

“Informe Experiencia Número 3 de Desarrollo Full Stack ”

Sección:001D

Docente: Viviana Poblete

Integrantes: Javier Albornoz

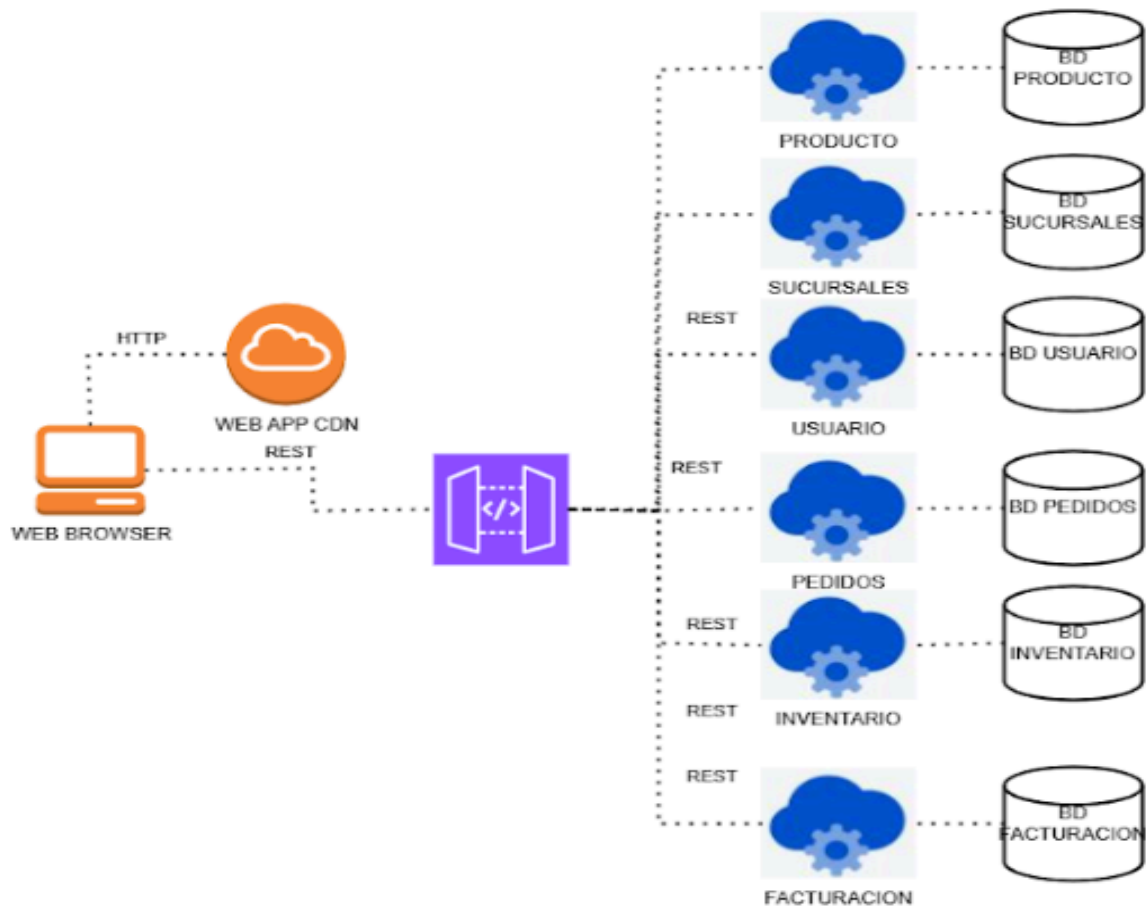
Vicente Hormazabal

Diego Neira

Índice

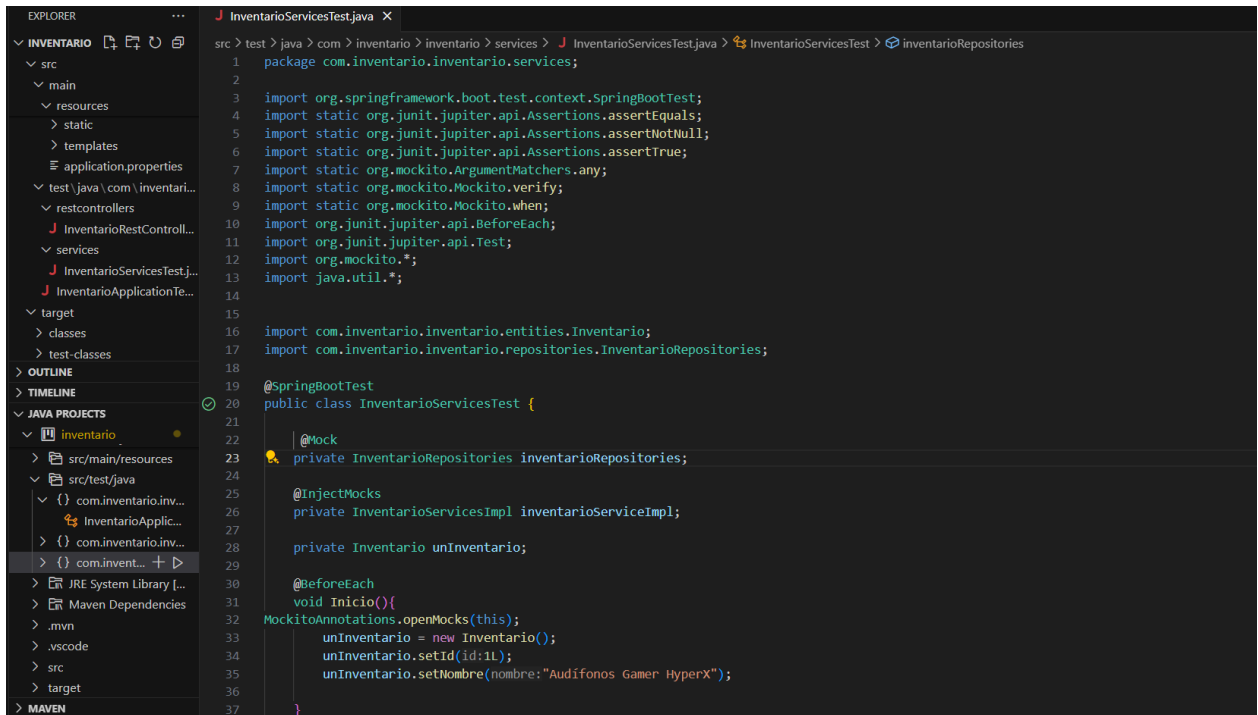
“Informe Experiencia Número 3 de Desarrollo Full Stack ”	1
índice	2
diagrama arquitectura de microservicios	3
Plan de pruebas	4
❖ Pruebas Unitarias.....	8
❖ Pruebas de integración.....	10
OAS	11
GITHUB	20
conclusión	21

