



Desarrollo de Aplicación Web: Implementación de EcoMarket+ con Spring Boot.

Integrantes del grupo:

- Bryckson Gutierrez – Microservicio Usuario
- Rodrigo Toro – Microservicio inventario
- Gonzalo Cortes – Microservicio Catálogo

Docente: Viviana Poblete

Asignatura: DESARROLLO FULLSTACK I_008V

Fecha: 26-05-2025

Índice de Contenidos

Índice de Contenidos	2
1. Introducción.....	3
2. Diagrama de Arquitectura	4
3. Creación del Proyecto.....	5
4. Estructura del Proyecto (Packages y Propósito)	6
5. Evidencia de Consumo de Servicios con Postman.....	12
6. Subida a GitHub.....	14
6. Conclusión	16

1. Introducción

Descripción General del Proyecto

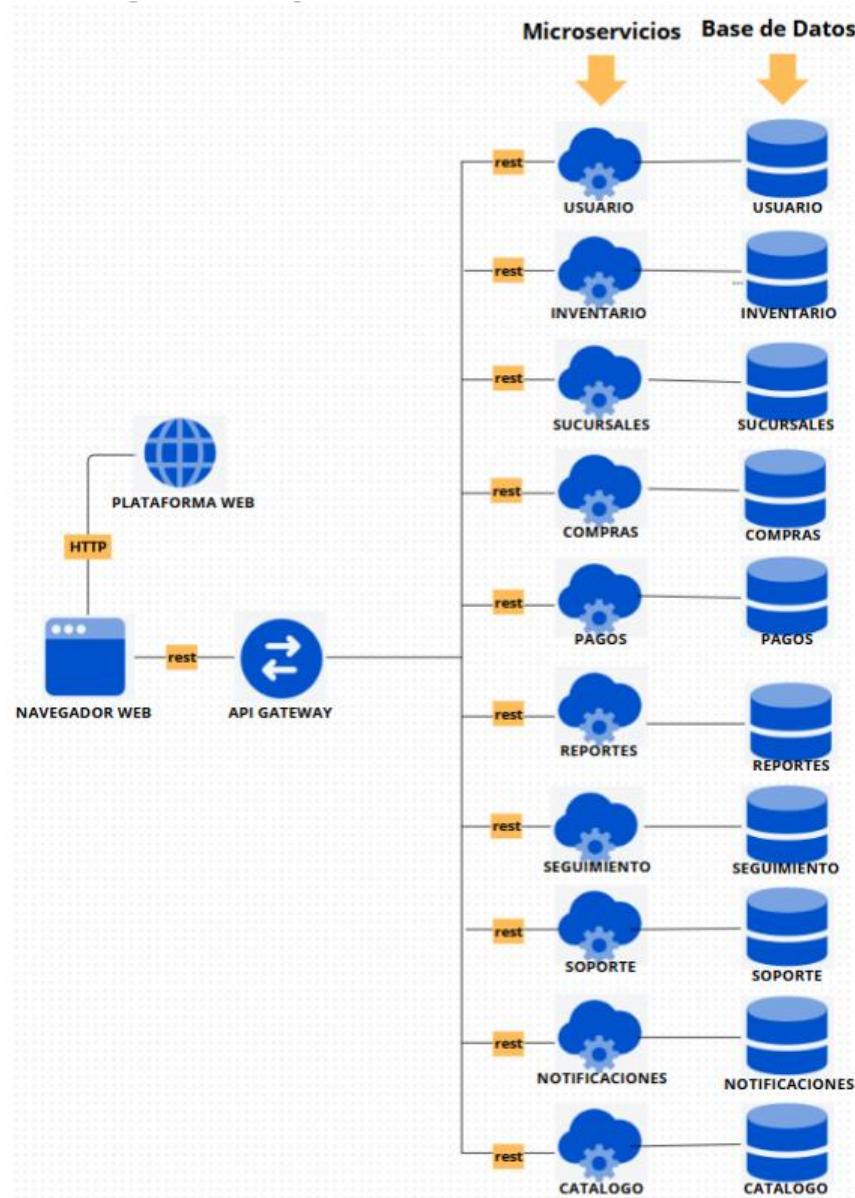
El sistema EcoMarket+ surge como una solución modular orientada a reemplazar un sistema monolítico mediante una arquitectura basada en microservicios, con el objetivo de mejorar la escalabilidad, mantenibilidad y eficiencia operativa de EcoMarket SPA, empresa dedicada a la venta de productos ecológicos y sustentables.

