**Caso 1: Perfulandia SPA**

**Integrantes:** Gabriel Avendaño – Jeremy Cárcamo – Camila Pino

**Asignatura**: Desarrollo FullStack I

**Profesor**: Eduardo Baeza

**Sección:** 010D

**Carrera**: Analista Programador Computacional

**Fecha de entrega:** 23 de Junio de 2025

**Tabla de contenidos**

Contenido

[1. Introducción 3](#_Toc201441103)

[2. Diagrama de arquitectura de microservicios 4](#_Toc201441104)

[3. Plan de Pruebas 5](#_Toc201441105)

[3.1 Pruebas Unitarias 6](#_Toc201441106)

[3.2 Pruebas de Integración 12](#_Toc201441107)

[4. Ejecución de Pruebas 20](#_Toc201441108)

[4.1 Pruebas Unitarias 20](#_Toc201441109)

[4.2 Pruebas de Integración 20](#_Toc201441110)

[4.3 Documentación con Open Api 21](#_Toc201441111)

[23](#_Toc201441112)

[5. Comandos Git-GitHub 26](#_Toc201441113)

[5.1 Anteriormente usados 26](#_Toc201441114)

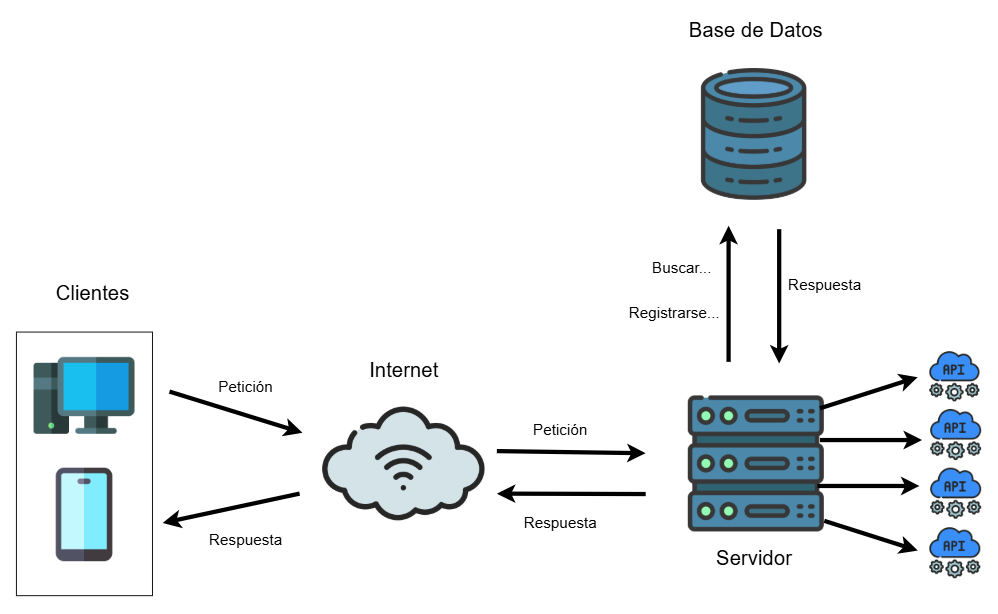
[5.2 Comandos nuevos 28](#_Toc201441115)

[6. Conclusión 30](#_Toc201441116)

# 1. Introducción

El presente informe expone el desarrollo e implementación de una arquitectura basada en microservicios para el sistema de gestión de Perfulandia SPA, una empresa en constante crecimiento que requiere soluciones tecnológicas escalables y adaptables. Para ello, se diseñó una estructura modular soportada en Spring Boot y MySQL, complementada con un enfoque sólido en pruebas unitarias e integración, lo que garantiza la calidad y confiabilidad del software. Además, se documenta el uso de herramientas de control de versiones como Git y GitHub, así como la automatización de pruebas mediante frameworks como JUnit 5, Mockito y MockMvc. Este informe refleja las buenas prácticas adoptadas durante el desarrollo y el trabajo colaborativo del equipo.

# 2. Diagrama de arquitectura de microservicios



El sistema se divide en distintos microservicios, cada uno es independiente del otro, lo que permite que puedan escalar a ritmos diferentes dependiendo de la demanda o el consumo de cada uno. En este sistema utilizamos 4 microservicios, el microservicio de **productos** (el símil a un microservicio de inventario), el microservicio de **boletas** (para llevar un control de las ventas), el microservicio de **envíos** (permite conocer el estado de los envíos de productos) y por último el microservicio de **usuarios** (para gestionar todo acerca de los usuarios del sistema).

## 

## 

## 

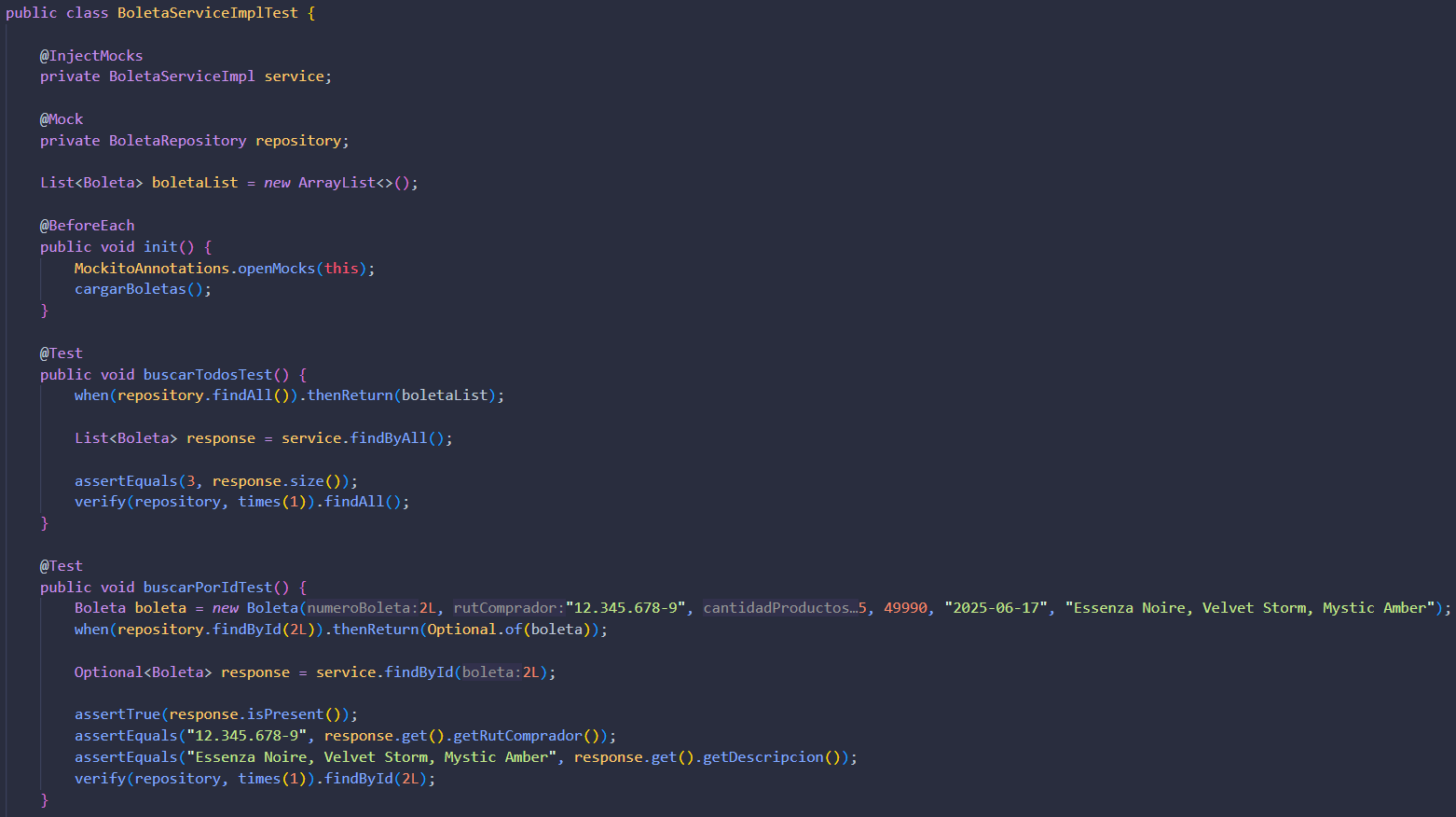
# 3. Plan de Pruebas

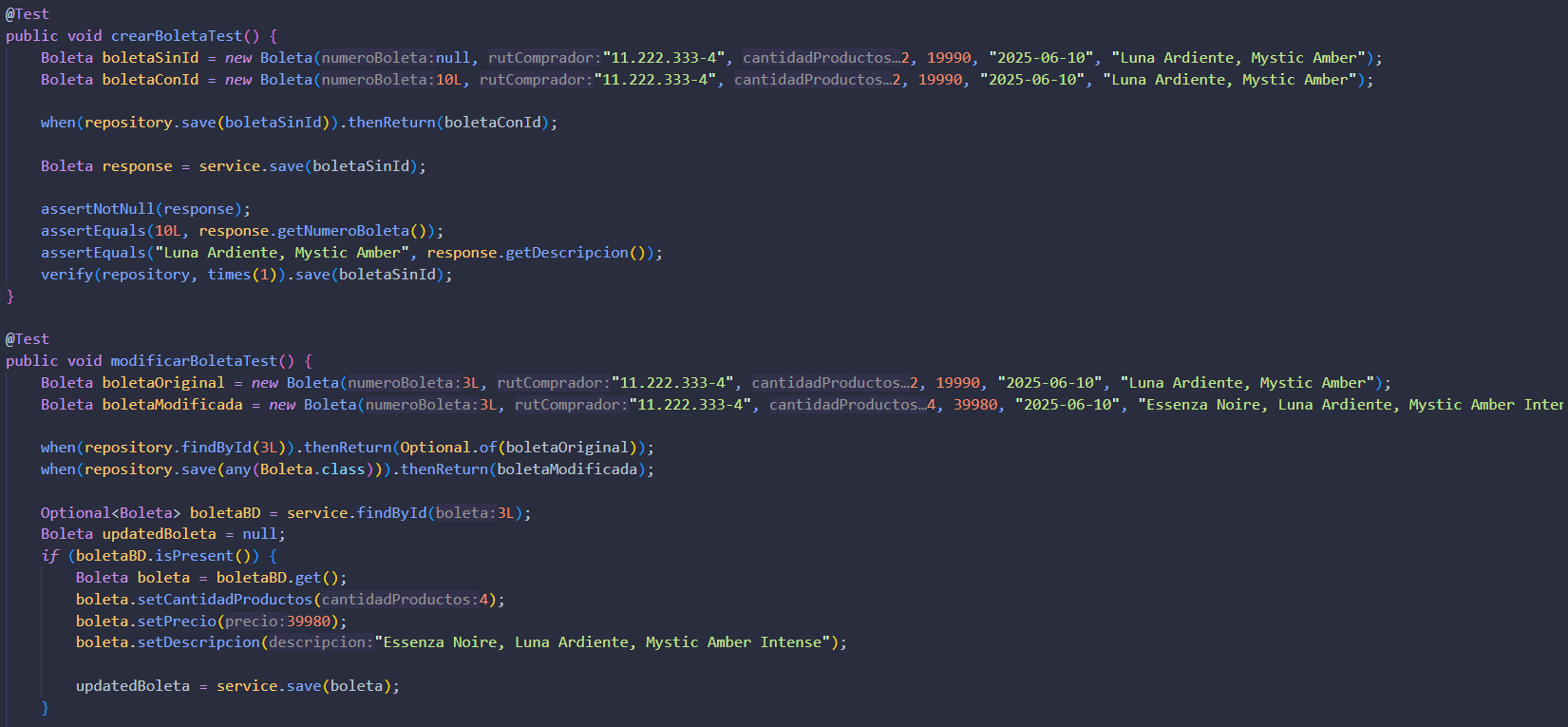
Herramientas y Frameworks utilizados:

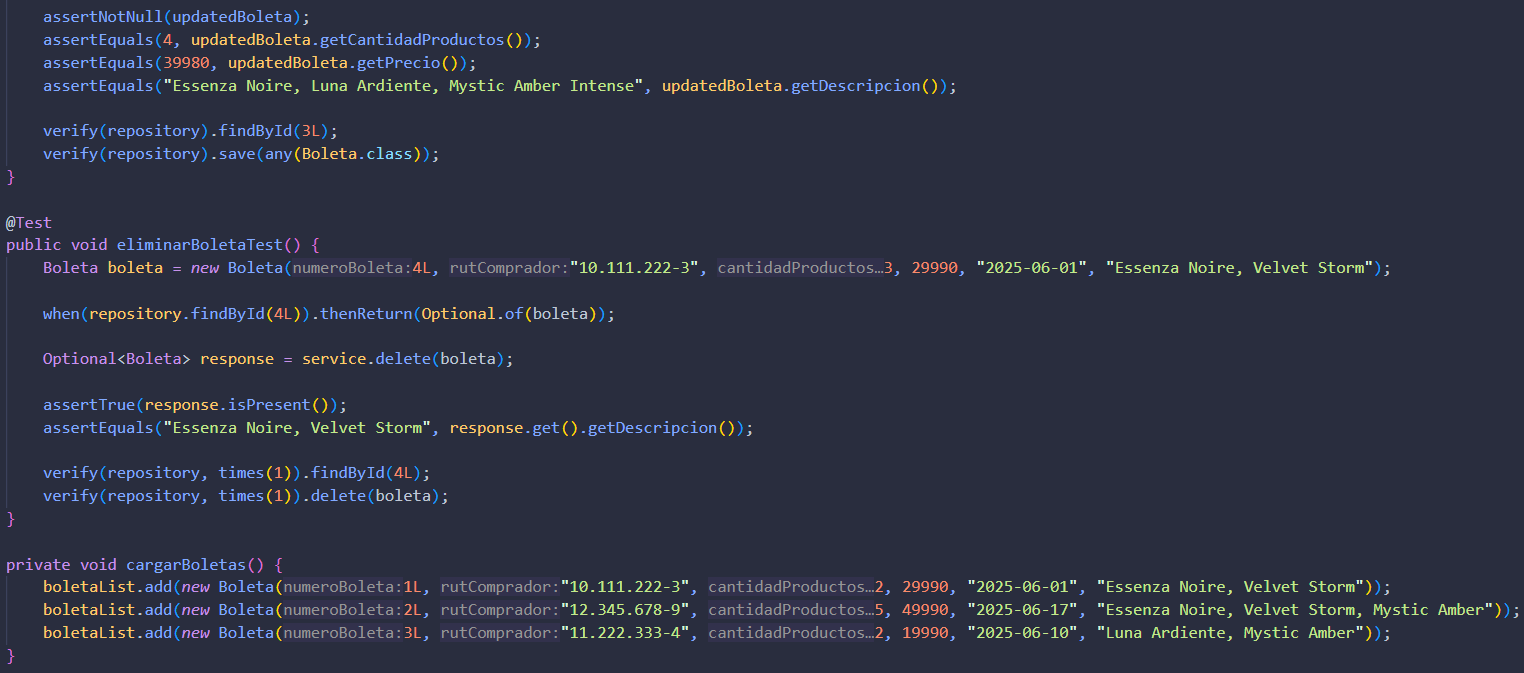
* JUnit 5:Se utilizó como framework principal para definir y ejecutar pruebas. Permite marcar métodos de prueba con la anotación @Test y organizar las pruebas de manera estructurada y automática.
* Spring Boot Test:Mediante las anotaciones @SpringBootTest y @AutoConfigureMockMvc, se cargó el contexto completo de la aplicación, permitiendo realizar pruebas integradas que simulan el comportamiento real del sistema.
* MockMvc:Esta herramienta permitió simular peticiones HTTP a los controladores (como GET o POST) sin necesidad de levantar el servidor. Fue útil para verificar respuestas y estados HTTP esperados.
* Mockito:Se empleó para simular servicios internos usando when() y thenReturn(), evitando así conexiones reales a bases de datos u otras dependencias. Esto facilita pruebas más controladas y predecibles.
* ObjectMapper:Se utilizó para convertir objetos Java en JSON y viceversa, lo que fue esencial para probar controladores que manejan datos en formato JSON.