Diagrama arquitectura de microservicios



Plan de pruebas

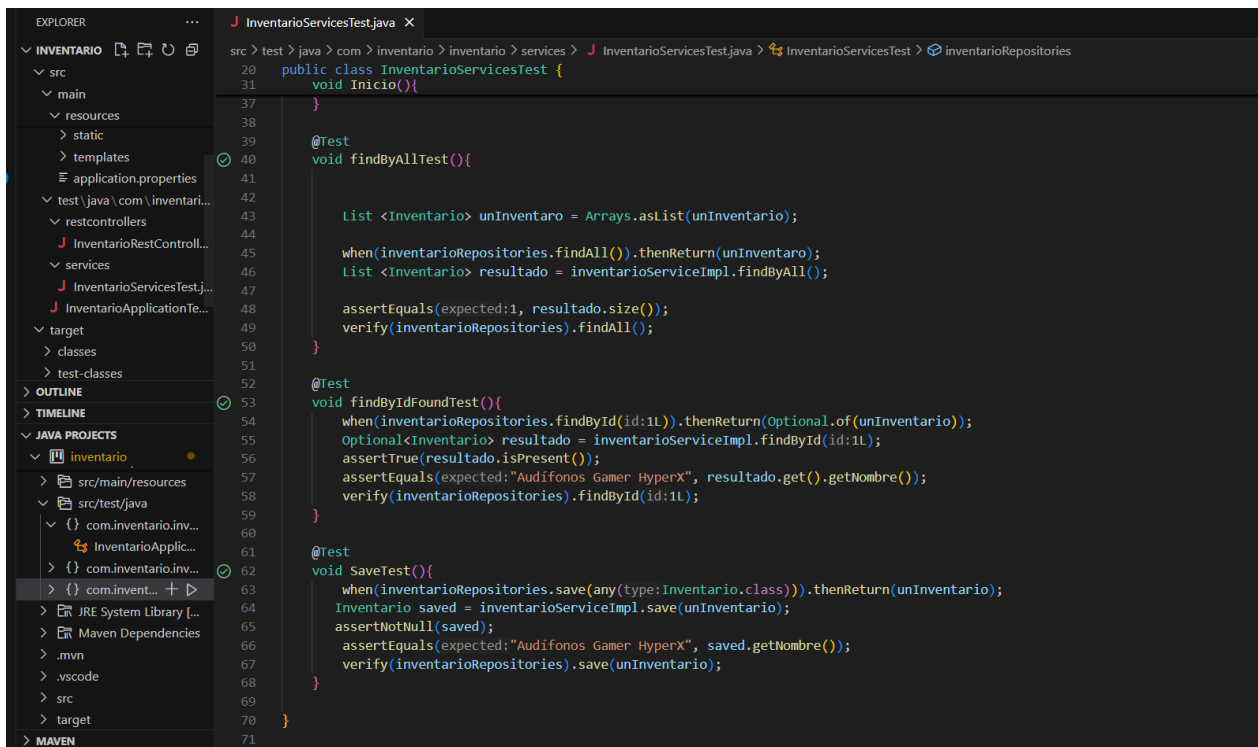
Codificación de Prueba Unitaria a Services:



```

1  package com.inventario.inventario.services;
2
3  import org.springframework.boot.test.context.SpringBootTest;
4  import static org.junit.jupiter.api.Assertions.assertEquals;
5  import static org.junit.jupiter.api.Assertions.assertNotNull;
6  import static org.junit.jupiter.api.Assertions.assertTrue;
7  import static org.mockito.ArgumentMatchers.any;
8  import static org.mockito.Mockito.verify;
9  import static org.mockito.Mockito.when;
10 import org.junit.jupiter.api.BeforeEach;
11 import org.junit.jupiter.api.Test;
12 import org.mockito.*;
13 import java.util.*;
14
15
16 import com.inventario.inventario.entities.Inventario;
17 import com.inventario.inventario.repositories.InventarioRepositories;
18
19 @SpringBootTest
20 public class InventarioServicesTest {
21
22     @Mock
23     private InventarioRepositories inventarioRepositories;
24
25     @InjectMocks
26     private InventarioServiceImpl inventarioServiceImpl;
27
28     private Inventario unInventario;
29
30     @BeforeEach
31     void Inicio(){
32         MockitoAnnotations.openMocks(this);
33         unInventario = new Inventario();
34         unInventario.setId(1L);
35         unInventario.setNombre(nombre: "Audifonos Gamer HyperX");
36
37     }

```



```

20 public class InventarioServicesTest {
31     void Inicio(){
37     }
38
39     @Test
40     void findByAllTest(){
41
42         List<Inventario> unInventario = Arrays.asList(unInventario);
43
44         when(inventarioRepositories.findAll()).thenReturn(unInventario);
45         List<Inventario> resultado = inventarioServiceImpl.findByAll();
46
47         assertEquals(expected:1, resultado.size());
48         verify(inventarioRepositories).findAll();
49     }
50
51     @Test
52     void findByIdFoundTest(){
53
54         when(inventarioRepositories.findById(id:1L)).thenReturn(Optional.of(unInventario));
55         Optional<Inventario> resultado = inventarioServiceImpl.findById(id:1L);
56         assertTrue(resultado.isPresent());
57         assertEquals(expected:"Audifonos Gamer HyperX", resultado.get().getNombre());
58         verify(inventarioRepositories).findById(id:1L);
59     }
60
61     @Test
62     void SaveTest(){
63
64         when(inventarioRepositories.save(any(type:Inventario.class))).thenReturn(unInventario);
65         Inventario saved = inventarioServiceImpl.save(unInventario);
66         assertNotNull(saved);
67         assertEquals(expected:"Audifonos Gamer HyperX", saved.getNombre());
68         verify(inventarioRepositories).save(unInventario);
69     }
70 }
71

```

Codificación de Pruebas de integración:

```

EXPLORER
INVENTARIO
  src
    main
      resources
        static
        templates
      application.properties
    test \java \com \inventari...
      restcontrollers
        J InventarioRestControll...
      services
        J InventarioServicesTest.j...
        J InventarioApplicationTe...
    target
      classes
      test-classes
  OUTLINE
  TIMELINE
  JAVA PROJECTS
    inventario
      src/main/resources
      src/test/java
        {} com.inventario.inv...
        InventarioApplic...
        {} com.inventario.inv...
        Inventario... + ▸
      {} com.invent... + ▸
      JRE System Library [...
      Maven Dependencies
      .mvn
      .vscode
      src
  MAVEN
    Spring Boot-InventarioApplications\inventario> (Inventario)
    Java: Ready

src > test > java > com > inventario > inventario > restcontrollers > J InventarioRestControllerTest.java > InventarioRestControllerTest > mockMvc

1 package com.inventario.inventario.restcontrollers;
2
3 import java.util.List;
4 import java.util.Optional;
5 import static org.junit.jupiter.api.Assertions.fail;
6
7 import org.junit.jupiter.api.Test;
8 import org.springframework.beans.factory.annotation.Autowired;
9 import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
10 import org.springframework.boot.test.context.SpringBootTest;
11 import org.springframework.http.MediaType;
12 import org.springframework.test.context.bean.override.mockito.MockitoBean;
13 import org.springframework.test.web.servlet.MockMvc;
14
15
16 import com.fasterxml.jackson.databind.ObjectMapper;
17 import com.inventario.inventario.entities.Inventario;
18 import com.inventario.inventario.services.InventarioServiceImpl;
19
20 import static org.mockito.Mockito.when;
21 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
22 import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
23 import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
24 import static org.mockito.ArgumentMatchers.any;
25
26 @SpringBootTest
27 @AutoConfigureMockMvc
28 public class InventarioRestControllerTest {
29
30     @Autowired
31     private MockMvc mockMvc;
32
33     @MockitoBean
34     private InventarioServiceImpl inventarioServiceImpl; //mocks, objeto simulado
35
36     @Autowired
37     private ObjectMapper objectMapper; //atributo para gestionar el método save

