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

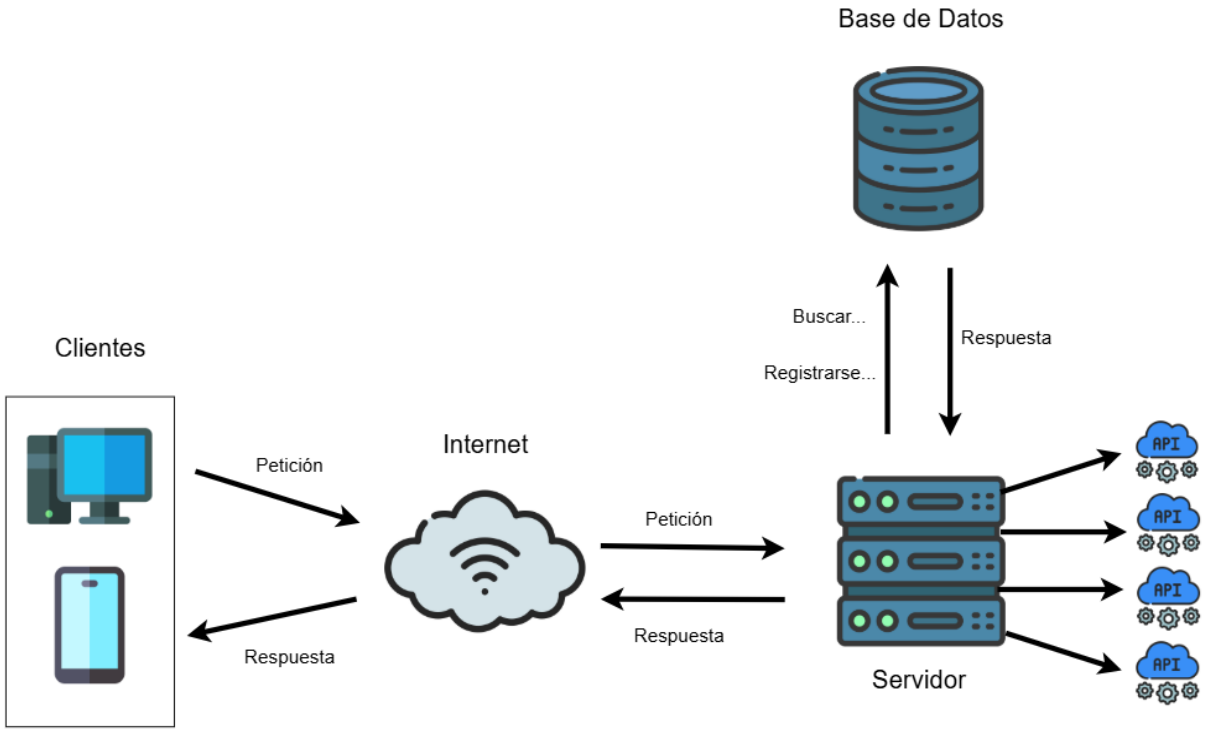
Tabla de contenidos

Contenido

1. Introducción

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

```java
public class BoletaServiceImplTest {

    @InjectMocks
    private BoletaServiceImpl service;

    @Mock
    private BoletaRepository repository;

    List<Boleta> boletaList = new ArrayList<>();

    @BeforeEach
    public void init() {
        MockitoAnnotations.openMocks(this);
        cargarBoletas();
    }

    @Test
    public void buscarTodosTest() {
        when(repository.findAll()).thenReturn(boletaList);

        List<Boleta> response = service.findByAll();

        assertEquals(3, response.size());
        verify(repository, times(1)).findAll();
    }

    @Test
    public void buscarPorIdTest() {
        Boleta boleta = new Boleta(numeroBoleta:2L, rutComprador:"12.345.678-9", cantidadProductos…5, 49990, "2025-06-17", "Essenza Noire, Velvet Storm, Mystic Amber");
        when(repository.findById(2L)).thenReturn(Optional.of(boleta));

        Optional<Boleta> response = service.findById(boleta:2L);

        assertTrue(response.isPresent());
        assertEquals("12.345.678-9", response.get().getRutComprador());
        assertEquals("Essenza Noire, Velvet Storm, Mystic Amber", response.get().getDescripcion());
        verify(repository, times(1)).findById(2L);
    }
```

```java
@Test
public void crearBoletaTest() {
    Boleta boletaSinId = new Boleta(numeroBoleta:null, rutComprador:"11.222.333-4", cantidadProductos…2, 19990, "2025-06-10", "Luna Ardiente, Mystic Amber");
    Boleta boletaConId = new Boleta(numeroBoleta:10L, rutComprador:"11.222.333-4", cantidadProductos…2, 19990, "2025-06-10", "Luna Ardiente, Mystic Amber");

    when(repository.save(boletaSinId)).thenReturn(boletaConId);

    Boleta response = service.save(boletaSinId);

    assertNotNull(response);
    assertEquals(10L, response.getNumeroBoleta());
    assertEquals("Luna Ardiente, Mystic Amber", response.getDescripcion());
    verify(repository, times(1)).save(boletaSinId);
}

@Test
public void modificarBoletaTest() {
    Boleta boletaOriginal = new Boleta(numeroBoleta:3L, rutComprador:"11.222.333-4", cantidadProductos…2, 19990, "2025-06-10", "Luna Ardiente, Mystic Amber");
    Boleta boletaModificada = new Boleta(numeroBoleta:3L, rutComprador:"11.222.333-4", cantidadProductos…4, 39980, "2025-06-10", "Essenza Noire, Luna Ardiente, Mystic Amber Inter

    when(repository.findById(3L)).thenReturn(Optional.of(boletaOriginal));
    when(repository.save(any(Boleta.class))).thenReturn(boletaModificada);

    Optional<Boleta> boletaBD = service.findById(boleta:3L);
    Boleta updatedBoleta = null;
    if (boletaBD.isPresent()) {
        Boleta boleta = boletaBD.get();
        boleta.setCantidadProductos(cantidadProductos:4);
        boleta.setPrecio(precio:39980);
        boleta.setDescripcion(descripcion:"Essenza Noire, Luna Ardiente, Mystic Amber Intense");

        updatedBoleta = service.save(boleta);
    }
```

```
    assertNotNull(updatedBoleta);
    assertEquals(4, updatedBoleta.getCantidadProductos());
    assertEquals(39980, updatedBoleta.getPrecio());
    assertEquals("Essenza Noire, Luna Ardiente, Mystic Amber Intense", updatedBoleta.getDescripcion());

    verify(repository).findById(3L);
    verify(repository).save(any(Boleta.class));
}

@Test
public void eliminarBoletaTest() {
