Una caricatura de una persona

El contenido generado por IA puede ser incorrecto.

Desarrollo de Aplicación Web: Fase de Pruebas, Documentación y Mejora Continua de EcoMarket+

Integrantes del grupo:

- Bryckson Gutierrez – Microservicios Usuario / Soporte

- Rodrigo Toro – Microservicios Inventario / Notificaciones

- Gonzalo Cortes – Microservicios Catálogo / Sucursales

Docente: Viviana Poblete

Asignatura: DESARROLLO FULLSTACK I\_008V

Fecha: 24-06-2025

# Índice de Contenidos

Contenido

[Índice de Contenidos 2](#_Toc201447590)

[1. Introducción 3](#_Toc201447591)

[2. Diagrama de Arquitectura 4](#_Toc201447592)

[3. Implementación de Nuevos Servicios. 5](#_Toc201447593)

[4. Controladores REST Implementados. 6](#_Toc201447594)

[5.Pruebas Unitarias (JUnit 5). 8](#_Toc201447595)

[6. Pruebas de Integración (REST Controllers). 10](#_Toc201447596)

[7. Documentación con OpenAPI (OAS) 14](#_Toc201447597)

[8.Git Hub. 15](#_Toc201447598)

[9. Buenas Prácticas Implementadas. 17](#_Toc201447599)

[10. Conclusión 17](#_Toc201447600)

# 1. Introducción

**Descripción General del Proyecto – Segunda Etapa.**

En esta segunda etapa del desarrollo del sistema *EcoMarket+*, se continúa con la evolución del proyecto iniciado previamente mediante una arquitectura basada en microservicios. En la primera entrega, se implementaron los microservicios fundamentales de Usuarios, Catálogo e Inventario, cubriendo operaciones CRUD y conexión a base de datos MySQL, pero sin incluir pruebas ni documentación formal con OpenAPI.

Para esta segunda entrega, se han desarrollado tres nuevos microservicios: Sucursales, Soporte y Notificaciones, los cuales amplían la funcionalidad del sistema permitiendo gestionar ubicaciones físicas, atención al cliente y envío de mensajes o alertas internas.

Además, se abordó un aspecto clave en cualquier proyecto de software profesional: la calidad y mantenibilidad del código. En esta fase se incorporaron:

* Pruebas unitarias con JUnit 5 para validar la lógica de negocio de todos los microservicios.
* Pruebas de integración sobre los controladores REST para asegurar el correcto comportamiento de los endpoints expuestos.
* Documentación de la API utilizando OpenAPI (OAS) mediante la integración de Swagger UI, permitiendo una interfaz visual de cada servicio REST.
* Uso controlado del versionamiento en Git y GitHub, con una estructura clara y comandos aplicados en consola.

Todas las prácticas mencionadas fueron replicadas de forma uniforme en los seis microservicios, asegurando coherencia, reutilización de patrones y facilidad de mantenimiento. Este informe presenta el detalle técnico de dichas implementaciones, evidencias de pruebas realizadas, decisiones de diseño y las herramientas utilizadas para robustecer la plataforma *EcoMarket+*.

# 2. Diagrama de Arquitectura

Gráfico

El contenido generado por IA puede ser incorrecto.

El sistema está dividido en varios microservicios independientes. Para este proyecto, se desarrollaron seis microservicios principales:

Usuario / Soporte

Inventario / Notificaciones

Catálogo / Sucursales

Cada uno de ellos tiene su propia base de datos y sus propios endpoints REST.

# 3. Implementación de Nuevos Servicios.

En esta segunda fase del proyecto *EcoMarket+*, se incorporaron tres nuevos microservicios: **Sucursal**, **Soporte** y **Notificacion**, complementando así la estructura previamente conformada por **Usuarios**, **Catálogo** e **Inventario**.

A continuación, se presenta una captura de la carpeta entities que contiene las clases de las seis entidades utilizadas en el sistema. Posteriormente, se detalla sus atributos.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto.

**Sucursal.java**  
Define información como nombre, dirección, ciudad, región, zona de cobertura y estado activo de cada sucursal.

**Soporte.java**  
Contiene los campos necesarios para gestionar reclamos, sugerencias o consultas, con fechas, estado y respuesta.

**Notificación.java**  
Modela los mensajes enviados dentro del sistema, incluyendo remitente, destinatario, tipo de evento y fecha de envío.

# 4. Controladores REST Implementados.

Cada uno de los seis microservicios incluye su propio **controlador REST**, el cual permite gestionar operaciones CRUD mediante solicitudes HTTP. Se utilizó el enfoque estándar de Spring Boot, aplicando anotaciones como @RestController, @GetMapping, @PostMapping, @PutMapping y @DeleteMapping.

A continuación, se muestra como ejemplo el controlador del microservicio **Soporte**, cuya estructura se replica de forma similar en los demás servicios del sistema.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Este patrón de implementación fue aplicado de forma coherente a los controladores de los microservicios de **Usuarios**, **Catálogo**, **Inventario**, **Sucursales**, **Soporte** y **Notificaciones**, asegurando consistencia en las rutas, respuestas y estructura interna.

# 5.Pruebas Unitarias (JUnit 5).

Para las pruebas unitarias se utilizó el framework **JUnit 5**, una herramienta ampliamente usada en Java para la creación de pruebas automatizadas.  
Además, se incorporó **Mockito**, una librería que permite simular (mockear) componentes externos como los repositorios, evitando la necesidad de conectarse a una base de datos real durante la prueba.

JUnit 5 permite definir métodos de prueba con la anotación @Test, mientras que Mockito aporta funcionalidades como @Mock para simular dependencias y @BeforeEach para inicializar datos o configuraciones antes de cada prueba.