```

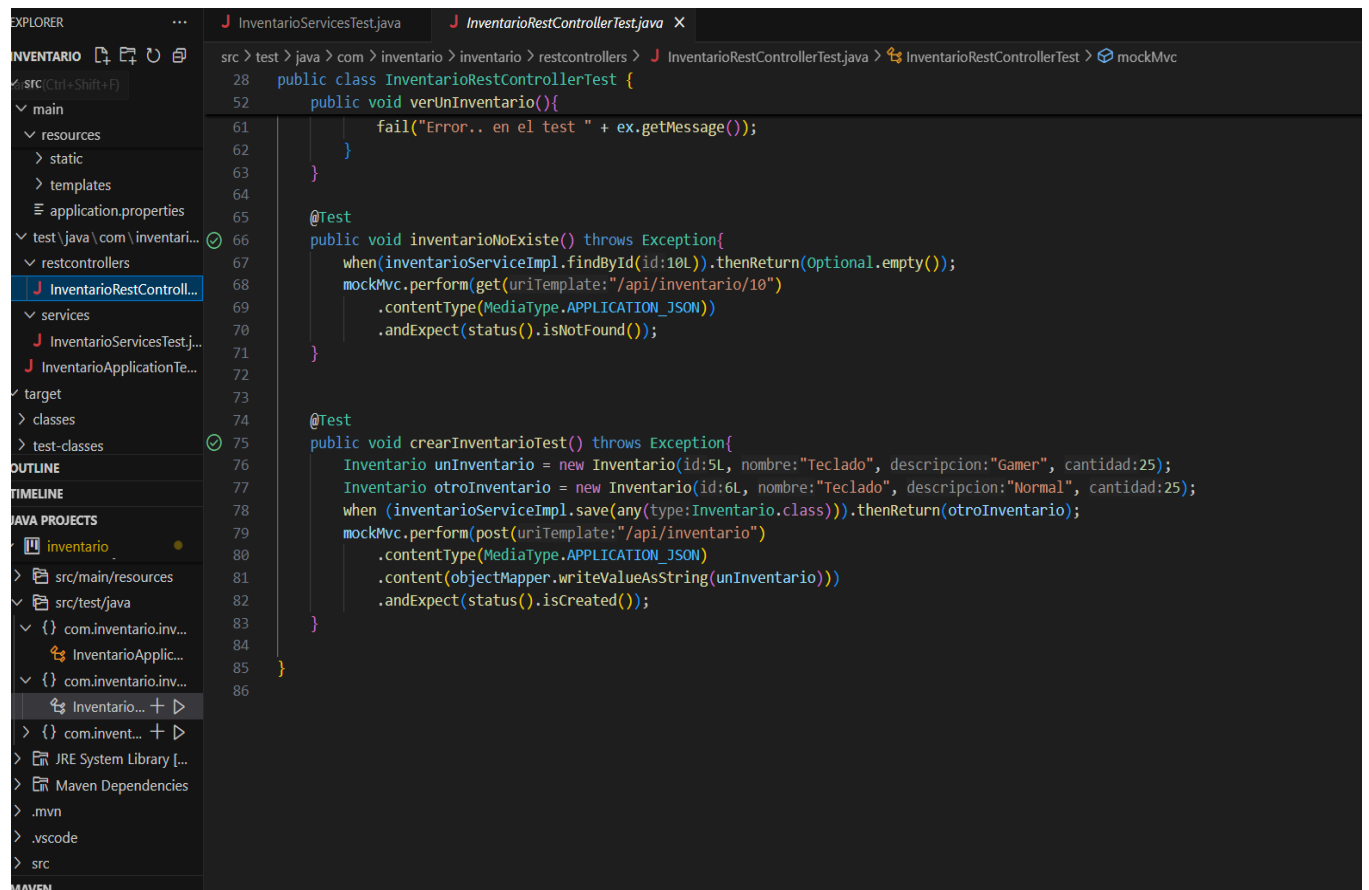
```

EXPLORER
INVENTARIO
  src
    main
      resources
        static
        templates
      application.properties
    test \java \com \inventari...
      restcontrollers
        J InventarioRestControll...
      services
        J InventarioServicesTest.j...
        J InventarioApplicationTe...
    target
      classes
      test-classes
  OUTLINE
  TIMELINE
  JAVA PROJECTS
    inventario
      src/main/resources
      src/test/java
        {} com.inventario.inv...
        InventarioApplic...
        {} com.inventario.inv...
        Inventario... + ▸
      {} com.invent... + ▸
      JRE System Library [...
      Maven Dependencies
      .mvn
      .vscode
      src
  MAVEN
    Spring Boot-InventarioApplications\inventario> (Inventario)
    Java: Ready

src > test > java > com > inventario > inventario > restcontrollers > J InventarioRestControllerTest.java > InventarioRestControllerTest > mockMvc

28 public class InventarioRestControllerTest {
29     @Autowired
30     private ObjectMapper objectMapper; //atributo para gestionar el método save
31
32     List<Inventario> listaInventarios;
33
34     @Test
35     public void verInventarios() throws Exception{
36         when(inventarioServiceImpl.findAll()).thenReturn(listaInventarios);
37         mockMvc.perform(get(uriTemplate:"/api/inventario")
38             .contentType(MediaType.APPLICATION_JSON))
39             .andExpect(status().isOk());
40     }
41
42     @Test
43     public void verUnInventario(){
44         Inventario unInventario = new Inventario (id:4L, nombre:"Audifonos Gamer HyperX", descripcion:"Audifonos con sonido envolvente 7.1 y mic
45         try{
46             when(inventarioServiceImpl.findById(id:4L)).thenReturn(Optional.of(unInventario));
47             mockMvc.perform(get(uriTemplate:"/api/inventario/4")
48                 .contentType(MediaType.APPLICATION_JSON))
49                 .andExpect(status().isOk());
50         }catch (Exception ex){
51             fail("Error.. en el test " + ex.getMessage());
52         }
53     }
54
55     @Test
56     public void inventarioNoExiste() throws Exception{
57         when(inventarioServiceImpl.findById(id:10L)).thenReturn(Optional.empty());
58         mockMvc.perform(get(uriTemplate:"/api/inventario/10")
59             .contentType(MediaType.APPLICATION_JSON))
60             .andExpect(status().isNotFound());
61     }
62 }

```



Tecnologías y Frameworks utilizados:

1. JUnit 5 (Jupiter)

Tipo de pruebas: Unitarias y de integración.

Uso:

JUnit es el framework principal para la escritura y ejecución de pruebas automatizadas en Java.

Anotaciones clave:

- **@Test** Indica un método de prueba.
- **@BeforeEach**: Método que se ejecuta antes de cada test.
- **@AfterEach**: Método que se ejecuta después de cada test (opcional).

2. Mockito

Tipo de pruebas: Principalmente **unitarias**

Uso:

Mockito se utiliza para **simular (mockear)** el comportamiento de componentes externos o dependencias de las clases que se están probando. Por ejemplo, en las pruebas unitarias de tu servicio, simulas la respuesta del repositorio.

Anotaciones clave:

- **@Mock:** Crea un objeto simulado (mock).
- **@InjectMocks:** Inyecta los mocks en la clase a testear.
- **when(...).thenReturn(...):** Define comportamientos simulados.
- **verify(...):** Verifica que un método fue llamado.

3.Spring Boot Test (spring-boot-starter-test)

Tipo de pruebas: Integración y unitarias (con configuración Spring)

Uso:

Spring Boot Test proporciona un entorno completo para pruebas con contexto de Spring cargado. Es ideal para pruebas de controladores (**@RestController**) y servicios con dependencias reales o simuladas.

Anotaciones clave:

- **@SpringBootTest:** Carga todo el contexto de la aplicación (ideal para pruebas de integración).
- **@WebMvcTest:** Carga solo los controladores y componentes de la capa web.
- **@AutoConfigureMockMvc:** Permite usar MockMvc para simular peticiones HTTP.
- **MockMvc:** Se usa para hacer peticiones simuladas (GET, POST, etc.) a los endpoints.