## 3.1 Pruebas Unitarias

ServiceImplTest de Boleta

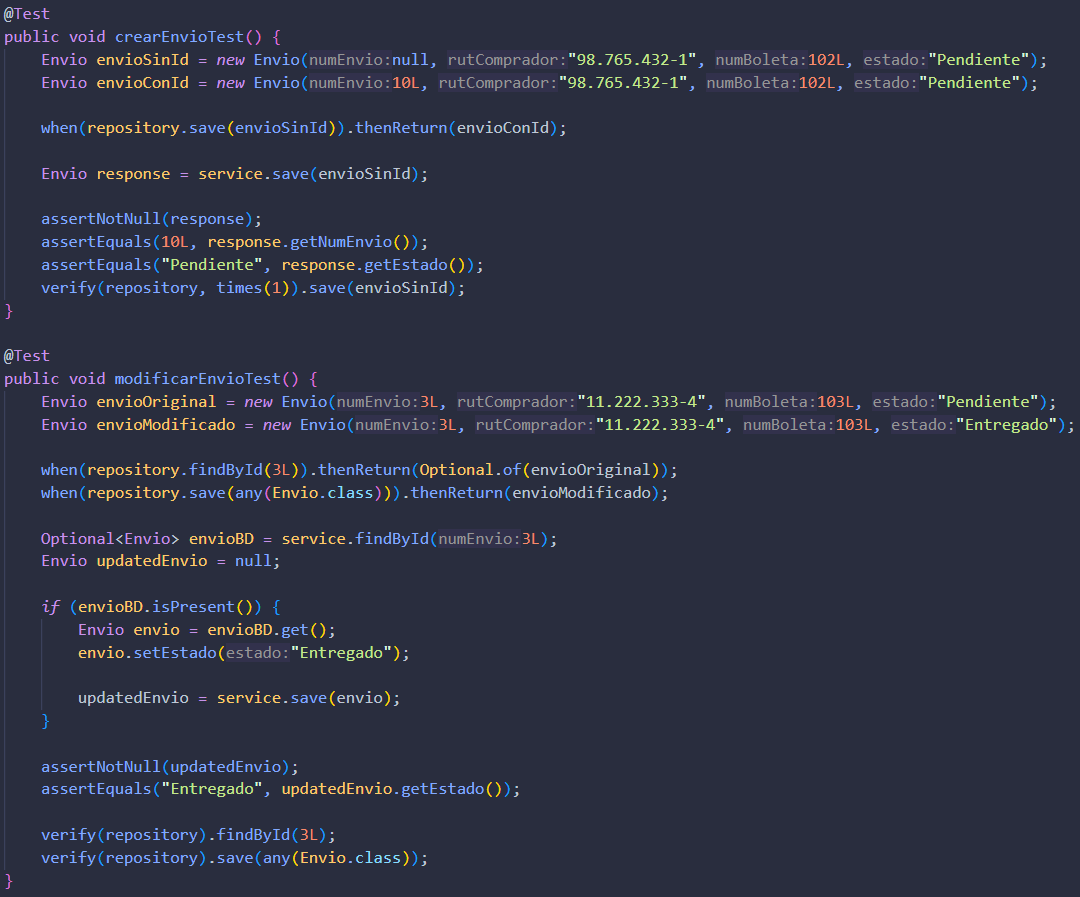






* ServiceImplTest de Envío

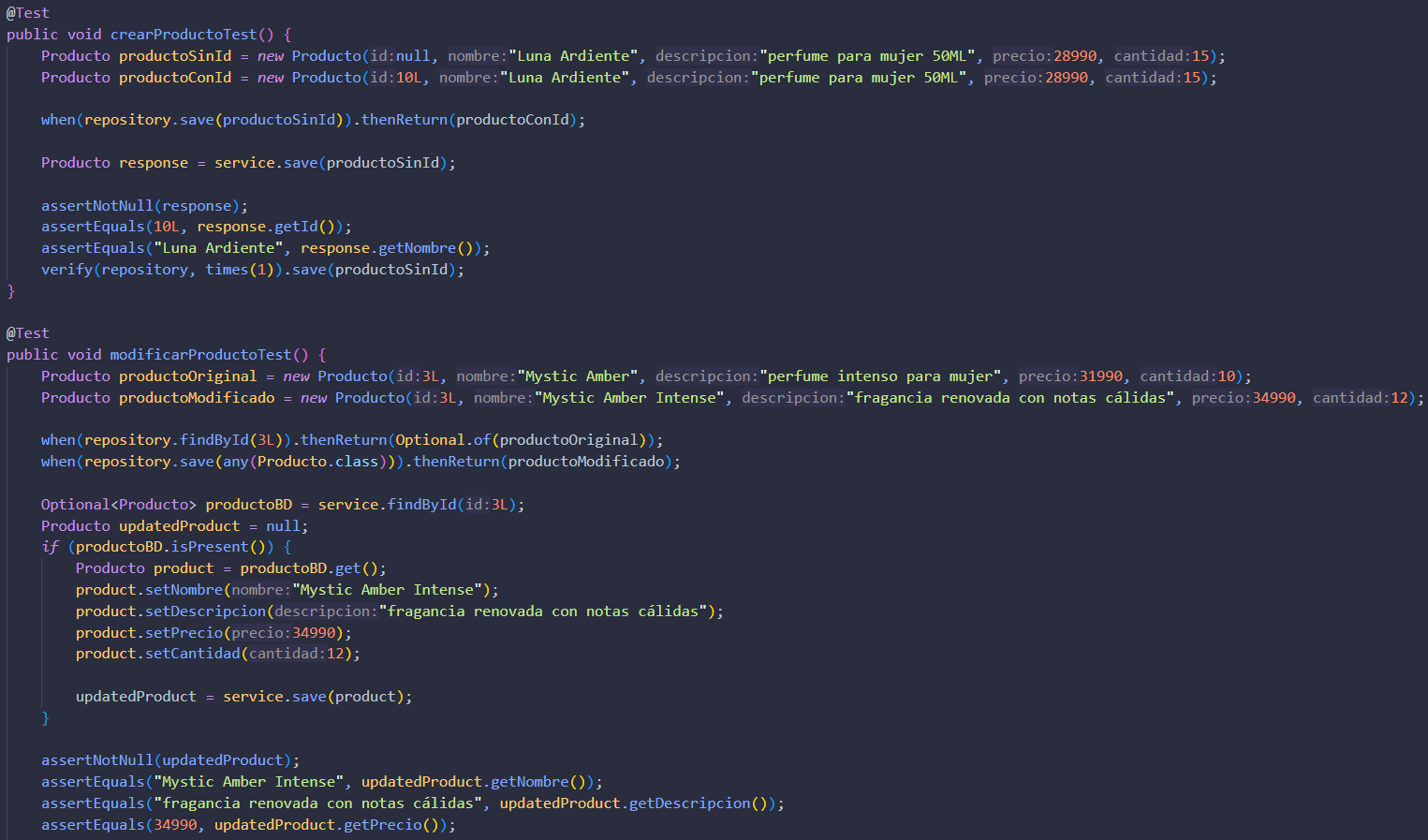






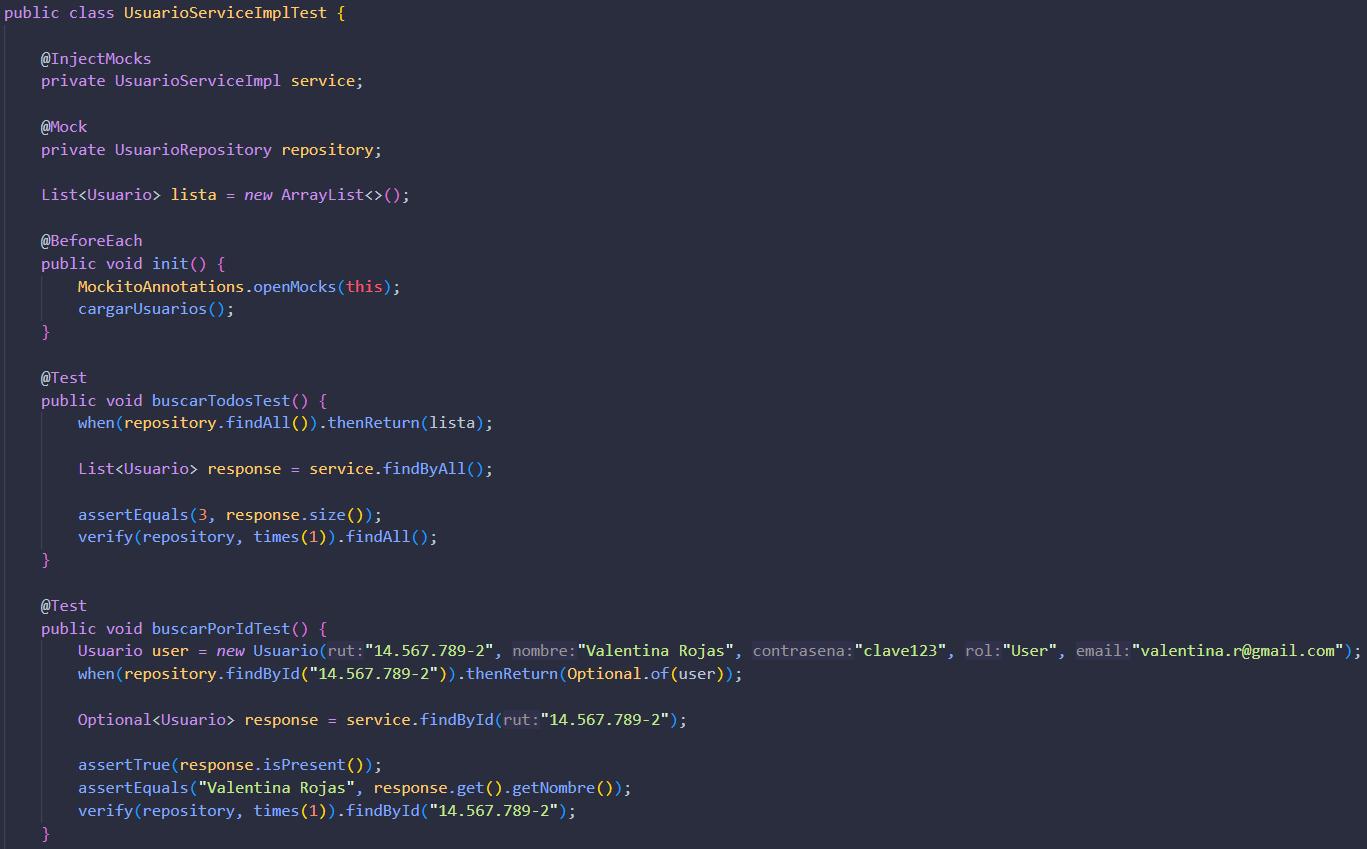
* ServiceImplTest de Producto

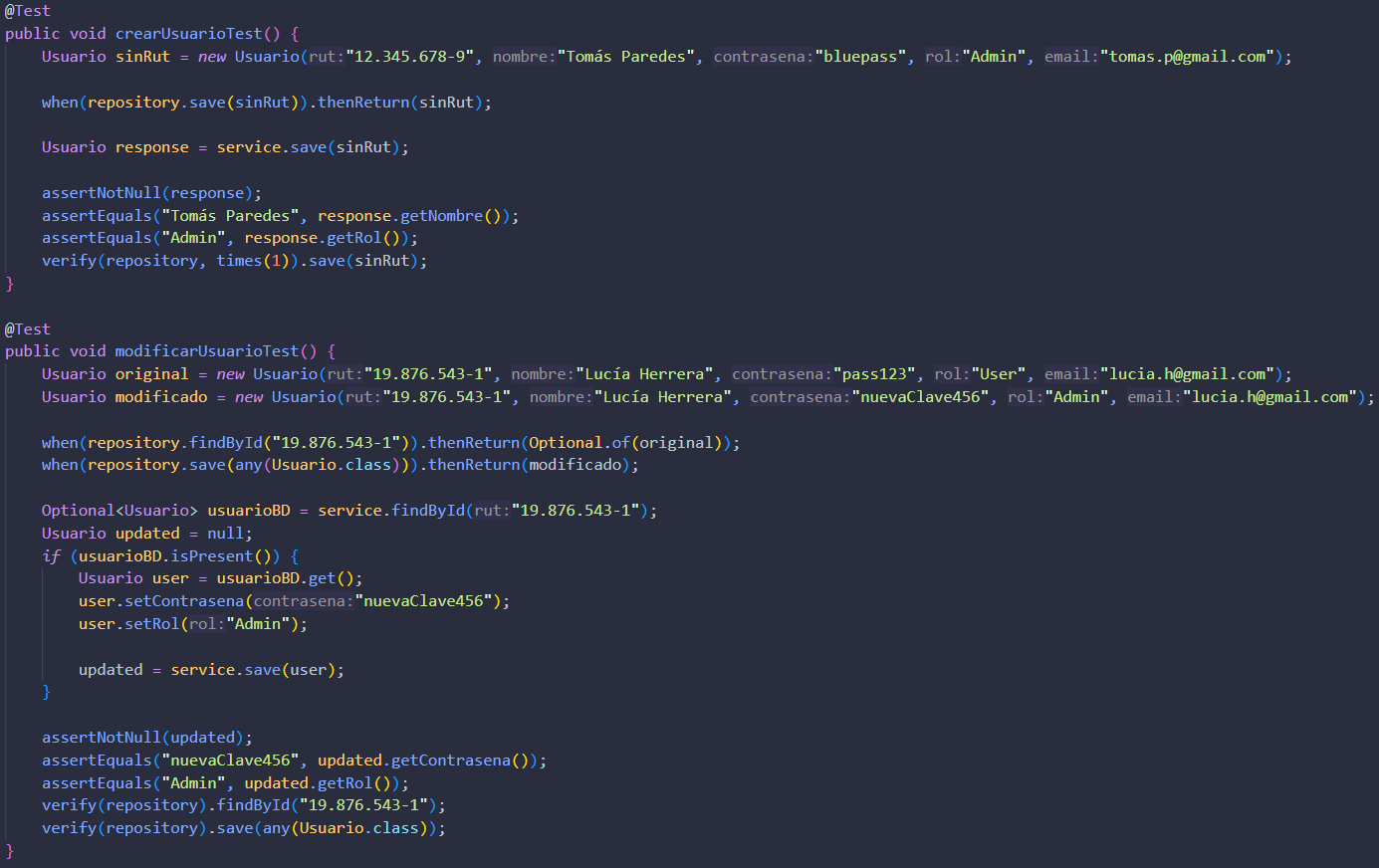