    Boleta boleta = new Boleta(numeroBoleta:4L, rutComprador:"10.111.222-3", cantidadProductos…3, 29990, "2025-06-01", "Essenza Noire, Velvet Storm");

    when(repository.findById(4L)).thenReturn(Optional.of(boleta));

    Optional<Boleta> response = service.delete(boleta);

    assertTrue(response.isPresent());
    assertEquals("Essenza Noire, Velvet Storm", response.get().getDescripcion());

    verify(repository, times(1)).findById(4L);
    verify(repository, times(1)).delete(boleta);
}

private void cargarBoletas() {
    boletaList.add(new Boleta(numeroBoleta:1L, rutComprador:"10.111.222-3", cantidadProductos…2, 29990, "2025-06-01", "Essenza Noire, Velvet Storm"));
    boletaList.add(new Boleta(numeroBoleta:2L, rutComprador:"12.345.678-9", cantidadProductos…5, 49990, "2025-06-17", "Essenza Noire, Velvet Storm, Mystic Amber"));
    boletaList.add(new Boleta(numeroBoleta:3L, rutComprador:"11.222.333-4", cantidadProductos…2, 19990, "2025-06-10", "Luna Ardiente, Mystic Amber"));
}
```

- ServiceImplTest de Envío

```
public class EnvioServiceImplTest {

    @InjectMocks
    private EnvioServiceImpl service;

    @Mock
    private EnvioRepository repository;

    private List<Envio> envioList = new ArrayList<>();

    @BeforeEach
    public void setUp() {
        MockitoAnnotations.openMocks(this);
        cargarEnvios();
    }

    @Test
    public void buscarTodosEnviosTest() {
        when(repository.findAll()).thenReturn(envioList);

        List<Envio> response = service.findByAll();

        assertEquals(3, response.size());
        verify(repository, times(1)).findAll();
    }

    @Test
    public void buscarEnvioPorIdTest() {
        Envio envioEjemplo = new Envio(numEnvio:2L, rutComprador:"12.345.678-9", numBoleta:101L, estado:"En camino");
        when(repository.findById(2L)).thenReturn(Optional.of(envioEjemplo));

        Optional<Envio> response = service.findById(numEnvio:2L);

        assertTrue(response.isPresent());
        assertEquals("12.345.678-9", response.get().getRutComprador());
        assertEquals("En camino", response.get().getEstado());
        verify(repository, times(1)).findById(2L);
    }
```

```java
@Test
public void crearEnvioTest() {
    Envio envioSinId = new Envio(numEnvio:null, rutComprador:"98.765.432-1", numBoleta:102L, estado:"Pendiente");
    Envio envioConId = new Envio(numEnvio:10L, rutComprador:"98.765.432-1", numBoleta:102L, estado:"Pendiente");

    when(repository.save(envioSinId)).thenReturn(envioConId);

    Envio response = service.save(envioSinId);

    assertNotNull(response);
    assertEquals(10L, response.getNumEnvio());
    assertEquals("Pendiente", response.getEstado());
    verify(repository, times(1)).save(envioSinId);
}

@Test
public void modificarEnvioTest() {
    Envio envioOriginal = new Envio(numEnvio:3L, rutComprador:"11.222.333-4", numBoleta:103L, estado:"Pendiente");
    Envio envioModificado = new Envio(numEnvio:3L, rutComprador:"11.222.333-4", numBoleta:103L, estado:"Entregado");

    when(repository.findById(3L)).thenReturn(Optional.of(envioOriginal));
    when(repository.save(any(Envio.class))).thenReturn(envioModificado);

    Optional<Envio> envioBD = service.findById(numEnvio:3L);
    Envio updatedEnvio = null;

    if (envioBD.isPresent()) {
        Envio envio = envioBD.get();
        envio.setEstado(estado:"Entregado");

        updatedEnvio = service.save(envio);
    }

    assertNotNull(updatedEnvio);
    assertEquals("Entregado", updatedEnvio.getEstado());

    verify(repository).findById(3L);
    verify(repository).save(any(Envio.class));