4. MockMvc

Tipo de pruebas: Integración (REST)

Uso:

MockMvc permite realizar llamadas HTTP simuladas a tus endpoints REST para verificar su funcionamiento. Es ideal para probar los **@RestController** sin arrancar el servidor completo.

5. AssertJ / Assertions (de JUnit)

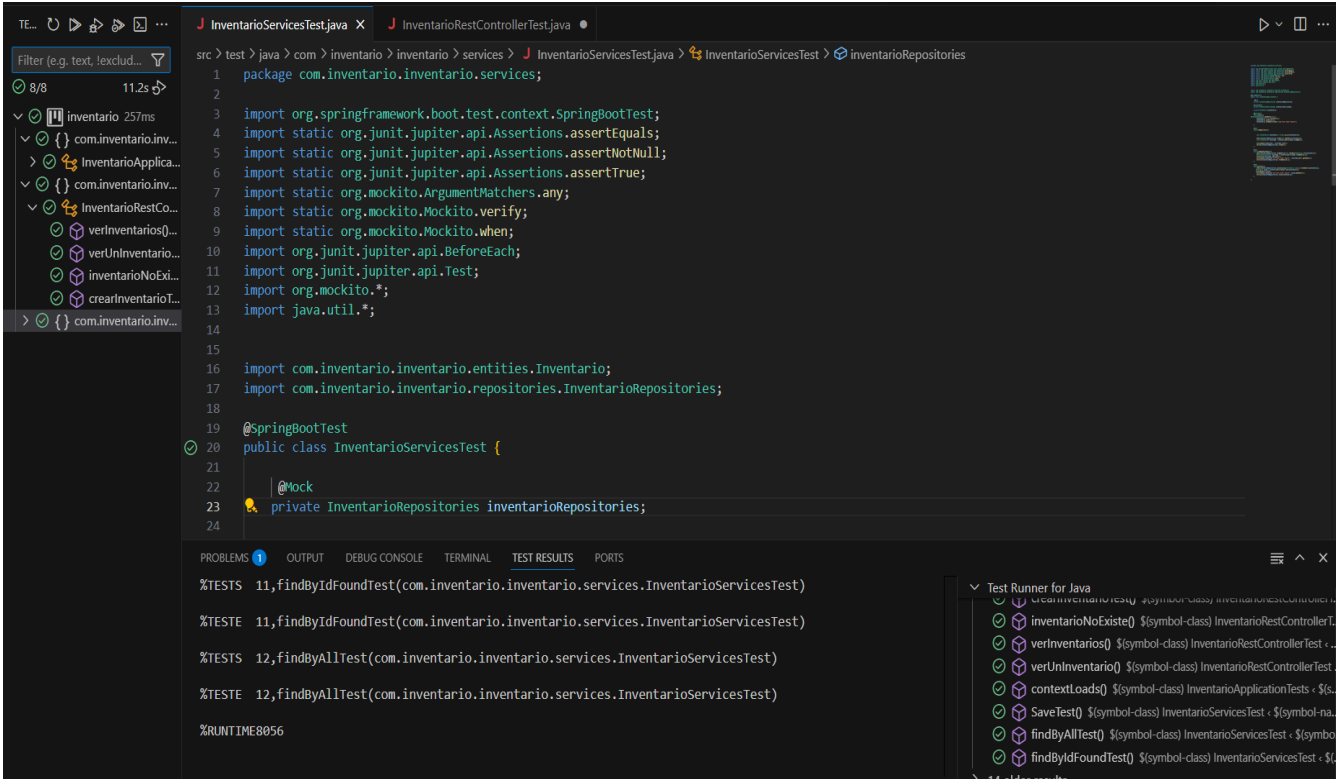
Tipo de pruebas: Ambas

Uso:

Utilizado para hacer afirmaciones (asserts) y verificar que los valores obtenidos en la prueba son los esperados.

❖ Pruebas Unitarias.

Prueba a Services:



The screenshot shows an IDE with a Java test file named `InventarioServicesTest.java` and its execution results. The code in the editor is as follows:

```

1 package com.inventario.inventario.services;
2
3 import org.springframework.boot.test.context.SpringBootTest;
4 import static org.junit.jupiter.api.Assertions.assertEquals;
5 import static org.junit.jupiter.api.Assertions.assertNotNull;
6 import static org.junit.jupiter.api.Assertions.assertTrue;
7 import static org.mockito.ArgumentMatchers.any;
8 import static org.mockito.Mockito.verify;
9 import static org.mockito.Mockito.when;
10 import org.junit.jupiter.api.BeforeEach;
11 import org.junit.jupiter.api.Test;
12 import org.mockito.*;
13 import java.util.*;
14
15
16 import com.inventario.inventario.entities.Inventario;
17 import com.inventario.inventario.repositories.InventarioRepositories;
18
19 @SpringBootTest
20 public class InventarioServicesTest {
21
22     @Mock
23     private InventarioRepositories inventarioRepositories;
24

```

The Test Results panel shows the following tests passed:

- `11, findByIdFoundTest(com.inventario.inventario.services.InventarioServicesTest)`
- `12, findByIdAllTest(com.inventario.inventario.services.InventarioServicesTest)`

The total runtime is 8056ms. The Test Runner for Java shows a list of tests with their status and the class they belong to.


```

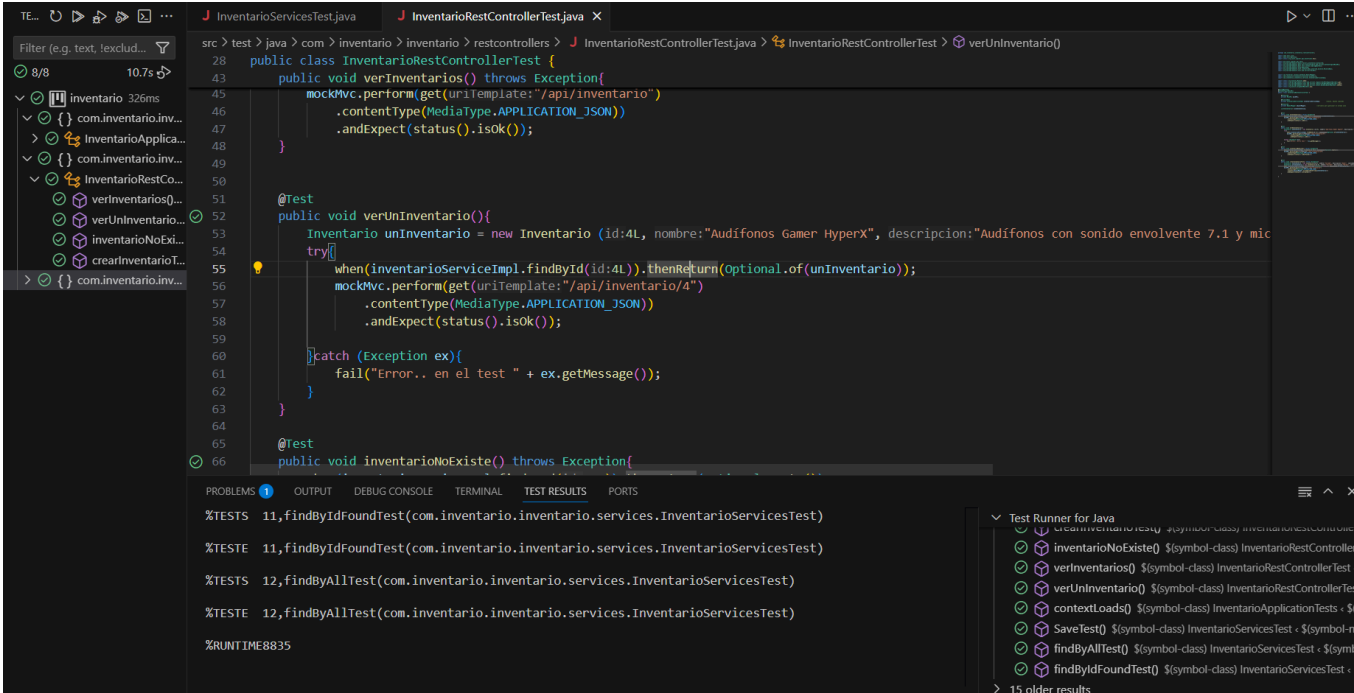
TE... 8/8 11.2s
src > test > java > com > inventario > inventario > services > J InventarioServicesTest.java > InventarioServicesTest > inventarioRepositories

20 public class InventarioServicesTest {
31 void Inicio(){
37 }
38
39 @Test
40 void findByAllTest(){
41
42     List <Inventario> unInventario = Arrays.asList(unInventario);
43
44     when(inventarioRepositories.findAll()).thenReturn(unInventario);
45     List <Inventario> resultado = inventarioServiceImpl.findByAll();
46
47     assertEquals(expected:1, resultado.size());
48     verify(inventarioRepositories).findAll();
49 }
50
51
52 @Test
53 void findByIdFoundTest(){
54     when(inventarioRepositories.findById(id:1L)).thenReturn(Optional.of(unInventario));
55     Optional<Inventario> resultado = inventarioServiceImpl.findById(id:1L);
56     assertTrue(resultado.isPresent());
57     assertEquals(expected:"Audifonos Gamer HyperX", resultado.get().getNombre());
58     verify(inventarioRepositories).findById(id:1L);
59 }
60
61
62 @Test
63 void SaveTest(){
64     when(inventarioRepositories.save(any(type:Inventario.class))).thenReturn(unInventario);
65     Inventario saved = inventarioServiceImpl.save(unInventario);
66     assertNotNull(saved);
67     assertEquals(expected:"Audifonos Gamer HyperX", saved.getNombre());
68     verify(inventarioRepositories).save(unInventario);
69 }
70
71

