

2025

LENGUAJE DE PROGRAMACIÓN AVANZADO 1 - 2501B04G1



INGENIERIA DE SISTEMAS

Realizado por:

Carlos Andres Gaviria Ocampo

Presentado a:

Martha Nicolasa Amaya Becerra

📌 Evaluación de API RESTful

Objetivo: Evaluar el conocimiento básico de los estudiantes en **API RESTful con Spring Boot** a través de 5 ejercicios prácticos y sencillos.

📁 Instrucciones Generales

- ✓ Los estudiantes deben **desarrollar los ejercicios en Spring Boot 3+**.
- ✓ Pueden probar sus endpoints con **Postman o el navegador**.
- ✓ Se recomienda documentar el código con comentarios.
- ✓ **Subir el código a GitHub** y compartir el enlace en Moodle.

📝 Ejercicio 1: Crear un Endpoint Simple

📌 **Descripción:** Crea un endpoint en un controlador REST que devuelva el mensaje

"¡Hola, API RESTful!" cuando el usuario acceda a /api/saludo.

📄 Código Esperado:

java

@RestController

@RequestMapping("/api")

```
public class SaludoController {
```

```
    @GetMapping("/saludo")    public
```

```
String obtenerSaludo() {  
    return "¡Hola, API RESTful!";  
  
}
```

📌 **Prueba:**

- **GET:** http://localhost:8080/api/saludo
- **Salida esperada:** "¡Hola, API RESTful!"

📝 **Ejercicio 2: Crear un CRUD Básico de Productos**

📌 **Descripción:** Implementa una API RESTful que maneje un CRUD (Crear, Leer, Actualizar, Eliminar) para un modelo **Producto**.

📄 **Código Esperado:**

```
java  
  
@Data  
  
@AllArgsConstructor  
tor @NoArgsConstructor  
  
public class Producto {  
  
    private String id;    private  
    String nombre;    private  
    double precio; }
```

📄 **Endpoints esperados:**

- POST /api/productos → Agregar un producto.
- GET /api/productos → Listar todos los productos.
- GET /api/productos/{id} → Obtener un producto por ID.
- PUT /api/productos/{id} → Actualizar un producto.
- DELETE /api/productos/{id} → Eliminar un producto.

📌 **Prueba:**

- **POST:** Agregar un producto con Postman. **GET:** Consultar todos los productos.

📝 **Ejercicio 3: Implementar Internacionalización (i18n)**

📌 **Descripción:** Modifica el **Ejercicio 1** para que el mensaje de saludo pueda ser traducido según el idioma solicitado en la URL (?lang=es o ?lang=en).

📄 **Archivos messages.properties:**

📄 **messages_es.properties**

properties

saludo=¡Hola, API RESTful en Español!

📄 **messages_en.properties**

properties

saludo=Hello, RESTful API in English!

📄 **Código Esperado:**

java

@RestController

@RequestMapping("/api")

public class SaludoController {

@Autowired

private MessageSource messageSource;

@GetMapping("/saludo")

public String obtenerSaludo(@RequestHeader(name = "Accept-Language",
required = false) Locale locale) { return messageSource.getMessage("saludo", null,
locale);

} }

📌 **Prueba:**

□ GET: <http://localhost:8080/api/saludo?lang=es> → "¡Hola, API RESTful en
Español!" □ GET: <http://localhost:8080/api/saludo?lang=en> → "Hello, RESTful API in
English!"

 **Ejercicio 4: Crear un Endpoint Reactivo con WebFlux**

 **Descripción:** Modifica el **Ejercicio 2** para que la lista de productos sea manejada de manera **reactiva** usando Flux.

 **Código Esperado:**

java

```
@RestController  
  
@RequestMapping("/api/productos")  
  
public class ProductoController {  
  
    @GetMapping    public  
    Flux<Producto> listarProductos() {  
  
        return Flux.just(  
  
            new Producto("1", "Laptop",  
1200.0),      new Producto("2", "Mouse", 25.0),  
  
            new Producto("3", "Teclado", 45.0)  
        );  
  
    }  
  
}
```

📌 **Prueba:**

- **GET:** <http://localhost:8080/api/productos>
- **Salida esperada:** JSON con los productos en formato reactivo.

📝 **Ejercicio 5: Implementar Pruebas con StepVerifier**

📌 **Descripción:** Crea una prueba unitaria con **StepVerifier** para validar que el endpoint de productos devuelve los valores esperados.

📄 **Código Esperado:**

java

```
@SpringBootTest
```

```
@AutoConfigureMockMvc
```

```
public class ProductoControllerTest {
```

```
    @Autowired
```

```
    private ProductoController productoController;
```

```
    @Test
```

```
    public void testListaProductos() {
```

```
        Flux<Producto> productos = productoController.listarProductos();
```

```
        StepVerifier.create(productos)
```

```

    .expectNextMatches(p -> p.getNombre().equals("Laptop"))

    .expectNextMatches(p -> p.getNombre().equals("Mouse"))

    .expectNextMatches(p -> p.getNombre().equals("Teclado"))

.verifyComplete();

}
}

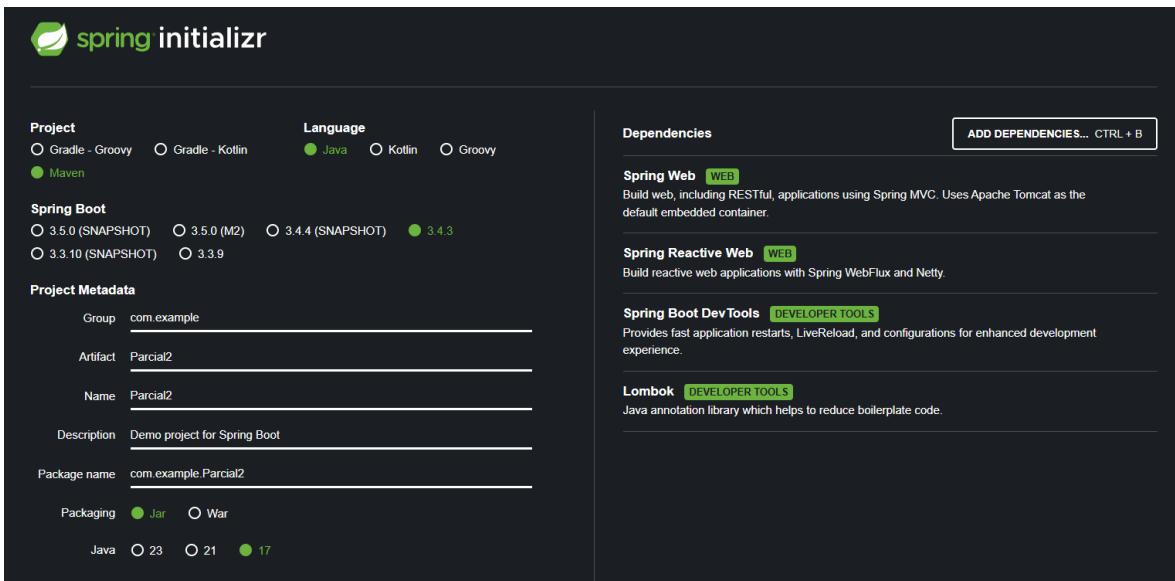
```

📌 Prueba:

- **Ejecutar la prueba con JUnit y StepVerifier.**
- **Salida esperada:** La prueba debe pasar correctamente.