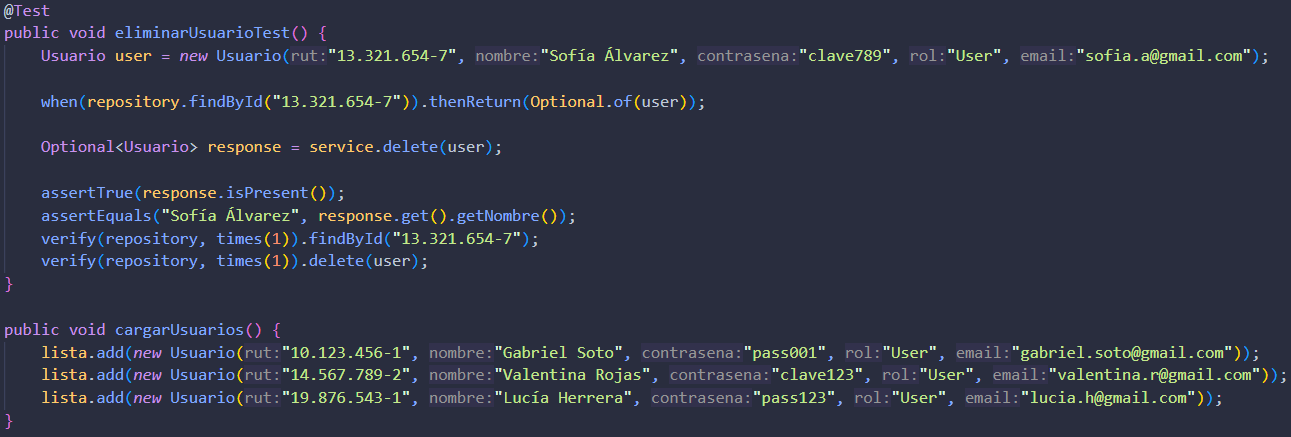




* ServiceImplTest de Usuario

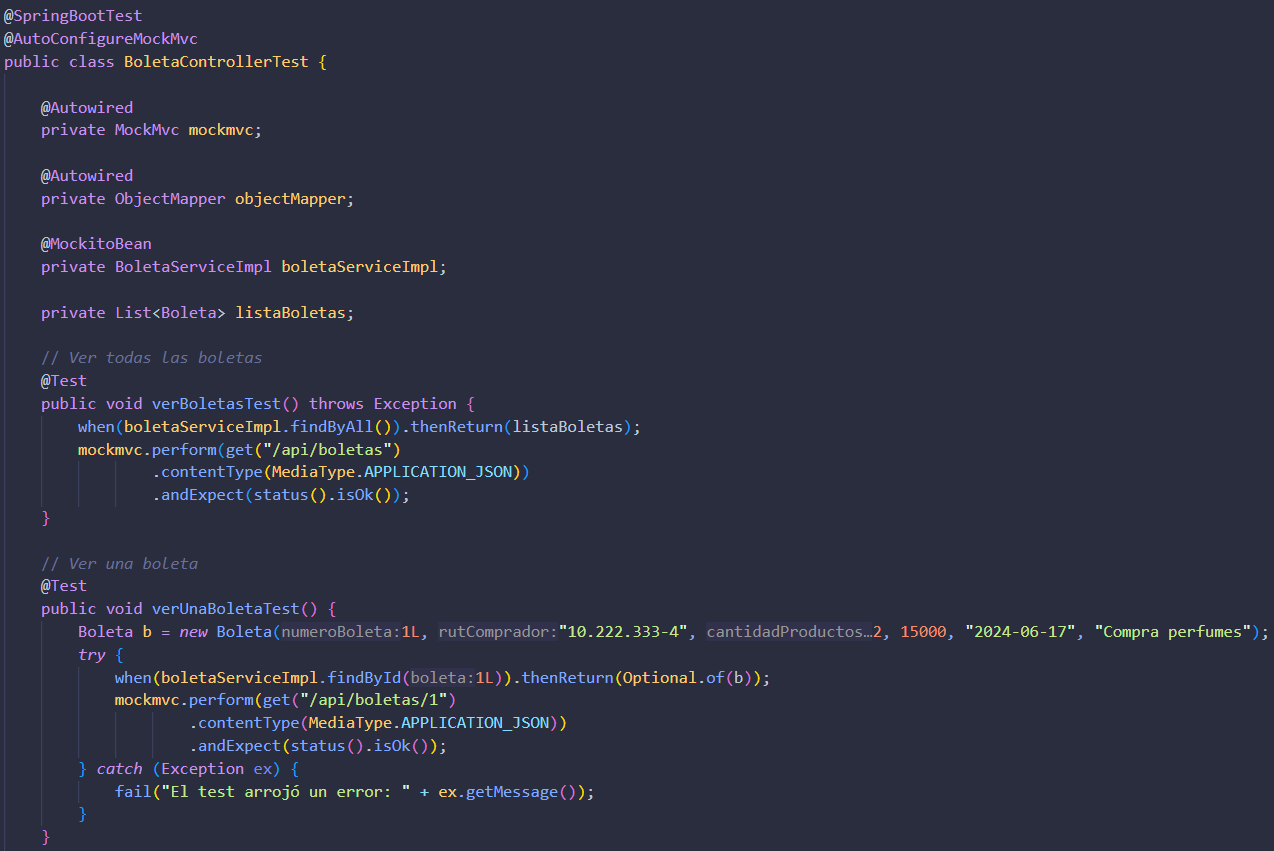


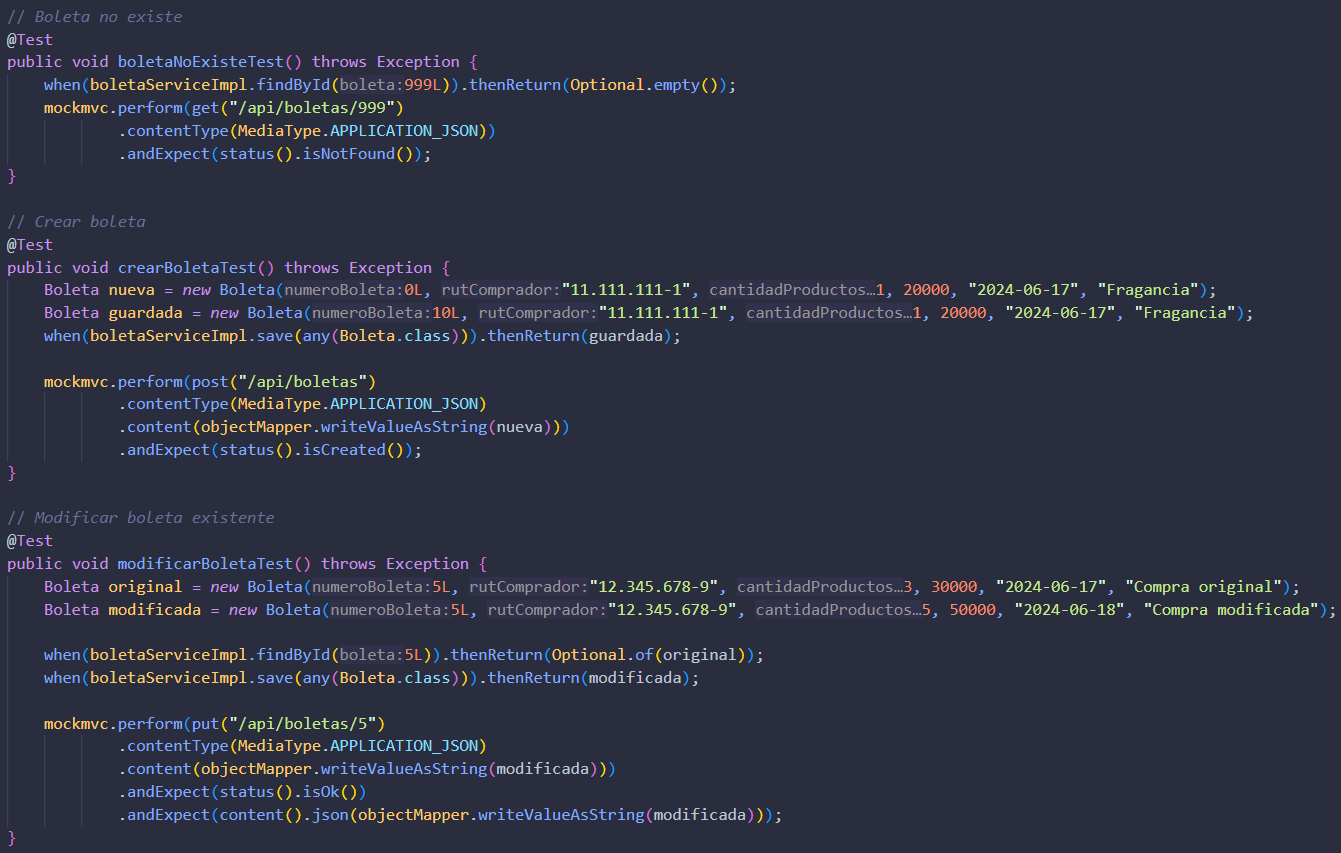




## 3.2 Pruebas de Integración

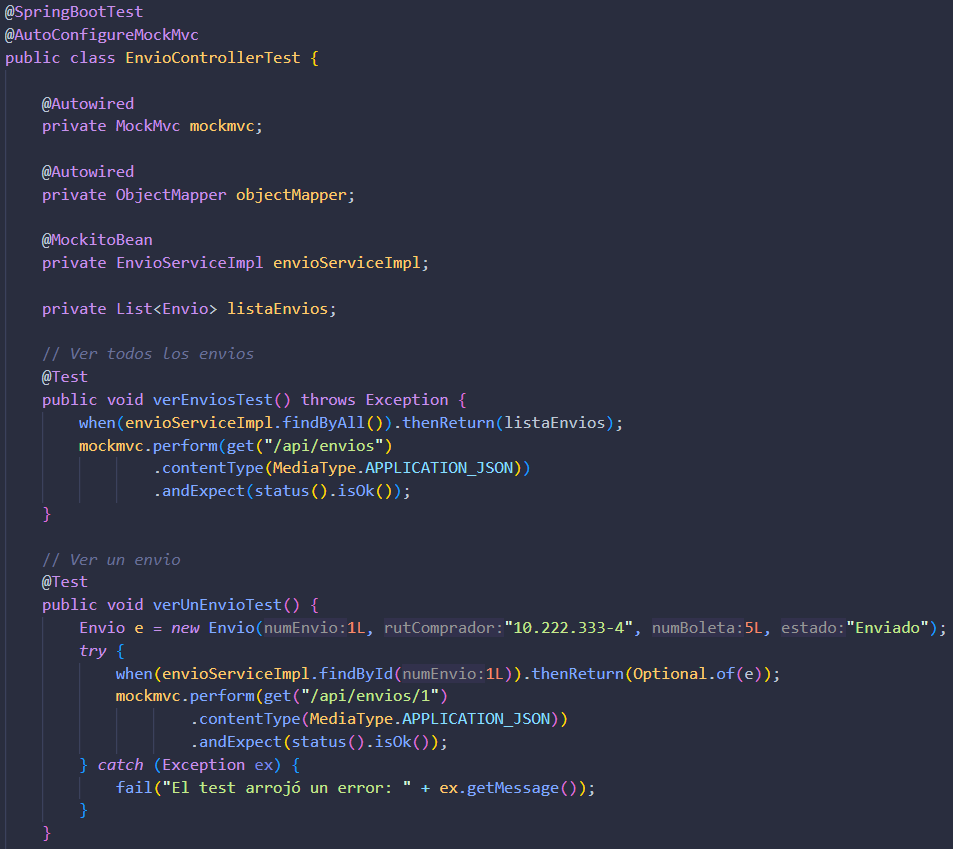
* RestControllerTest de Boleta