}
```

```java
@Test
public void eliminarEnvioTest() {
    Envio envio = new Envio(numEnvio:4L, rutComprador:"22.333.444-5", numBoleta:104L, estado:"Cancelado");

    when(repository.findById(4L)).thenReturn(Optional.of(envio));

    Optional<Envio> response = service.delete(envio);

    assertTrue(response.isPresent());
    assertEquals("Cancelado", response.get().getEstado());

    verify(repository, times(1)).findById(4L);
    verify(repository, times(1)).delete(envio);
}

private void cargarEnvios() {
    envioList.add(new Envio(numEnvio:1L, rutComprador:"10.111.222-3", numBoleta:100L, estado:"Pendiente"));
    envioList.add(new Envio(numEnvio:2L, rutComprador:"12.345.678-9", numBoleta:101L, estado:"En camino"));
    envioList.add(new Envio(numEnvio:3L, rutComprador:"11.222.333-4", numBoleta:103L, estado:"Pendiente"));
}
```

- ServiceImplTest de Producto

```java
public class ProductoServiceImplTest {

    @InjectMocks
    private ProductServiceImpl service;

    @Mock
    private ProductoRepository repository;

    List<Producto> list = new ArrayList<>();

    @BeforeEach
    public void init() {
        MockitoAnnotations.openMocks(this);
        cargarProductos();
    }

    @Test
    public void buscarTodosTest() {
        when(repository.findAll()).thenReturn(list);

        List<Producto> response = service.findByAll();

        assertEquals(3, response.size());
        verify(repository, times(1)).findAll();
    }

    @Test
    public void buscarPorIdTest() {
        Producto producto = new Producto(id:2L, nombre:"Velvet Storm", descripcion:"perfume para hombre 100ML", precio:35990, cantidad:20);
        when(repository.findById(2L)).thenReturn(Optional.of(producto));

        Optional<Producto> response = service.findById(id:2L);

        assertTrue(response.isPresent());
        assertEquals("Velvet Storm", response.get().getNombre());
        verify(repository, times(1)).findById(2L);
    }
```

```java
@Test
public void crearProductoTest() {
    Producto productoSinId = new Producto(id:null, nombre:"Luna Ardiente", descripcion:"perfume para mujer 50ML", precio:28990, cantidad:15);
    Producto productoConId = new Producto(id:10L, nombre:"Luna Ardiente", descripcion:"perfume para mujer 50ML", precio:28990, cantidad:15);

    when(repository.save(productoSinId)).thenReturn(productoConId);

    Producto response = service.save(productoSinId);

    assertNotNull(response);
    assertEquals(10L, response.getId());
    assertEquals("Luna Ardiente", response.getNombre());
    verify(repository, times(1)).save(productoSinId);
}

@Test
public void modificarProductoTest() {
    Producto productoOriginal = new Producto(id:3L, nombre:"Mystic Amber", descripcion:"perfume intenso para mujer", precio:31990, cantidad:10);
    Producto productoModificado = new Producto(id:3L, nombre:"Mystic Amber Intense", descripcion:"fragancia renovada con notas cálidas", precio:34990, cantidad:12);

    when(repository.findById(3L)).thenReturn(Optional.of(productoOriginal));
    when(repository.save(any(Producto.class))).thenReturn(productoModificado);

    Optional<Producto> productoBD = service.findById(id:3L);
    Producto updatedProduct = null;
    if (productoBD.isPresent()) {
        Producto product = productoBD.get();
        product.setNombre(nombre:"Mystic Amber Intense");
        product.setDescripcion(descripcion:"fragancia renovada con notas cálidas");
        product.setPrecio(precio:34990);
        product.setCantidad(cantidad:12);