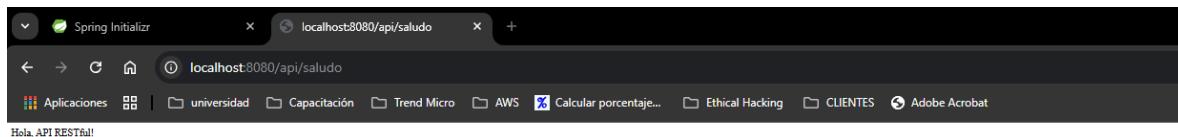
SOLUCION

Punto 1



The screenshot shows the Spring Initializr interface for creating a new Spring Boot project. The configuration is as follows:

- Project:** Maven (selected)
- Language:** Java (selected)
- Spring Boot:** 3.4.3 (selected)
- Project Metadata:**
 - Group: com.example
 - Artifact: Parcial2
 - Name: Parcial2
 - Description: Demo project for Spring Boot
 - Package name: com.example.Parcial2
 - Packaging: Jar (selected)
 - Java version: 23 (selected)
 - Java version: 21 (selected)
 - Java version: 17 (selected)
- Dependencies:**
 - Spring Web** (WEB): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
 - Spring Reactive Web** (WEB): Build reactive web applications with Spring WebFlux and Netty.
 - Spring Boot DevTools** (DEVELOPER TOOLS): Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
 - Lombok** (DEVELOPER TOOLS): Java annotation library which helps to reduce boilerplate code.



```

project ~
  Parcial2 C:\Parcial2
    > idea
    > mvn
    > src
      > main
        > java
          > com.example.Parcial2
            > Parcial2Application
          > controller
            > SaludoController
              > SaludoController.java
            > model
            > resources
            > static
            > templates
            > application.properties
      > test
    > target
    > .gitattributes
    > .gitignore
    > HELP.md
    > mvnw
    > mvnw.cmd
    > pom.xml
  > External Libraries

```

```

application.properties   SaludoController.java   Parcial2Application.java
1 package com.example.Parcial2;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RequestMapping;
5 import org.springframework.web.bind.annotation.RestController;
6
7 @RestController
8 @RequestMapping("/api")
9 public class SaludoController {
10
11     @GetMapping("/saludo")
12     public String obtenerSaludo() {
13         return "Hola, API RESTful!";
14     }
15 }

```

Saludo

```
package com.example.parcial2.controller;
```

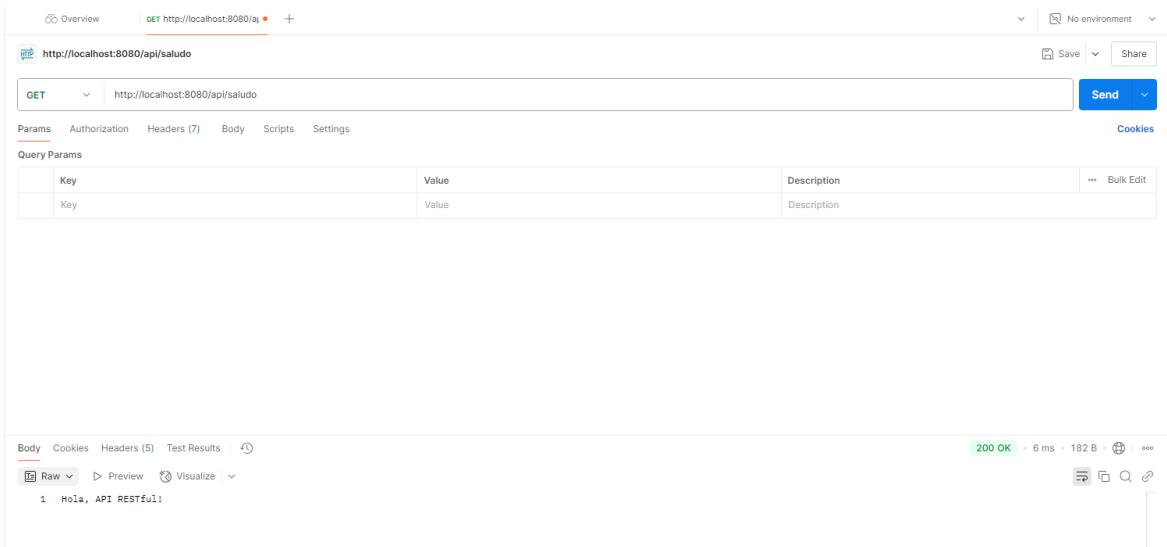
```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
```

@RestController

```
@RequestMapping("/api")
public class SaludoController {

    @GetMapping("/saludo")
    public String obtenerSaludo() {
        return "Hola, API RESTful!";
    }
}
```

Postman



The screenshot shows the Postman interface with a GET request to `http://localhost:8080/api/saludo`. The response is a 200 OK status with the body containing the string `Hola, API RESTful!`.

Punto 2

```
package com.example.parcial2.controller;
```

```
import com.example.parcial2.model.Producto;  
  
import org.springframework.web.bind.annotation.*;  
  
import java.util.ArrayList;  
  
import java.util.List;  
  
import java.util.UUID;  
  
  
@RestController  
  
@RequestMapping("/api/productos")  
  
public class ProductoController {  
  
  
    private List<Producto> productos = new ArrayList<>();  
  
  
  
    @PostMapping  
    public Producto agregarProducto(@RequestBody Producto producto) {  
  
        producto.setId(UUID.randomUUID().toString());  
  
        productos.add(producto);  
  
        return producto;  
    }  
  
  
    @GetMapping  
    public List<Producto> listarProductos() {
```

```
return productos;  
}  
  
@GetMapping("/{id}")  
public Producto obtenerProducto(@PathVariable String id) {  
  
    return productos.stream()  
        .filter(p -> p.getId().equals(id))  
        .findFirst()  
        .orElseThrow(() -> new RuntimeException("Producto no encontrado"));  
}  
  
@PutMapping("/{id}")  
public Producto actualizarProducto(@PathVariable String id, @RequestBody Producto  
producto) {  
  
    Producto productoExistente = obtenerProducto(id);  
    productoExistente.setNombre(producto.getNombre());  
    productoExistente.setPrecio(producto.getPrecio());  
  
    return productoExistente;  
}  
  
@DeleteMapping("/{id}")  
public void eliminarProducto(@PathVariable String id) {
```