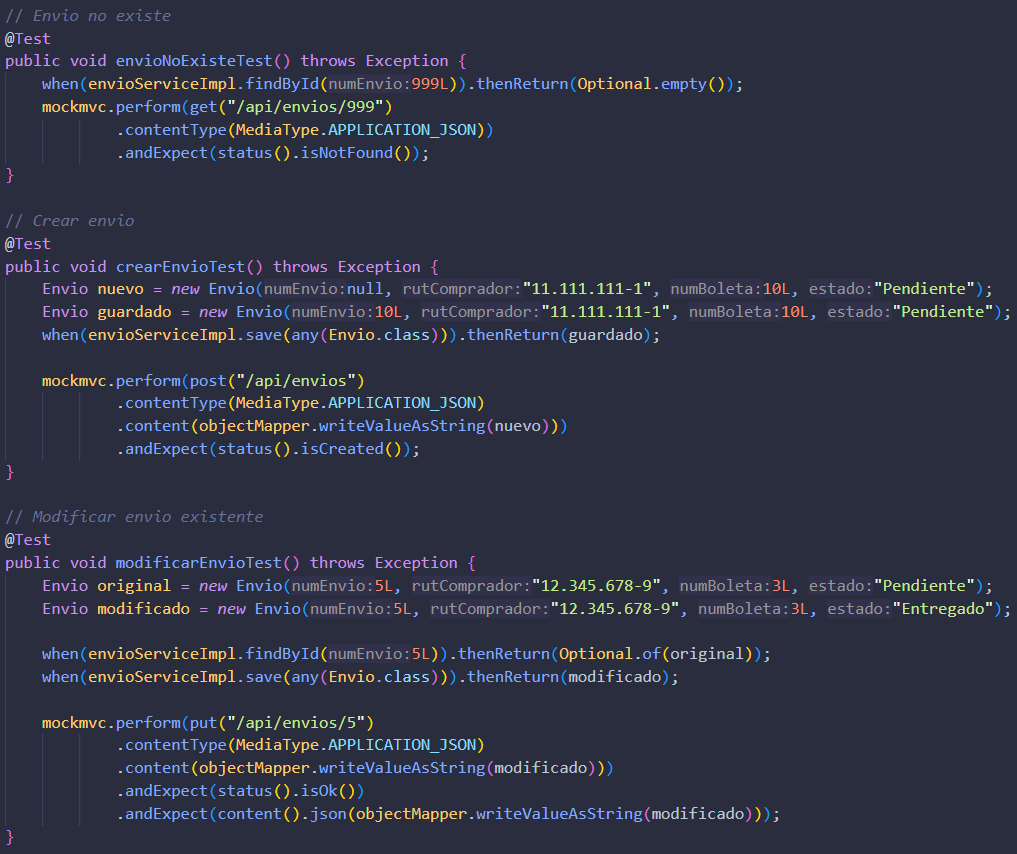






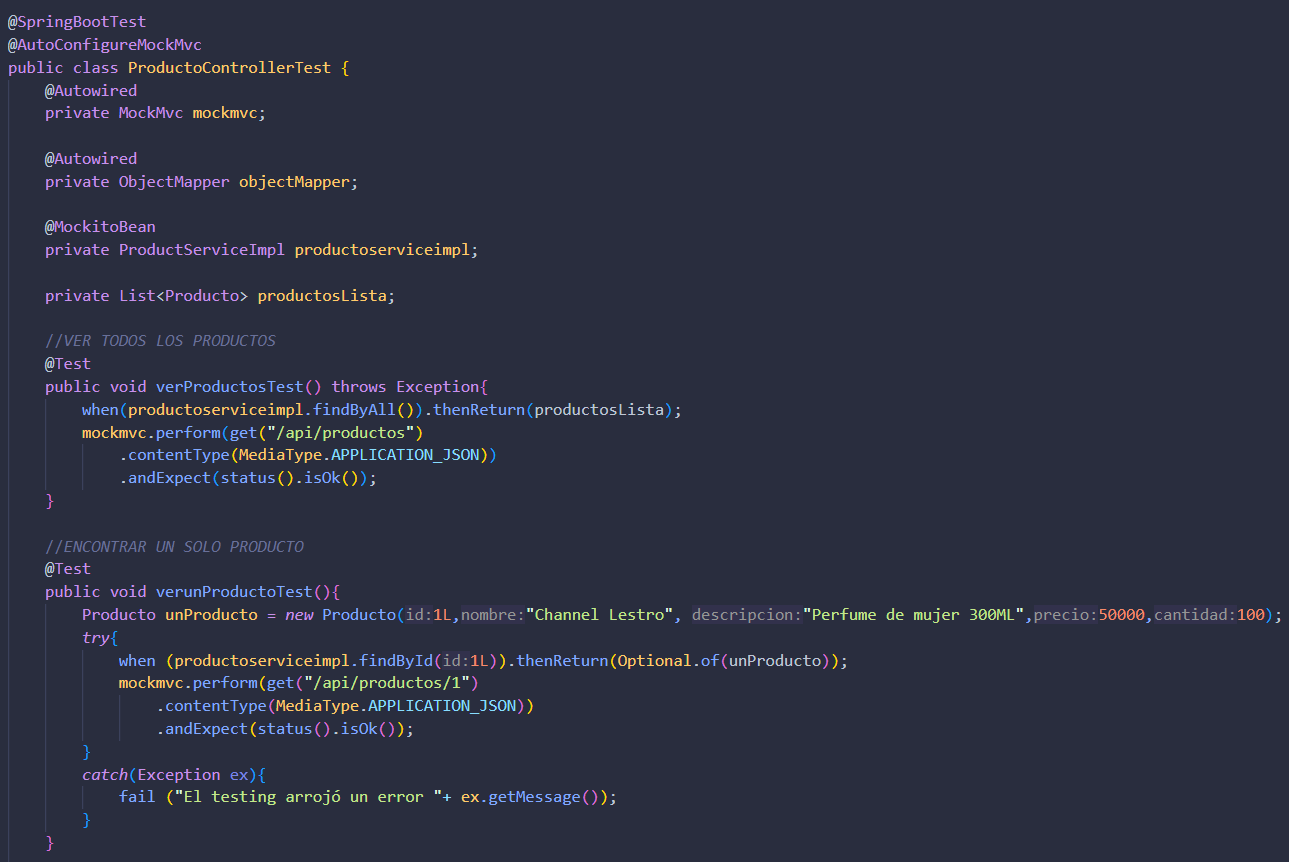
* RestControllerTest de Envío

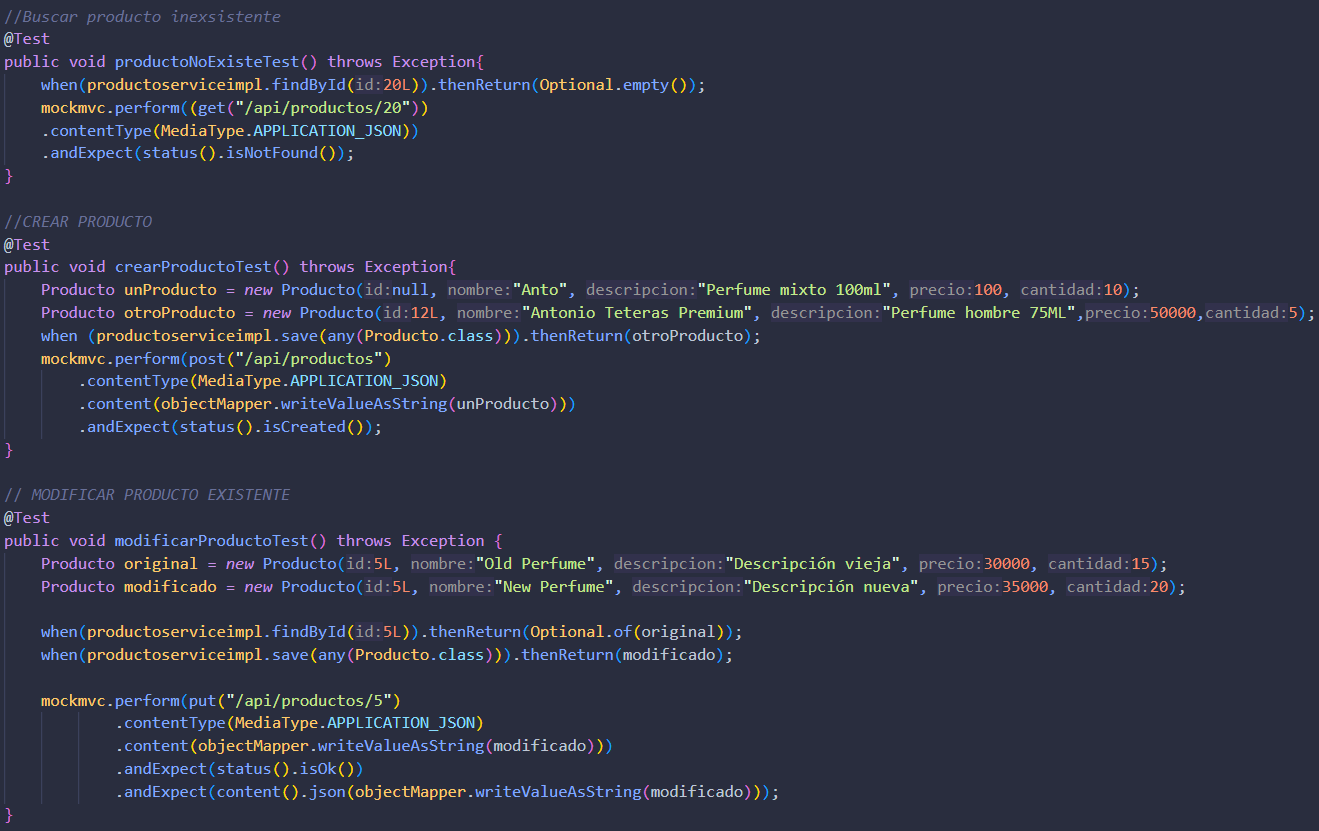






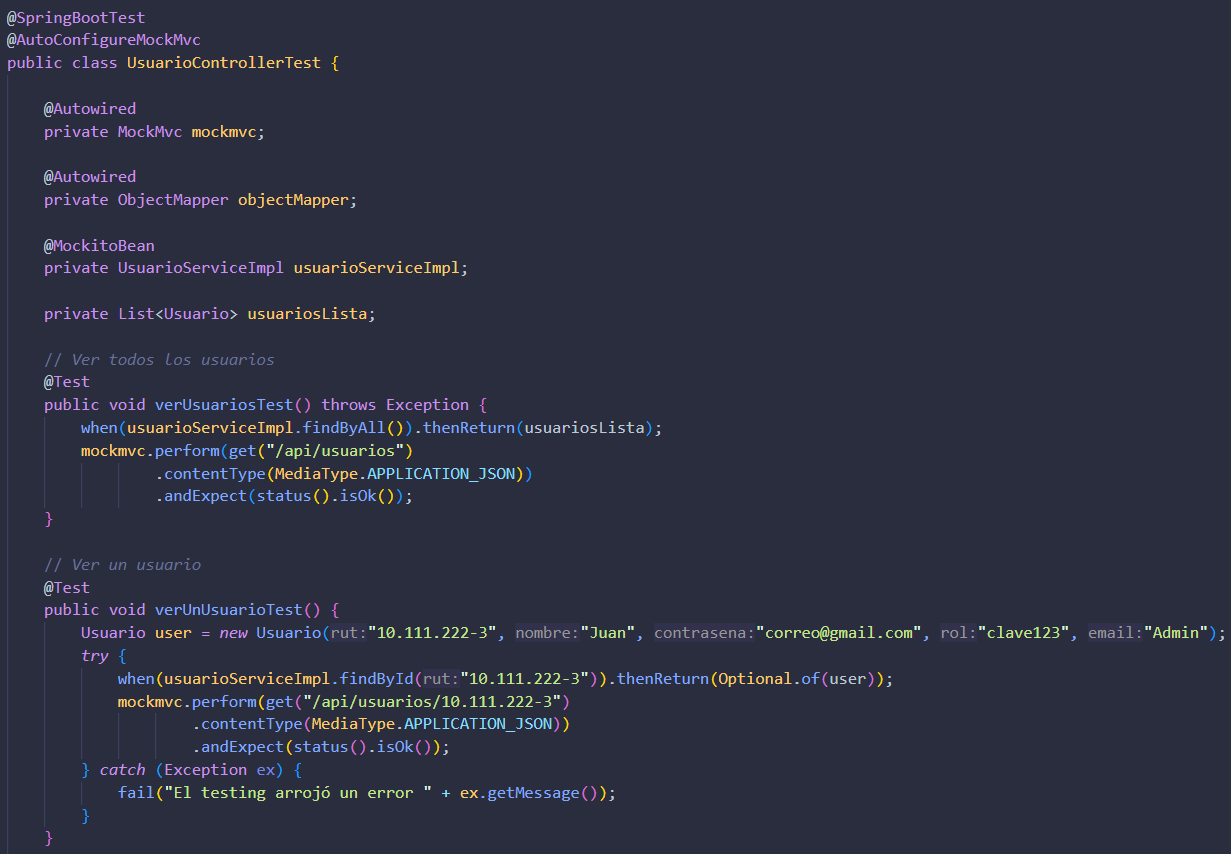
* RestControllerTest de Producto

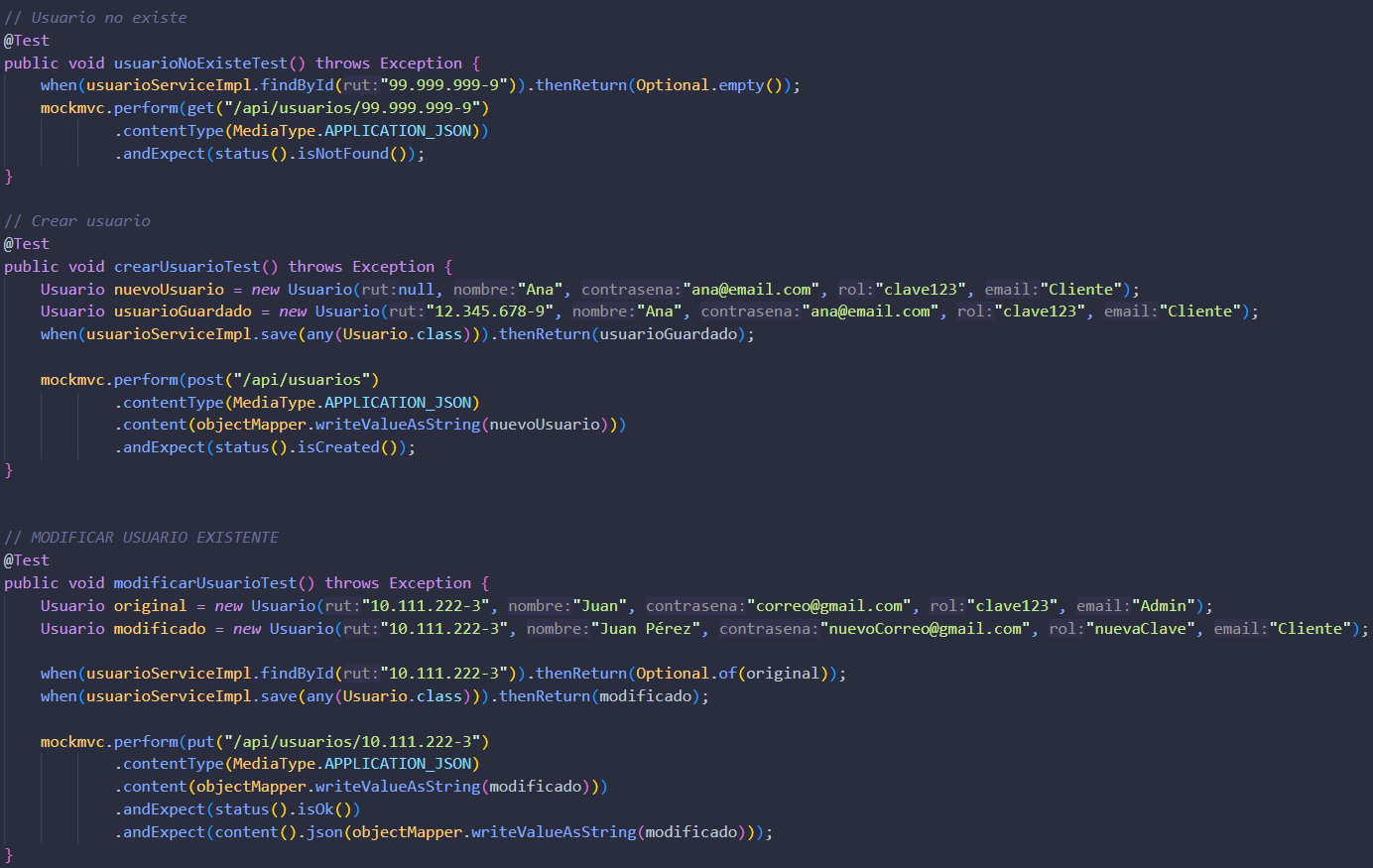