Este proyecto fue desarrollado utilizando Spring Boot, implementando varios microservicios, de los cuales, en este informe, desarrollaremos específicamente 3: usuario, inventario y catálogo, cada uno con su propia base de datos en MySQL y expuesto mediante API REST. Estos servicios permiten realizar operaciones como registro y gestión de usuarios, control de stock de productos y visualización del catálogo, asegurando una interacción desacoplada y eficiente entre componentes.

Cada microservicio puede desplegarse de forma independiente, facilitando el desarrollo colaborativo, la integración continua y la escalabilidad horizontal del sistema.

El proyecto fue creado desde cero utilizando Spring Initializr, y cada integrante del grupo asumió la implementación de uno de los servicios mencionados. La solución incluye vistas con Thymeleaf, controladores REST, lógica de negocio y persistencia de datos, así como pruebas realizadas con Postman y subida del código a GitHub.

2. Diagrama de Arquitectura



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

Usuario

Inventario

Catálogo

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

3. Creación del Proyecto

La aplicación fue creada utilizando Spring Boot versión 3.4.5, con el JDK 21. Se utilizó el gestor de dependencias Maven, y el proyecto fue estructurado desde cero utilizando la opción “Spring Initializr”, seleccionando las siguientes dependencias esenciales:

Dependencia	Descripción
spring-boot-starter-data-jpa	Facilita el acceso a bases de datos relacionales usando JPA y anotaciones para operaciones CRUD.
spring-boot-starter-web	Proporciona soporte para crear aplicaciones web RESTful con controladores, rutas y servidor embebido.
spring-boot-devtools	Incluye herramientas útiles para desarrollo como recarga automática ante cambios.
mysql-connector-j	Driver JDBC necesario para conectar la aplicación a bases de datos MySQL.
spring-boot-starter-thymeleaf	Motor de plantillas para renderizar vistas HTML dinámicas desde el backend.