```

❖ Pruebas de integración

Prueba a RestController:



```
src > test > java > com > inventario > inventario > restcontrollers > J InventarioRestControllerTest.java > InventarioRestControllerTest > verUnInventario()
28 public class InventarioRestControllerTest {
43 public void verInventarios() throws Exception{
45     mockMvc.perform(get(uriTemplate:"/api/inventario")
46         .contentType(MediaType.APPLICATION_JSON))
47         .andExpect(status().isOk());
48 }
50
51 @Test
52 public void verUnInventario(){
53     Inventario unInventario = new Inventario (id:4L, nombre:"Audifonos Gamer HyperX", descripcion:"Audifonos con sonido envolvente 7.1 y mic
54     try{
55         when(inventarioServiceImpl.findById(id:4L)).thenReturn(Optional.of(unInventario));
56         mockMvc.perform(get(uriTemplate:"/api/inventario/4")
57             .contentType(MediaType.APPLICATION_JSON))
58             .andExpect(status().isOk());
59     }
60 }catch (Exception ex){
61     fail("Error.. en el test " + ex.getMessage());
62 }
63 }
64
65 @Test
66 public void inventarioNoExiste() throws Exception{
67
68 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS PORTS

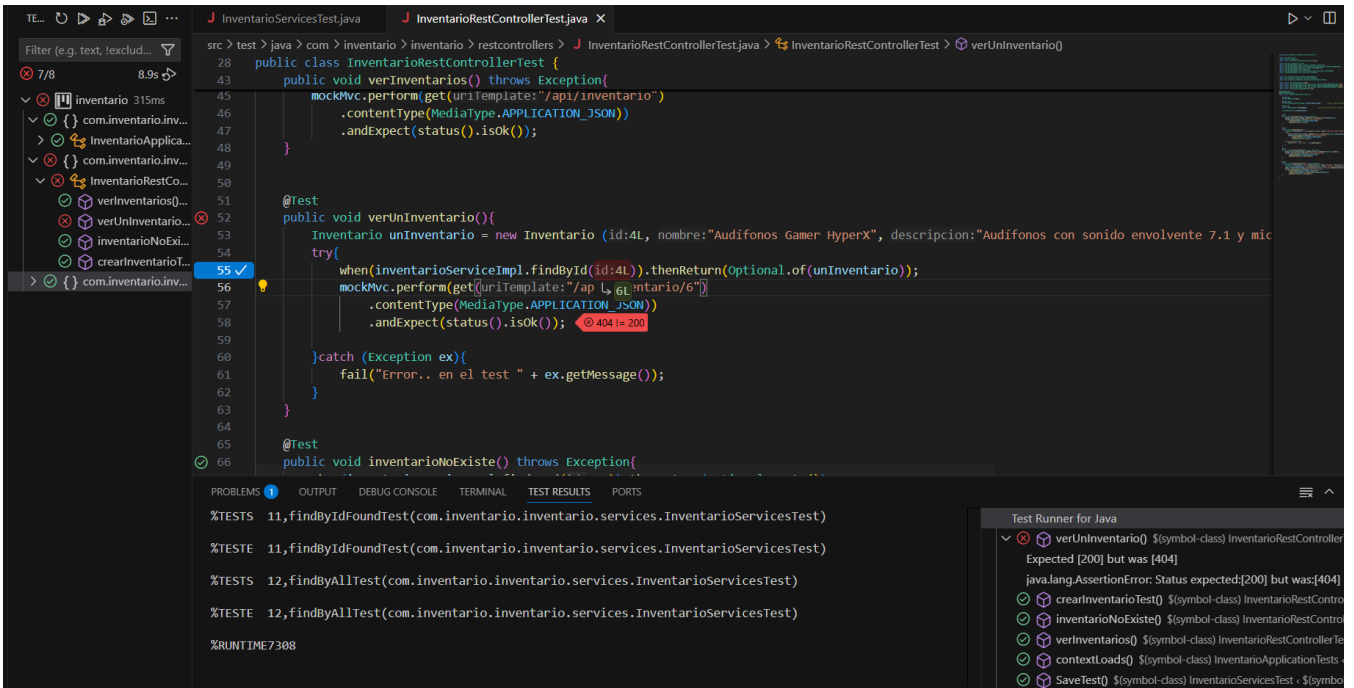
```
%TESTS 11,findByIdFoundTest(com.inventario.inventario.services.InventarioServicesTest)
%TESTE 11,findByIdFoundTest(com.inventario.inventario.services.InventarioServicesTest)
%TESTS 12,findAllTest(com.inventario.inventario.services.InventarioServicesTest)
%TESTE 12,findAllTest(com.inventario.inventario.services.InventarioServicesTest)
%RUNTIME8835
```

Test Runner for Java

- ✓ inventarioNoExiste() \$(symbol-class) InventarioRestControllerTest
- ✓ verInventarios() \$(symbol-class) InventarioRestControllerTest
- ✓ verUnInventario() \$(symbol-class) InventarioRestControllerTest
- ✓ contextLoads() \$(symbol-class) InventarioApplicationTests
- ✓ SaveTest() \$(symbol-class) InventarioServicesTest
- ✓ findByIdAllTest() \$(symbol-class) InventarioServicesTest
- ✓ findByIdFoundTest() \$(symbol-class) InventarioServicesTest

> 15 older results

Pruebas de integración funcionando correctamente.



