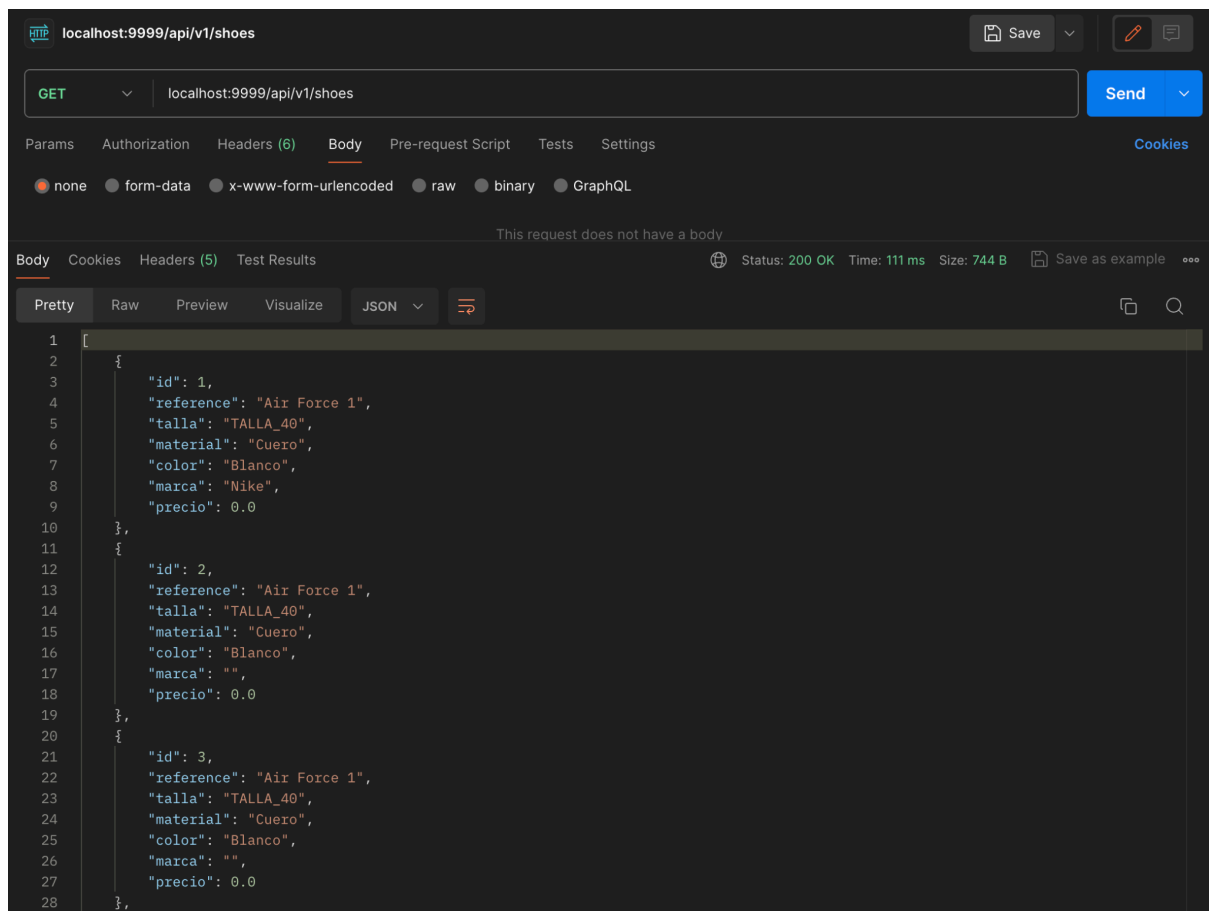


Como se puede observar en la captura, la respuesta que nos devuelve es de 200, por lo tanto funciona correctamente.