Pom.xml:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>

<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-thymeleaf</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```



```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

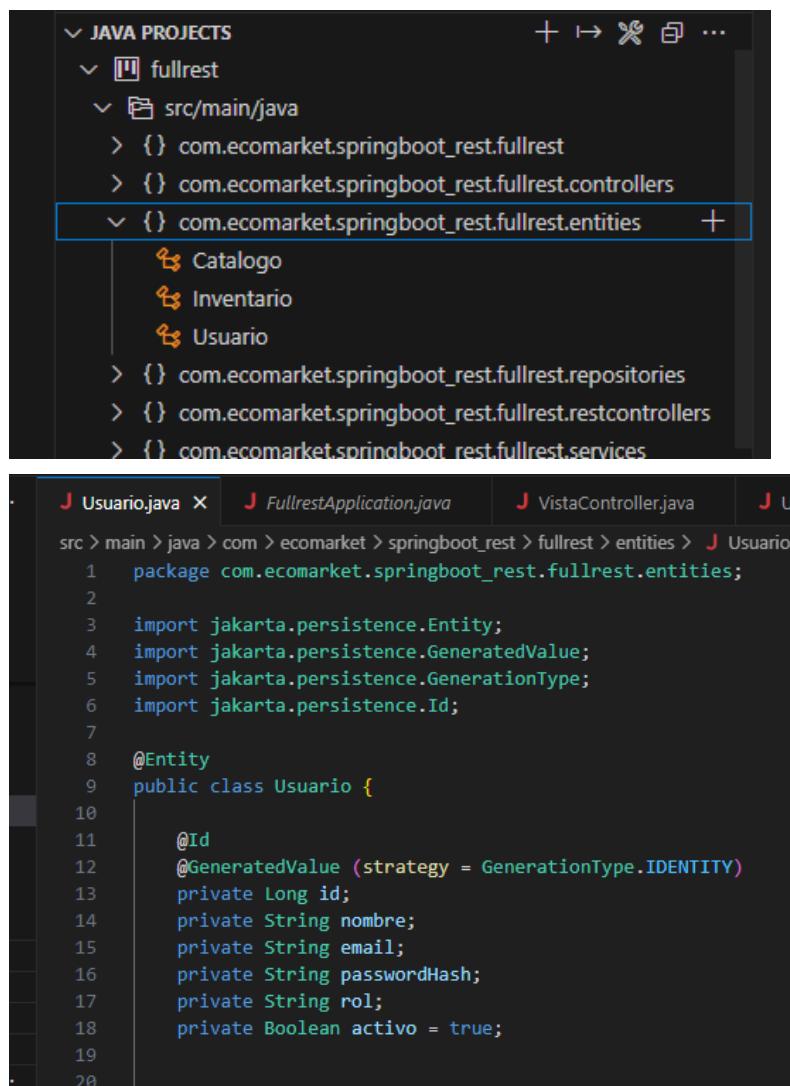
Estas dependencias permiten conectar con la base de datos, crear controladores REST, manejar vistas HTML y trabajar con entidades persistentes.

4. Estructura del Proyecto (Packages y Propósito)

A continuación, se detallan los paquetes implementados, su propósito y su funcionalidad:

Entity:

Contiene las clases que representan las entidades del sistema (por ejemplo: Usuario, Inventario, Catalogo). Cada entidad está mapeada con anotaciones JPA a su tabla correspondiente en la base de datos.



The screenshot shows a Java IDE interface. At the top, there's a toolbar with icons for new project, import, export, etc. Below it is a tree view of the project structure under 'JAVA PROJECTS'. The 'fullrest' project is expanded, showing its sub-packages: 'src/main/java', 'com.ecomarket.springboot_rest.fullrest', 'com.ecomarket.springboot_rest.fullrest.controllers', and 'com.ecomarket.springboot_rest.fullrest.entities'. The 'entities' package is selected and highlighted with a blue border. Inside 'entities', three classes are listed: 'Catalogo', 'Inventario', and 'Usuario'. The 'Usuario' class is currently open in the code editor. The code editor shows the following Java code:

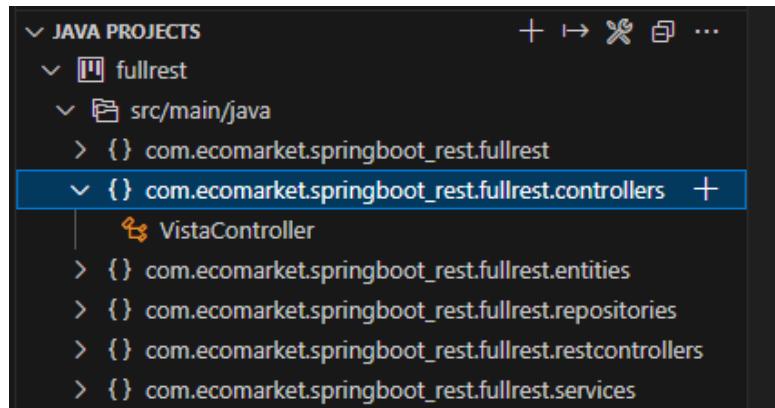
```
1 package com.ecomarket.springboot_rest.fullrest.entities;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 @Entity
9 public class Usuario {
10
11     @Id
12     @GeneratedValue (strategy = GenerationType.IDENTITY)
13     private Long id;
14     private String nombre;
15     private String email;
16     private String passwordHash;
17     private String rol;
18     private Boolean activo = true;
19
20 }
```

```
6 import jakarta.persistence.Id;
7
8 @Entity
9 public class Inventario {
10
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long id;
14
15     private String nombreProducto;
16     private String categoria;
17     private Integer cantidad;
18     private Double precioUnitario;
19     private Boolean activo = true;
20
21     public Inventario() {
22
23         public Inventario(Long id, String nombreProducto, String cate
```