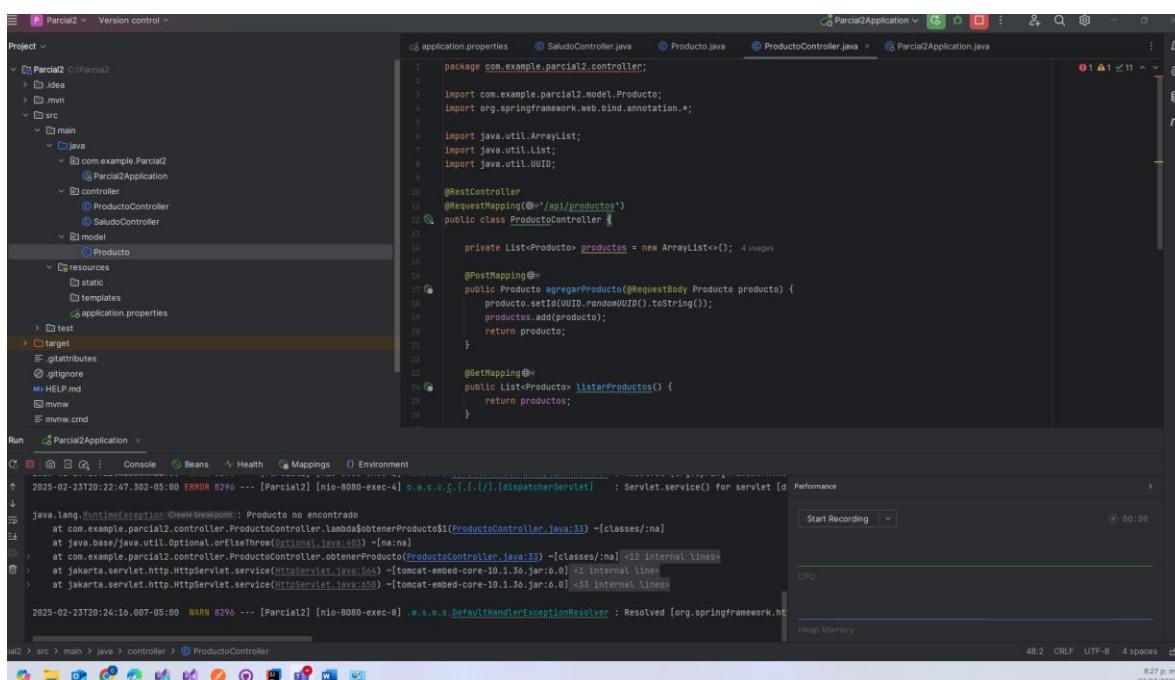
```

        productos.removeIf(p -> p.getId().equals(id));

    }

}

```



The screenshot shows the IntelliJ IDEA interface. The code editor displays a Java controller class named `ProductoController.java`. The run tab at the bottom shows logs from a Tomcat server, indicating a 404 error for a non-existent product and a warning about a default exception resolver.

```

package com.example.parcial2.controller;

import com.example.parcial2.model.Producto;
import org.springframework.web.bind.annotation.*;

import java.util.ArrayList;
import java.util.List;
import java.util.UUID;

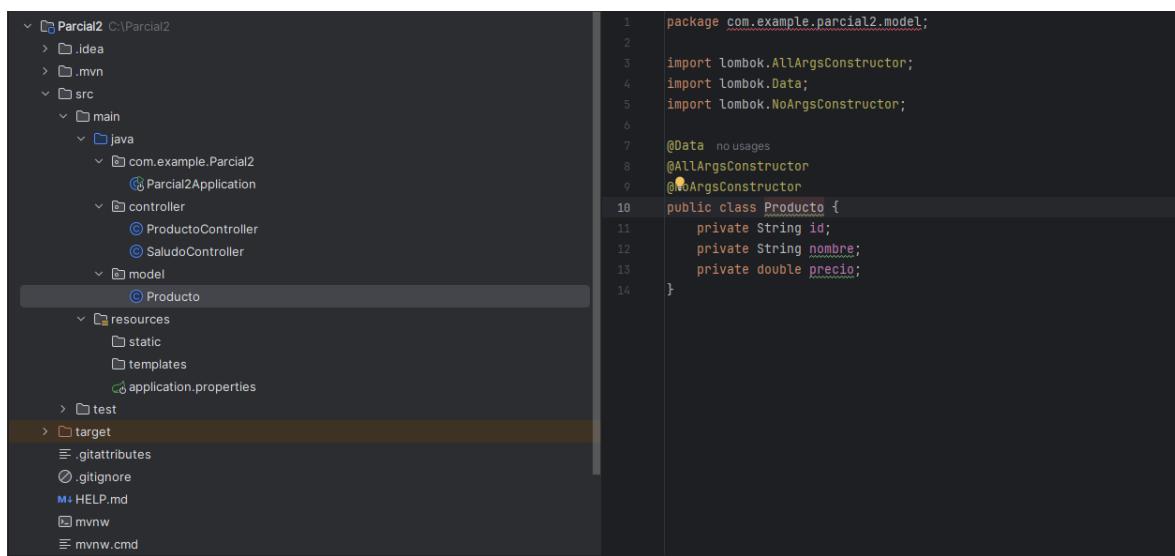
@RestController
@RequestMapping("/api/products")
public class ProductoController {

    private List<Producto> productos = new ArrayList<>(); 4 usages

    @PostMapping
    public Producto agregarProducto(@RequestBody Producto producto) {
        producto.setId(UUID.randomUUID().toString());
        productos.add(producto);
        return producto;
    }

    @GetMapping
    public List<Producto> listarProductos() {
        return productos;
    }
}

```



The screenshot shows the IntelliJ IDEA interface. The code editor displays a Java model class named `Producto.java`. The run tab at the bottom shows logs from a Tomcat server, indicating a 404 error for a non-existent product and a warning about a default exception resolver.

