

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
1. Introducción.....	3
2. Diagrama de Arquitectura	4
3. Implementación de Nuevos Servicios.....	5
4. Controladores REST Implementados	6
5. Pruebas Unitarias (JUnit 5).....	8
6. Pruebas de Integración (REST Controllers).....	10
7. Documentación con OpenAPI (OAS).....	14
8. Git Hub.....	15
9. Buenas Prácticas Implementadas.....	17
10. Conclusión	17

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.

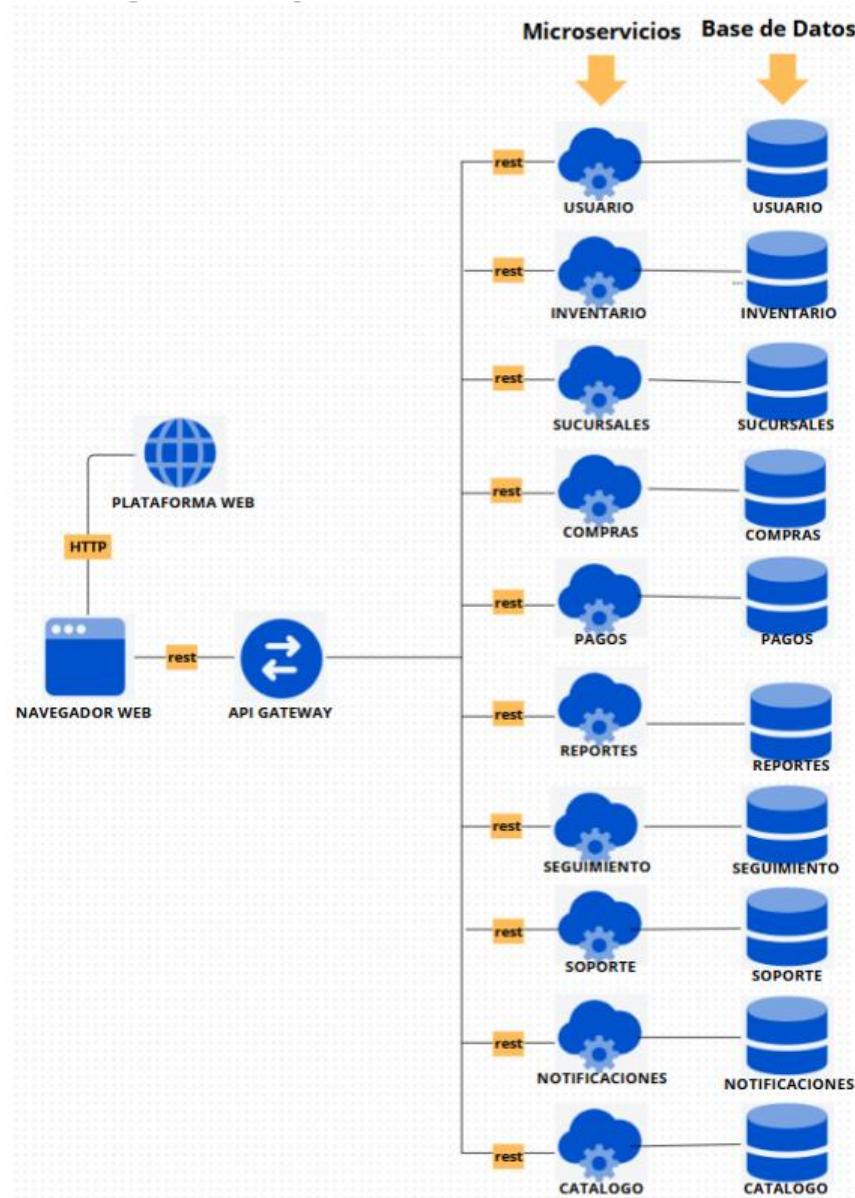
Para esta segunda entrega, se han desarrollado tres nuevos microservicios: Sucursales, Soporte y Notificaciones, los cuales amplían la funcionalidad del sistema permitiendo gestionar ubicaciones físicas, atención al cliente y envío de mensajes o alertas internas.

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



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 **Notificación**, 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.