        updatedProduct = service.save(product);
    }

    assertNotNull(updatedProduct);
    assertEquals("Mystic Amber Intense", updatedProduct.getNombre());
    assertEquals("fragancia renovada con notas cálidas", updatedProduct.getDescripcion());
    assertEquals(34990, updatedProduct.getPrecio());
```

```
    verify(repository).findById(3L);
    verify(repository).save(any(Producto.class));
}

@Test
public void eliminarProductoTest() {
    Producto producto = new Producto(id:4L, nombre:"Oceanus Bleu", descripcion:"perfume fresco para hombre 100ML", precio:44990, cantidad:10);

    when(repository.findById(4L)).thenReturn(Optional.of(producto));

    Optional<Producto> response = service.delete(producto);

    assertTrue(response.isPresent());
    assertEquals("Oceanus Bleu", response.get().getNombre());

    verify(repository, times(1)).findById(4L);
    verify(repository, times(1)).delete(producto);
}

public void cargarProductos() {
    list.add(new Producto(id:1L, nombre:"Essenza Noire", descripcion:"perfume floral para mujer 100ML", precio:32990, cantidad:25));
    list.add(new Producto(id:2L, nombre:"Velvet Storm", descripcion:"perfume para hombre 100ML", precio:35990, cantidad:20));
    list.add(new Producto(id:3L, nombre:"Mystic Amber", descripcion:"perfume intenso para mujer", precio:31990, cantidad:10));
}
```

- ServiceImplTest de Usuario

```
public class UsuarioServiceImplTest {

    @InjectMocks
    private UsuarioServiceImpl service;

    @Mock
    private UsuarioRepository repository;

    List<Usuario> lista = new ArrayList<>();

    @BeforeEach
    public void init() {
        MockitoAnnotations.openMocks(this);
        cargarUsuarios();
    }

    @Test
    public void buscarTodosTest() {
        when(repository.findAll()).thenReturn(lista);

        List<Usuario> response = service.findByAll();

        assertEquals(3, response.size());
        verify(repository, times(1)).findAll();
    }

    @Test
    public void buscarPorIdTest() {
        Usuario user = new Usuario(rut:"14.567.789-2", nombre:"Valentina Rojas", contrasena:"clave123", rol:"User", email:"valentina.r@gmail.com");
        when(repository.findById("14.567.789-2")).thenReturn(Optional.of(user));

        Optional<Usuario> response = service.findById(rut:"14.567.789-2");

        assertTrue(response.isPresent());
        assertEquals("Valentina Rojas", response.get().getNombre());
        verify(repository, times(1)).findById("14.567.789-2");
    }
```

```java
@Test
public void crearUsuarioTest() {
    Usuario sinRut = new Usuario(rut:"12.345.678-9", nombre:"Tomás Paredes", contrasena:"bluepass", rol:"Admin", email:"tomas.p@gmail.com");

    when(repository.save(sinRut)).thenReturn(sinRut);

    Usuario response = service.save(sinRut);

    assertNotNull(response);
    assertEquals("Tomás Paredes", response.getNombre());
    assertEquals("Admin", response.getRol());
    verify(repository, times(1)).save(sinRut);
}

@Test
public void modificarUsuarioTest() {
    Usuario original = new Usuario(rut:"19.876.543-1", nombre:"Lucía Herrera", contrasena:"pass123", rol:"User", email:"lucia.h@gmail.com");
    Usuario modificado = new Usuario(rut:"19.876.543-1", nombre:"Lucía Herrera", contrasena:"nuevaClave456", rol:"Admin", email:"lucia.h@gmail.com");

    when(repository.findById("19.876.543-1")).thenReturn(Optional.of(original));
    when(repository.save(any(Usuario.class))).thenReturn(modificado);

    Optional<Usuario> usuarioBD = service.findById(rut:"19.876.543-1");