Como se puede comprobar, al no introducir el precio el valor base es de 0.0 ya que este se estableció en la clase de la entidad.

```
@Schema(description = "Precio unitario", example = "120.99", defaultValue="0")  
@Min(value = 0)  
private float precio;  
  
}
```


- GET ALL



localhost:9999/api/v1/shoes

GET localhost:9999/api/v1/shoes

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

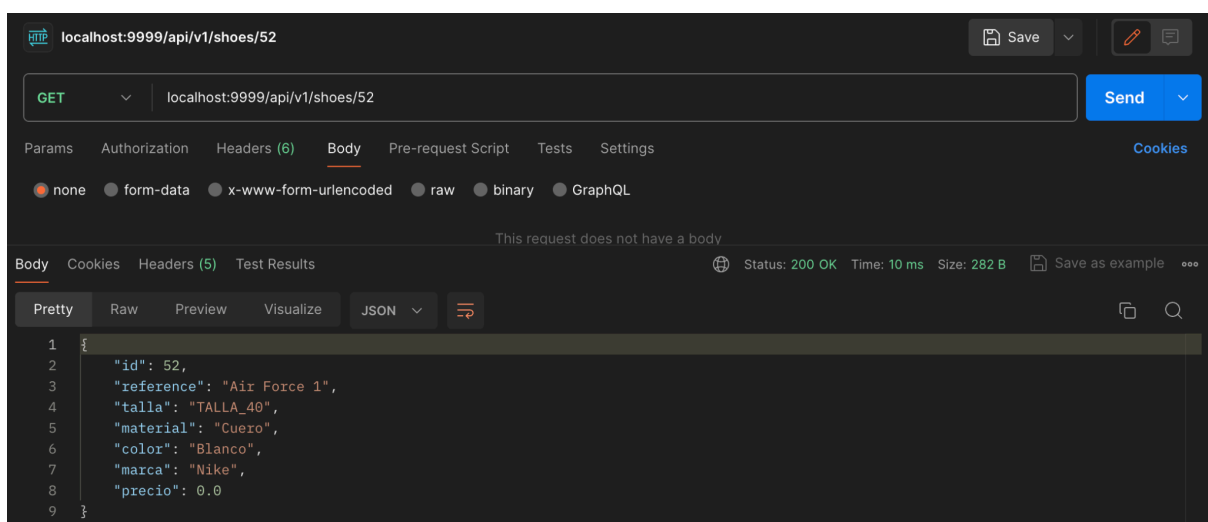
This request does not have a body

Body Cookies Headers (5) Test Results Status: 200 OK Time: 111 ms Size: 744 B Save as example