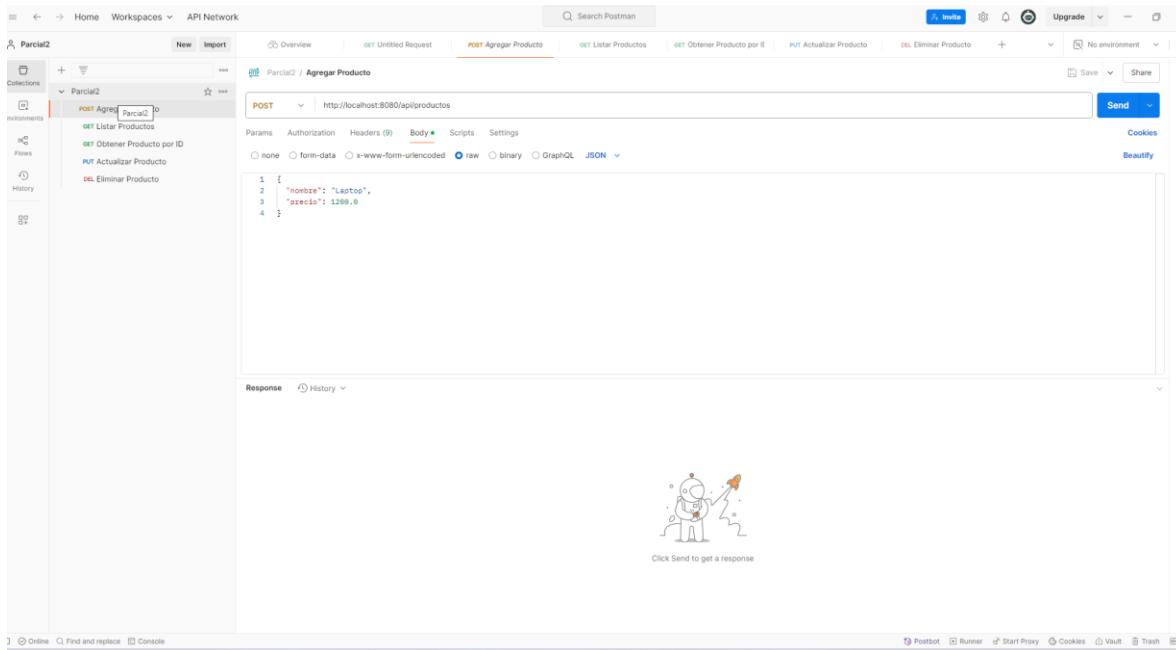
```

package com.example.parcial2.model;

import lombok.AllArgsConstructorConstructor;
import lombok.Data;
import lombok.NoArgsConstructorConstructor;

@Data no usages
@AllArgsConstructorConstructor
@NoArgsConstructorConstructor
public class Producto {
    private String id;
    private String nombre;
    private double precio;
}

```



The screenshot shows the Postman application interface. A collection named 'Parcial2' is selected. Inside it, a request named 'POST Agregar Producto' is highlighted. The request method is POST, and the URL is http://localhost:8080/api/productos. The 'Body' tab is selected, showing a JSON payload:

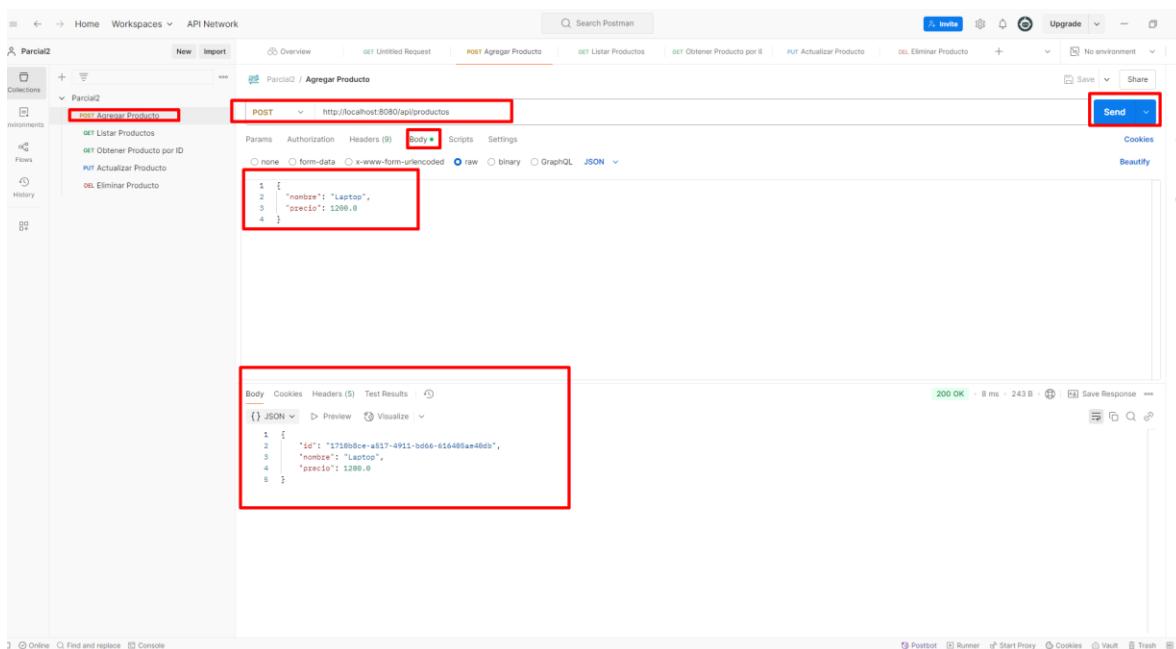
```

1 {
2   "nombre": "Laptop",
3   "precio": 1200.0
4 }

```

The response area is currently empty, showing a small cartoon character icon and the text 'Click Send to get a response'.

Agregar producto



The screenshot shows the same Postman interface after the request has been sent. The 'Send' button is highlighted with a red box. The response area displays the JSON response from the server:

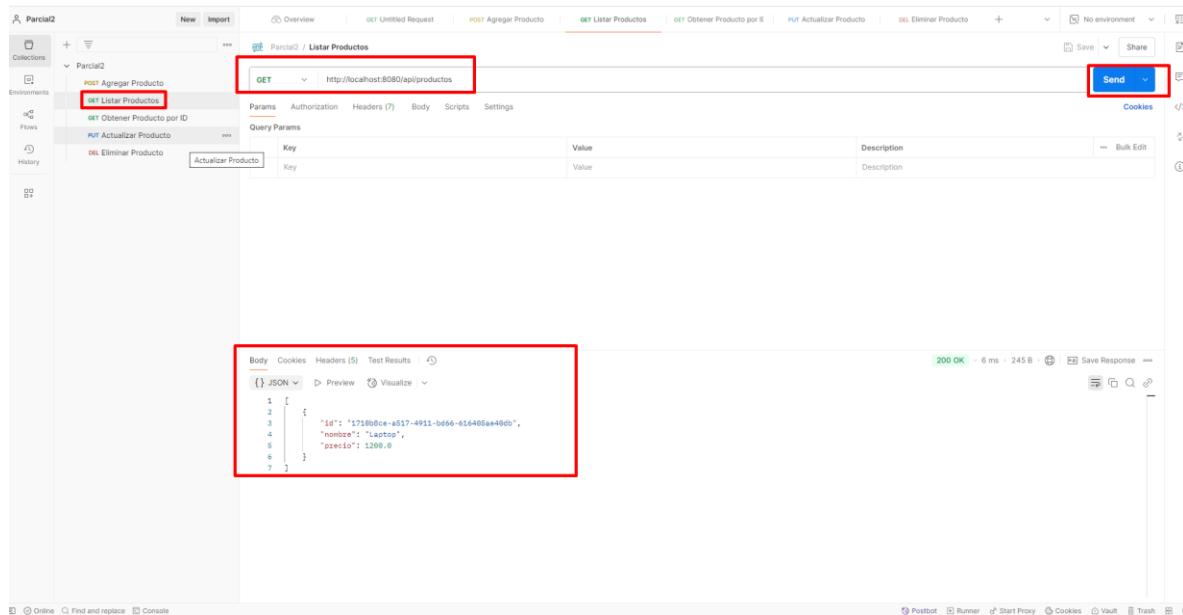
```

{
  "id": "1710b08ce-a537-4911-bd66-616485ae48d5",
  "nombre": "Laptop",
  "precio": 1200.0
}

```

The entire response area is highlighted with a large red box.