    Usuario updated = null;
    if (usuarioBD.isPresent()) {
        Usuario user = usuarioBD.get();
        user.setContrasena(contrasena:"nuevaClave456");
        user.setRol(rol:"Admin");

        updated = service.save(user);
    }

    assertNotNull(updated);
    assertEquals("nuevaClave456", updated.getContrasena());
    assertEquals("Admin", updated.getRol());
    verify(repository).findById("19.876.543-1");
    verify(repository).save(any(Usuario.class));
}
```

```java
@Test
public void eliminarUsuarioTest() {
    Usuario user = new Usuario(rut:"13.321.654-7", nombre:"Sofía Álvarez", contrasena:"clave789", rol:"User", email:"sofia.a@gmail.com");

    when(repository.findById("13.321.654-7")).thenReturn(Optional.of(user));

    Optional<Usuario> response = service.delete(user);

    assertTrue(response.isPresent());
    assertEquals("Sofía Álvarez", response.get().getNombre());
    verify(repository, times(1)).findById("13.321.654-7");
    verify(repository, times(1)).delete(user);
}

public void cargarUsuarios() {
    lista.add(new Usuario(rut:"10.123.456-1", nombre:"Gabriel Soto", contrasena:"pass001", rol:"User", email:"gabriel.soto@gmail.com"));
    lista.add(new Usuario(rut:"14.567.789-2", nombre:"Valentina Rojas", contrasena:"clave123", rol:"User", email:"valentina.r@gmail.com"));
    lista.add(new Usuario(rut:"19.876.543-1", nombre:"Lucía Herrera", contrasena:"pass123", rol:"User", email:"lucia.h@gmail.com"));
}
```

3.2 Pruebas de Integración

- RestControllerTest de Boleta

```java
@SpringBootTest
@AutoConfigureMockMvc
public class BoletaControllerTest {

    @Autowired
    private MockMvc mockmvc;

    @Autowired
    private ObjectMapper objectMapper;

    @MockitoBean
    private BoletaServiceImpl boletaServiceImpl;

    private List<Boleta> listaBoletas;

    // Ver todas las boletas
    @Test
    public void verBoletasTest() throws Exception {
        when(boletaServiceImpl.findByAll()).thenReturn(listaBoletas);
        mockmvc.perform(get("/api/boletas")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk());
    }

    // Ver una boleta
    @Test
    public void verUnaBoletaTest() {
        Boleta b = new Boleta(numeroBoleta:1L, rutComprador:"10.222.333-4", cantidadProductos…2, 15000, "2024-06-17", "Compra perfumes");
        try {
            when(boletaServiceImpl.findById(boleta:1L)).thenReturn(Optional.of(b));
            mockmvc.perform(get("/api/boletas/1")
                    .contentType(MediaType.APPLICATION_JSON))
                    .andExpect(status().isOk());
        } catch (Exception ex) {
            fail("El test arrojó un error: " + ex.getMessage());
        }
    }
```

```java
// Boleta no existe
@Test
public void boletaNoExisteTest() throws Exception {
    when(boletaServiceImpl.findById(boleta:999L)).thenReturn(Optional.empty());
    mockmvc.perform(get("/api/boletas/999")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
}

// Crear boleta
@Test
public void crearBoletaTest() throws Exception {
    Boleta nueva = new Boleta(numeroBoleta:0L, rutComprador:"11.111.111-1", cantidadProductos…1, 20000, "2024-06-17", "Fragancia");
    Boleta guardada = new Boleta(numeroBoleta:10L, rutComprador:"11.111.111-1", cantidadProductos…1, 20000, "2024-06-17", "Fragancia");
    when(boletaServiceImpl.save(any(Boleta.class))).thenReturn(guardada);

    mockmvc.perform(post("/api/boletas")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(nueva)))
            .andExpect(status().isCreated());