The screenshot shows a Java IDE interface with the following details:

- Project Structure:** A tree view under "JAVA PROJECTS" shows a project named "fullrest" containing "src/main/java". Inside "src/main/java", there are packages for controllers and entities. The "entities" package is selected, revealing sub-packages for Catalogo, Inventario, Notificacion, Soporte, Sucursal, and Usuario.
- Sucursal.java:** A code snippet for the "Sucursal" entity class. It includes annotations for Entity, Id, and GeneratedValue, and defines fields for id, nombre, direccion, ciudad, region, zonaCobertura, and activa.
- Soporte.java:** A code snippet for the "Soporte" entity class. It includes annotations for Entity, Id, and GeneratedValue, and defines fields for id, idUsuario, tipo, mensaje, fechaCreacion, estado, respuesta, fechaRespuesta, and activo.
- Notificacion.java:** A code snippet for the "Notificacion" entity class. It includes annotations for Entity, Id, and GeneratedValue, and defines fields for id, idUsuario, tipo, mensaje, fechaCreacion, and leido. It also includes a constructor and a copy constructor.

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.

```
@RestController  
@RequestMapping("api/Soportes")  
public class SoporteRestController {
```

```
@GetMapping("/{id}")  
public ResponseEntity<?> verSoporte(@PathVariable Long id){  
    Optional<Soporte> soporteOptional = soporteservice.findById(id);  
    if (soporteOptional.isPresent()) {  
        return ResponseEntity.ok(soporteOptional.orElseThrow());  
    }  
    return ResponseEntity.notFound().build();  
}
```

```
@PostMapping  
public ResponseEntity<Soporte> crearSoporte(@RequestBody Soporte unSoporte){  
    return ResponseEntity.status (HttpStatus.CREATED). body(soporteservice.sav  
}
```

```
@PutMapping("/{id}")  
public ResponseEntity<?> modificarSoporte(@PathVariable Long id, @Requ  
Optional<Soporte> soporteOptional = soporteservice.findById(id);  
  
if (soporteOptional.isPresent()) {  
    Soporte soporteExistente = soporteOptional.get();  
  
    soporteExistente.setIdUsuario(unSoporte.getIdUsuario());  
    soporteExistente.setTipo(unSoporte.getTipo());  
    . . .
```

```
@DeleteMapping("/{id}")
public ResponseEntity<?> eliminarSoporte (@PathVariable Long id){
    Soporte unSoporte = new Soporte();
    unSoporte.setId(id);
    Optional <Soporte> soporteOptional = soporteservice.delete(unSoporte);
    if (soporteOptional.isPresent()){
        return ResponseEntity.noContent().build();
    }
    return ResponseEntity.notFound().build();
}
```

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.

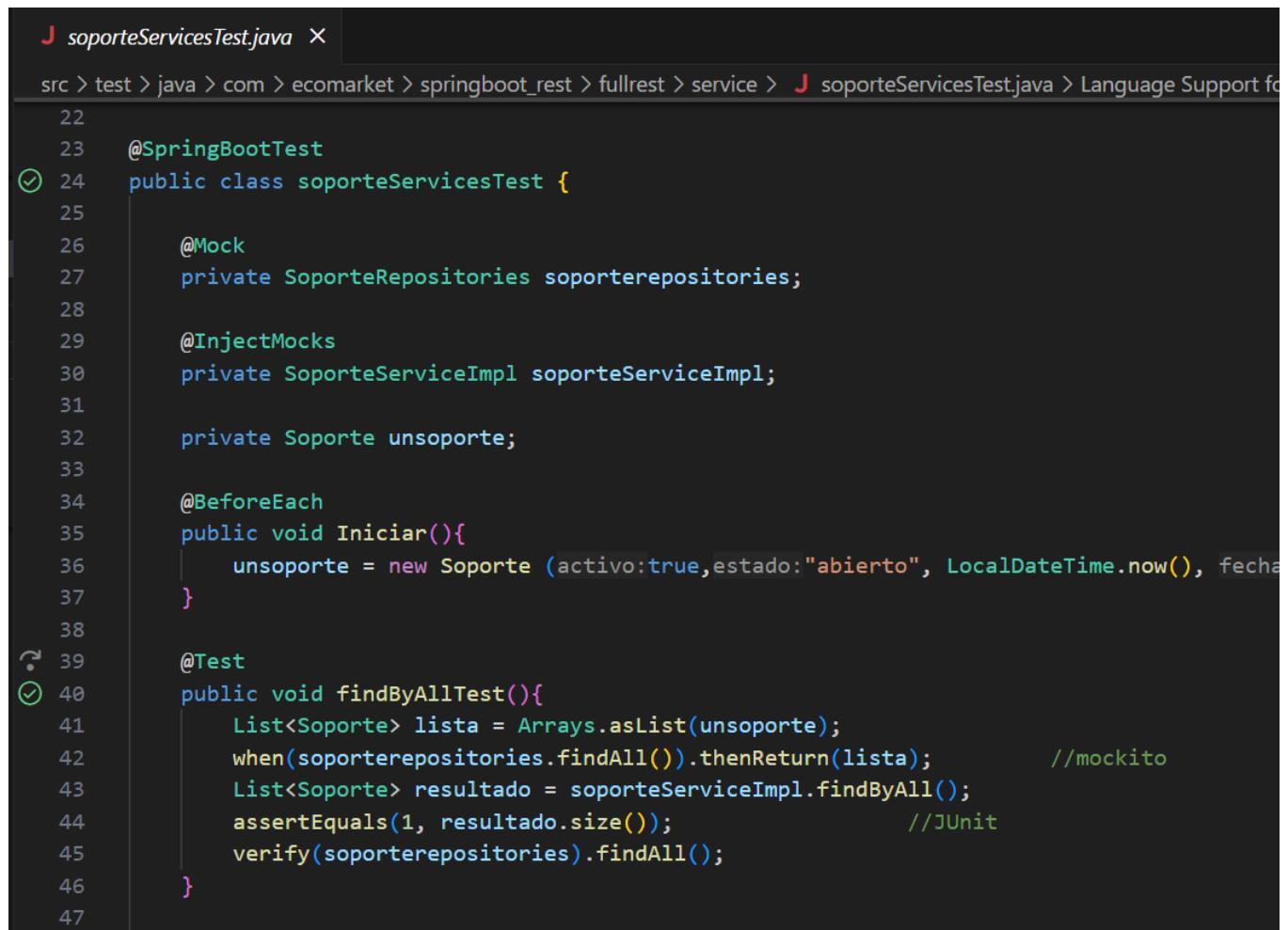
5. Pruebas Unitarias (JUnit 5).

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`).



```
soporteServicesTest.java
src > test > java > com > ecomarket > springboot_rest > fullrest > service > soporteServicesTest.java > Language Support fo
22
23  @SpringBootTest
24  public class soporteServicesTest {
25
26      @Mock
27      private SoporteRepositories soporterepositories;
28
29      @InjectMocks
30      private SoporteServiceImpl soporteServiceImpl;
31
32      private Soporte unsoporte;
33
34      @BeforeEach
35      public void Iniciar(){
36          unsoporte = new Soporte (activo:true,estado:"abierto", LocalDateTime.now(), fecha);
37      }
38
39      @Test
40      public void findByAllTest(){
41          List<Soporte> lista = Arrays.asList(unsoporte);
42          when(soporterepositories.findAll()).thenReturn(lista);           //mockito
43          List<Soporte> resultado = soporteServiceImpl.findByAll();
44          assertEquals(1, resultado.size());                                //JUnit
45          verify(soporterepositories).findAll();
46      }
47  }
```

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

```
|> ✓ { } com.ecomarket.springboot_rest.fullrest.service 219ms
|  > ✓ 📂 catalogoServicesTest 40ms
|< ✓ 📂 soporteServicesTest 129ms
|   ✓ ⚡ findByAllTest() 12ms
|   ✓ ⚡ findByIdTest() 12ms
|   ✓ ⚡ saveTest() 93ms
|   ✓ ⚡ deleteTest() 12ms
|> ✓ 📂 sucursalServicesTest 26ms
|> ✓ 📂 usuarioServicesTest 24ms
```

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.

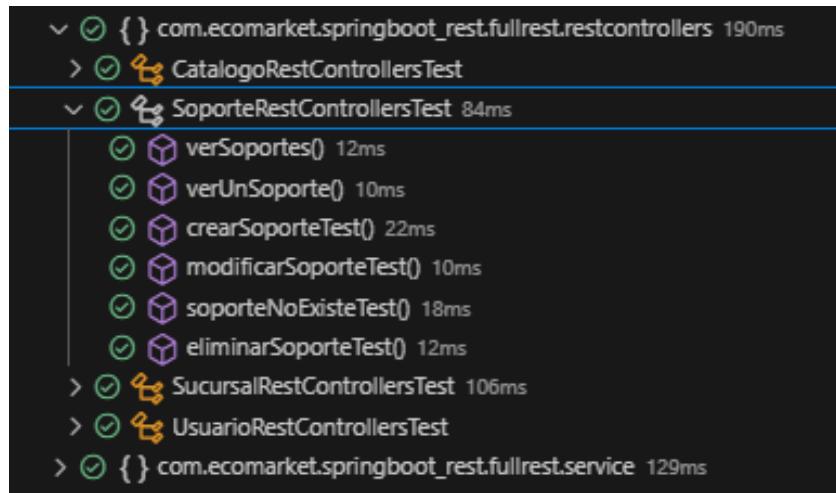
A continuación, se muestra un ejemplo del microservicio **Soporte**, incluyendo el controlador, su clase de prueba y la ejecución correspondiente. Esta misma estructura fue replicada en los demás microservicios del sistema.

```
J SoporteRestController.java X
src > main > java > com > ecomarket > springboot_rest > fullrest > restcontrollers > J SoporteRestController.java > ...
26 import io.swagger.v3.oas.annotations.tags.Tag;
27
28 @Tag(name="Soportes", description="API de soporte")
29 @RestController
30 @RequestMapping("api/Soportes")
31 public class SoporteRestController {
32
33     @Autowired
34     private SoporteServices soporteservice;
35
36
37     @Operation(summary = "Obtener lista de Soportes", description = "Devuelve todos los soportes",
38     @ApiResponse(responseCode = "200", description = "Lista de soporte retornada correctamente",
39                 content = @Content(mediaType = "application/json",
40                 schema = @Schema(implementation = Soporte.class)))
41
42     @GetMapping
43     public List<Soporte> mostrarSoportes(){
44         return soporteservice.findByAll();
45     }
46
47
48     @Operation(summary = "Obtener soporte por ID", description = "Obtiene el detalle de un soporte",
49     @ApiResponses(value = {
50         @ApiResponse(responseCode = "200", description = "soporte encontrado",
51                     content = @Content(mediaType = "application/json", schema = @Schema(implementation = Soporte.class)),
52         @ApiResponse(responseCode = "404", description = "soporte no encontrado")
53     })
54
55     @GetMapping("/{id}")
56     public ResponseEntity<?> verSoporte(@PathVariable Long id){
57         Optional<Soporte> soporteOptional = soporteservice.findById(id);
58         if (soporteOptional.isPresent()) {
59             return ResponseEntity.ok(soporteOptional.orElseThrow());
60         }
61         return ResponseEntity.notFound().build();
62     }
63 }
```

```
J SoporteRestController.java X
src > main > java > com > ecomarket > springboot_rest > fullrest > restcontrollers > J SoporteRestController.java > ...
31  public class SoporteRestController {
32
33
34      @Operation(summary = "Crear un nuevo soporte", description = "Crea un soporte con los datos proporcionados")
35      @ApiResponse(responseCode = "201", description = "soporte creado correctamente",
36                  content = @Content(mediaType = "application/json", schema = @Schema(implementations = @ApiResponse))
37
38      @PostMapping
39      public ResponseEntity<Soporte> crearSoporte(@RequestBody Soporte unSoporte){
40          return ResponseEntity.status(HttpStatus.CREATED). body(soporteservice.save(unSoporte));
41      }
42
43
44      @Operation(summary = "Modificar soporte por ID", description = "Modificar información de un soporte existente")
45      @ApiResponses(value = {
46          @ApiResponse(responseCode = "200", description = "soporte modificado",
47                      content = @Content(mediaType = "application/json", schema = @Schema(implementations = @ApiResponse))
48          @ApiResponse(responseCode = "404", description = "soporte no encontrado")
49      })
50
51      @PutMapping("/{id}")
52      public ResponseEntity<?> modificarSoporte(@PathVariable Long id, @RequestBody Soporte unSoporte){
53          Optional<Soporte> soporteOptional = soporteservice.findById(id);
54
55          if (soporteOptional.isPresent()){
56              Soporte soporteExistente = soporteOptional.get();
57
58              soporteExistente.setIdUsuario(unSoporte.getIdUsuario());
59              soporteExistente.setTipo(unSoporte.getTipo());
60              soporteExistente.setMensaje(unSoporte.getMensaje());
61              soporteExistente.setFechaCreacion(unSoporte.getFechaCreacion());
62              soporteExistente.setEstado(unSoporte.getEstado());
63              soporteExistente.setRespuesta(unSoporte.getRespuesta());
64              soporteExistente.setFechaRespuesta(unSoporte.getFechaRespuesta());
65              soporteExistente.setActivo(unSoporte.getActivo());
66
67              Soporte soporteModificado = soporteservice.save(soporteExistente);
68              return ResponseEntity.ok(soporteModificado);
69          }
70
71          return ResponseEntity.notFound().build();
72      }
73
74      @Operation(summary = "Eliminar soporte por ID", description = "Elimina un objeto específico")
75      @ApiResponses(value = {
76          @ApiResponse(responseCode = "200", description = "soporte eliminado",
77                      content = @Content(mediaType = "application/json", schema = @Schema(implementations = @ApiResponse))
78          @ApiResponse(responseCode = "404", description = "soporte no encontrado")
79      })
80
81      @DeleteMapping("/{id}")
82      public ResponseEntity<?> eliminarSoporte (@PathVariable Long id){
83          Soporte unSoporte = new Soporte();
84          unSoporte.setId(id);
85          Optional <Soporte> soporteOptional = soporteservice.delete(unSoporte);
86          if (soporteOptional.isPresent()){
87              return ResponseEntity.noContent().build();
88          }
89          return ResponseEntity.notFound().build();
90      }
91
92      }
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
```

```
J SoporteRestControllersTest.java X
ket > springboot_rest > fullrest > restcontrollers > J SoporteRestControllersTest.java > Language Support for Java(TM) by Red Hat >
Search (Ctrl+Shift+F) botTest
28 @AutoConfigureMockMvc
29 public class SoporteRestControllersTest {
30
31     @Autowired
32     private MockMvc mockMvc;
33
34     @MockitoBean
35     private SoporteServiceImpl SoporteServiceImpl;           //mocks, objeto simulado
36
37     @Autowired
38     private ObjectMapper objectMapper;                      //atributo para gestionar el método save
39
40     List<Soporte> listaSoportes;
41
42
43     @Test
44     public void verSoportes() throws Exception{
45         when(SoporteServiceImpl.findAll()).thenReturn(listaSoportes);
46         mockMvc.perform(get(uriTemplate:"/api/Soportes")
47                         .contentType(MediaType.APPLICATION_JSON)
48                         .andExpect(status().isOk());
49     }
50
51
52     @Test
53     public void verUnSoporte(){
54         Soporte unSoporte = new Soporte (activo:true,estado:"abierto", LocalDateTime.now(),
55         try{
56             when(SoporteServiceImpl.findById(id:1L)).thenReturn(Optional.of(unSoporte));
57             mockMvc.perform(get(uriTemplate:"/api/Soportes/1")
58                             .contentType(MediaType.APPLICATION_JSON))
59                             .andExpect(status().isOk());
60
61         }catch (Exception ex){
62             fail("Error.. en el test " + ex.getMessage());
63         }
64     }
65
66     @Test
67     public void crearSoporteTest() throws Exception {
68         Soporte soporteEntrada = new Soporte(activo:null, estado:"nuevo", LocalDateTime.now());
69         Soporte soporteGuardado = new Soporte(activo:true, estado:"nuevo", LocalDateTime.now());
70
71         when(SoporteServiceImpl.save(any(type:Soporte.class))).thenReturn(soporteGuardado);
72
73         mockMvc.perform(post(uriTemplate:"/api/Soportes")
74                         .contentType(MediaType.APPLICATION_JSON)
75                         .content(objectMapper.writeValueAsString(soporteEntrada)))
76                         .andExpect(status().isCreated());
77     }
```

```
J SoporteRestControllersTest.java X
ket > springboot_rest > fullrest > restcontrollers > J SoporteRestControllersTest.java > Language Support for Java(TM) by Red Hat > {} c
29  public class SoporteRestControllersTest {
30
31      @Test
32      public void modificarSoporteTest() throws Exception {
33          Soporte soporteEntrada = new Soporte(activo:null, estado:"en_progreso", LocalDateTime.now());
34          Soporte soporteModificado = new Soporte(activo:true, estado:"en_progreso", LocalDateTime.now());
35
36          // Simula que el soporte con ID 5 sí existe
37          when(SoporteServiceImpl.findById(id:5L)).thenReturn(Optional.of(soporteModificado));
38
39          when(SoporteServiceImpl.save(any(type:Soporte.class))).thenReturn(soporteModificado);
40
41          mockMvc.perform(put(uriTemplate:"/api/Soportes/5")
42                          .contentType(MediaType.APPLICATION_JSON)
43                          .content(objectMapper.writeValueAsString(soporteEntrada)))
44                          .andExpect(status().isOk());
45      }
46
47
48      @Test
49      public void soporteNoExisteTest() throws Exception {
50          when(SoporteServiceImpl.findById(id:99L)).thenReturn(Optional.empty());
51
52          mockMvc.perform(get(uriTemplate:"/api/Soportes/99")
53                          .contentType(MediaType.APPLICATION_JSON))
54                          .andExpect(status().isNotFound());
55      }
56
57
58      @Test
59      public void eliminarSoporteTest() throws Exception {
60          Soporte soporteExistente = new Soporte(activo:true, estado:"cerrado", LocalDateTime.now());
61
62          // Simula que el soporte con ID 6 fue encontrado y eliminado
63          when(SoporteServiceImpl.delete(any(type:Soporte.class))).thenReturn(Optional.of(soporteExistente));
64
65          mockMvc.perform(delete(uriTemplate:"/api/Soportes/6"))
66                          .andExpect(status().isNoContent());
67      }
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118}
```



