



PALMERSTON
TECH.

2025

EcoMarket SPA

2025

ANY CITY



WELCOME TO

INTRODUCCION

En esta fase se aplicaron pruebas unitarias, pruebas de integración y documentación de endpoints.

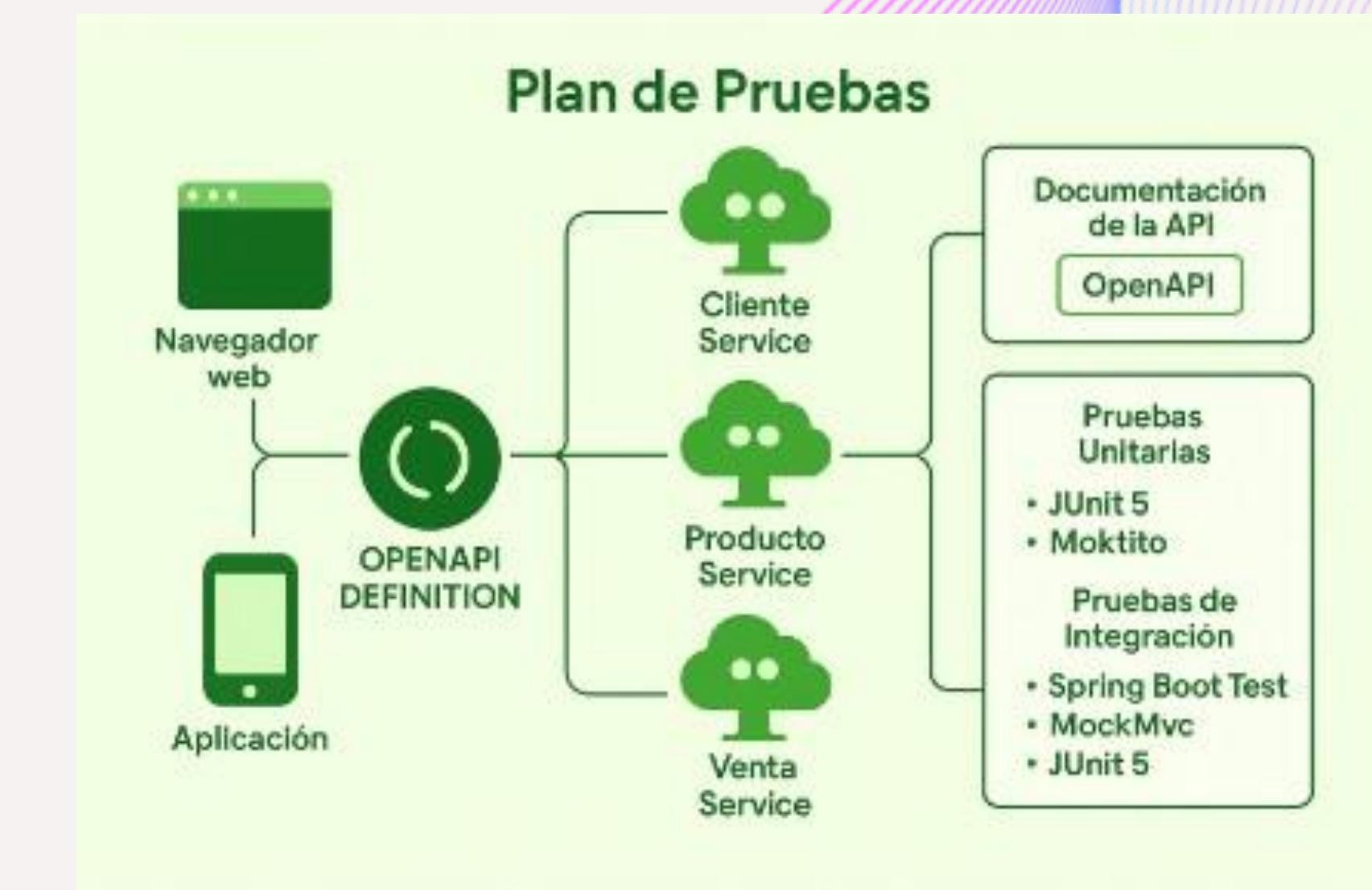
Se utilizaron JUnit 5, Mockito, Spring Boot Test, MockMvc y Swagger UI. El objetivo fue garantizar la calidad y el correcto funcionamiento del sistema.

ANY CITY



Arquitectura general

La arquitectura está dividida en tres microservicios independientes: USUARIO, PEDIDO y VENTA. Cada uno se conecta a la base de datos.

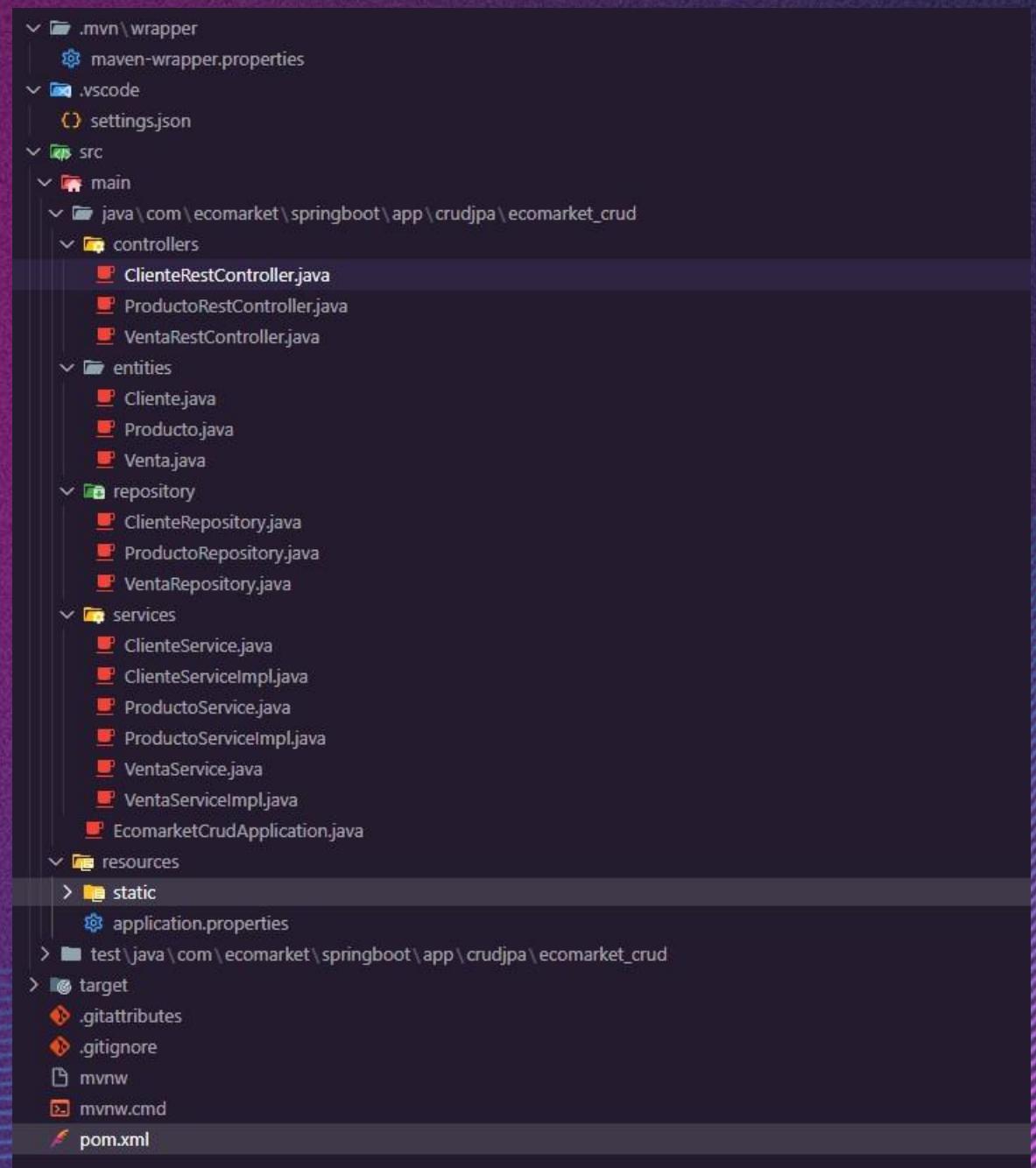




Estructura del proyecto

El proyecto sigue la estructura estándar de Spring Boot, con carpetas bien definidas:

- Controllers: exponen las rutas para las peticiones de los clientes.
- Services: contienen la lógica de negocio y las implementaciones de las interfaces.
- Repositories: permiten acceder y manipular los datos en la base de datos.
- Entities: representan las tablas y columnas en la base de datos.
- Además, utilizamos Maven como gestor de dependencias, y la configuración de la base de datos se encuentra en el archivo application.properties."





Servicio de Cliente

Este servicio gestiona la información de los clientes registrados en Eco Market SPA. La entidad Cliente incluye atributos como id, nombre, gmail y edad.

Se implementaron los siguientes endpoints CRUD: GET:

- obtener todos los clientes o uno en específico.
- POST: registrar un nuevo cliente.
- PUT: actualizar los datos de un cliente.
- DELETE: eliminar un cliente existente.
- Las pruebas se realizaron utilizando Postman, verificando que las respuestas sean correctas y que los datos se almacenen adecuadamente en la base de datos."

Servicio de Cliente

The image shows four Postman requests for a Client Service API, each with a different method and URL:

- POST localhost:8080/api/clientes**: A POST request with a JSON body containing a client's name, email, and age.
- GET localhost:8080/api/clientes/1**: A GET request to retrieve a client by ID.
- PUT localhost:8080/api/clientes/1**: A PUT request to update a client's information, including a modified name and email.
- DELETE localhost:8080/api/clientes/1**: A DELETE request to delete a client by ID.

Each request includes tabs for Body, Cookies, Headers (8), and Test Results, and a preview of the JSON response.

```
POST localhost:8080/api/clientes
Body:
1. {
  "nombre": "Juan Pérez",
  "gmail": "juanperez@gmail.com",
  "edad": 30
}

GET localhost:8080/api/clientes/1
Body:
1. Ctrl+Alt+P for Postbot

PUT localhost:8080/api/clientes/1
Body:
1. {
  "nombre": "Juan Pérez Modificado",
  "gmail": "juanperezmodificado@gmail.com",
  "edad": 31
}

DELETE localhost:8080/api/clientes/1
Body:
1. Ctrl+Alt+P for Postbot
```

```
{} JSON
1. {
  "id": 1,
  "nombre": "Juan Pérez",
  "gmail": "juanperez@gmail.com",
  "edad": 30
}

{} JSON
1. {
  "id": 1,
  "nombre": "Juan Pérez",
  "gmail": "juanperez@gmail.com",
  "edad": 30
}

{} JSON
1. {
  "id": 1,
  "nombre": "Juan Pérez Modificado",
  "gmail": "juanperezmodificado@gmail.com",
  "edad": 31
}

{} JSON
1. {
  "id": 1,
  "nombre": null,
  "gmail": null,
  "edad": 0
}
```

Servicio de Producto

El Servicio de Producto permite administrar el catálogo de productos ofrecidos por EcoMarket SPA.

La entidad Producto incluye atributos como id, nombre, descripción y precio.

Se desarrollaron las siguientes operaciones CRUD:

- GET: listar todos los productos o buscar uno específico.
- POST: agregar un nuevo producto.
- PUT: editar los datos de un producto.
- DELETE: eliminar un producto del catálogo.
- Todas las operaciones fueron validadas en Postman, garantizando que los datos se almacenen correctamente en la base de datos."

Servicio de Producto

The image shows a screenshot of a REST API testing tool with four panels, each representing a different HTTP request:

- POST** | <localhost:8080/api/productos>:
Body tab selected. Body content:

```
1 {  
2   "nombre": "Pan Integral",  
3   "descripcion": "Pan integral",  
4   "precio": 1500  
5 }
```

Body tab selected. Body content:

```
{ } JSON ▾
```

Body content:

```
1 {  
2   "id": 2,  
3   "nombre": "Pan Integral",  
4   "descripcion": "Pan integral",  
5   "precio": 1500  
6 }
```
- GET** | <localhost:8080/api/productos/2>:
Body tab selected. Body content:

```
1 Ctrl+Alt+P for Postbot
```

Body tab selected. Body content:

```
{ } JSON ▾
```

Body content:

```
1 {  
2   "id": 2,  
3   "nombre": "Pan Integral",  
4   "descripcion": "Pan integral",  
5   "precio": 1500  
6 }
```
- PUT** | <localhost:8080/api/productos/2>:
Body tab selected. Body content:

```
1 {  
2   "nombre": "Pan blanco",  
3   "descripcion": "Pan blanco",  
4   "precio": 1500  
5 }
```

Body tab selected. Body content:

```
{ } JSON ▾
```

Body content:

```
1 {  
2   "id": 2,  
3   "nombre": "Pan blanco",  
4   "descripcion": "Pan blanco",  
5   "precio": 1500  
6 }
```
- DELETE** | <localhost:8080/api/productos/2>:
Body tab selected. Body content:

```
1 Ctrl+Alt+P for Postbot
```

Body tab selected. Body content:

```
{ } JSON ▾
```

Body content:

```
1 {  
2   "id": 2,  
3   "nombre": null,  
4   "descripcion": null,  
5   "precio": 0  
6 }
```



Servicio de Venta

El Servicio de Venta, también desarrollado por David Albornoz, se encarga de registrar y gestionar las transacciones de compra de los clientes.

La entidad Venta tiene atributos como id, fecha y total.

Endpoints CRUD:

- GET: consultar todas las ventas o una específica.
- POST: crear un nuevo registro de venta.
- PUT: actualizar detalles de una venta.
- DELETE: eliminar una venta.
- Estas operaciones permiten mantener un historial de compras y garantizar la integridad de la información.



The image shows four screenshots of the Postman application interface, each demonstrating a different HTTP method on the same endpoint: `localhost:8080/api/ventas/1`.

- POST Request:** Shows the creation of a new sale record. The request body is a JSON object with `"fecha": "2025-05-26"` and `"total": 3500`. The response body shows the newly created record with `"id": 1`, `"fecha": "2025-05-26"`, and `"total": 3500.0`.
- GET Request:** Shows the retrieval of the first sale record. The response body is identical to the POST response: `{"id": 1, "fecha": "2025-05-26", "total": 3500.0}`.
- PUT Request:** Shows the update of the first sale record. The request body is updated to `"total": 5300`. The response body shows the updated record with `"id": 1`, `"fecha": "2025-05-26"`, and `"total": 5300.0`.
- DELETE Request:** Shows the deletion of the first sale record. The response body is updated to reflect the deletion: `"id": 1, "fecha": null, "total": 0.0`.

Servicio de Venta



Base de datos y configuración

La base de datos utilizada es MySQL, diseñada para almacenar y gestionar la información de los servicios.

Tablas principales:

- Cliente: almacena los datos personales de los clientes.
- Producto: contiene la información de los productos disponibles.
- Venta: guarda los registros de las transacciones realizadas.

Se utilizó MySQL Workbench para modelar la estructura de las tablas y XAMPP como servidor local para levantar el motor de base de datos.

La conexión con Spring Boot se configura en application.properties, asegurando la correcta integración entre el backend y la base de datos.

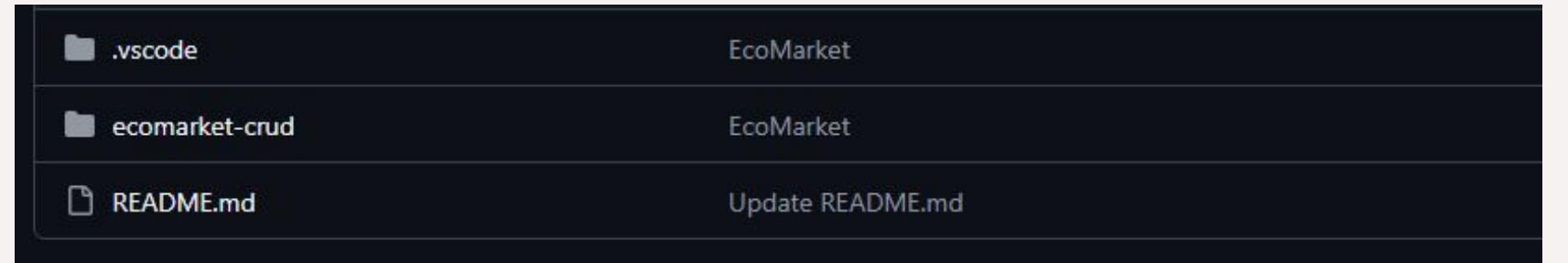


```
spring.application.name=eco_market

# parámetros de la conexión
spring.datasource.url=jdbc:mysql://localhost:3306/eco_market?serverTimezone=UTC&useSSL=false
spring.datasource.username=root
spring.datasource.password=

# conexión de JPA
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

# configuración de Logs (opcional)
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```



```
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Desktop/Exp3_Jilberto_Salgado_Albornoz
$ git init
Initialized empty Git repository in C:/Users/tomas/Desktop/Exp3_Jilberto_Salgado_Albornoz/.git/
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Desktop/Exp3_Jilberto_Salgado_Albornoz (master)
$ git add .
```

```
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Desktop/Exp3_Jilberto_Salgado_Albornoz (master)
$ git config --global user.name "tomasjilberto"
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Desktop/Exp3_Jilberto_Salgado_Albornoz (master)
$ git config --global user.email "to.jilberto@duocuc.cl"
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Desktop/Exp3_Jilberto_Salgado_Albornoz (master)
$ git branch -M main
```

```
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Desktop/Exp3_Jilberto_Salgado_Albornoz (main)
$ git remote add origin https://github.com/tomasjilberto/Exp3_Jilberto_Albornoz_Salgado.git
```

```
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Desktop/Exp3_Jilberto_Salgado_Albornoz
$ git push -u origin main
Enumerating objects: 106, done.
Counting objects: 100% (106/106), done.
Delta compression using up to 12 threads
Compressing objects: 100% (76/76), done.
Writing objects: 100% (106/106), 16.81 MiB | 7.00 MiB/s, done.
Total 106 (delta 27), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (27/27), done.
To https://github.com/tomasjilberto/Exp3_Jilberto_Albornoz_Salgado.git
 * [new branch]      main -> main
branch 'main' set up to track 'origin/main'.
```

Uso de GitHub

```
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos  
$ git clone https://github.com/tomasjilberto/Exp3_Jilberto_Albornoz_Salgado.git  
Cloning into 'Exp3_Jilberto_Albornoz_Salgado'...  
remote: Enumerating objects: 114, done.  
remote: Counting objects: 100% (114/114), done.  
remote: Compressing objects: 100% (56/56), done.  
remote: Total 114 (delta 30), reused 104 (delta 27), pack-reused 0 (from 0)  
Receiving objects: 100% (114/114), 20.01 MiB | 22.17 MiB/s, done.  
Resolving deltas: 100% (30/30), done.
```

```
$ git checkout -b araceli  
Switched to a new branch 'araceli'  
  
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos/Exp3_Jilberto_Albornoz_Salgado (araceli)  
$ git status  
On branch araceli  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git restore <file>..." to discard changes in working directory)  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/main/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/ClienteService.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/main/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/ClienteServiceImpl.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/main/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/ProductoService.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/main/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/ProductoServiceImpl.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/main/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/VentaService.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/main/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/VentaServiceImpl.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/test/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/ClienteServiceImplTest.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/test/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/ProductoServiceImplTest.java  
    modified: Experiencia 3/ProyectoEvaluacion3/ecomarket-crud/src/test/java/com/ecomarket/springboot/app/crudjpa/ecomarket_crud/services/VentaServiceImplTest.java
```

```
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos/Exp3_Jilberto_Albornoz_Salgado (araceli)  
$ git add .  
  
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos/Exp3_Jilberto_Albornoz_Salgado (araceli)  
$ git commit -m "Creacion de metodo modificar para pruebas unitarias"  
[araceli 651cb22] Creacion de metodo modificar para pruebas unitarias  
 10 files changed, 121 insertions(+)  
 create mode 100644 .vscode/launch.json  
  
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos/Exp3_Jilberto_Albornoz_Salgado (araceli)  
$ git push origin araceli  
Enumerating objects: 58, done.  
Counting objects: 100% (58/58), done.  
Delta compression using up to 12 threads.  
Compressing objects: 100% (20/20), done.  
Writing objects: 100% (36/36), 5.00 KiB | 640.00 KiB/s, done.  
Total 36 (delta 11), reused 0 (delta 0), pack-reused 0 (from 0)  
remote: Resolving deltas: 100% (11/11), completed with 8 local objects.  
remote:  
remote: Create a pull request for 'araceli' on GitHub by visiting:  
remote:   https://github.com/tomasjilberto/Exp3_Jilberto_Albornoz_Salgado/pull/new/araceli  
remote:  
To https://github.com/tomasjilberto/Exp3_Jilberto_Albornoz_Salgado.git  
 * [new branch]      araceli -> araceli
```

Uso de GitHub

Pruebas unitarias - Cliente

BIBLIOTECAS:

```
package com.ecomarket.springboot.app.crudjpa.ecomarket_crud.ecomarket_crud.services;
import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.entities.Cliente;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.repository.ClienteRepository;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.services.ClienteServiceImpl;
```

MÉTODO PARA CARGAR LOS DATOS PARA PODER TESTEAR:

```
public void chargeCliente(){
    Cliente cliente1 = new Cliente(Long.valueOf(1L), nombre:"Tomas Jilberto",gmail:"tomas.jilberto@gmail.com",edad:90);
    Cliente cliente2 = new Cliente(Long.valueOf(1L), nombre:"Juan Jara",gmail:"juan.jara@gmail.com",edad:40);
    Cliente cliente3 = new Cliente(Long.valueOf(1L), nombre:"David Albornoz",gmail:"david@gmail.com",edad:50);
    list.add(cliente1);
    list.add(cliente2);
    list.add(cliente3);
}
```

MÉTODO PARA BUSCAR AL TEST, BUSCAR UN CLIENTE Y ELIMINAR:

```
@Test
public void findByAllTest(){
    when(repository.findAll()).thenReturn(list);
    List<Cliente> response = service.findByAll();
    assertEquals(3, response.size());
    verify(repository, times(wantedNumberOfInvocations:1)).findAll();
}

@Test
public void testBuscarClientePorId() {
    Cliente clienteBuscado = list.get(index:0);
    when(repository.findById(1L)).thenReturn(Optional.of(clienteBuscado));

    Optional<Cliente> resultado = service.findById(id:1L);

    assertEquals("Tomas Jilberto", resultado.get().getNombre());
}

@Test
public void testEliminarCliente() {
    Cliente clienteAEliminar = list.get(index:1);
    when(repository.findById(2L)).thenReturn(Optional.of(clienteAEliminar));

    Optional<Cliente> resultado = service.delete(clienteAEliminar);

    assertEquals("Juan Jara", resultado.get().getNombre());
}
```

CONFIGURACIÓN DE PRUEBA UNITARIA:

```
@InjectMocks
private ClienteServiceImpl service;
@Mock
private ClienteRepository repository;
List<Cliente> list = new ArrayList<Cliente>();

@BeforeEach
public void init(){
    MockitoAnnotations.openMocks(this);
    this.chargeCliente();
}
```

Pruebas unitarias - Producto

BIBLIOTECAS

```
package com.ecomarket.springboot.app.crudjpa.ecomarket_crud.ecomarket_crud.services;
import static org.junit.jupiter.api.Assertions.*;
import java.util.ArrayList;
import java.util.List;
import java.util.Optional;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.entities.Producto;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.repository.ProductoRepository;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.services.ProductoServiceImpl;

public class ProductoServiceImplTest {
```

MÉTODO PARA CARGAR LOS DATOS PARA PODER TESTEAR:

```
///CARGA DATOS PARA PODER TESTEAR
public void chargeProducto(){
    Producto prod1 = new Producto(Long.valueOf(1L), nombre:"Miel Organica",descripcion:"dulce",precio:6000);
    Producto prod2 = new Producto(Long.valueOf(1L), nombre:"Café Organico",descripcion:"sin cafeina",precio:3000);
    Producto prod3 = new Producto(Long.valueOf(1L), nombre:"Cepillo de Dientes",descripcion:"de Bambú",precio:4000);
    list.add(prod1);
    list.add(prod2);
    list.add(prod3);
}
```

MÉTODO PARA BUSCAR AL TEST, BUSCAR UN CLIENTE Y ELIMINAR:

```
///MÉTODO PARA ENCONTRAR AL TEST
@Test
public void findByAllTest(){
    when(repository.findAll()).thenReturn(list);
    List<Producto> response = service.findByAll();
    assertEquals(3, response.size());
    verify(repository, times(wantedNumberOfInvocations:1)).findAll();
}

@Test
public void testBuscarProductoPorId() {
    Producto buscado = list.get(index:0); // Miel Organica
    when(repository.findById(id:1L)).thenReturn(Optional.of(buscado));

    Optional<Producto> resultado = service.findById(id:1L);

    assertEquals("Miel Organica", resultado.get().getNombre());
    assertEquals("dulce", resultado.get().getDescripcion());
    assertEquals(6000, resultado.get().getPrecio());
}

@Test
public void testEliminarProducto() {
    Producto aEliminar = list.get(index:2); // Cepillo de Dientes
    when(repository.findById(id:3L)).thenReturn(Optional.of(aEliminar));

    Optional<Producto> resultado = service.delete(aEliminar);

    assertEquals("Cepillo de Dientes", resultado.get().getNombre());
    assertEquals("de Bambú", resultado.get().getDescripcion());
    assertEquals(4000, resultado.get().getPrecio());
}
```

CONFIGURACIÓN DE PRUEBA UNITARIA:

```
@InjectMocks
private ProductoServiceImpl service;
@Mock
private ProductoRepository repository;
List<Producto> list = new ArrayList<Producto>();

@BeforeEach
public void init(){
    MockitoAnnotations.openMocks(this);
    this.chargeProducto();
}
```

Pruebas de unitarias - Venta

BIBLIOTECAS:

```
package com.ecomarket.springboot.app.crudjpa.ecomarket_crud.ecomarket_crud.services;

import static org.junit.jupiter.api.Assertions.assertEquals;
import static org.mockito.Mockito.times;
import static org.mockito.Mockito.verify;
import static org.mockito.Mockito.when;

import java.util.ArrayList;
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.entities.Venta;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.repository.VentaRepository;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.services.VentaServiceImpl;
```

MÉTODO PARA CARGAR LOS DATOS PARA PODER TESTEAR:

```
public void chargeVenta(){
    Venta venta1 = new Venta(Long.valueOf(1:1), fecha:"22/06/2025",total:9000000);
    Venta venta2 = new Venta(Long.valueOf(1:2), fecha:"31/04/2025",total:10000);
    Venta venta3 = new Venta(Long.valueOf(1:3), fecha:"12/06/2025",total:4000000);
    list.add(venta1);
    list.add(venta2);
    list.add(venta3);
}
```

MÉTODO PARA BUSCAR AL TEST, BUSCAR UN CLIENTE Y ELIMINAR:

```
@Test
public void findByAllTest(){
    when(repository.findAll()).thenReturn(list);
    List<Venta> response = service.findByAll();
    assertEquals(3, response.size());
    verify(repository, times(wantedNumberOfInvocations:1)).findAll();
}

@Test
public void testBuscarVentaPorId() {
    Venta buscada = list.get(index:0); // 22/06/2025
    when(repository.findById(id:1L)).thenReturn(Optional.of(buscada));

    Optional<Venta> resultado = service.findById(id:1L);

    assertEquals("22/06/2025", resultado.get().getFecha());
    assertEquals(9000000, resultado.get().getTotal());
}

@Test
public void testEliminarVenta() {
    Venta aEliminar = list.get(index:1); // 31/04/2025
    when(repository.findById(id:2L)).thenReturn(Optional.of(aEliminar));

    Optional<Venta> resultado = service.delete(aEliminar);

    assertEquals("31/04/2025", resultado.get().getFecha());
    assertEquals(10000, resultado.get().getTotal());
}
```

CONFIGURACIÓN DE PRUEBA UNITARIA:

```
@InjectMocks
private VentaServiceImpl service;
@Mock
private VentaRepository repository;
List<Venta> list = new ArrayList<Venta>();

@BeforeEach
public void init(){
    MockitoAnnotations.openMocks(this);
    this.chargeVenta();
}
```

Pruebas de integración - Producto

BIBLIOTECAS

```
import static org.mockito.Mockito.any;
import static org.mockito.Mockito.when;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;
import static org.junit.jupiter.api.Assertions.fail;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.entities.Producto;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.services.ProductoServiceImpl;
import com.fasterxml.jackson.databind.ObjectMapper;

import org.springframework.http.MediaType;
```

CONFIGURACIÓN DE PRUEBA INTEGRACION:

```
@Autowired
private MockMvc mockMvc;

@Autowired
private ObjectMapper objectMapper;

@MockBean
private ProductoServiceImpl productoServiceimpl;

private List<Producto> productosLista;
public void setUp() {
    Producto producto1 = new Producto(id:1L, nombre:"Cepillo", descripcion:"Ecologico", precio:1500);
    Producto producto2 = new Producto(id:2L, nombre:"Bombilla", descripcion:"de Papel", precio:200);
    productosLista = Arrays.asList(producto1, producto2);
}
```

MÉTODO PARA PRODUCTO NO EXISTENTE, PARA CREAR UNO Y MODIFICAR UNO:

```
@Test
public void productoNoExisteTest() throws Exception{
    when(productoserviceimpl.findById(id:10L)).thenReturn(Optional.empty());
    mockMvc.perform(get(uriTemplate:"/api/productos/10")
        .contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isNotFound());
}

@Test
public void crearProductoTest() throws Exception{
    Producto unProducto = new Producto(id:null,nombre:"Bombilla",descripcion:"de Papel",precio:200);
    Producto otroProducto = new Producto(id:4L,nombre:"Calzado",descripcion:"bueno con el medio ambiente",precio:40000);
    when(productoserviceimpl.save(any(type:Producto.class))).thenReturn(otroProducto);
    mockMvc.perform(post(uriTemplate:"/api/productos")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(unProducto))
        .andExpect(status().isCreated());
}

@Test
public void modificarProductoTest() throws Exception {
    Producto original = new Producto(id:1L, nombre:"Cepillo", descripcion:"Ecologico", precio:1500);
    Producto modificado = new Producto(id:1L, nombre:"Cepillo Bamboo", descripcion:"Mejorado", precio:2000);

    when(productoserviceimpl.findById(id:1L)).thenReturn(Optional.of(original));
    when(productoserviceimpl.save(any(type:Producto.class))).thenReturn(modificado);

    mockMvc.perform(put(uriTemplate:"/api/productos/1")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(modificado)))
        .andExpect(status().isOk());
}
```

MÉTODO PARA VER LOS PRODUCTOS:

```
@Test
public void verProductosTest() throws Exception {
    when(productoserviceimpl.findByAll()).thenReturn(productosLista);

    mockMvc.perform(get(uriTemplate:"/api/productos")
        .contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isOk());
}

@Test
public void verunProductoTest(){
    Producto unProducto = new Producto(id:1L,nombre:"Cepillo", descripcion:"Ecologico", precio:1500);
    try{
        when(productoserviceimpl.findById(id:1L)).thenReturn(Optional.of(unProducto));
        mockMvc.perform(get(uriTemplate:"/api/productos/1")
            .contentType(MediaType.APPLICATION_JSON)
            .andExpect(status().isOk());
    }
    catch(Exception ex){
        fail("El testing lanzo un error " + ex.getMessage());
    }
}
```

MÉTODO PARA ELIMINAR UN PRODUCTO:

```
@Test
public void eliminarProductoTest() throws Exception {
    Producto productoAEliminar = new Producto(id:1L, nombre:"Bombilla", descripcion:"de papel", precio:200);

    when(productoserviceimpl.delete(any(type:Producto.class))).thenReturn(Optional.of(productoAEliminar));
    mockMvc.perform(delete(uriTemplate:"/api/productos/1")
        .contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isOk());
}
```

Pruebas de integración - Venta

BIBLIOTECAS:

```
import static org.junit.jupiter.api.Assertions.fail;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.entities.Venta;
import com.ecomarket.springboot.app.crudjpa.ecomarket_crud.services.VentaServiceImpl;
import com.fasterxml.jackson.databind.ObjectMapper;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;
```

MÉTODO PARA PRODUCTO NO EXISTENTE, PARA CREAR UNO Y MODIFICAR UNO:

```
@Test
public void ventaNoExisteTest() throws Exception{
    when(ventaServiceImpl.findById(id:10L)).thenReturn(Optional.empty());
    mockMvc.perform(get(uriTemplate:"/api/ventas/10")
        .contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isNotFound());
}

@Test
public void crearVentaTest() throws Exception{
    Venta unaVenta = new Venta(id:6L,fecha:"23/03/2011",total:600000);
    Venta otraVenta = new Venta(id:5L,fecha:"21/03/2021",total:40000);
    when(ventaServiceImpl.save(any(type:Venta.class))).thenReturn(otraVenta);
    mockMvc.perform(post(uriTemplate:"/api/ventas")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(unaVenta)))
        .andExpect(status().isCreated());
}

@Test
public void modificarVentaTest() throws Exception {
    Venta original = new Venta(id:1L,fecha:"21/08/23", total:22332);
    Venta modificado = new Venta(id:1L, fecha:"21/08/23", total:199900);

    when(ventaServiceImpl.findById(id:1L)).thenReturn(Optional.of(original));
    when(ventaServiceImpl.save(any(type:Venta.class))).thenReturn(modificado);

    mockMvc.perform(put(uriTemplate:"/api/ventas/1")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(modificado)))
        .andExpect(status().isOk());
}
```

CONFIGURACIÓN DE PRUEBA INTEGRACION:

```
1  @SpringBootTest
2  @AutoConfigureMockMvc
3  public class VentaRestControllerTest {
4      @Autowired
5      private MockMvc mockMvc;
6
7      @Autowired
8      private ObjectMapper objectMapper;
9      @MockBean
10     private VentaServiceImpl ventaServiceImpl;
11     private List<Venta> ventasLista;
12
13
14     public void setUp() {
15         Venta venta1 = new Venta(id:1L, fecha:"21/08/23", total:22332);
16         Venta venta2 = new Venta(id:2L, fecha:"29/01/24", total:19990);
17         ventasLista = Arrays.asList(venta1, venta2);
18     }
19 }
```

MÉTODO PARA VER LOS PRODUCTOS:

```
@Test
public void verVentaTest() throws Exception {
    when(ventaServiceImpl.findByAll()).thenReturn(ventasLista);
    mockMvc.perform(get(uriTemplate:"/api/ventas")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk());
}

@Test
public void verunaVentaTest(){
    Venta unaVenta = new Venta(id:1L,fecha:"21/08/23", total:22332);
    try{
        when(ventaServiceImpl.findById(id:1L)).thenReturn(Optional.of(unaVenta));
        mockMvc.perform(get(uriTemplate:"/api/ventas/1")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk());
    }catch(Exception ex){
        fail("El testing lanza un error " + ex.getMessage());
    }
}
```

MÉTODO PARA ELIMINAR UN PRODUCTO:

```
public void eliminarVentaTest() throws Exception {
    Venta ventaAEliminar = new Venta(id:1L, fecha:"21/08/23", total:199900);

    when(ventaServiceImpl.delete(any(type:Venta.class))).thenReturn(Optional.of(ventaAEliminar));
    mockMvc.perform(delete(uriTemplate:"/api/ventas/1")
        .contentType(MediaType.APPLICATION_JSON))
        .andExpect(status().isOk());
}
```

Pruebas de integración - Cliente

BIBLIOTECAS:

```
import static org.junit.jupiter.api.Assertions.fail;
import static org.mockito.ArgumentMatchers.any;
import static org.mockito.Mockito.when;

import java.util.Arrays;
import java.util.List;
import java.util.Optional;

import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.boot.test.mock.mockito.MockBean;
import org.springframework.http.MediaType;
import org.springframework.test.web.servlet.MockMvc;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.get;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.post;
import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.delete;
import static org.springframework.test.web.servlet.request.MockMvcRequestBuilders.put;
import com.ecomarket.springboot.app.crud.jpa.ecomarket_crud.entities.Cliente;
import com.ecomarket.springboot.app.crud.jpa.ecomarket_crud.services.ClienteServiceImpl;
import com.fasterxml.jackson.databind.ObjectMapper;
```

CONFIGURACIÓN DE PRUEBA INTEGRACION:

```
@SpringBootTest
@AutoConfigureMockMvc
public class ClienteRestControllerTest {
    @Autowired
    private MockMvc mockMvc;

    @Autowired
    private ObjectMapper objectMapper;
    @MockBean
    private ClienteServiceImpl clienteServiceImpl;
    private List<Cliente> clienteLista;
    public void setUp() {
        Cliente cliente1 = new Cliente(id:1L, nombre:"Josefina", gmail:"josefina@gmail.com", edad:22);
        Cliente cliente2 = new Cliente(id:2L, nombre:"Tomas", gmail:"tomas@gmail.com", edad:19);
        clienteLista = Arrays.asList(cliente1, cliente2);
    }
}
```

MÉTODO PARA PRODUCTO NO EXISTENTE, PARA CREAR UNO Y MODIFICAR UNO:

```
@Test
public void clienteNoExisteTest() throws Exception{
    when(clienteServiceImpl.findById(id:10L)).thenReturn(Optional.empty());
    mockMvc.perform(get(uriTemplate:"/api/clientes/10")
        .contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isNotFound());
}

public void crearClienteTest() throws Exception{
    Cliente unCliente = new Cliente(id:null,nombre:"Miguel",gmail:"miguel@gmail.com",edad:22);
    Cliente otroCliente = new Cliente(id:3L,nombre:"Rene",gmail:"rene@gmail.com",edad:18);
    when(clienteServiceImpl.save(any(type:Cliente.class))).thenReturn(otroCliente);
    mockMvc.perform(post(uriTemplate:"/api/clientes")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(unCliente))
        .andExpect(status().isCreated());
}

@Test
public void modificarClienteTest() throws Exception {
    Cliente original = new Cliente(id:2L, nombre:"Tomas", gmail:"tomas@gmail.com", edad:19);
    Cliente modificado = new Cliente(id:2L, nombre:"Agustin", gmail:"agustin51@gmail.com", edad:2000);

    when(clienteServiceImpl.findById(id:2L)).thenReturn(Optional.of(original));
    when(clienteServiceImpl.save(any(type:Cliente.class))).thenReturn(modificado);

    mockMvc.perform(put(uriTemplate:"/api/clientes/2")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(modificado))
        .andExpect(status().isOk());
}
```

MÉTODO PARA VER LOS CLIENTES:

```
@Test
public void verClientesTest() throws Exception {
    when(clienteServiceImpl.findByAll()).thenReturn(clienteLista);

    mockMvc.perform(get(uriTemplate:"/api/clientes")
        .contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isOk());
}

@Test
public void verUnClienteTest(){
    Cliente unCliente = new Cliente(id:1L,nombre:"Josefina", gmail:"josefina@gmail.com", edad:22);
    try{
        when(clienteServiceImpl.findById(id:1L)).thenReturn(Optional.of(unCliente));
        mockMvc.perform(get(uriTemplate:"/api/clientes/1")
            .contentType(MediaType.APPLICATION_JSON)
            .andExpect(status().isOk());
    } catch(Exception ex){
        fail("El testing lanzo un error " + ex.getMessage());
    }
}
```

MÉTODO PARA ELIMINAR UN PRODUCTO:

```
@Test
public void eliminarClienteTest() throws Exception {
    Cliente clienteAEliminar = new Cliente(id:2L, nombre:"Tomas", gmail:"tomas@gmail.com", edad:18);
    when(clienteServiceImpl.delete(any(type:Cliente.class))).thenReturn(Optional.of(clienteAEliminar));
    mockMvc.perform(delete(uriTemplate:"/api/clientes/2")
        .contentType(MediaType.APPLICATION_JSON)
        .andExpect(status().isOk());
}
```

Documentación y prueba manual con Swagger UI

A través de Swagger, se puede ejecutar operaciones como:

- GET /api/clientes para obtener todos los clientes
- GET /api/clientes/{id} para obtener un cliente específico
- POST /api/clientes para crear un nuevo cliente
- PUT /api/clientes/{id} para modificar un cliente existente
- DELETE /api/clientes/{id} para eliminar un cliente

Servers

http://localhost:8080 - Generated server url

Cientes

Operaciones relacionadas con clientes

GET	/api/clientes/{id}	Obtener clientes por ID
PUT	/api/clientes/{id}	Modificar un cliente
DELETE	/api/clientes/{id}	Eliminar un cliente
GET	/api/clientes	Obtener lista de clientes
POST	/api/clientes	Crear un nuevo cliente

Productos

Operaciones relacionadas con productos

GET	/api/productos/{id}	Obtener productos por ID
PUT	/api/productos/{id}	Modificar un producto
DELETE	/api/productos/{id}	Eliminar un producto
GET	/api/productos	Obtener lista de productos
POST	/api/productos	Crear un nuevo producto

Ventas

Operaciones relacionadas con Ventas

GET	/api/ventas/{id}	Obtener Ventas por ID
PUT	/api/ventas/{id}	Modificar una venta
DELETE	/api/ventas/{id}	Eliminar una Venta
GET	/api/ventas	Obtener lista de Ventas
POST	/api/ventas	Crear una nueva Venta

Implementación de Git-Hub

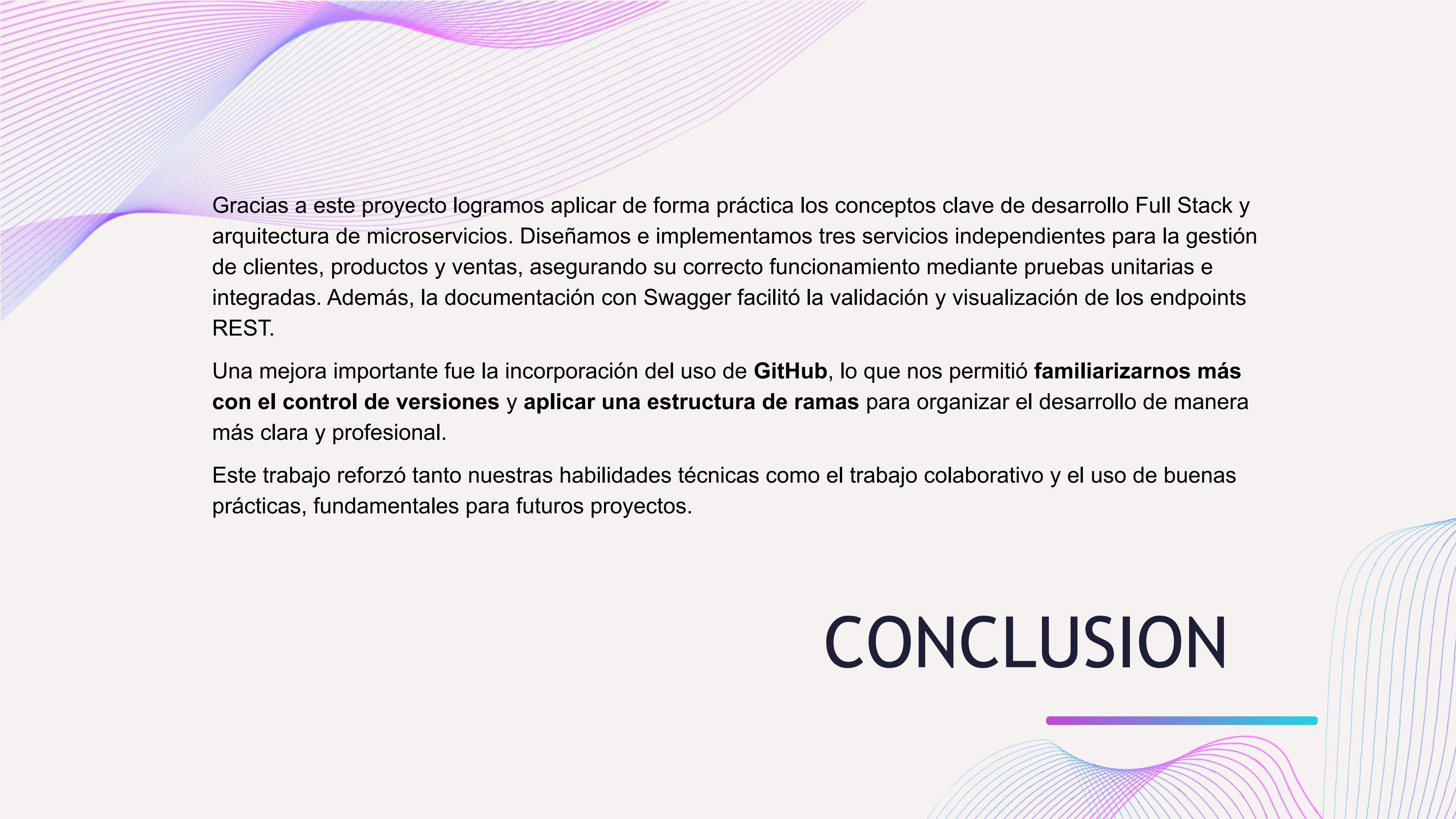
El proyecto implementa una estructura de control de versiones basada en tres ramas principales utilizando Git, con el objetivo de facilitar el desarrollo colaborativo, mantener la estabilidad del código en producción y organizar el flujo de trabajo de forma eficiente.

Esta estrategia permite separar claramente los entornos de desarrollo, integración y producción. Gracias a esto, las nuevas funcionalidades o correcciones pueden desarrollarse de forma aislada, sin afectar el código estable, y luego integrarse de manera controlada y segura. Además, mejora la trazabilidad de los cambios y favorece un trabajo más ordenado, especialmente en equipos o proyectos que puedan escalar.

```
tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos/Exp3_Jilberto_Albornoz_Salgado (araceli)
$ git add .

tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos/Exp3_Jilberto_Albornoz_Salgado (araceli)
$ git commit -m "Creacion de metodo modificar para pruebas unitarias"
[araceli 651cb22] Creacion de metodo modificar para pruebas unitarias
 10 files changed, 121 insertions(+)
 create mode 100644 .vscode/launch.json

tomas@DESKTOP-5HJ07R5 MINGW64 ~/Videos/Exp3_Jilberto_Albornoz_Salgado (araceli)
$ git push origin araceli
Enumerating objects: 58, done.
Counting objects: 100% (58/58), done.
Delta compression using up to 12 threads
Compressing objects: 100% (20/20), done.
Writing objects: 100% (36/36), 5.00 KiB | 640.00 KiB/s, done.
Total 36 (delta 11), reused 0 (delta 0), pack-reused 0 (from 0)
remote: Resolving deltas: 100% (11/11), completed with 8 local objects.
remote:
remote: Create a pull request for 'araceli' on GitHub by visiting:
remote:   https://github.com/tomasjilberto/Exp3_Jilberto_Albornoz_Salgado/pull/new/araceli
remote:
To https://github.com/tomasjilberto/Exp3_Jilberto_Albornoz_Salgado.git
 * [new branch]      araceli -> araceli
```



Gracias a este proyecto logramos aplicar de forma práctica los conceptos clave de desarrollo Full Stack y arquitectura de microservicios. Diseñamos e implementamos tres servicios independientes para la gestión de clientes, productos y ventas, asegurando su correcto funcionamiento mediante pruebas unitarias e integradas. Además, la documentación con Swagger facilitó la validación y visualización de los endpoints REST.

Una mejora importante fue la incorporación del uso de **GitHub**, lo que nos permitió **familiarizarnos más con el control de versiones y aplicar una estructura de ramas** para organizar el desarrollo de manera más clara y profesional.

Este trabajo reforzó tanto nuestras habilidades técnicas como el trabajo colaborativo y el uso de buenas prácticas, fundamentales para futuros proyectos.

CONCLUSION