}

// Modificar boleta existente
@Test
public void modificarBoletaTest() throws Exception {
    Boleta original = new Boleta(numeroBoleta:5L, rutComprador:"12.345.678-9", cantidadProductos…3, 30000, "2024-06-17", "Compra original");
    Boleta modificada = new Boleta(numeroBoleta:5L, rutComprador:"12.345.678-9", cantidadProductos…5, 50000, "2024-06-18", "Compra modificada");

    when(boletaServiceImpl.findById(boleta:5L)).thenReturn(Optional.of(original));
    when(boletaServiceImpl.save(any(Boleta.class))).thenReturn(modificada);

    mockmvc.perform(put("/api/boletas/5")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificada)))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(modificada)));
}
```

```java
// Intentar modificar boleta inexistente
@Test
public void modificarBoletaNoExisteTest() throws Exception {
    Boleta modificada = new Boleta(numeroBoleta:99L, rutComprador:"9.999.999-9", cantidadProductos…1, 10000, "2024-06-17", "Mod inexistente");

    when(boletaServiceImpl.findById(boleta:99L)).thenReturn(Optional.empty());

    mockmvc.perform(put("/api/boletas/99")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificada)))
            .andExpect(status().isNotFound());
}

// Eliminar boleta existente
@Test
public void eliminarBoletaTest() throws Exception {
    Boleta boletaEliminar = new Boleta(numeroBoleta:7L, rutComprador:"22.222.222-2", cantidadProductos…1, 10000, "2024-06-17", "Eliminar prueba");

    when(boletaServiceImpl.delete(any(Boleta.class))).thenReturn(Optional.of(boletaEliminar));

    mockmvc.perform(delete("/api/boletas/7")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(boletaEliminar)));
}
```

- RestControllerTest de Envío

```java
@SpringBootTest
@AutoConfigureMockMvc
public class EnvioControllerTest {

    @Autowired
    private MockMvc mockmvc;

    @Autowired
    private ObjectMapper objectMapper;

    @MockitoBean
    private EnvioServiceImpl envioServiceImpl;

    private List<Envio> listaEnvios;

    // Ver todos los envios
    @Test
    public void verEnviosTest() throws Exception {
        when(envioServiceImpl.findByAll()).thenReturn(listaEnvios);
        mockmvc.perform(get("/api/envios")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk());
    }

    // Ver un envio
    @Test
    public void verUnEnvioTest() {
        Envio e = new Envio(numEnvio:1L, rutComprador:"10.222.333-4", numBoleta:5L, estado:"Enviado");
        try {
            when(envioServiceImpl.findById(numEnvio:1L)).thenReturn(Optional.of(e));
            mockmvc.perform(get("/api/envios/1")
                    .contentType(MediaType.APPLICATION_JSON))
                    .andExpect(status().isOk());
        } catch (Exception ex) {
            fail("El test arrojó un error: " + ex.getMessage());
        }
    }
```

```java
// Envio no existe
@Test
public void envioNoExisteTest() throws Exception {
    when(envioServiceImpl.findById(numEnvio:999L)).thenReturn(Optional.empty());
    mockmvc.perform(get("/api/envios/999")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
}

// Crear envio
@Test
public void crearEnvioTest() throws Exception {
    Envio nuevo = new Envio(numEnvio:null, rutComprador:"11.111.111-1", numBoleta:10L, estado:"Pendiente");
    Envio guardado = new Envio(numEnvio:10L, rutComprador:"11.111.111-1", numBoleta:10L, estado:"Pendiente");
    when(envioServiceImpl.save(any(Envio.class))).thenReturn(guardado);

    mockmvc.perform(post("/api/envios")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(nuevo)))