```
6 import jakarta.persistence.Id;
7
8 @Entity
9 public class Catalogo {
10
11     @Id
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long id;
14
15     private String nombre;
16     private String descripcion;
17     private String categoria;
18     private Double precio;
19     private String sucursal;
20     private Boolean promocion = true;
21
22     public Catalogo() {
23 }
```

Controller:

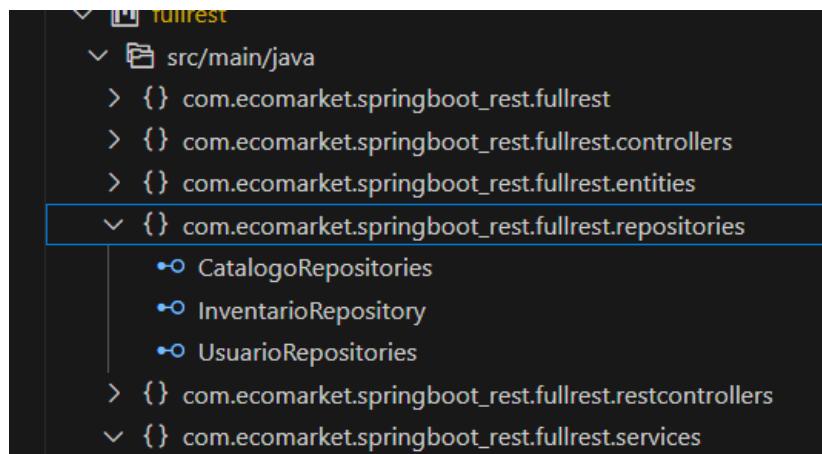
Este paquete contiene controladores que permiten gestionar la navegación de vistas en Thymeleaf. Son usados para retornar páginas HTML. En esta ocasión, tenemos solo una clase la cual tendrá los 3 métodos de los servicios.



```
16
17     @Controller
18     public class VistaController {
19
20         @Autowired
21         private UsuarioRepositories usuariorepositories;
22
23         @GetMapping("/usuarios")
24         public String verUsuarios(Model model){
25             List<Usuario> usuariditos = (List<Usuario>) usuariorepositories.findAll();
26             model.addAttribute(attributeName:"usuarios", usuariditos);
27             return "usuario";
28         }
29
30
31         @Autowired
32         private CatalogoRepositories catalogorepositories;
33
34         @GetMapping("/catalogos")
35         public String verCatalogos(Model model){
36             List<Catalogo> catalogitos = (List<Catalogo>) catalogorepositories.findAll();
37             model.addAttribute(attributeName:"catalogo", catalogitos);
38             return "catalogo";
39         }
40
41
42         @Autowired
43         private InventarioRepository inventariopositories;
44
45         @GetMapping("/inventario")
46         public String listarInventario(Model model) [
47             List<Inventario> invent = (List<Inventario>) inventariopositories.findAll();
48             model.addAttribute(attributeName:"inventario", invent);
49             return "inventario";
50         ]
51
```

Repositories:

Aquí se encuentran las interfaces que extienden JpaRepository o CrudRepository. Son responsables de la interacción con la base de datos (operaciones CRUD). Este formato de las imágenes, se replico en los 3 servicios.



```
src > main > java > com > ecomarket > springboot_rest > fullrest > repositories > UsuarioRepositories.java
1 package com.ecomarket.springboot_rest.fullrest.repositories;
2
3 import org.springframework.data.repository.CrudRepository;
4
5 import com.ecomarket.springboot_rest.fullrest.entities.Usuario;
6
7 public interface UsuarioRepositories extends CrudRepository<Usuario, Long> {
8
9 }
```

Services:

Define la lógica de negocio del sistema. Contiene interfaces y sus implementaciones (como UsuarioServices y UsuarioServiceImpl). Este formato de las imágenes, se replicó en los 3 servicios.

The screenshot shows a Java project structure and several code editor panes.

Java Projects:

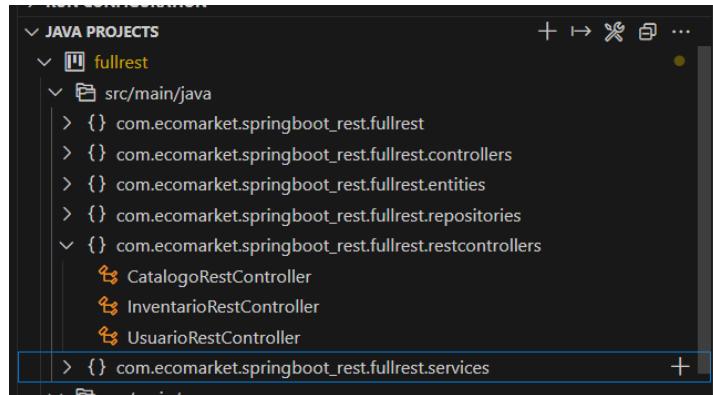
- fullrest
- src/main/java
 - com.ecomarket.springboot_rest.fullrest
 - com.ecomarket.springboot_rest.fullrest.controllers
 - com.ecomarket.springboot_rest.fullrest.entities
 - com.ecomarket.springboot_rest.fullrest.repositories
 - com.ecomarket.springboot_rest.fullrest.restcontrollers
 - com.ecomarket.springboot_rest.fullrest.services
 - CatalogoService
 - CatalogoServiceImpl
 - InventarioService
 - InventarioServiceImpl
 - UsuarioService
 - UsuarioServiceImpl
 - UsuarioServices

Code Editor Panes:

- VistaController.java:** Shows code for a service implementation. The cursor is at line 13, which starts with `@Service`. Lines 1-12 are imports.
- UsuarioRestController.java:** Shows code for a REST controller. The cursor is at line 1, which starts with `> {}`.
- UsuarioServices.java:** Shows the interface `UsuarioServices` with methods `findByAll()`, `findById(Long id)`, and `save(Usuario unUsuario)`.
- UsuarioServiceImpl.java:** Shows the implementation of `UsuarioServices` using `UsuarioRepository`. It includes methods for `delete(Usuario unUsuario)`, `findByAll()`, `findById(Long id)`, and `save(Usuario unUsuario)`.
- CatalogoService.java:** Shows the interface `CatalogoService` with methods `findByAll()`, `findById(Long id)`, `save(Catalogo unCatalogo)`, and `delete(Catalogo unCatalogo)`.
- CatalogoServiceImpl.java:** Shows the implementation of `CatalogoService` using `CatalogoRepository`. It includes methods for `findByAll()`, `findById(Long id)`, `save(Catalogo unCatalogo)`, and `delete(Catalogo unCatalogo)`.

Restcontroller:

Controladores REST expuestos públicamente para interactuar mediante peticiones HTTP (GET, POST, PUT, DELETE). Manejan peticiones desde Postman y otras interfaces de cliente. Este formato de las imágenes, se replicó en los 3 servicios.



```

src > main > java > com > ecomarket > springboot_rest > fullrest > restcontrollers > J UsuarioRestController.java > Lang
1 package com.ecomarket.springboot_rest.fullrest.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.ecomarket.springboot_rest.fullrest.entities.Usuario;
19 import com.ecomarket.springboot_rest.fullrest.services.UsuarioServices;
20
21 @RestController
22 @RequestMapping("api/usuarios")
23 public class UsuarioRestController {
24
25     @Autowired
26     private UsuarioServices usuarioservice;
27
28     @GetMapping
29     public List<Usuario> mostrarUsuarios(){
30         return usuarioservice.findAll();
31     }
32
33     @GetMapping("/{id}")
34     public ResponseEntity<?> verUsuario(@PathVariable Long id){
35         Optional<Usuario> usuarioOptional = usuarioservice.findById(id);
36         if (usuarioOptional.isPresent()) {
37             return ResponseEntity.ok(usuarioOptional.orElseThrow());
38         }
39         return ResponseEntity.notFound().build();
40     }
41
42     @PostMapping
43     public ResponseEntity<Usuario> crearUsuario(@RequestBody Usuario unUsuario){
44         return ResponseEntity.status(HttpStatus.CREATED).body(usuarioservice.save(
45     )
46
47     @PutMapping("/{id}")
48     public ResponseEntity<?> modificarUsuario(@PathVariable Long id, @RequestBody Usuario unUsuario) {
49         Optional<Usuario> usuarioOptional = usuarioservice.findById(id);
50         if (usuarioOptional.isPresent()){
51             Usuario usuarioexistente = usuarioOptional.get();
52             usuarioexistente.setNombre(unUsuario.getNombre());
53             usuarioexistente.setEmail(unUsuario.getEmail());
54             usuarioexistente.setPasswordHash(unUsuario.getPasswordHash());
55             usuarioexistente.setRol(unUsuario.getRol());
56             Usuario productomodificado = usuarioservice.save(usuarioexistente);
57             return ResponseEntity.ok(productomodificado);
58         }
59     }
60     return ResponseEntity.notFound().build();
61 }
62
63     @DeleteMapping("/{id}")
64     public ResponseEntity<?> eliminarUsuario (@PathVariable Long id){
65         Usuario unUsuario = new Usuario();
66         unUsuario.setId(id);
67         Optional <Usuario> usuarioOptional = usuarioservice.delete(unUsuario);
68         if (usuarioOptional.isPresent()){
69             return ResponseEntity.ok(usuarioOptional.orElseThrow());
70         }
71     }
72 }
73
74

```

5. Evidencia de Consumo de Servicios con Postman

Para validar el correcto funcionamiento de los servicios desarrollados en Spring Boot, se realizaron pruebas utilizando Postman, una herramienta que permite enviar solicitudes HTTP y visualizar las respuestas de una API.

A continuación, se documentan las peticiones realizadas a los endpoints REST (GET, GET por ID, POST, PUT y DELETE), acompañadas de capturas de pantalla que demuestran su ejecución exitosa.

- **GET:**

The screenshot shows a Postman workspace with a single collection named "Getting starte". A request is made to "http://localhost:8080/api/usuarios" using the "GET" method. The response status is "200 OK" with a duration of 9 ms and a size of 721 B. The response body is a JSON array containing three user objects:

```
[{"id": 1, "nombre": "Bryckson Gutierrez", "email": "bryc_gu@ecomarket.com", "passwordHash": "$2a$10$claveHash", "rol": "VENDEDOR", "activo": null}, {"id": 2, "nombre": "Camila Torres", "email": "camila_to@ecomarket.com", "passwordHash": "$2a$10$Camil145Segura2024", "rol": "GERENTE", "activo": null}, {"id": 3, "nombre": "Fernanda Diaz", "email": "fer_di@ecomarket.com", "passwordHash": "$2a$10$Fer123456Hash", "rol": "LOGISTICA", "activo": null}]
```

GET (id):

The screenshot shows a Postman workspace with a single collection named "Getting starte". A request is made to "http://localhost:8080/api/usuarios/3" using the "GET" method. The response status is "200 OK" with a duration of 9 ms and a size of 299 B. The response body is a JSON object representing a single user:

```
{"id": 3, "nombre": "Fernanda Diaz", "email": "fer_di@ecomarket.com", "passwordHash": "$2a$10$Fer123456Hash", "rol": "LOGISTICA", "activo": null}
```

- **POST:**

```

1 {
2   "nombre": "Martin Bravo Ramirez",
3   "email": "martin_b@ecomarket.com",
4   "passwordHash": "$2a$10$Mart1nS3guro",
5   "rol": "LOGISTICA",
6   "activo": null
7 }
  
```

Body 201 Created 77 ms 313 B

{ } JSON ▾ Preview Visualize

1 {
2 "id": 26,
3 "nombre": "Martin Bravo Ramirez",
4 "email": "martin_b@ecomarket.com",
5 "passwordHash": "\$2a\$10\$Mart1nS3guro",
6 "rol": "LOGISTICA",
7 "activo": null
8 }

- **PUT:**

***CONSULTA GET ID(3) ROL:"GERENTE"**

```

1 {
2   "id": 3,
3   "nombre": "Fernanda Diaz",
4   "email": "fer_di@ecomarket.com",
5   "passwordHash": "$2a$10$Fer123456Hash",
6   "rol": "GERENTE",
7   "activo": null
8 }
  
```

Body 200 OK

{ } JSON ▾ Preview Visualize

***MODICAMOS ROL:"GERENTE GENERAL"**

```

1 {
2   "nombre": "Fernanda Diaz",
3   "email": "fer_di@ecomarket.com",
4   "passwordHash": "$2a$10$Fer123456Hash",
5   "rol": "GERENTE REGIONAL",
6   "activo": null
7 }
  
```

Body 200 OK

{ } JSON ▾ Preview Visualize

```

1 {
2   "id": 3,
3   "nombre": "Fernanda Diaz",
4   "email": "fer_di@ecomarket.com",
5   "passwordHash": "$2a$10$Fer123456Hash",
6   "rol": "GERENTE REGIONAL",
7   "activo": null
8 }
  
```

- **DELETE:**

The screenshot shows the Postman interface. At the top, there are tabs for Workspaces and More, along with various icons for search, add, settings, and notifications. A prominent 'Upgrade' button is visible. Below the header, there's a navigation bar with links for 'Getting starte' (GET http://localhost:8080), 'Overview' (DEL http://localhost:8080), and 'No environment'. The main workspace shows a request to 'http://localhost:8080/api/usuarios/26' using the 'DELETE' method. The 'Body' tab is selected, displaying a JSON object with the following content:

```
1 {  
2   "id": 26,  
3   "nombre": null,  
4   "email": null,  
5   "passwordHash": null,  
6   "rol": null,  
7   "activo": true  
8 }
```

The response status is '200 OK' with a response time of 23 ms and a size of 245 B. The response body is identical to the JSON sent in the request.

6. Subida a GitHub

El proyecto fue subido a un repositorio remoto utilizando los siguientes comandos:

git init

```
MINGW64:/c/Users/DELL/OneDrive/Desktop/Nueva carpeta  
  
DELL@Wixo MINGW64 ~/OneDrive/Desktop/Nueva carpeta (master)  
$ git init  
Reinitialized existing Git repository in C:/Users/DELL/OneDrive/Desktop/Nueva ca  
rpeta/.git/
```

git add .

```
DELL@Wixo MINGW64 ~/OneDrive/Desktop/Nueva carpeta (master)  
$ git add .
```

git commit -m "Primer commit EcoMarket"

```
DELL@Wixo MINGW64 ~/OneDrive/Desktop/Nueva carpeta (master)  
$ git commit -m "Implementacion de servicios REST"  
[master (root-commit) 8206702] Implementacion de servicios REST  
 31 files changed, 1540 insertions(+)  
 create mode 100644 basedato_ecomarket.sql
```

git remote add origin https://github.com/Bryckson/EvaluacionN2_008V

```
DELL@Wixo MINGW64 ~/OneDrive/Desktop/Nueva carpeta (master)
$ git remote add origin https://github.com/Bryckson/EvaluacionN2_008V
```

```
git push -u origin main
```

```
DELL@Wixo MINGW64 ~/OneDrive/Desktop/Nueva carpeta (master)
$ git push -u origin main
remote: error: refusing to update checked-out branch 'main'
remote: error: main does not match any
```

REPOSITORIO GITHUB:

The screenshot shows a GitHub repository page for 'EvaluacionN2_008V'. The repository is private and has 3 commits. The README file contains instructions for running services in Postman and lists API endpoints for User and Inventory management. The repository also includes a database script 'basedato_ecomarket.sql'.

Commits:

- Bryckson Update README.md (c411fc8 · now) 3 Commits
- fullrestEcoMarket/fullrestEcoMarket/fullrest Implementacion de servicios REST 6 minutes ago
- README.md Update README.md now
- basedato_ecomarket.sql Implementacion de servicios REST 6 minutes ago

README Content:

Paths para ejecutar consultas de los servicios en Postman.

Usuario:

- GET: <http://localhost:8080/api usuarios>
- GET{id}: <http://localhost:8080/api usuarios/4>
- POST: <http://localhost:8080/api usuarios>
- PUT: <http://localhost:8080/api usuarios/4>
- DELETE: <http://localhost:8080/api usuarios/4>

Inventario:

About:

No description, website, or top-level README.

Activity:

- Readme
- Activity
- 0 stars
- 0 watching
- 0 forks

Releases:

No releases published. Create a new release.

Packages:

No packages published. Publish your first package.

Languages:

Java 70.3% HTML 29.7%

Suggested workflows:

Based on your tech stack

6. Conclusión

El desarrollo de este proyecto representó una experiencia significativa desde el punto de vista académico y práctico. Como estudiantes, tuvimos la oportunidad de aplicar conceptos clave en la construcción de software, utilizando herramientas actuales como Spring Boot, MySQL y Postman, en el contexto de una arquitectura basada en microservicios.

Dividir el trabajo en equipo por microservicio (usuario, inventario y catálogo) no solo permitió una mejor organización, sino que también simuló un entorno colaborativo similar al de equipos profesionales. Cada integrante asumió la responsabilidad de un servicio completo, lo que implicó comprender la lógica de negocio, estructurar controladores REST, validar interacciones con base de datos y documentar pruebas funcionales.

Además de afianzar habilidades técnicas, enfrentamos desafíos reales como la integración entre servicios, la conexión a bases de datos externas y la organización del proyecto en GitHub. Todo esto nos permitió reforzar la importancia de la planificación, la comunicación clara y la documentación adecuada.

En definitiva, esta experiencia nos preparó mejor para el desarrollo de software moderno, destacando la importancia de construir soluciones modulares, escalables y mantenibles.