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "id": 1,
4     "reference": "Air Force 1",
5     "talla": "TALLA_40",
6     "material": "Cuero",
7     "color": "Blanco",
8     "marca": "Nike",
9     "precio": 0.0
10  },
11  {
12    "id": 2,
13    "reference": "Air Force 1",
14    "talla": "TALLA_40",
15    "material": "Cuero",
16    "color": "Blanco",
17    "marca": "",
18    "precio": 0.0
19  },
20  {
21    "id": 3,
22    "reference": "Air Force 1",
23    "talla": "TALLA_40",
24    "material": "Cuero",
25    "color": "Blanco",
26    "marca": "",
27    "precio": 0.0
28  },
29 ]
```

- GET ONE



localhost:9999/api/v1/shoes/52

GET localhost:9999/api/v1/shoes/52

Params Authorization Headers (6) Body Pre-request Script Tests Settings Cookies

none form-data x-www-form-urlencoded raw binary GraphQL

This request does not have a body

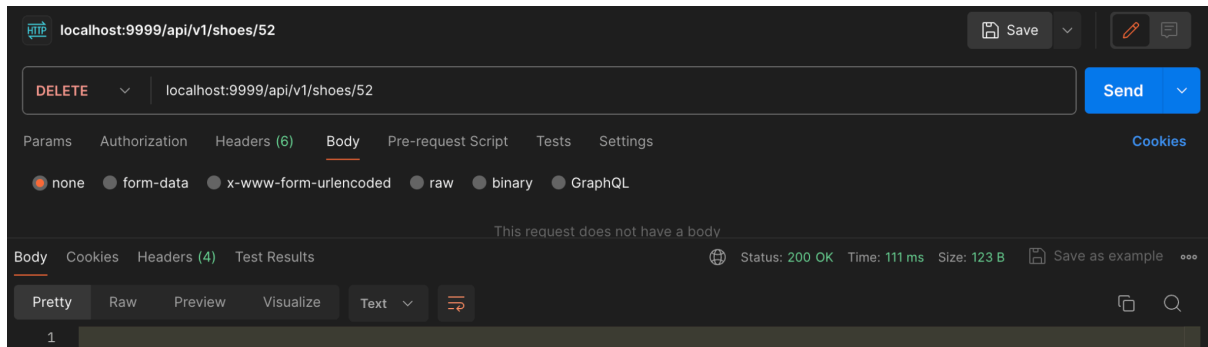
Body Cookies Headers (5) Test Results Status: 200 OK Time: 10 ms Size: 282 B Save as example

Pretty Raw Preview Visualize JSON