```
src > test > java > com > inventario > inventario > restcontrollers > J InventarioRestControllerTest.java > InventarioRestControllerTest > verUnInventario()
28 public class InventarioRestControllerTest {
43 public void verInventarios() throws Exception{
45     mockMvc.perform(get(uriTemplate:"/api/inventario")
46         .contentType(MediaType.APPLICATION_JSON))
47         .andExpect(status().isOk());
48 }
50
51 @Test
52 public void verUnInventario(){
53     Inventario unInventario = new Inventario (id:4L, nombre:"Audifonos Gamer HyperX", descripcion:"Audifonos con sonido envolvente 7.1 y mic
54     try{
55         when(inventarioServiceImpl.findById(id:4L)).thenReturn(Optional.of(unInventario));
56         mockMvc.perform(get(uriTemplate:"/api/inventario/6")
57             .contentType(MediaType.APPLICATION_JSON))
58             .andExpect(status().isOk());
59     }
60 }catch (Exception ex){
61     fail("Error.. en el test " + ex.getMessage());
62 }
63 }
64
65 @Test
66 public void inventarioNoExiste() throws Exception{
67
68 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL TEST RESULTS PORTS

```
%TESTS 11,findByIdFoundTest(com.inventario.inventario.services.InventarioServicesTest)
%TESTE 11,findByIdFoundTest(com.inventario.inventario.services.InventarioServicesTest)
%TESTS 12,findAllTest(com.inventario.inventario.services.InventarioServicesTest)
%TESTE 12,findAllTest(com.inventario.inventario.services.InventarioServicesTest)
%RUNTIME7308
```

Test Runner for Java

- ✗ verUnInventario() \$(symbol-class) InventarioRestControllerTest
- Expected [200] but was [404]
- java.lang.AssertionError: Status expected:[200] but was:[404]
- ✓ crearInventarioTest() \$(symbol-class) InventarioRestControllerTest
- ✓ inventarioNoExiste() \$(symbol-class) InventarioRestControllerTest
- ✓ verInventarios() \$(symbol-class) InventarioRestControllerTest
- ✓ contextLoads() \$(symbol-class) InventarioApplicationTests
- ✓ SaveTest() \$(symbol-class) InventarioServicesTest

Cambiamos de id a uno que no existe para hacer que el test nos arroje 404 y comprobar que funciona.

OAS

Codificación de OAS:

```

J InventarioServicesTest.java × J InventarioRestControllerTest.java ● J InventarioRestController.java ×
src > main > java > com > inventario > inventario > restcontrollers > J InventarioRestController.java > ...
1 package com.inventario.inventario.restcontrollers;
2
3 import java.util.List;
4 import java.util.Optional;
5
6 import org.springframework.beans.factory.annotation.Autowired;
7 import org.springframework.http.HttpStatus;
8 import org.springframework.http.ResponseEntity;
9 import org.springframework.web.bind.annotation.DeleteMapping;
10 import org.springframework.web.bind.annotation.GetMapping;
11 import org.springframework.web.bind.annotation.PathVariable;
12 import org.springframework.web.bind.annotation.PostMapping;
13 import org.springframework.web.bind.annotation.PutMapping;
14 import org.springframework.web.bind.annotation.RequestBody;
15 import org.springframework.web.bind.annotation.RequestMapping;
16 import org.springframework.web.bind.annotation.RestController;
17
18 import com.inventario.inventario.entities.Inventario;
19 import com.inventario.inventario.services.InventarioServices;
20 import io.swagger.v3.oas.annotations.Operation;
21 import io.swagger.v3.oas.annotations.media.Content;
22 import io.swagger.v3.oas.annotations.media.Schema;
23 import io.swagger.v3.oas.annotations.responses.ApiResponse;
24 import io.swagger.v3.oas.annotations.responses.ApiResponses;
25 import io.swagger.v3.oas.annotations.tags.Tag;
26
27 @Tag(name = "Inventario", description = "Operaciones relacionadas con el inventario")
28 @RestController
29 @RequestMapping("/api/inventario")
30 public class InventarioRestController {
31     @Autowired
32     private InventarioServices inventarioServices;
33
34     @Operation(summary = "Obtener lista de productos", description = "Devuelve todos los productos disponibles")
35     @ApiResponse(responseCode = "200", description = "Lista de productos retornada correctamente",
36                 content = @Content(mediaType = "application/json",
37                                   schema = @Schema(implementation = Inventario.class)))

```

añadimos los siguientes imports de swagger.

```

J InventarioServicesTest.java  J InventarioRestControllerTest.java  J InventarioRestController.java X
src > main > java > com > inventario > inventario > restcontrollers > J InventarioRestController.java > ...
25 import io.swagger.v3.oas.annotations.tags.Tag;
26
27 @Tag(name = "Inventario", description = "Operaciones relacionadas con el inventario")
28 @RestController
29 @RequestMapping("/api/inventario")
30 public class InventarioRestController {
31     @Autowired
32     private InventarioServices inventarioServices;
33
34     @Operation(summary = "Obtener lista de productos", description = "Devuelve todos los productos disponibles")
35     @ApiResponse(responseCode = "200", description = "Lista de productos retornada correctamente",
36                 content = @Content(mediaType = "application/json",
37                                   schema = @Schema(implementation = Inventario.class)))
38     @GetMapping
39     public List<Inventario> verInventarios(){
40         return (List<Inventario>) inventarioServices.findAll();
41     }
42     @Operation(summary = "Obtener producto por ID", description = "Obtiene el detalle de un producto específico")
43     @ApiResponses(value = {
44         @ApiResponse(responseCode = "200", description = "Producto encontrado",
45                     content = @Content(mediaType = "application/json", schema = @Schema(implementation = Inventario.class))),
46         @ApiResponse(responseCode = "404", description = "Producto no encontrado")
47     })
48     @GetMapping("/{id}")
49     public ResponseEntity<?> verDetalle(@PathVariable Long id){
50         Optional<Inventario> inventarioOptional = inventarioServices.findById(id);
51         if (inventarioOptional.isPresent()){
52             return ResponseEntity.ok(inventarioOptional.orElseThrow());
53         }
54         return ResponseEntity.notFound().build();
55     }
56     @Operation(summary = "Crear un nuevo producto", description = "Crea un producto con los datos proporcionados")
57     @ApiResponse(responseCode = "201", description = "Producto creado correctamente",
58                 content = @Content(mediaType = "application/json", schema = @Schema(implementation = Inventario.class)))
59     @PostMapping
60     public ResponseEntity<Inventario> crear(@RequestBody Inventario unInventario){
61         Application<inventario> (inventario)  Java: Ready
Ln 21, Col 27  Spaces: 4  U

```

```

J InventarioServicesTest.java  J InventarioRestControllerTest.java  J InventarioRestController.java X
src > main > java > com > inventario > inventario > restcontrollers > J InventarioRestController.java > ...
30 public class InventarioRestController {
60 @PostMapping
61 public ResponseEntity<Inventario> crear(@RequestBody Inventario unInventario){
62     return ResponseEntity.status(HttpStatus.CREATED).body(inventarioServices.save(unInventario));
63 }
64
65 @Operation(summary = "Modificar producto por ID", description = "Modificar información de un producto")
66 @ApiResponses(value = {
67     @ApiResponse(responseCode = "200", description = "Producto modificado",
68                 content = @Content(mediaType = "application/json", schema = @Schema(implementation = Inventario.class))),
69     @ApiResponse(responseCode = "404", description = "Producto no encontrado")
70 })
71 @PutMapping("/{id}")
72 public ResponseEntity<?> modificarInventario(@PathVariable Long id, @RequestBody Inventario unInventario){
73     Optional<Inventario> inventarioOptional = inventarioServices.findById(id);
74     if (inventarioOptional.isPresent()){
75         Inventario inventarioExiste = inventarioOptional.get();
76         inventarioExiste.setNombre(unInventario.getNombre());
77
78         Inventario inventarioModificado = inventarioServices.save(inventarioExiste);
79         return ResponseEntity.ok(inventarioModificado);
80     }
81     return ResponseEntity.notFound().build();
82 }
83
84 @Operation(summary = "Eliminar producto por ID", description = "Elimina un objeto específico")
85 @ApiResponses(value = {
86     @ApiResponse(responseCode = "200", description = "Producto eliminado",
87                 content = @Content(mediaType = "application/json", schema = @Schema(implementation = Inventario.class))),
88     @ApiResponse(responseCode = "404", description = "Producto no encontrado")
89 })
90 @DeleteMapping("/{id}")
91 public ResponseEntity<?> eliminarInventario(@PathVariable Long id){
92     Inventario unInventario = new Inventario();
93     unInventario.setId(id);
94     Optional<Inventario> inventarioOptional = inventarioServices.delete(unInventario);
95     if (inventarioOptional.isPresent()){
6
Application<inventario> (inventario)  Java: Ready
Ln 21, Col 27  Spaces: 4  UTF-8

```

En cada método agregamos anotaciones de swagger para la documentación.