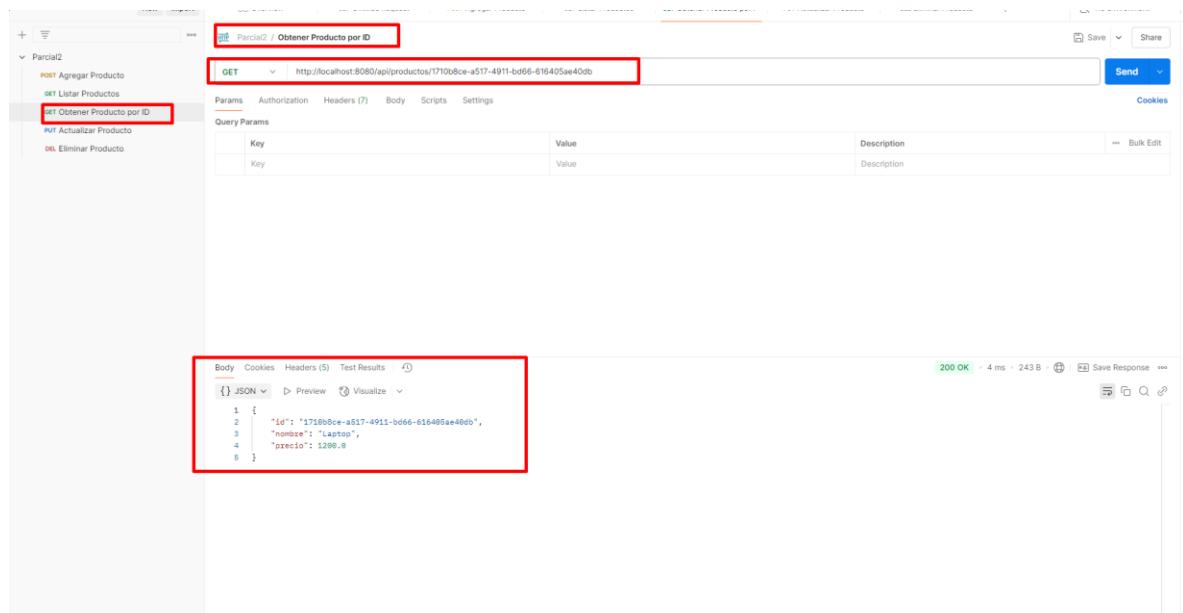
Listar Productos



The screenshot shows the Postman interface with a collection named 'Parcial2'. A GET request is selected for the endpoint `/api/productos`. The response body is displayed as JSON:

```
{
  "id": "1710b0ce-a517-4911-bd66-616405ae40db",
  "nombre": "Laptop",
  "precio": 1200.0
}
```

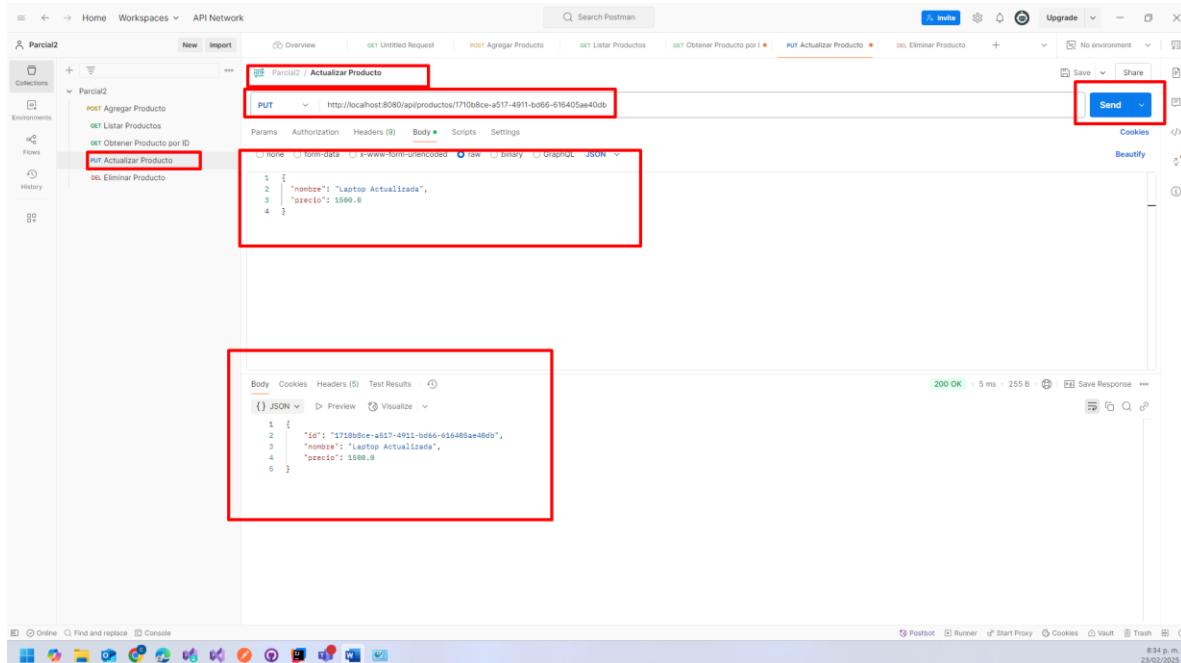
Obtener Producto por ID



The screenshot shows the Postman interface with a collection named 'Parcial2'. A GET request is selected for the endpoint `/api/productos/1710b0ce-a517-4911-bd66-616405ae40db`. The response body is displayed as JSON:

```
{
  "id": "1710b0ce-a517-4911-bd66-616405ae40db",
  "nombre": "Laptop",
  "precio": 1200.0
}
```