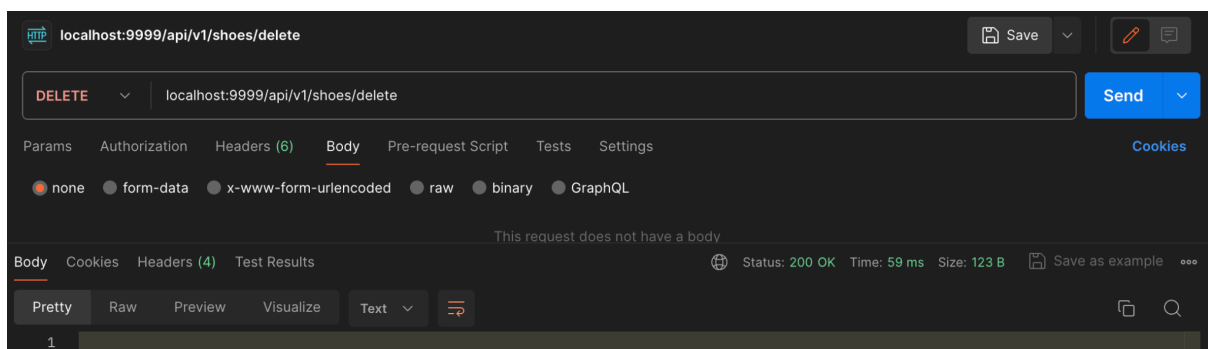
```
1 {
2   "id": 52,
3   "reference": "Air Force 1",
4   "talla": "TALLA_40",
5   "material": "Cuero",
6   "color": "Blanco",
7   "marca": "Nike",
8   "precio": 0.0
9 }
```

Los métodos de get funcionan a la perfección ya que nos devuelve tanto el estado de 200 OK, como las entidades agregadas anteriormente.

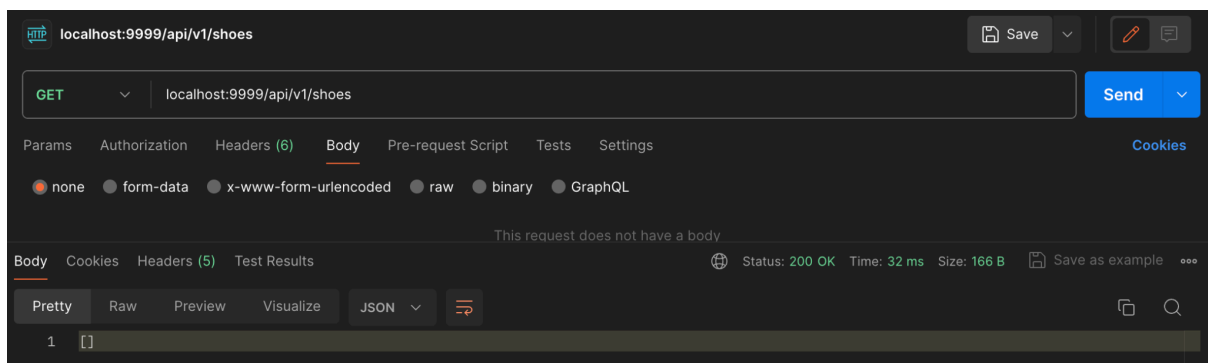
- DELETE



- DELETE ALL

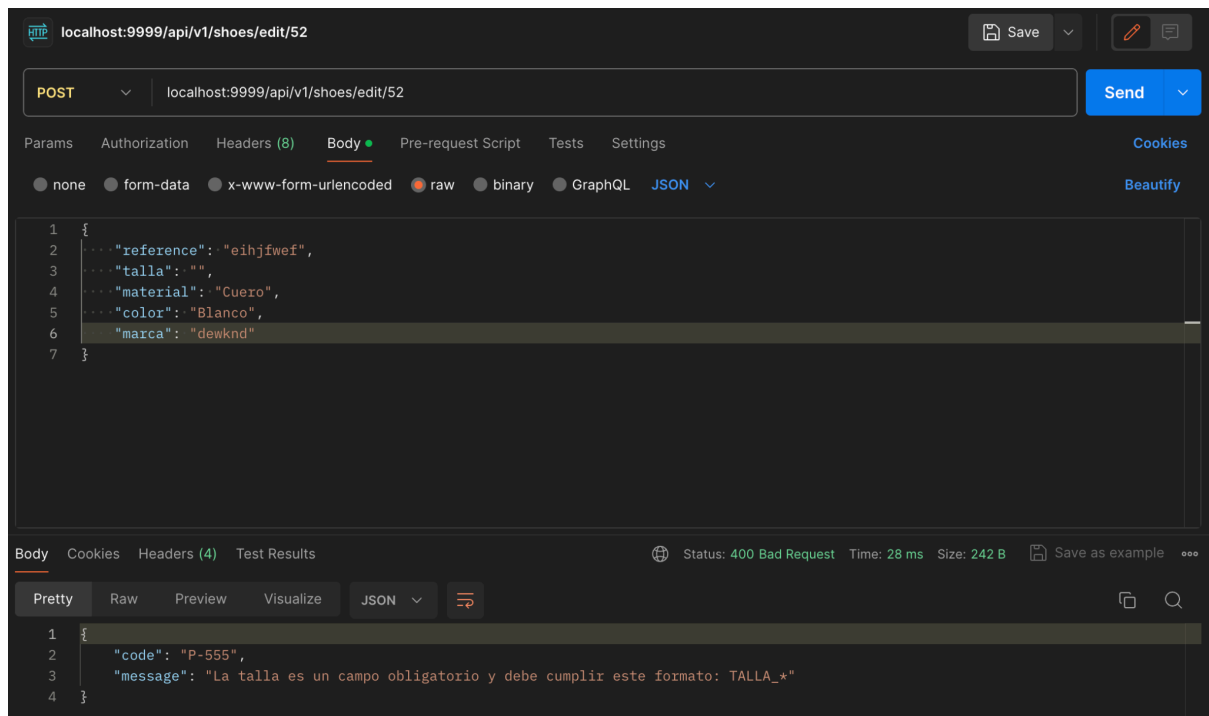


RESULTADO:



Los delete funcionan a la perfección porque el estado de las sentencias son 200 y al realizar un **get shoes** no devuelve ningún resultado, por lo tanto funcionan a la perfección.

- POST edit



El método funciona a la perfección, pero en este caso se ha puesto de ejemplo un error para probar el manejo de errores en la api. En este caso se ha creado un error personalizado para ser de ayuda y más explicativo para los usuarios.

En esta imagen se deja el campo talla vacío, por lo tanto el mensaje personalizado en este caso es este:

