

**Eco Market SPA.**

**Nombre de los Estudiantes: Araceli Salgado**

**Tomas Jilberto**

**David Albornoz**

**Nombre del Docente: Eduardo Baeza**

**Nombre Asignatura: Full Stack 1**

**Sección: 011D**

# **Índice**

**Diagrama de Arquitectura de Microservicios**3

**Plan de Pruebas**4-17

**Ejecución de Pruebas**18-23

**Git-GitHub**24

**Conclusión**25

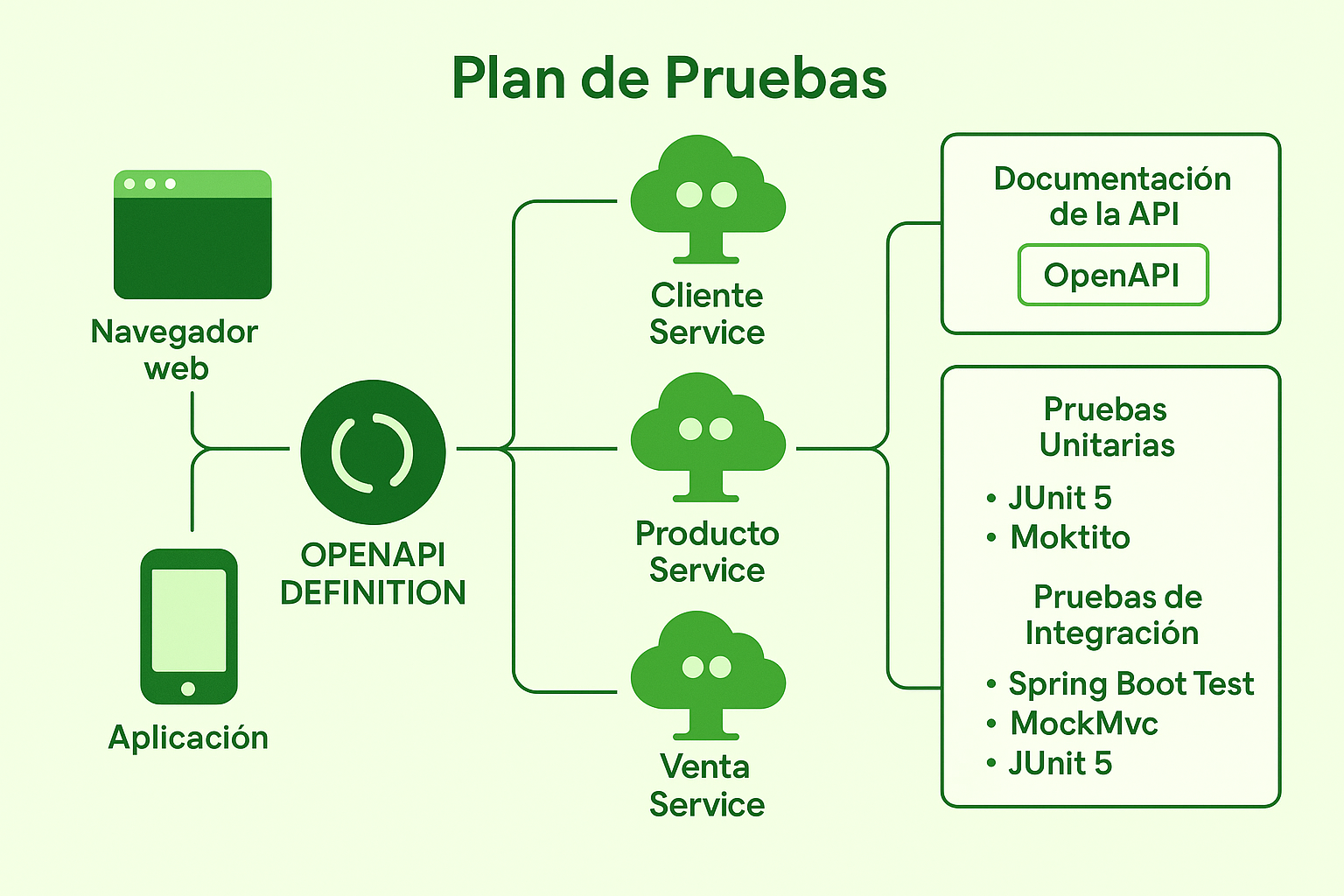
# **Introducción**

El presente informe tiene como objetivo documentar el desarrollo del proyecto "EcoMarket SPA", el cual consistió en la creación de una aplicación utilizando arquitectura de microservicios bajo el enfoque Full Stack. Durante su ejecución, trabajamos en la implementación de servicios independientes para la gestión de clientes, productos y ventas, aplicando principios de modularidad, reutilización de código y buenas prácticas de desarrollo.

A lo largo del proyecto, empleamos diversas herramientas y frameworks modernos, como Spring Boot, JUnit 5, Mockito y MockMvc, que nos permitieron realizar pruebas unitarias e integradas para asegurar la funcionalidad y estabilidad del sistema. Este proceso no solo reforzó nuestros conocimientos técnicos, sino que también nos permitió trabajar de manera colaborativa, enfrentando desafíos reales del desarrollo de software.

El siguiente documento presenta de forma detallada el diseño de la arquitectura, la planificación y ejecución de pruebas, así como las conclusiones obtenidas a partir de esta experiencia de aprendizaje.

***Diagrama De Arquitectura De Microservicios:***

**

El diagrama muestra la arquitectura basada en microservicios del sistema de EcoMarket SPA, donde los usuarios acceden a través de un navegador web o aplicación móvil, y las solicitudes se gestionan mediante la definición OpenAPI, que documenta y organiza los endpoints REST disponibles. Los tres microservicios principales —**Cliente Service, Producto Service y Venta Service**— operan de forma independiente, ejecutando las funciones **CRUD** correspondientes. El sistema incluye pruebas unitarias (**JUnit 5 y Mockito**) para validar la lógica de negocio de los servicios de manera aislada, y pruebas de integración (**Spring Boot Test, MockMvc y JUnit 5**) para verificar la correcta interacción entre controladores y servicios. Esta estructura mejora la calidad del software, facilita el mantenimiento y asegura un desarrollo eficiente.

***Plan de Pruebas:***

## **Justificación de Herramientas y Frameworks Utilizados para la Creación de Pruebas**

En el desarrollo del proyecto **EcoMarket CRUD**, se utilizaron diversas herramientas y frameworks con el propósito de asegurar la calidad del software mediante pruebas unitarias y de integración. A continuación, se detalla qué tecnologías se utilizaron para cada tipo de prueba y por qué fueron elegidas:

### Pruebas Unitarias

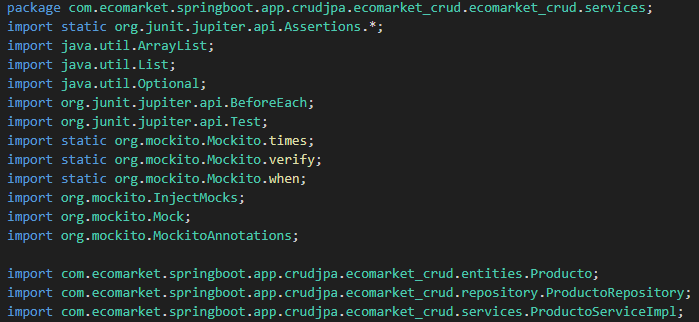
Las **pruebas unitarias** tienen como objetivo verificar que cada componente individual del sistema funcione correctamente de forma aislada. En este proyecto, se enfocaron principalmente en la lógica de los servicios (ClienteServiceImpl, ProductoServiceImpl, VentaServiceImpl), sin involucrar capas externas como la base de datos o los controladores.

#### Herramientas utilizadas:

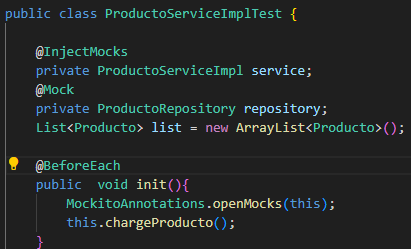
* **JUnit 5**: Es el motor principal para ejecutar las pruebas. Es ampliamente reconocido por su facilidad de uso y su integración con herramientas modernas de desarrollo.
* **Mockito**: Permite simular (mockear) dependencias como los repositorios, con el fin de probar exclusivamente la lógica del servicio sin que intervenga la conexión a una base de datos u otros componentes.

1. **Producto**

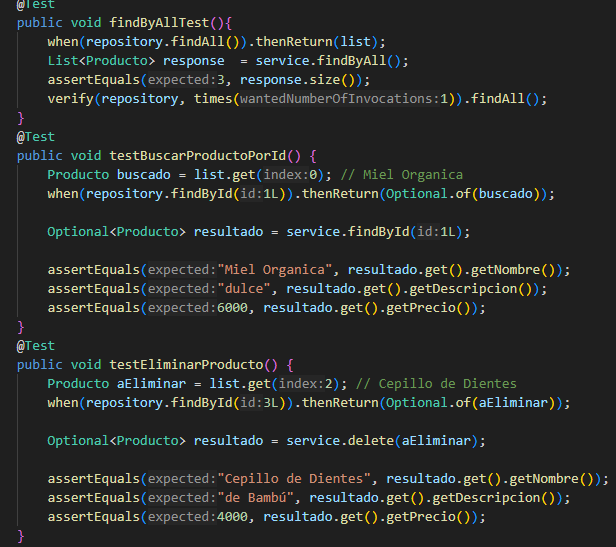
***Bibliotecas Implementadas:***

******

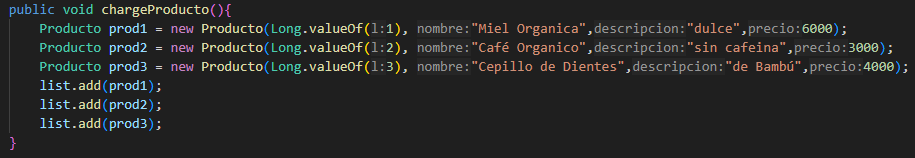
***Configuración de Prueba Unitaria:***

******

***Método para buscar al test, buscar un producto y eliminar:***

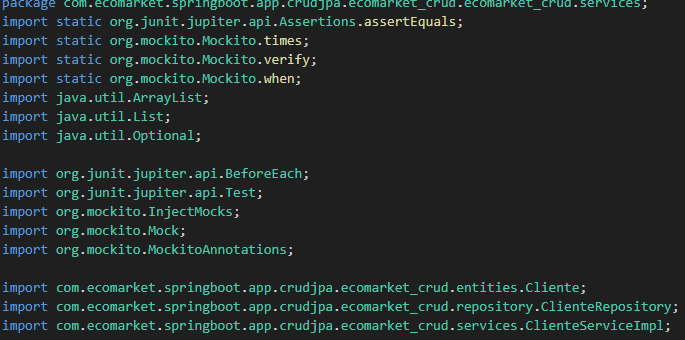
******

***Método para cargar los datos para poder testear:***

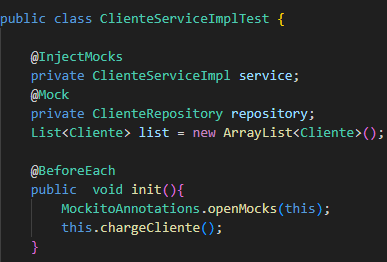
******

1. **Cliente**

***Bibliotecas Implementadas:***

******

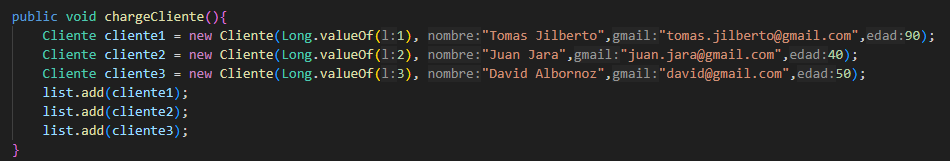
***Configuración de Prueba Unitaria:***

******

***Método para buscar al test, buscar un cliente y eliminar:***

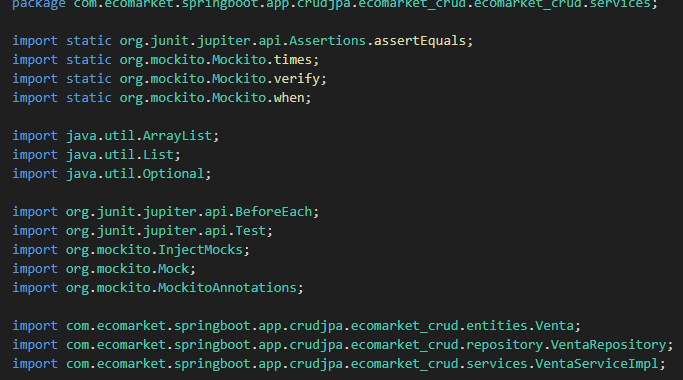
******

***Método para cargar los datos para poder testear:***

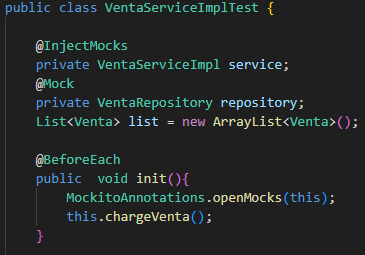
******

1. **Ventas**

***Bibliotecas Implementadas:***

******

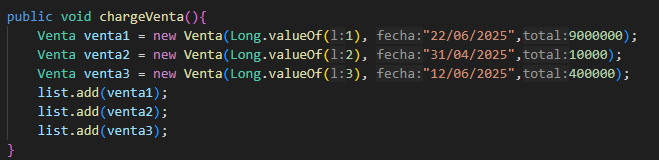
***Configuración de Prueba Unitaria:***

******

***Método para buscar al test, buscar una venta y eliminar:***

******

***Método para cargar los datos para poder testear:***

******

### Pruebas de Integración

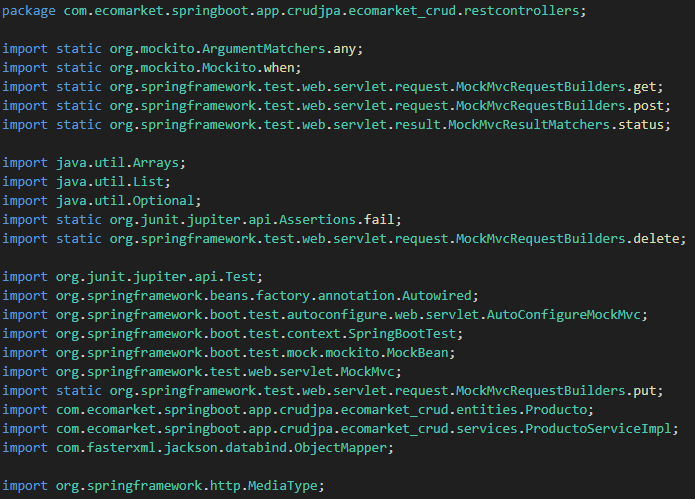
Las **pruebas de integración** se utilizan para asegurar que los diferentes módulos del sistema trabajen correctamente en conjunto. En este caso, se probaron los controladores REST, validando que interactúan adecuadamente con los servicios y que responden correctamente ante solicitudes HTTP.

#### Herramientas utilizadas:

* **Spring Boot Test**: Proporciona un entorno de prueba realista que integra múltiples capas de la aplicación (controlador, servicio, repositorio).
* **MockMvc**: Permite simular peticiones HTTP a los endpoints REST sin necesidad de iniciar un servidor real.
* **JUnit 5**: También utilizado aquí como framework base para la ejecución de pruebas.

**1.Producto**

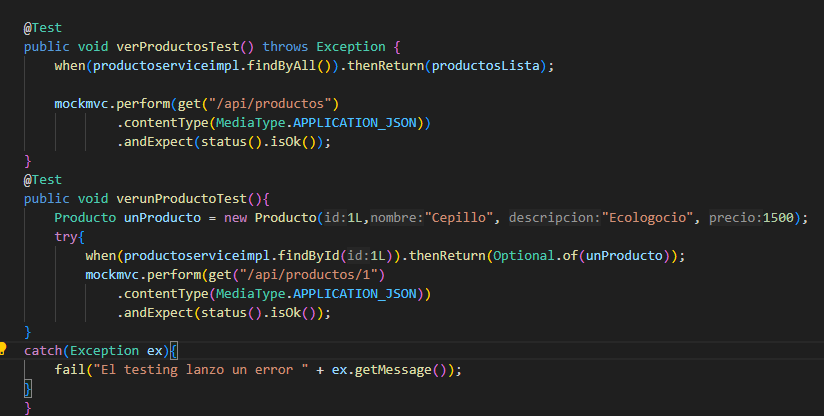
***Bibliotecas Implementadas:***



***Configuración de prueba de integración***

******

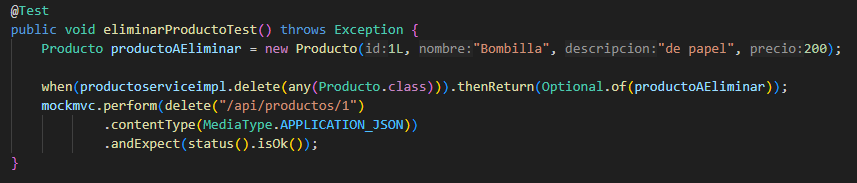
***Método Para ver los Productos:***

******

***Método para producto no existente, para crear uno y modificar uno:***

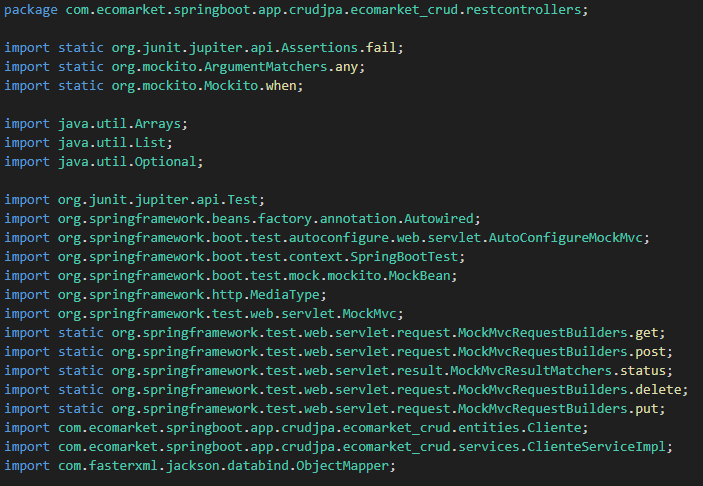
******

***Método para eliminar un producto:***

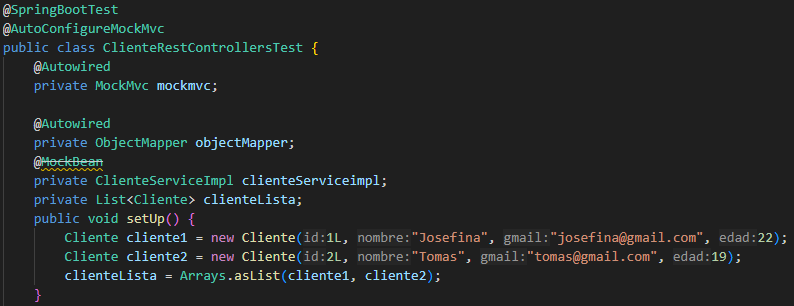
******

**2.Cliente**

***Bibliotecas Implementadas:***

******

***Test de integración en Spring Boot:***



***Método Para ver los Clientes:***

******

***Método para cliente no existente, para crear uno y modificar uno:***

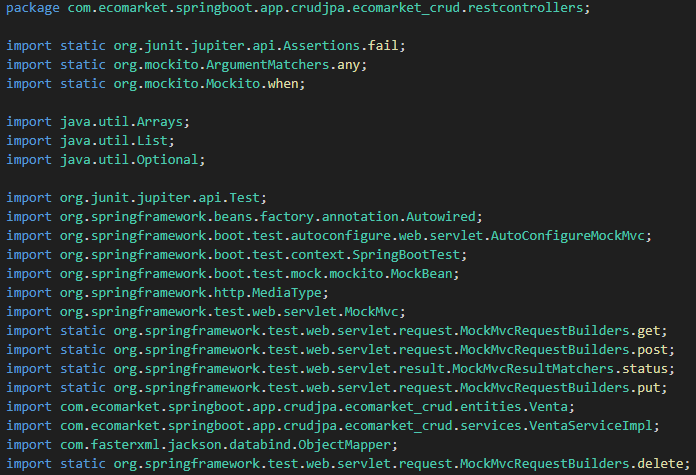


***Método para eliminar un Cliente:***

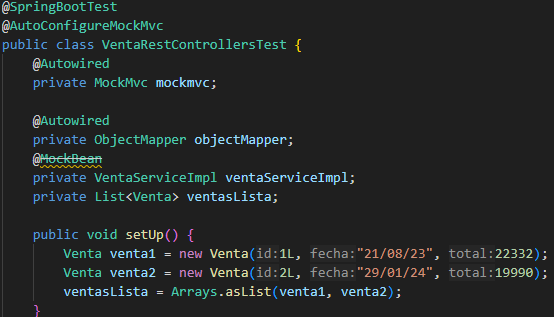
# 

**3.Ventas**

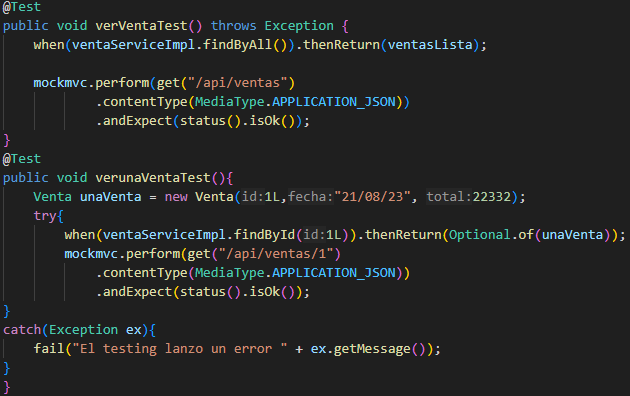
***Bibliotecas Implementadas:***

******

***Test de integración en Spring Boot:***

******

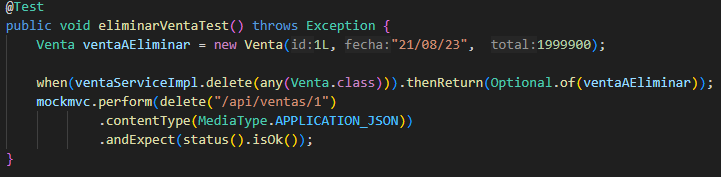
***Método Para ver las Ventas:***

******

***Método para venta no existente, para crear una y modificar una:***

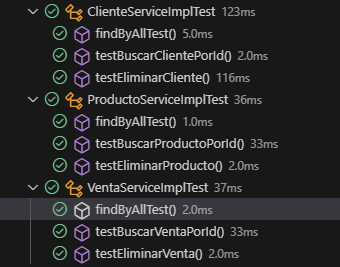
******

***Método para eliminar una venta:***



***Ejecución de Pruebas***

1. *Unitarias:*

**

## **Documentación:**

### ClienteServiceImplTest

* **Prueba:** findByAllTest()  
   **Tipo:** Unitaria  
   **Descripción:** Verifica que el método findByAll() devuelva correctamente la lista de todos los clientes utilizando un repositorio simulado (mock).  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** testBuscarClientePorId()  
   **Tipo:** Unitaria  
   **Descripción:** Evalúa que se pueda buscar un cliente por su ID de forma correcta.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** testEliminarCliente()  
   **Tipo:** Unitaria  
   **Descripción:** Verifica que el servicio sea capaz de eliminar correctamente un cliente simulado.  
   **Resultado:** Prueba ejecutada exitosamente.

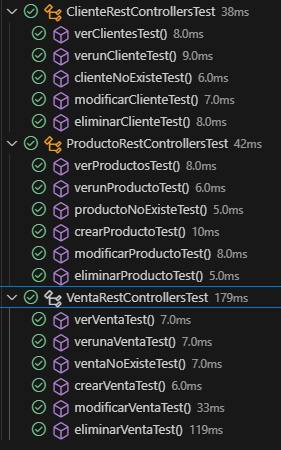
### ProductoServiceImplTest

* **Prueba:** findByAllTest()  
   **Tipo:** Unitaria  
   **Descripción:** Verifica que el servicio retorne correctamente la lista de productos.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** testBuscarProductoPorId()  
   **Tipo:** Unitaria  
   **Descripción:** Evalúa la búsqueda de productos por ID usando datos simulados.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** testEliminarProducto()  
   **Tipo:** Unitaria  
   **Descripción:** Comprueba que un producto pueda ser eliminado correctamente desde el servicio.  
   **Resultado:** Prueba ejecutada exitosamente.

### VentaServiceImplTest

* **Prueba:** findByAllTest()  
   **Tipo:** Unitaria  
   **Descripción:** Valida que el método findByAll() recupere todas las ventas registradas.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** testBuscarVentaPorId()  
   **Tipo:** Unitaria  
   **Descripción:** Comprueba que el servicio devuelva correctamente una venta específica por su ID.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** testEliminarVenta()  
   **Tipo:** Unitaria  
   **Descripción:** Evalúa que la lógica de eliminación de ventas funcione correctamente usando mock.  
   **Resultado:** Prueba ejecutada exitosamente.

1. *Integración:*



## **Documentación:**

### ClienteRestControllersTest

* **Prueba:** verClientesTest()  
   **Tipo:** Integración  
   **Descripción:** Simula una petición GET al endpoint /api/clientes para listar todos los clientes.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** verunClienteTest()  
   **Tipo:** Integración  
   **Descripción:** Evalúa que se pueda recuperar un cliente por su ID.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** clienteNoExisteTest()  
   **Tipo:** Integración  
   **Descripción:** Comprueba el comportamiento del sistema cuando se consulta un cliente que no existe.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** modificarClienteTest()  
   **Tipo:** Integración  
   **Descripción:** Verifica la modificación de datos de un cliente vía PUT.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** eliminarClienteTest()  
   **Tipo:** Integración  
   **Descripción:** Simula una petición DELETE para eliminar un cliente por su ID.  
   **Resultado:** Prueba ejecutada exitosamente.

### ProductoRestControllersTest

* **Prueba:** verProductosTest()  
   **Tipo:** Integración  
   **Descripción:** Simula una solicitud GET para listar todos los productos.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** verunProductoTest()  
   **Tipo:** Integración  
   **Descripción:** Verifica que el controlador pueda retornar un producto específico por ID.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** productoNoExisteTest()  
   **Tipo:** Integración  
   **Descripción:** Evalúa la respuesta ante la búsqueda de un producto inexistente.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** crearProductoTest()  
   **Tipo:** Integración  
   **Descripción:** Simula una solicitud POST para registrar un nuevo producto.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** modificarProductoTest()  
   **Tipo:** Integración  
   **Descripción:** Valida que un producto pueda modificarse correctamente.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** eliminarProductoTest()  
   **Tipo:** Integración  
   **Descripción:** Comprueba la eliminación correcta de un producto mediante el endpoint DELETE.  
   **Resultado:** Prueba ejecutada exitosamente.

### VentaRestControllersTest

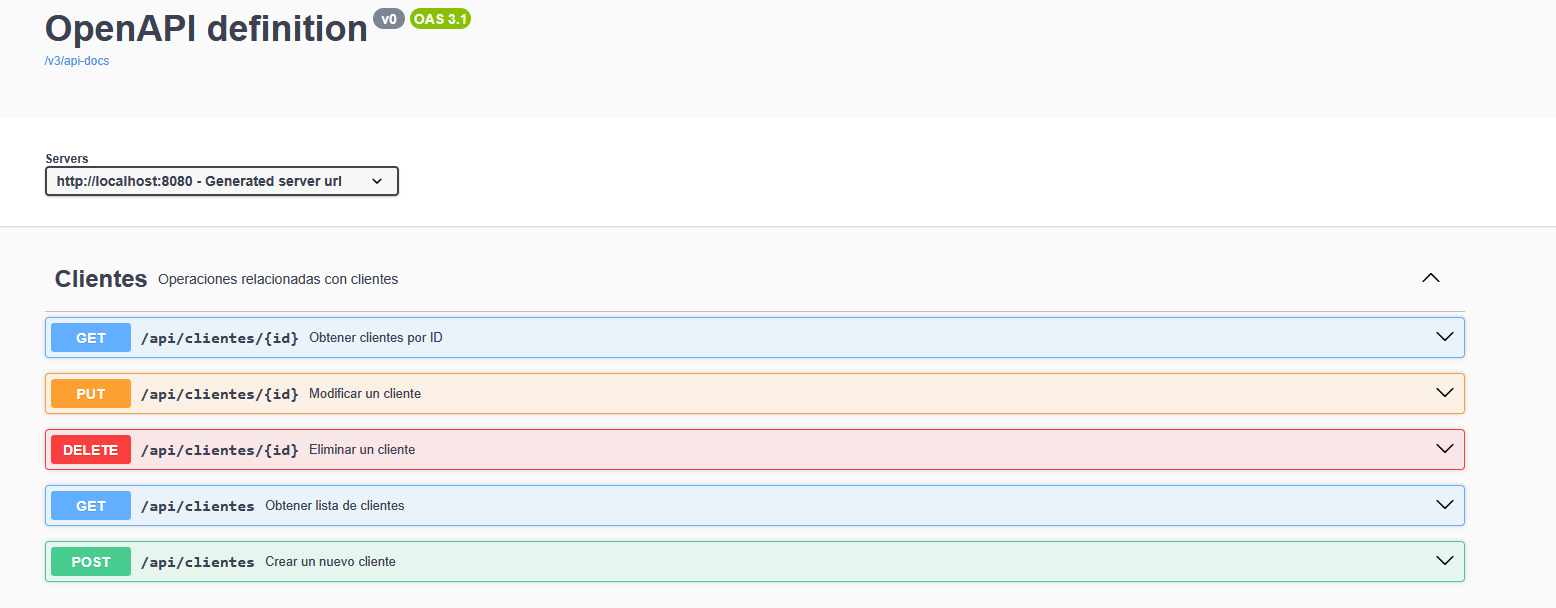
* **Prueba:** verVentaTest()  
   **Tipo:** Integración  
   **Descripción:** Realiza una petición GET para obtener todas las ventas registradas.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** verunaVentaTest()  
   **Tipo:** Integración  
   **Descripción:** Valida que una venta específica se recupere correctamente por ID.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** ventaNoExisteTest()  
   **Tipo:** Integración  
   **Descripción:** Evalúa la respuesta al intentar buscar una venta que no existe.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** crearVentaTest()  
   **Tipo:** Integración  
   **Descripción:** Simula la creación de una venta a través del endpoint POST, validando su integración con cliente y productos.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** modificarVentaTest()  
   **Tipo:** Integración  
   **Descripción:** Verifica que una venta pueda actualizarse correctamente mediante PUT.  
   **Resultado:** Prueba ejecutada exitosamente.
* **Prueba:** eliminarVentaTest()  
   **Tipo:** Integración  
   **Descripción:** Evalúa la correcta eliminación de una venta existente.  
   **Resultado:** Prueba ejecutada exitosamente.

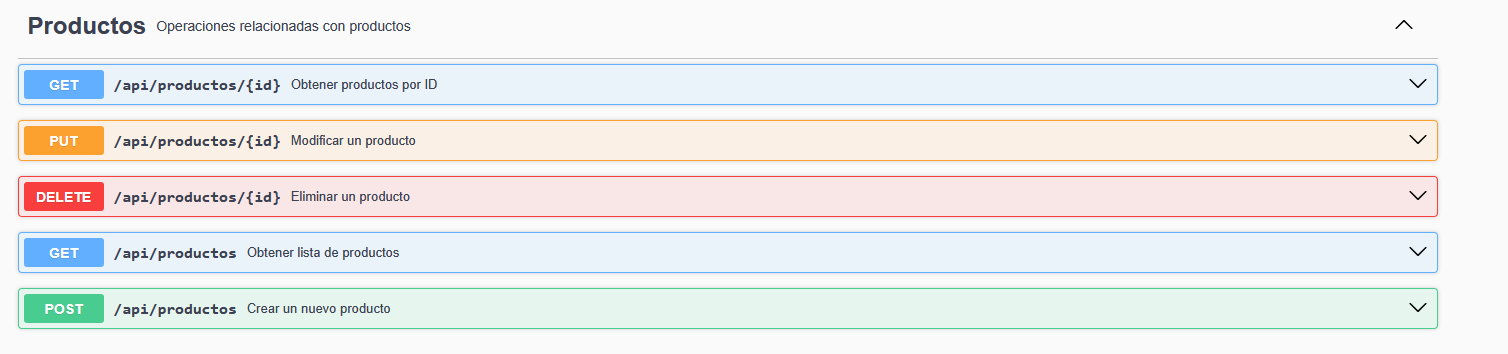
### Documentación y prueba manual con Swagger UI

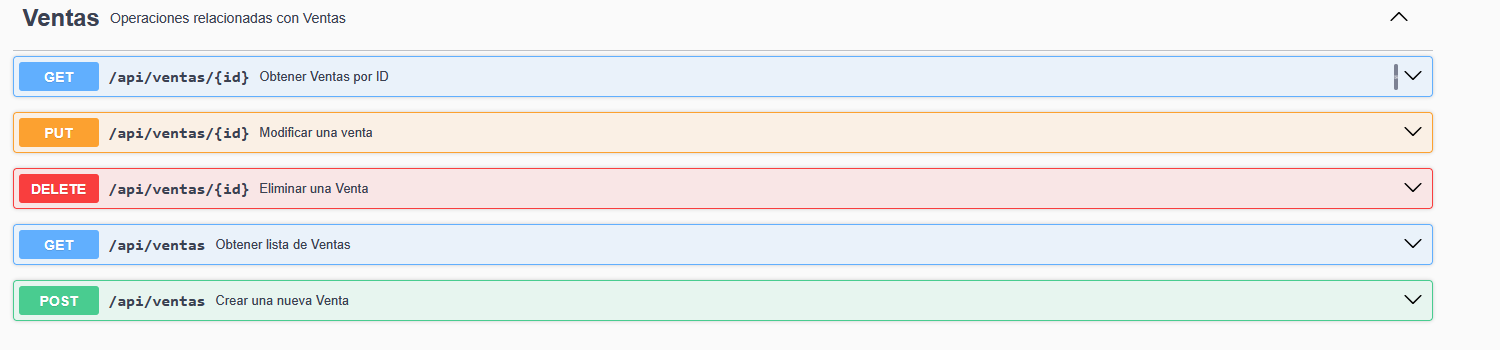
Durante el desarrollo del proyecto, también utilizamos **Swagger UI**(OpenAPI) Esta interfaz nos permitió visualizar y probar manualmente cada uno de los endpoints disponibles para el servicio de clientes, esto facilitó la verificación rápida del correcto funcionamiento.

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

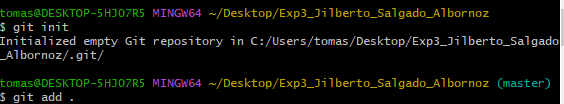






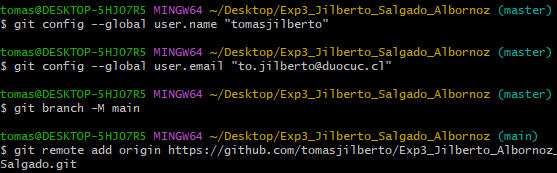
Uso de GITHUB

Se hizo utilización del Github tanto como para desarrollar de manera colaborativa el proyecto como para subirlo, a continuación dejaré las capturas de pantalla de los comandos que se utilizaron para subir el proyecto a github.

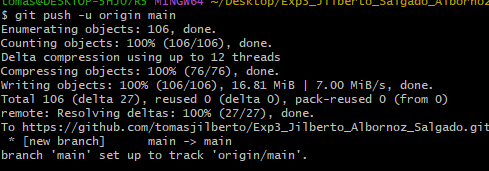


***Git init = inicializa el git en nuestra carpeta***

***git add . = es para añadir todos los archivos.***

******

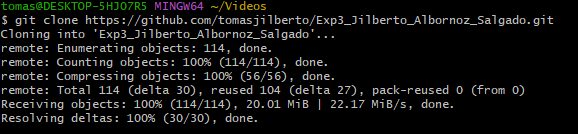
***Aca se usaron comandos para poder configurar la cuenta en la cual se subiran (posteriormente se utilizo un token)***

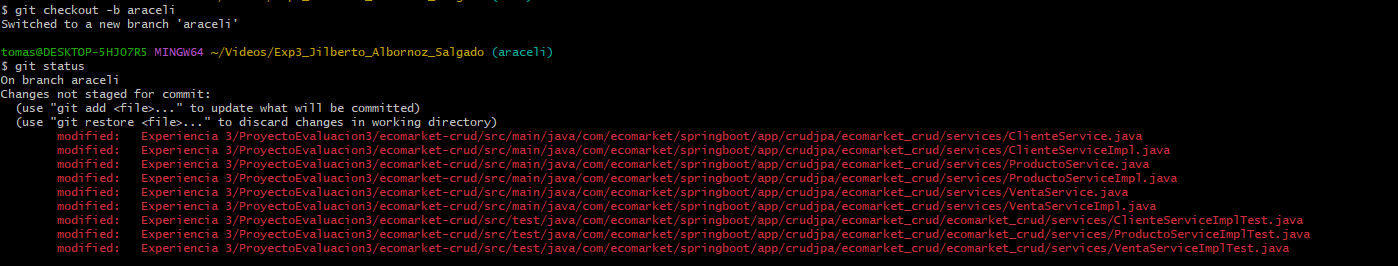
******

*Aca se muestra como se subieron todos los archivos*

Utilización de branchs con ejemplo

Se clonó el repositorio original desde GitHub a la computadora local utilizando el comando git clone, con el objetivo de trabajar de forma independiente y segura sobre el proyecto. Posteriormente, se creó una nueva branch llamada **araceli**, destinada a implementar y probar funcionalidades específicas asignadas a un integrante del equipo. De esta forma, se mantuvo separada la rama principal (main), evitando conflictos y permitiendo una integración controlada de los cambios al finalizar el desarrollo.







***Conclusión:***

El desarrollo del proyecto EcoMarket SPA nos ayudó significativamente a comprender y aplicar de forma práctica los conceptos fundamentales del desarrollo Full Stack. A lo largo del proceso, aprendimos a estructurar un sistema utilizando una arquitectura basada en microservicios, lo cual nos permitió dividir las funcionalidades en módulos independientes, facilitando tanto el mantenimiento como la escalabilidad del software.

Además, el uso de herramientas como JUnit 5, Mockito, Spring Boot Test y MockMvc nos permitió implementar pruebas unitarias e integradas que aseguraron el correcto funcionamiento de cada servicio, tanto de forma aislada como en conjunto. Esto reforzó nuestra capacidad para identificar errores, validar funcionalidades y garantizar una mayor calidad en el producto final.

En conjunto, este proyecto no solo fortaleció nuestras habilidades técnicas, sino también nuestra colaboración en equipo, la organización del código y el pensamiento crítico en la resolución de problemas reales. Nos llevamos una experiencia valiosa que sin duda aplicaremos en futuros desafíos del mundo del desarrollo de software.

El desarrollo del proyecto EcoMarket SPA nos ayudó significativamente a comprender y aplicar de forma práctica los conceptos fundamentales del desarrollo Full Stack. A lo largo del proceso, aprendimos a estructurar un sistema utilizando una arquitectura basada en microservicios, lo cual nos permitió dividir las funcionalidades en módulos independientes, facilitando tanto el mantenimiento como la escalabilidad del software.

Además, el uso de herramientas como JUnit 5, Mockito, Spring Boot Test y MockMvc nos permitió implementar pruebas unitarias e integradas que aseguraron el correcto funcionamiento de cada servicio, tanto de forma aislada como en conjunto. Esto reforzó nuestra capacidad para identificar errores, validar funcionalidades y garantizar una mayor calidad en el producto final.

En conjunto, este proyecto no solo fortaleció nuestras habilidades técnicas, sino también nuestra colaboración en equipo, la organización del código y el pensamiento crítico en la resolución de problemas reales. Nos llevamos una experiencia valiosa que sin duda aplicaremos en futuros desafíos del mundo del desarrollo de software.