Actualizar Producto



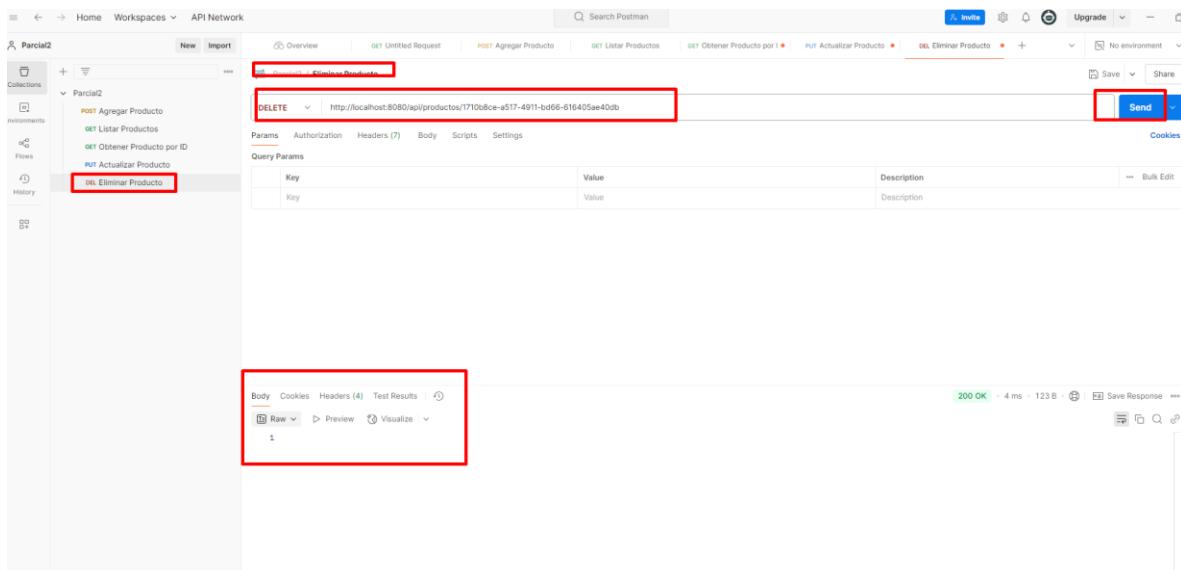
The screenshot shows the Postman interface for an API endpoint named "Actualizar Producto". The URL is `http://localhost:8080/api/productos/1710b8ce-a517-4911-bd66-616405ae40db`. The request method is PUT. The Body tab contains the following JSON:

```

1 [
2   {
3     "id": "1710b8ce-a517-4911-bd66-616405ae40db",
4     "nombre": "Laptop Actualizada",
5     "precio": 1800.0
6   }
7 ]
  
```

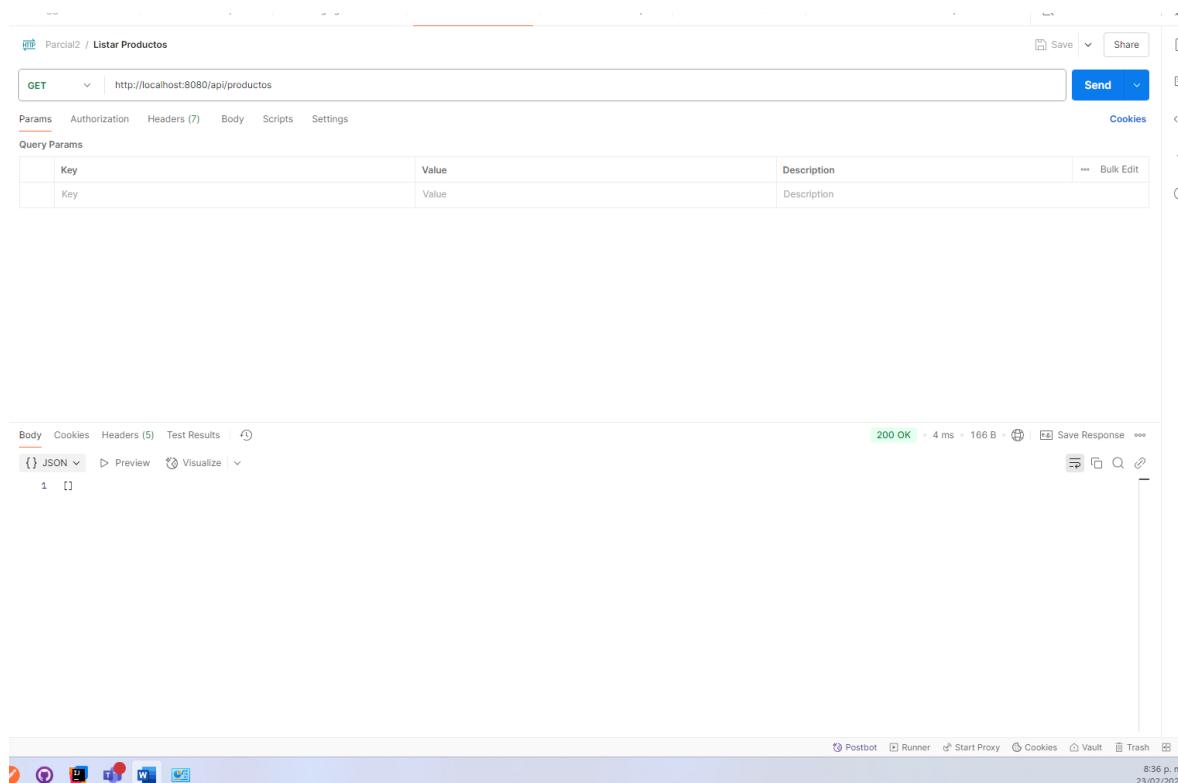
The response status is 200 OK, with a timestamp of 8:34 p.m. on 23/02/2025.

Eliminar producto



The screenshot shows the Postman interface for an API endpoint named "Eliminar Producto". The URL is `http://localhost:8080/api/productos/1710b8ce-a517-4911-bd66-616405ae40db`. The request method is DELETE. The Body tab shows a single key "Key" with the value "Value". The response status is 200 OK, with a timestamp of 4 ms on 23/02/2025.

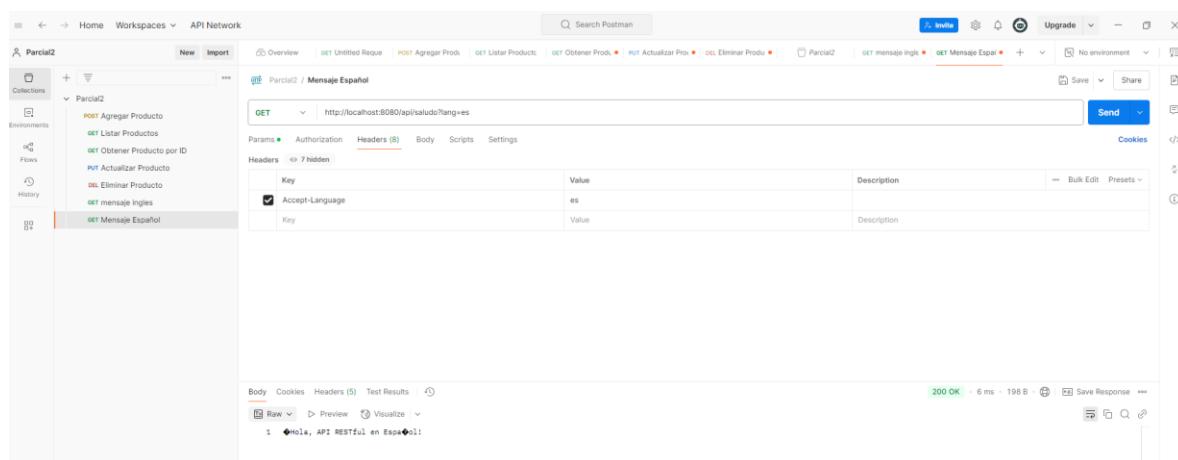
Consulta final



The screenshot shows the Postman interface with a collection named "Parcial2". A GET request is made to `http://localhost:8080/api/productos`. The response status is 200 OK, with a response time of 4 ms, 166 B, and a JSON body containing an empty array: `[]`.