Para asegurar el correcto funcionamiento de la lógica de negocio en cada microservicio, se implementaron pruebas unitarias enfocadas principalmente en los métodos de la capa de servicios (ServiceImpl).

**Texto

El contenido generado por IA puede ser incorrecto.**

Vista de las pruebas unitarias ejecutadas exitosamente sobre la capa ServiceImpl utilizando JUnit y Mockito.

**Texto

El contenido generado por IA puede ser incorrecto.**

**Ambas imágenes corresponden al microservicio Soporte, y representan la estructura y ejecución de las pruebas unitarias. Esta misma lógica fue replicada en los otros cinco microservicios del sistema.**

# 6. Pruebas de Integración (REST Controllers).

Las pruebas de integración fueron desarrolladas para validar el funcionamiento completo de los controladores REST, asegurando que cada endpoint responda correctamente ante solicitudes reales simuladas. Este tipo de pruebas se enfoca en verificar la comunicación entre el controlador y la capa de servicios, evaluando el sistema como un todo.

En este caso, los métodos testeados incluyen operaciones como GET, POST, PUT y DELETE, que interactúan directamente con las rutas HTTP definidas en cada microservicio.

A continuación, se muestra un ejemplo del microservicio **Soporte**, incluyendo el controlador, su clase de prueba y la ejecución correspondiente. Esta misma estructura fue replicada en los demás microservicios del sistema.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto.

# 7. Documentación con OpenAPI (OAS)

Para documentar los endpoints de cada microservicio, se integró la herramienta **Swagger UI** a través de la dependencia springdoc-openapi. Esta implementación permite generar una interfaz visual y navegable donde se puede consultar, probar y validar cada una de las rutas HTTP disponibles en los controladores REST.

Swagger detecta automáticamente las rutas expuestas en cada servicio y muestra detalles como el método HTTP (GET, POST, PUT, DELETE), los parámetros requeridos y las respuestas esperadas.

A continuación, se presenta un ejemplo correspondiente al microservicio **Soporte**. Esta documentación fue generada de forma automática y replicada en todos los microservicios restantes.

Interfaz de usuario gráfica, Aplicación

El contenido generado por IA puede ser incorrecto.

# 8.Git Hub.

Para el control de versiones del proyecto *EcoMarket+*, se utilizó el sistema de control de versiones **Git**, junto con la plataforma **GitHub** para el almacenamiento remoto y colaboración en línea.

Durante el desarrollo se aplicaron los comandos básicos de Git:

* git init: para inicializar el repositorio local.
* git add .: para agregar todos los archivos al área de staging.
* git commit -m "mensaje": para guardar cambios con una descripción.
* git push: para subir los cambios al repositorio remoto.

El repositorio contiene el código fuente completo de los seis microservicios, sus respectivas pruebas y el archivo README.md. Este archivo incluye link de Swagger y los **paths de cada servicio REST** y las operaciones que se pueden ejecutar, facilitando su consulta y prueba.

A continuación, se muestran capturas del uso de Git en consola y de la estructura del repositorio en GitHub.

Interfaz de usuario gráfica, Texto

El contenido generado por IA puede ser incorrecto.

git add .



git commit -m "Experiencia 3 Testing"

Texto

El contenido generado por IA puede ser incorrecto.

git remote add origin <https://github.com/Bryckson/Exp3_Gutierrez_Toro_Gonzales>



git push -u origin main  
Texto

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Correo electrónico, Sitio web

El contenido generado por IA puede ser incorrecto.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

El contenido generado por IA puede ser incorrecto.

# 9. Buenas Prácticas Implementadas.

Durante el desarrollo de los seis microservicios se aplicaron diversas buenas prácticas que contribuyen a mantener un código limpio, escalable y fácilmente mantenible. Algunas de las más destacadas son:

Separación por capas: cada microservicio está estructurado en paquetes como entities, repositories, services, restcontrollers, respetando el principio de responsabilidad única.

Uso de anotaciones de Spring Boot: se utilizaron de forma consistente @RestController, @Service, @Repository, @Autowired, entre otras, para aprovechar la inyección de dependencias y el ciclo de vida de los componentes.

Respuestas controladas: los controladores REST usan ResponseEntity para retornar respuestas HTTP con estados claros y personalizados (como 200, 201, 404).

Pruebas automatizadas: todos los microservicios cuentan con pruebas unitarias y de integración bien estructuradas, utilizando JUnit 5 y Mockito.

Documentación automática: se integró Swagger UI en todos los servicios para visualizar y probar los endpoints desde el navegador.

Estas prácticas fueron aplicadas de forma coherente en los seis microservicios, lo que permitió estandarizar el desarrollo y facilitar su comprensión por parte de todo el equipo.

# 10. Conclusión

La segunda etapa del proyecto *EcoMarket+* permitió consolidar el sistema basado en microservicios, no solo con la incorporación de nuevos módulos (**Sucursales**, **Soporte** y **Notificaciones**), sino también mediante la integración de herramientas clave para asegurar su calidad.

La implementación de **pruebas unitarias**, **pruebas de integración** y la **documentación automática con Swagger** aportó mayor robustez al proyecto, permitiendo validar el comportamiento de cada componente de forma controlada y trazable. Además, el uso de **Git y GitHub** permitió mantener una gestión organizada del código y facilitar el trabajo colaborativo.

Este proceso no solo fortaleció las habilidades técnicas del equipo, sino que también reforzó la importancia de aplicar buenas prácticas en el desarrollo de software real, modular y escalable.