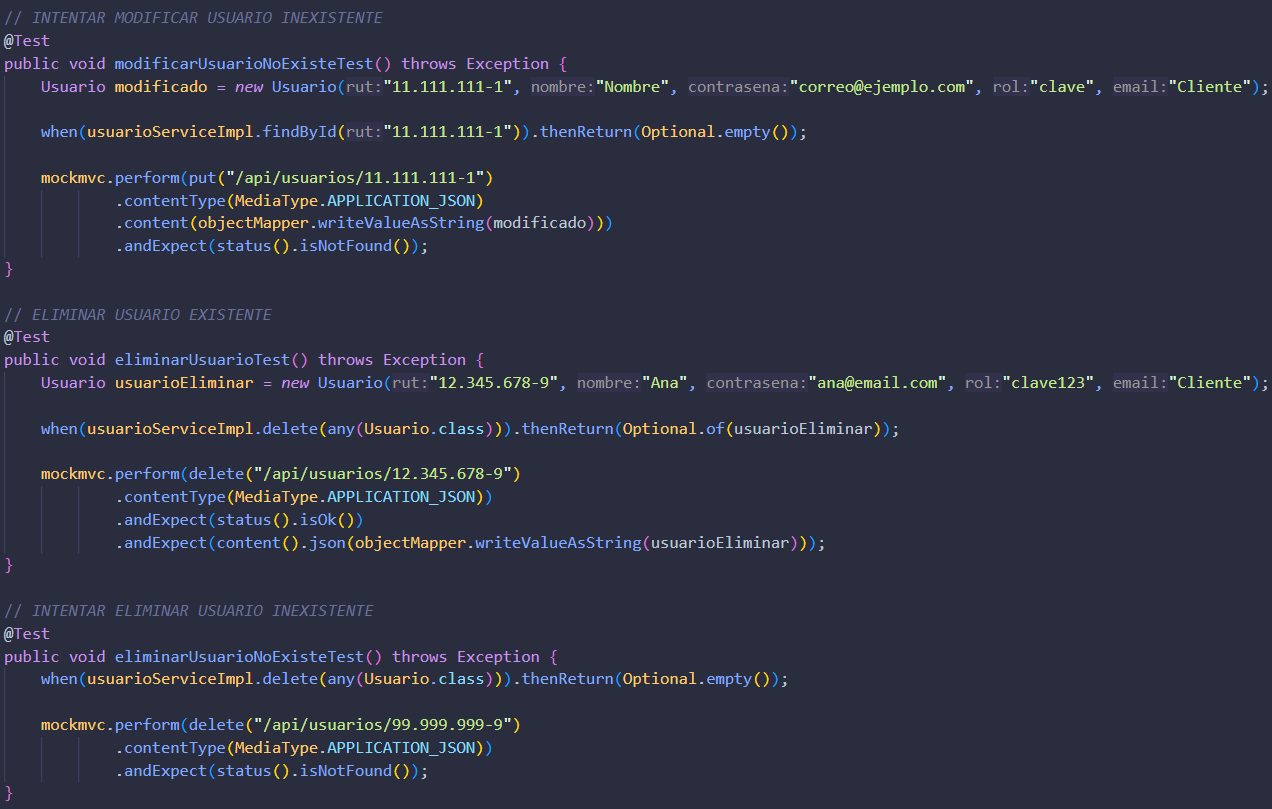




* RestControllerTest de Usuario

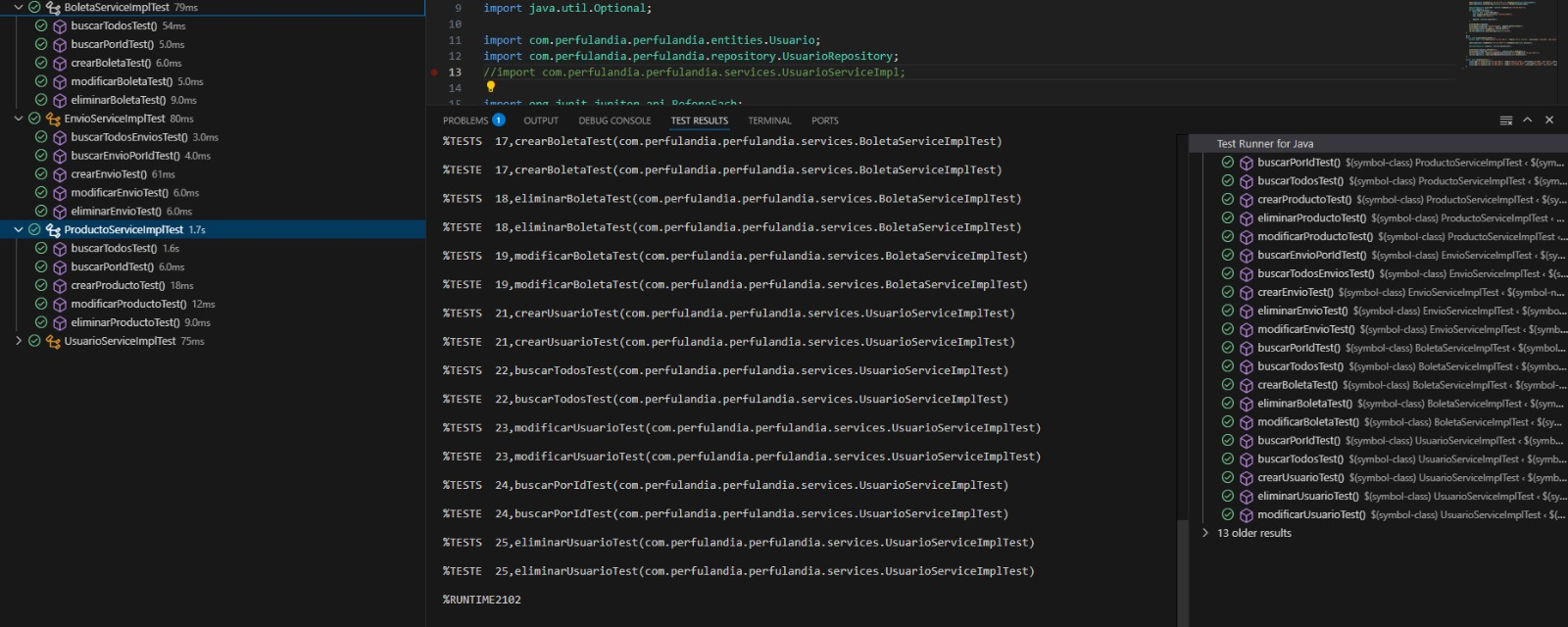






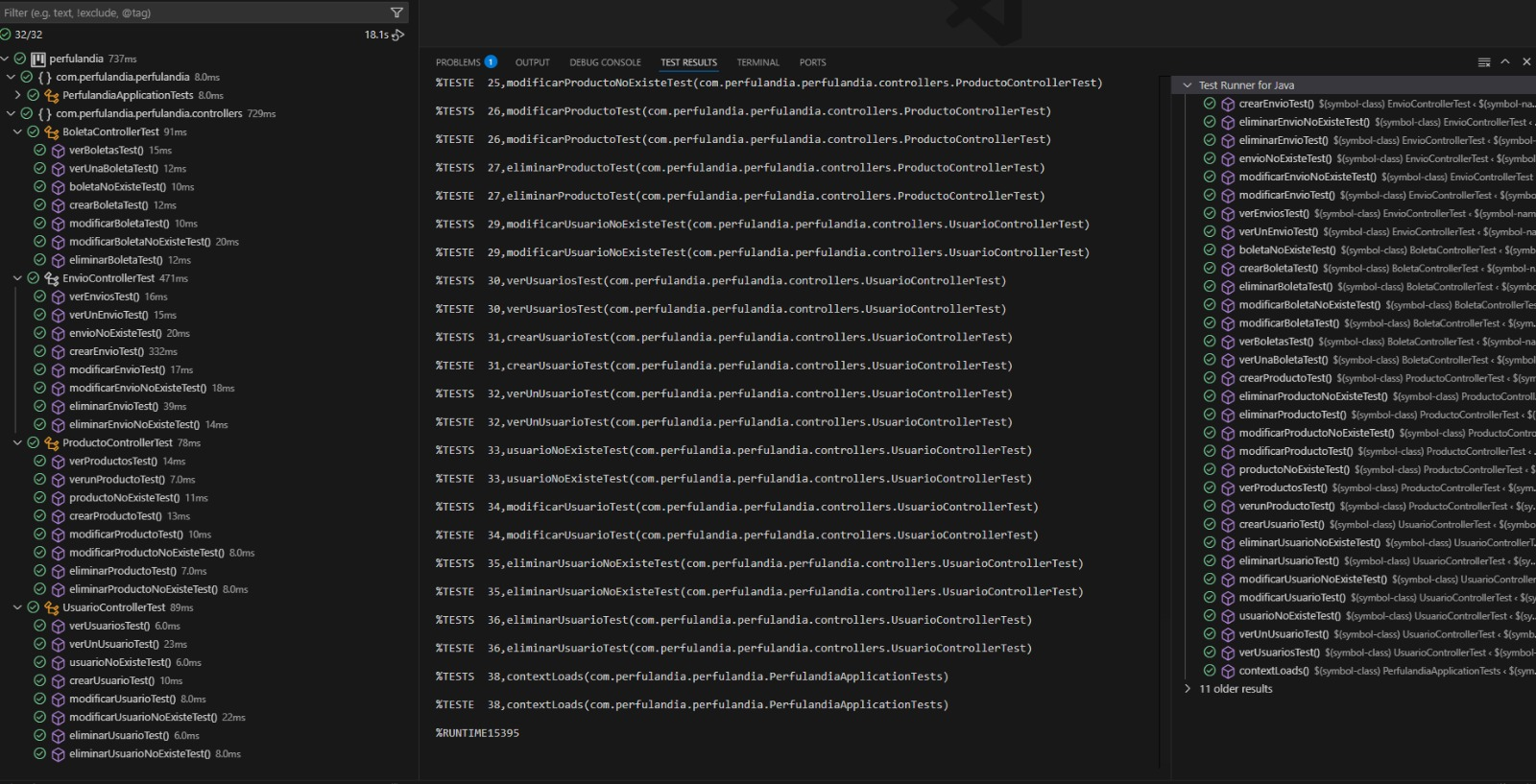
# 4. Ejecución de Pruebas

## 4.1 Pruebas Unitarias



En la imagen se logra apreciar las pruebas realizadas con éxito con cada servicio.

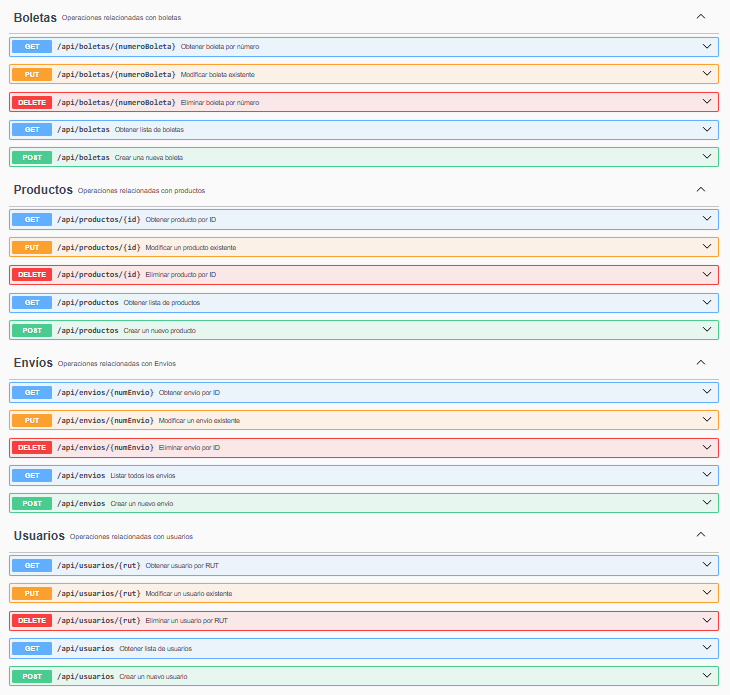
## 4.2 Pruebas de Integración



La imagen presenta los resultados realizados de forma satisfactoria con la conexión de la base de datos.

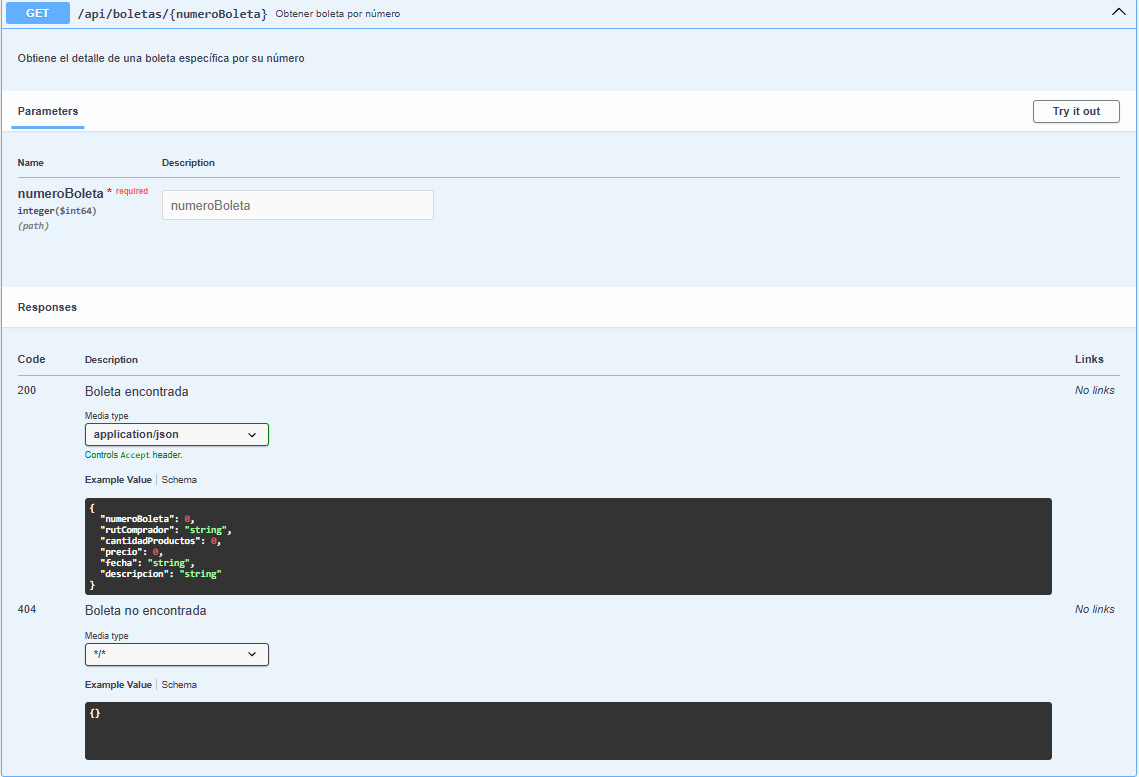
## 4.3 Documentación con Open Api

Documentación de todos los métodos creados para todos los microservicios



La documentación de métodos es similar para todos los microservicios, por lo que solo se adjuntan imágenes de los métodos del microservicio de boletas.

Documentación método verDetalle() (findById)

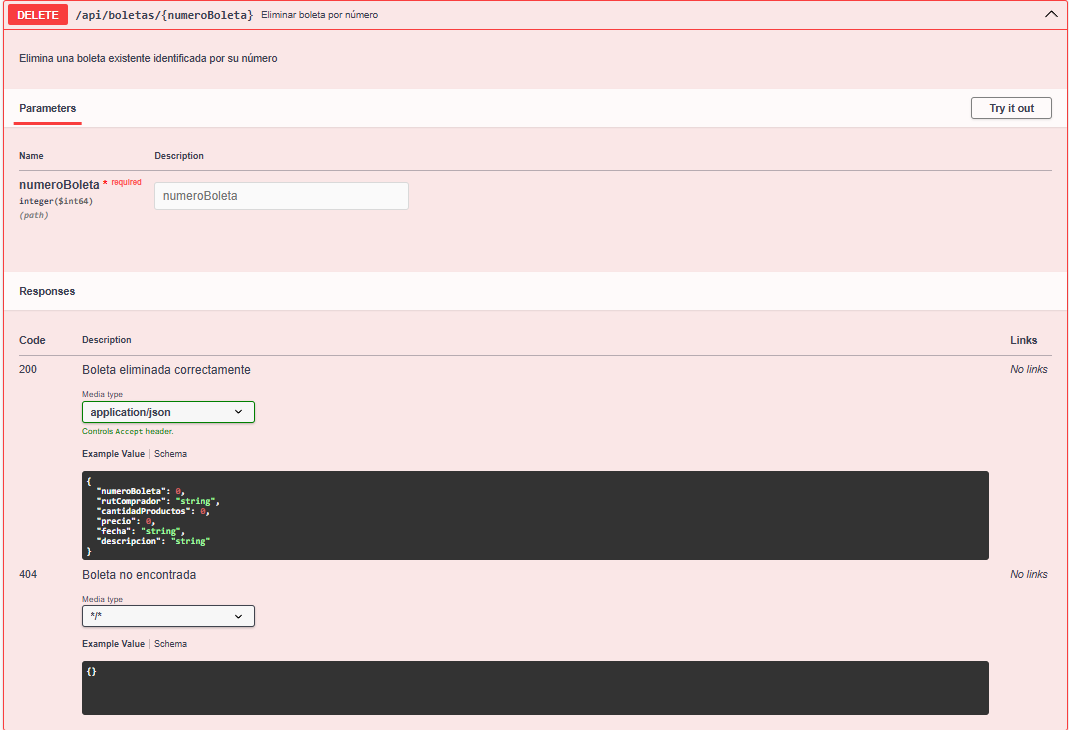


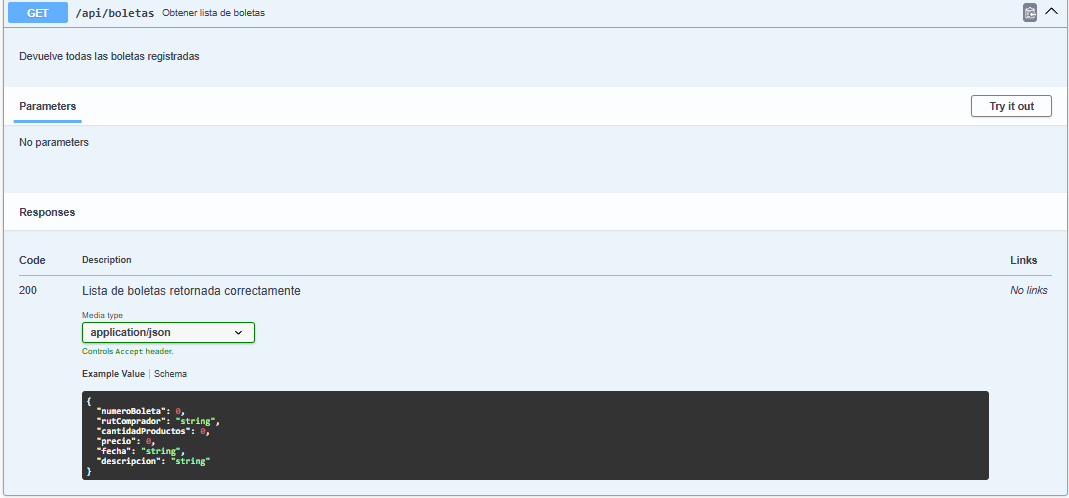
Documentación metodo modificar()