Documentacion Swagger:

Lista de Sucursales

Swagger UI

localhost:8080/swagger-ui/index.html#/Inventario/verDetalle

Servers

http://localhost:8080 - Generated server url

Inventario

Operaciones relacionadas con el inventario

GET

/api/inventario/{id}

Obtener producto por ID

Obtiene el detalle de un producto específico

Parameters

Cancel

Name	Description
id * <small>required</small>	
<small>integer(\$int64)</small>	4
<small>(path)</small>	

Execute

Clear

Responses

Curl

Lista de Sucursales

Swagger UI

localhost:8080/swagger-ui/index.html#/Inventario/verDetalle

Curl

curl -X 'GET' \n'http://localhost:8080/api/inventario/4' \n-H 'accept: application/json'

Request URL

http://localhost:8080/api/inventario/4

Server response

Code	Details
200	<div><div>Response body</div><div><pre>{ "id": 4, "nombre": "Audífonos Gamer HyperX", "descripcion": "Audífonos con sonido envolvente 7.1 y micrófono con cancelación de ruido", "cantidad": 0 }</pre></div><div>Download</div></div> <div><div>Response headers</div><div><pre>connection: keep-alive content-type: application/json date: Tue, 24 Jun 2025 09:38:42 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre></div></div>

Responses

Code	Description	Links
200	Producto encontrado	No links

Media type

application/json

Controls Accept header.

Example Value | Schema

Lista de Sucursales

Swagger UI

localhost:8080/swagger-ui/index.html#/Inventario/verDetalle

Responses

Code	Description	Links
200	Producto encontrado	No links
Media type application/json		
Controls Accept header		
Example Value Schema		
<pre>{ "id": 0, "nombre": "string", "descripcion": "string", "cantidad": 0}</pre>		
404	Producto no encontrado	No links
Media type */*		
Example Value Schema		
<pre>{}</pre>		

PUT

/api/inventario/{id}

Modificar producto por ID

DELETE

/api/inventario/{id}

Eliminar producto por ID

GET

/api/inventario

Obtener lista de productos

Aca vemos si es que esta funcionando correctamente el método buscar por id.

Swagger UI

localhost:8080/swagger-ui/index.html#/Inventario/verDetalle

PUT /api/inventario/{id} Modificar producto por ID

Modificar información de un producto

Parameters Cancel Reset

Name	Description
id * required integer(int64) (path)	<input type="text" value="4"/>

Request body required application/json

```
{
  "id": 4,
  "nombre": "Audifonos Gamer HyperX",
  "descripcion": "Audifonos con sonido envolvente 7.1 y micrófono con cancelación de ruido",
  "cantidad": 5
}
```

Lista de Sucursales

Swagger UI

localhost:8080/swagger-ui/index.html#/Inventario/verDetalle

Curl

```
curl -X 'PUT' \
  'http://localhost:8080/api/inventario/4' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "id": 4,
    "nombre": "Audífonos Gamer HyperX",
    "descripcion": "Audífonos con sonido envolvente 7.1 y micrófono con cancelación de ruido",
    "cantidad": 5
  }'
```

Request URL

http://localhost:8080/api/inventario/4

Server response

Code

Details

200

Response body

```
{
  "id": 4,
  "nombre": "Audífonos Gamer HyperX",
  "descripcion": "Audífonos con sonido envolvente 7.1 y micrófono con cancelación de ruido",
  "cantidad": 0
}
```

Download

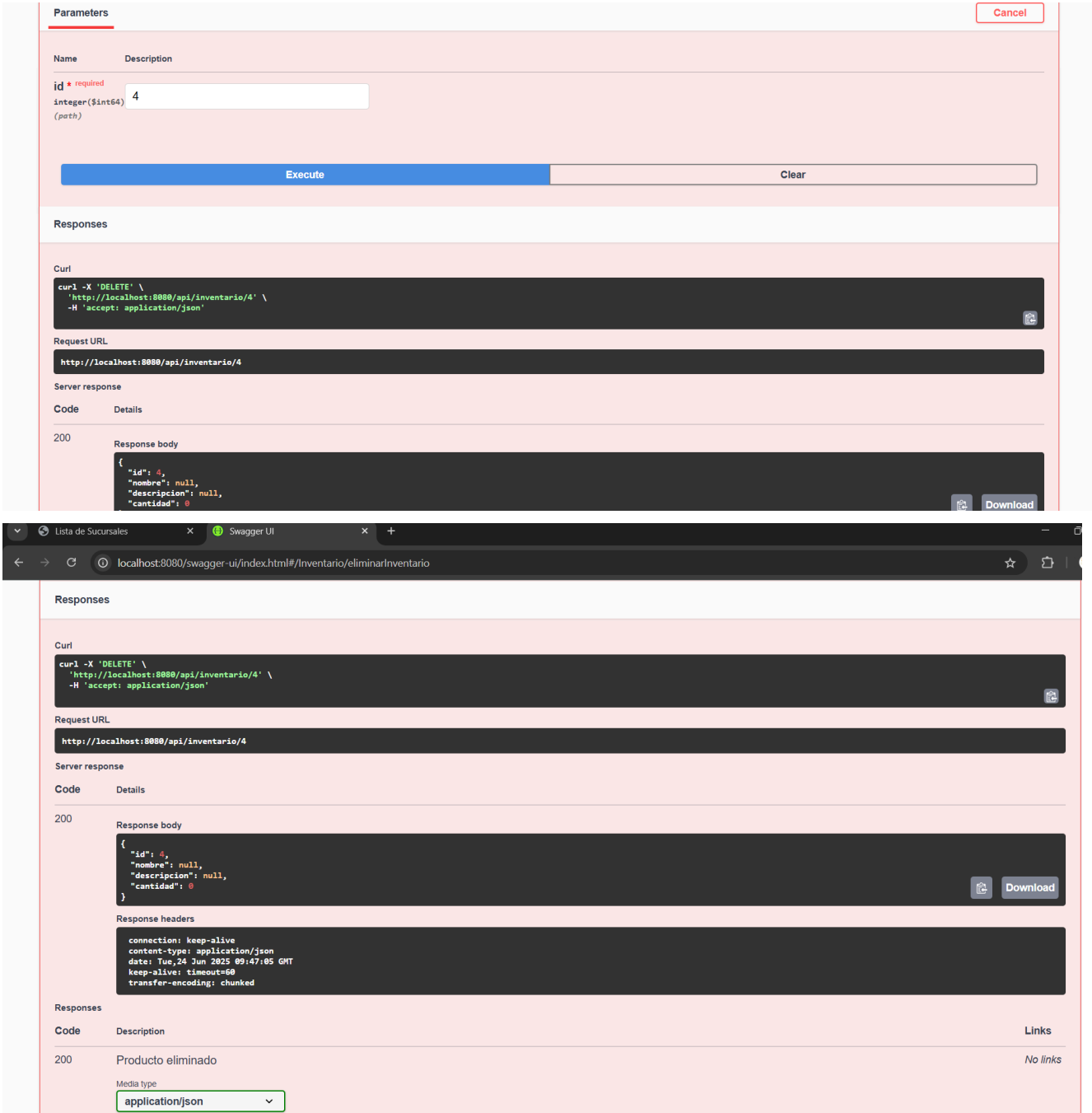
Response headers

```
connection: keep-alive
content-type: application/json
date: Tue, 24 Jun 2025 09:43:35 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code	Description	Links
200	Producto modificado	No links

Aca se prueba el método Put, se cambia la cantidad de 0 a 5 y devuelve code 200.



The image shows two screenshots of the Swagger UI interface. The top screenshot displays the 'Parameters' section for a DELETE endpoint. A parameter named 'id' is defined as a required integer (path) with a value of 4. Below the parameters, there are 'Execute' and 'Clear' buttons. The 'Responses' section shows a 200 status code with a JSON response body: {"id": 4, "nombre": null, "descripcion": null, "cantidad": 0}. The bottom screenshot shows the same interface after the request has been executed. It displays the 'Curl' command, the 'Request URL' (http://localhost:8080/api/inventario/4), and the 'Server response' (200). The 'Response body' is the same JSON object. Below the response body, the 'Response headers' are listed: connection: keep-alive, content-type: application/json, date: Tue, 24 Jun 2025 09:47:05 GMT, keep-alive: timeout=60, and transfer-encoding: chunked. At the bottom, a table of responses shows a 200 status code with the description 'Producto eliminado' and a media type of 'application/json'.

Parameters

Name	Description
id ★ required	
integer(\$int64)	4
(path)	

Execute **Clear**

Responses

Curl