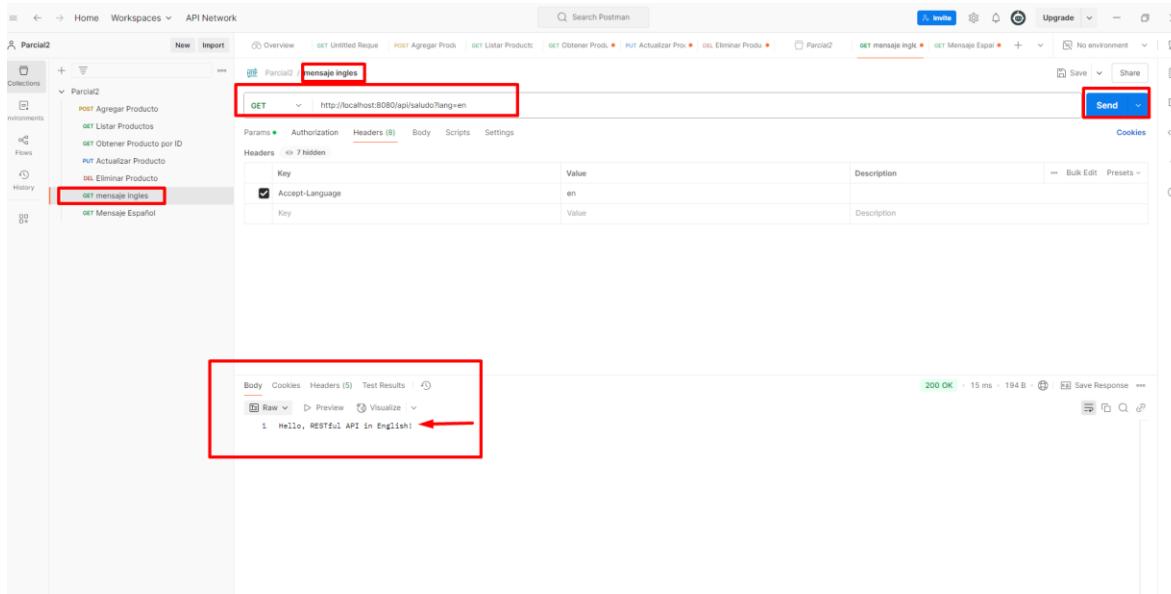
Punto 3 postman

Español



The screenshot shows the Postman interface with a collection named "Parcial2". A GET request is made to `http://localhost:8080/api/saludo?lang=es`. The response status is 200 OK, with a response time of 6 ms, 198 B, and a JSON body containing the message: `"Hola, API RESTful en Español"`.

Ingles



The screenshot shows the Postman interface with a collection named "Parcial2". A GET request is selected with the URL `http://localhost:8080/api/saludo?lang=en`. The Headers section includes `Accept-Language: en`. The response body shows the message "Hello, RESTful API in English!".

Punto 4

Web flux

```
package com.example.parcial2.controller;

import com.example.parcial2.model.Producto;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;
import reactor.core.publisher.Flux;
```

@RestController

@RequestMapping("/api/productos")

```
public class ProductoController {
```

@GetMapping

```
public Flux<Producto> listarProductos() {
```

```
    return Flux.just(
```

```
        new Producto("1", "Laptop", 1200.0),
```

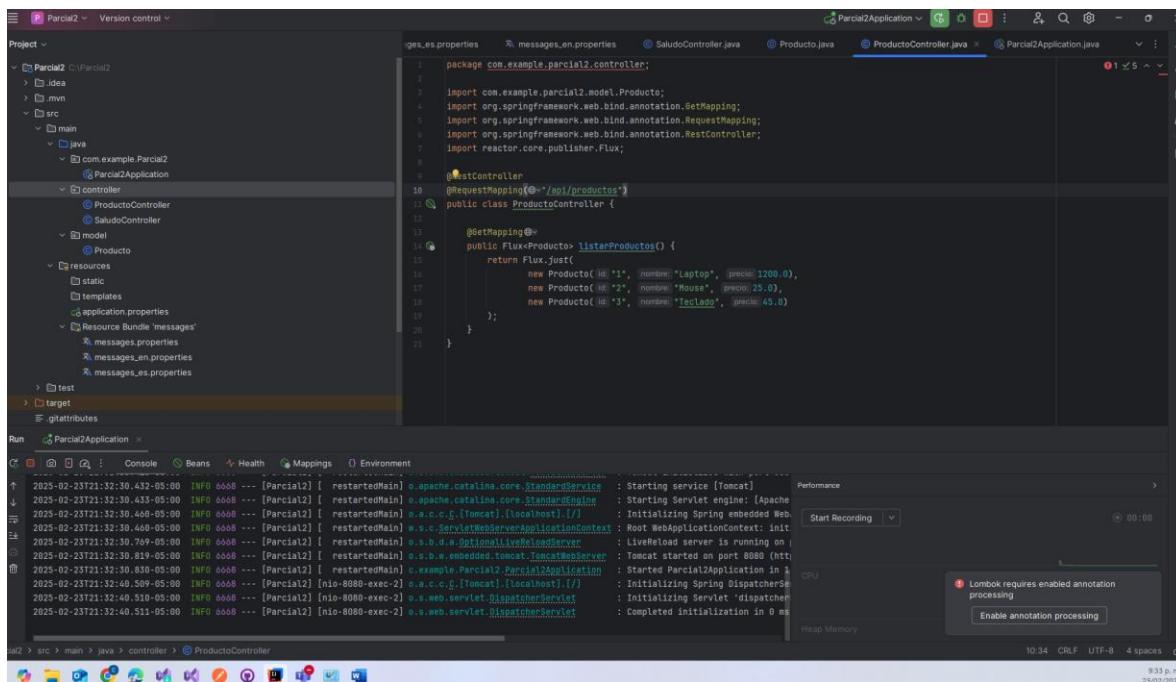
```
        new Producto("2", "Mouse", 25.0),
```

```
        new Producto("3", "Teclado", 45.0)
```

```
    );
```

```
}
```