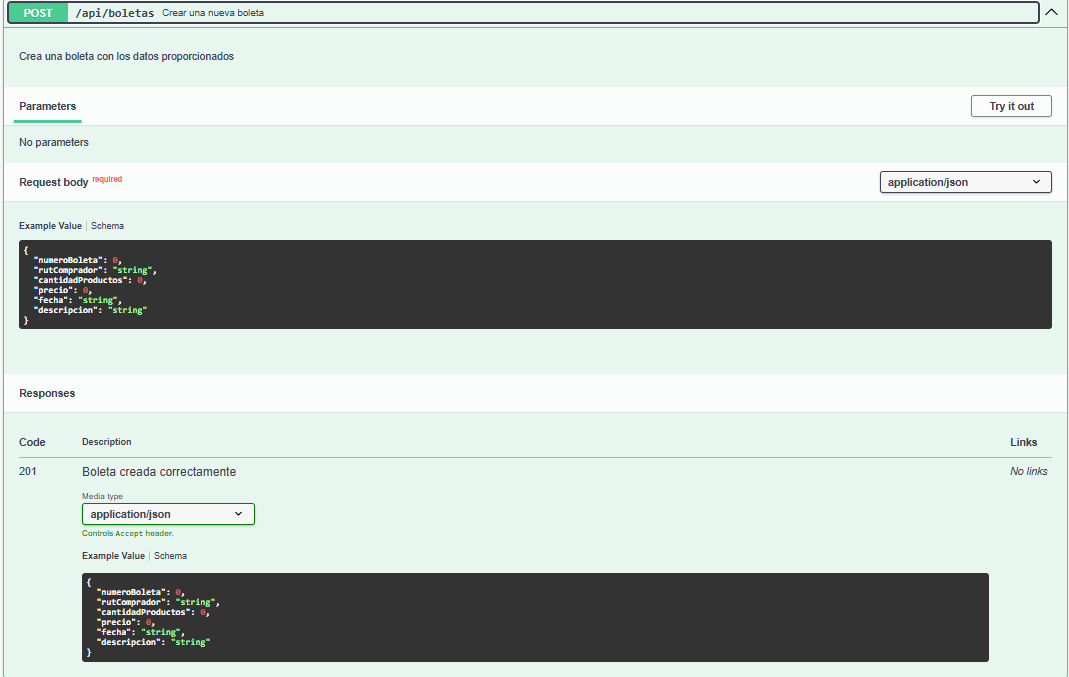
## 

## 

Documentación metodo eliminar()



Documentación método list() (findByAll)

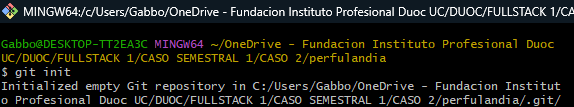
Documentación metodo crear() (save)

## 

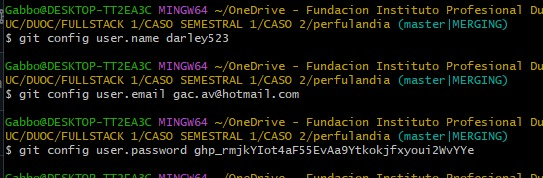
# 5. Comandos Git-GitHub

## 5.1 Anteriormente usados

Comando para inicializar un repositorio de git en local.



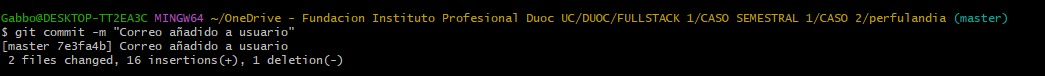
Comando para identificarse al momento de hacer los commit, agregando una contraseña para el repositorio online.



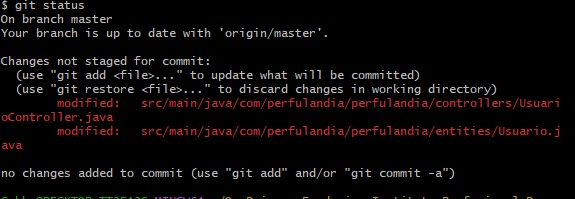
Comando para añadir todos los archivos creados o modificados al repositorio.



Comando para guardar una “versión/snapshot” del repositorio (local).



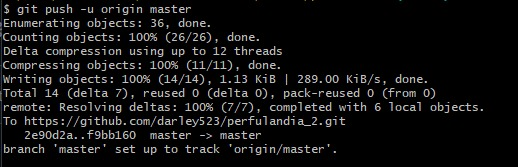
Comando para comprobar que cosas no han sido guardadas en el repositorio.



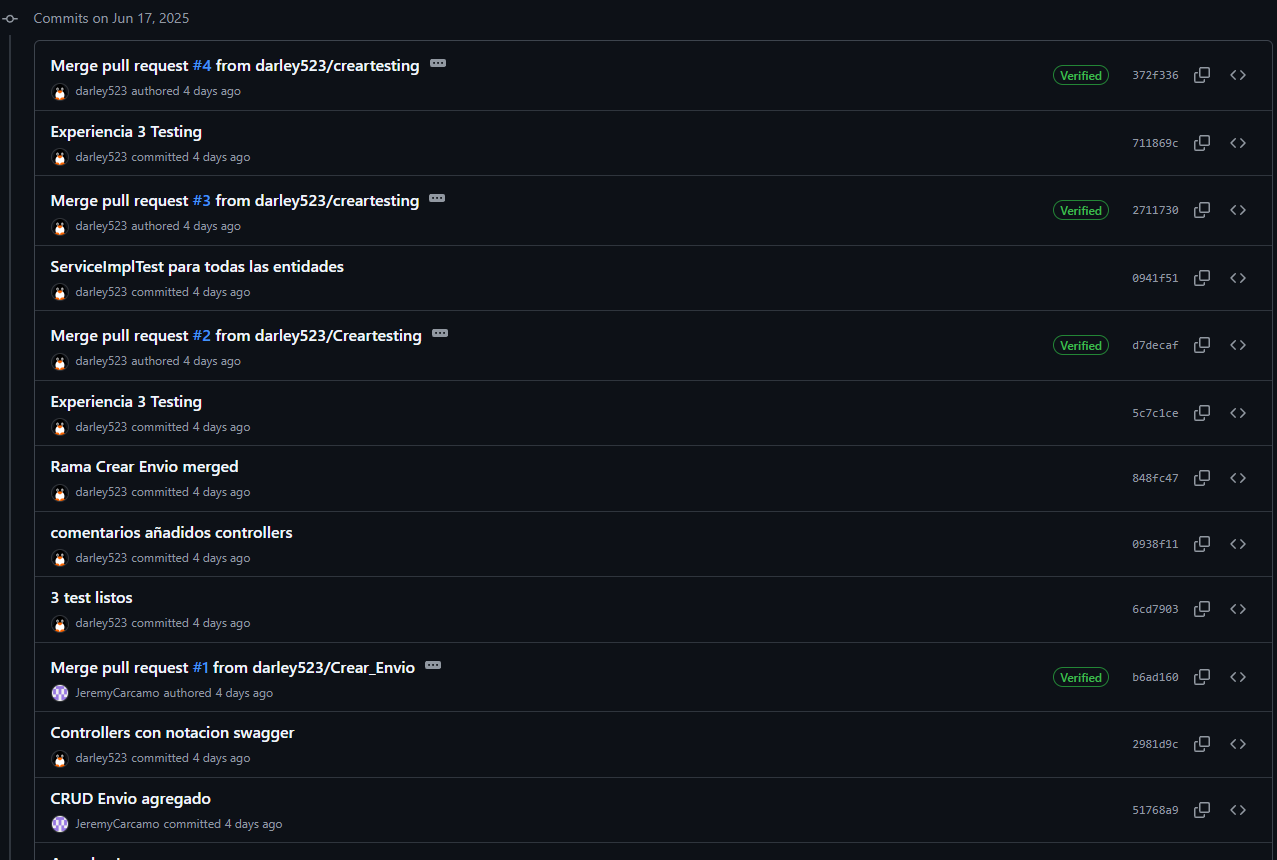
Comando para vincular el repositorio local con uno online.



Comando para subir los archivos al repositorio online (sube todos los commit).



Historial de commits colaborativos durante el desarrollo del proyecto.

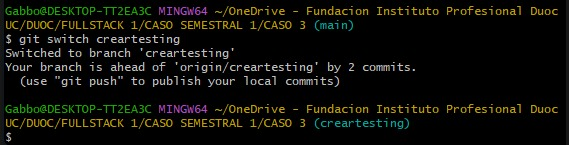


## 5.2 Comandos nuevos

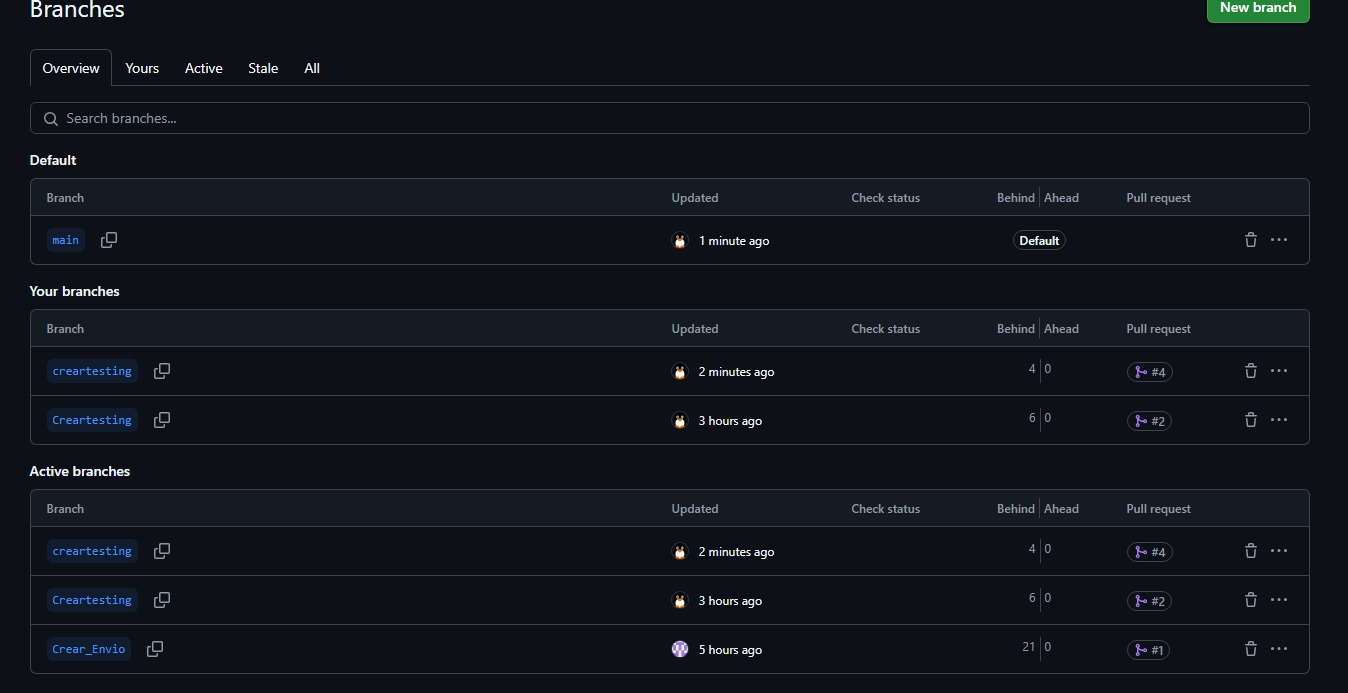
Para crear ramas.



Para cambiar entre ramas.



Visualización de las ramas (branches) en GitHub.



# 

# 

# 

# 6. Conclusión

La adopción de una arquitectura de microservicios ha supuesto un avance significativo en la evolución del sistema de Perfulandia SPA, mejorando su escalabilidad, mantenibilidad y capacidad de adaptación a futuras necesidades. La implementación de pruebas unitarias y de integración, a través de herramientas como JUnit 5, Mockito y MockMvc, permitió validar la correcta funcionalidad de cada componente, garantizando su integración con la base de datos y el comportamiento esperado frente a diversos escenarios. Y así, el uso de Git y GitHub fortaleció el trabajo colaborativo, permitiendo un control eficiente de versiones y facilitando la integración continua. El proyecto establece así una base tecnológica robusta para el crecimiento futuro de la empresa.