```
curl -X 'DELETE' \
'http://localhost:8080/api/inventario/4' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:8080/api/inventario/4
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "id": 4, "nombre": null, "descripcion": null, "cantidad": 0 }</pre> <p>Response headers</p> <pre>connection: keep-alive content-type: application/json date: Tue, 24 Jun 2025 09:47:05 GMT keep-alive: timeout=60 transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	Producto eliminado	No links

Media type: **application/json**

Se prueba el método Delete, se borra el producto con id 4 y devuelvo un code 200, con valores en null.

localhost:8080/swagger-ui/index.html#/Inventario/verInventarios

Request URL

http://localhost:8080/api/inventario

Server response

Code

Details

200

Response body

```
[
  {
    "id": 1,
    "nombre": "Teclado Mecánico RGB",
    "description": "Teclado retroiluminado con switches azules y 104 teclas anti-ghosting",
    "cantidad": 0
  },
  {
    "id": 2,
    "nombre": "Mouse Inalámbrico Logitech",
    "description": "Mouse ergonómico con conexión Bluetooth y sensor óptico de alta precisión",
    "cantidad": 0
  },
  {
    "id": 3,
    "nombre": "Monitor LED 24\"",
    "description": "Pantalla Full HD con tecnología antirreflejo y puerto HDMI",
    "cantidad": 0
  }
]
```

Download

Response headers

```
connection: keep-alive
content-type: application/json
date: Tue, 24 Jun 2025 09:49:55 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code

Description

Links

200

Lista de productos retornada correctamente

No links

Se prueba método Get el cual retorna todos los productos.

localhost:8080/swagger-ui/index.html#/Inventario/crear

Crea un producto con los datos proporcionados

Parameters

Cancel

Reset

No parameters

Request body required

application/json

```
{
  "nombre": "Taza",
  "descripcion": "Loza de vidrio",
  "cantidad": 1
}
```

Execute

Clear

Responses

Lista de Sucursales | Swagger UI

localhost:8080/swagger-ui/index.html#/Inventario/crear

Curl

```
curl -X 'POST' \
  'http://localhost:8080/api/inventario' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
    "nombre": "Taza",
    "descripcion": "Loza de vidrio",
    "cantidad": 1
  }'
```

Request URL

http://localhost:8080/api/inventario

Server response

Code

Details

201

Response body

```
{
  "id": 5,
  "nombre": "Taza",
  "descripcion": "Loza de vidrio",
  "cantidad": 1
}
```

Download

Response headers

```
connection: keep-alive
content-type: application/json
date: Tue, 24 Jun 2025 09:51:34 GMT
keep-alive: timeout=60
transfer-encoding: chunked
```

Responses

Code	Description	Links
201	Producto creado correctamente	No links

Media type

Se prueba el método Post, creamos un producto nuevo. y devuelve un code 201.

Tecnologías utilizadas:

Swagger/OpenAPI (springdoc-openapi)

- Es un Framework de documentación automática de APIs REST.
- ¿Cómo se usa aquí?:
 - @Tag: Agrupa los endpoints bajo una misma categoría.
 - @Operation: Describe el propósito del endpoint.
 - @ApiResponse, @ApiResponses: Detalla los posibles códigos de respuesta HTTP.
 - @Schema: Define el tipo de dato que devuelve el endpoint.

GITHUB

```
javie@GearSixth MINGW64 ~/OneDrive/Escritorio
$ git init
hint: Using 'master' as the name for the initial branch. This default branch name
hint: is subject to change. To configure the initial branch name to use in all
hint: of your new repositories, which will suppress this warning, call:
hint:
hint:   git config --global init.defaultBranch <name>
hint:
hint: Names commonly chosen instead of 'master' are 'main', 'trunk' and
hint: 'development'. The just-created branch can be renamed via this command:
hint:
hint:   git branch -m <name>
Initialized empty Git repository in C:/Users/javie/OneDrive/Escritorio/.git/

javie@GearSixth MINGW64 ~/OneDrive/Escritorio (master)
$ git config user.name DiegoNeira339

javie@GearSixth MINGW64 ~/OneDrive/Escritorio (master)
$ git config user.email diegoneiracardenas@gmail.com

javie@GearSixth MINGW64 ~/OneDrive/Escritorio (master)
$ git config user.password ghp_9qQkMOMB9Nr4XsTNeSieU3U3zrXA1E3Q7f49

javie@GearSixth MINGW64 ~/OneDrive/Escritorio (master)
$ git remote add origin https://github.com/DiegoNeira339/Exp3_Albornoz_Hormazabal_Neira.git

javie@GearSixth MINGW64 ~/OneDrive/Escritorio (master)
$ |
```

Inicializamos git, configuramos nuestras credenciales. Luego vinculamos el repositorio con el local con el comando `git remote add origin`.

Luego se agregaran las 3 carpetas de las experiencias con `git add .`
Luego hacer commit con `git commit -m`
y finalmente `git push -u origin master`.

Conclusión

Durante el desarrollo de este proyecto logramos implementar de manera efectiva los distintos microservicios que componen el sistema de Perfulandia, según nuestro diagrama de microservicios. A través de las pruebas realizadas, tanto unitarias como de integración, validamos el correcto funcionamiento de las conexiones entre servicios y las operaciones CRUD.

El uso de herramientas como Spring Boot, JPA, MySQL y Git facilitó una estructura modular, ordenada y escalable. Además, nos aseguramos de seguir buenas prácticas de desarrollo, como la separación por capas, la documentación con OpenAPI y el versionamiento del código con GitHub.

Gracias a este trabajo pudimos apreciar una de las etapas más importantes de un software, el testing, para comprobar que todo este funcionando correctamente y prevenir errores en el lanzamiento.