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.

The screenshot shows the Swagger UI interface for the 'Soportes' API. At the top, there's a header bar with a back arrow, forward arrow, and refresh icon, followed by the URL 'localhost:8080/swagger-ui/index.html#/'. Below the header, the title 'Soportes API de soporte' is displayed. The main content area lists five API endpoints:

- GET /api/Soportes/{id}** Obtener soporte por ID
- PUT /api/Soportes/{id}** Modificar soporte por ID
- DELETE /api/Soportes/{id}** Eliminar soporte por ID
- GET /api/Soportes** Obtener lista de Soportes
- POST /api/Soportes** Crear un nuevo soporte

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.

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

git add .

```
DELL@Wixo MINGW64 ~/Desktop/FullStack (master)
$ git add .
```

git commit -m "Experiencia 3 Testing"

```
DELL@Wixo MINGW64 ~/Desktop/FullStack (master)
$ git commit -m "Experiencia 3 Testing"
[master bbbc4c4] Experiencia 3 Testing
 195 files changed, 0 insertions(+), 0 deletions(-)
```

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

```
DELL@Wixo MINGW64 ~/Desktop/FullStack (master)
$ git remote add origin https://github.com/Bryckson/Bryckson/Exp3_Gutierrez_Toro_Gonzales.git
```

git push -u origin main

```
DELL@Wixo MINGW64 ~/Desktop/FullStack (master)
$ git push -u origin master
remote: Not Found
```

Bryckson / Exp3_Gutierrez_Toro_Gonzales Private

main 1 Branch 0 Tags

Bryckson Experiencia 3 Testing

bbbc4c4 · 12 minutes ago 2 Commits

Carpeta Exp1 Experiencia 3 Testing 12 minutes ago

Carpeta Exp2 Experiencia 3 Testing 12 minutes ago

Carpeta Exp3 Experiencia 3 Testing 12 minutes ago

README

Add a README

Activity 0 stars 0 watching 0 forks

Resumen del Repositorio

Resumir este repositorio

About

No description, website, or topics provided.

Releases

No releases published

Bryckson / Exp3_Gutierrez_Toro_Gonzales

Files

main 1 Branch 0 Tags

Exp3_Gutierrez_Toro_Gonzales / Carpeta Exp3 /

Bryckson Experiencia 3 Testing

bbbc4c4 · 14 minutes ago History

Name	Last commit message	Last commit date
..		
.vscode	Experiencia 3 Testing	14 minutes ago
fullrest	Experiencia 3 Testing	14 minutes ago
Diagrama de clases corregido.pdf	Experiencia 3 Testing	14 minutes ago
Informe Evaluacion3_fullsatck.docx	Experiencia 3 Testing	14 minutes ago
README.md	Experiencia 3 Testing	14 minutes ago
basedato_ecomarket.sql	Experiencia 3 Testing	14 minutes ago

README.md

Documentación de la API generada automáticamente con OAS 3.1 y Swagger UI:
<http://localhost:8080/swagger-ui/index.html#/>

Paths para ejecutar consultas de los servicios en Postman.

Usuario:

GET: <http://localhost:8080/api/Usuarios>

GET(id): <http://localhost:8080/api/Usuarios/3>

POST: <http://localhost:8080/api/Usuarios>

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.