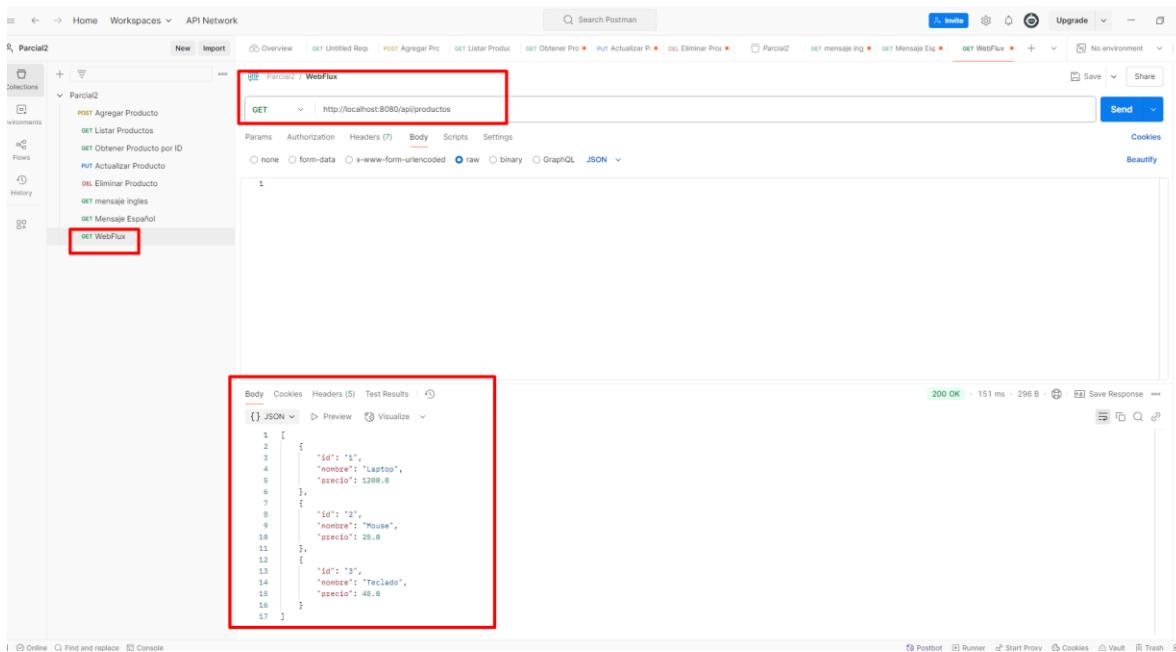
```
}
```



```

1 package com.example.parcial2.controller;
2
3 import com.example.parcial2.model.Producto;
4 import org.springframework.web.bind.annotation.GetMapping;
5 import org.springframework.web.bind.annotation.RequestMapping;
6 import org.springframework.web.bind.annotation.RestController;
7 import reactor.core.publisher.Flux;
8
9
10 @RestController
11 @RequestMapping("/api/productos")
12 public class ProductoController {
13
14     @GetMapping()
15     public Flux<Producto> listarProductos() {
16
17         return Flux.just(
18             new Producto(id = "1", nombre = "Laptop", precio = 1200.0),
19             new Producto(id = "2", nombre = "Mouse", precio = 25.0),
20             new Producto(id = "3", nombre = "Teclado", precio = 45.0)
21         );
22     }
23 }

```



Body

```

1 [
2   {
3     "id": "1",
4     "nombre": "Laptop",
5     "precio": 1200.0
6   },
7   {
8     "id": "2",
9     "nombre": "Mouse",
10    "precio": 25.0
11   },
12   {
13     "id": "3",
14     "nombre": "Teclado",
15     "precio": 45.0
16   }
17 ]

```

Punto 5

Producto controller test

```
package com.example.Parcial2;

import com.example.parcial2.controller.ProductoController;
import com.example.parcial2.model.Producto;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.context.ApplicationContext;
import reactor.core.publisher.Flux;
import reactor.test.StepVerifier;

@SpringBootTest
public class ProductoControllerTest {

    @Autowired
    private ApplicationContext context;

    private ProductoController productoController;

    @BeforeEach
    void setUp() {
        productoController = context.getBean(ProductoController.class);
    }

    @Test
    public void testListaProductos() {
        Flux<Producto> productos = productoController.listarProductos();

        StepVerifier.create(productos)
            .expectNextMatches(p -> "Laptop".equals(p.getNombre()) && p.getPrecio() == 1200.0)
            .expectNextMatches(p -> "Mouse".equals(p.getNombre()) && p.getPrecio() == 25.0)
            .expectNextMatches(p -> "Teclado".equals(p.getNombre()) && p.getPrecio() == 45.0)
            .verifyComplete();
    }
}
```

POM

```
<?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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
```

```
<parent>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-parent</artifactId>
  <version>3.4.3</version>
  <relativePath/>
</parent>

<groupId>com.example</groupId>
<artifactId>Parcial2</artifactId>
<version>0.0.1-SNAPSHOT</version>
<name>Parcial2</name>
<description>Demo project for Spring Boot</description>

<properties>
  <java.version>17</java.version>
  <mockito.version>5.7.0</mockito.version> <!-- Centralización de la versión de Mockito -->
</properties>

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

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

  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
  </dependency>

  <dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <scope>provided</scope>
  </dependency>

  <!-- Dependencias de pruebas -->
  <dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
  </dependency>

  <dependency>
    <groupId>io.projectreactor</groupId>
```

```
<artifactId>reactor-test</artifactId>
<scope>test</scope>
</dependency>

<!-- Mockito para JUnit 5 -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>${mockito.version}</version>
    <scope>test</scope>
</dependency>

<!-- Mockito Inline (para evitar advertencias del mock-maker) -->
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-inline</artifactId>
    <version>4.11.0</version> <!-- Última versión estable -->
    <scope>test</scope>
</dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <configuration>
                <annotationProcessorPaths>
                    <path>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </path>
                </annotationProcessorPaths>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <configuration>
                <excludes>
                    <exclude>
                        <groupId>org.projectlombok</groupId>
                        <artifactId>lombok</artifactId>
                    </exclude>
                </excludes>
            </configuration>
        </plugin>
    <!-- Plugin para evitar la advertencia de Mockito y permitir el uso de agentes -->
    <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
```

```
<version>3.0.0-M9</version>
<configuration>
    <argLine>-javaagent:${settings.localRepository}/org/mockito/mockito-inline/4.11.0/mockito-inline-4.11.0.jar -XX:+EnableDynamicAgentLoading</argLine>
</configuration>
</plugin>
</plugins>
</build>
</project>
```

Resultados punto 5

The screenshot shows the IntelliJ IDEA interface with the 'Run' tool window open. The title bar says 'Run' and 'Parcial2Application x'. Below it, the tab bar shows 'ProductoControllerTest x'. The main area displays the test results:

- Tests passed: 1 of 1 test – 557ms**
- testListaProductos 557 ms**

Below the results, the terminal output shows:

```
*C:\Program Files\Java\jdk-22\bin\java.exe* ...
21:44:33.608 [main] INFO org.springframework.test.context.support.AnnotationConfigContextLoaderUtils -- Could not de...
21:44:33.823 [main] INFO org.springframework.boot.test.context.SpringBootTestTestContextBootstrapper -- Found @SpringBoo...
21:44:33.978 [main] INFO org.springframework.boot.devtools.restart.RestartApplicationListener -- Restart disabled du...
```

Process finished with exit code 0

El proceso terminó con exit code 0, lo que significa que las pruebas pasaron

exitosamente. 