            .andExpect(status().isCreated());
}

// Modificar envio existente
@Test
public void modificarEnvioTest() throws Exception {
    Envio original = new Envio(numEnvio:5L, rutComprador:"12.345.678-9", numBoleta:3L, estado:"Pendiente");
    Envio modificado = new Envio(numEnvio:5L, rutComprador:"12.345.678-9", numBoleta:3L, estado:"Entregado");

    when(envioServiceImpl.findById(numEnvio:5L)).thenReturn(Optional.of(original));
    when(envioServiceImpl.save(any(Envio.class))).thenReturn(modificado);

    mockmvc.perform(put("/api/envios/5")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificado)))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(modificado)));
}
```

```java
// Intentar modificar envio inexistente
@Test
public void modificarEnvioNoExisteTest() throws Exception {
    Envio modificado = new Envio(numEnvio:99L, rutComprador:"9.999.999-9", numBoleta:1L, estado:"Cancelado");

    when(envioServiceImpl.findById(numEnvio:99L)).thenReturn(Optional.empty());

    mockmvc.perform(put("/api/envios/99")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificado)))
            .andExpect(status().isNotFound());
}

// Eliminar envio existente
@Test
public void eliminarEnvioTest() throws Exception {
    Envio envioEliminar = new Envio(numEnvio:7L, rutComprador:"22.222.222-2", numBoleta:1L, estado:"Pendiente");

    when(envioServiceImpl.delete(any(Envio.class))).thenReturn(Optional.of(envioEliminar));

    mockmvc.perform(delete("/api/envios/7")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(envioEliminar)));
}

// Intentar eliminar envio inexistente
@Test
public void eliminarEnvioNoExisteTest() throws Exception {
    when(envioServiceImpl.delete(any(Envio.class))).thenReturn(Optional.empty());

    mockmvc.perform(delete("/api/envios/999")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
}
```

- RestControllerTest de Producto

```java
@SpringBootTest
@AutoConfigureMockMvc
public class ProductoControllerTest {
    @Autowired
    private MockMvc mockmvc;

    @Autowired
    private ObjectMapper objectMapper;

    @MockitoBean
    private ProductServiceImpl productoserviceimpl;

    private List<Producto> productosLista;

    //VER TODOS LOS PRODUCTOS
    @Test
    public void verProductosTest() throws Exception{
        when(productoserviceimpl.findByAll()).thenReturn(productosLista);
        mockmvc.perform(get("/api/productos")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk());
    }

    //ENCONTRAR UN SOLO PRODUCTO
    @Test
    public void verunProductoTest(){
        Producto unProducto = new Producto(id:1L,nombre:"Channel Lestro", descripcion:"Perfume de mujer 300ML",precio:50000,cantidad:100);
        try{
            when (productoserviceimpl.findById(id:1L)).thenReturn(Optional.of(unProducto));
            mockmvc.perform(get("/api/productos/1")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk());
        }
        catch(Exception ex){
            fail ("El testing arrojó un error "+ ex.getMessage());
        }
    }
```

```java
//Buscar producto inexsistente
@Test
public void productoNoExisteTest() throws Exception{
    when(productoserviceimpl.findById(id:20L)).thenReturn(Optional.empty());
    mockmvc.perform((get("/api/productos/20"))
    .contentType(MediaType.APPLICATION_JSON))
    .andExpect(status().isNotFound());
}

//CREAR PRODUCTO
@Test
public void crearProductoTest() throws Exception{
    Producto unProducto = new Producto(id:null, nombre:"Anto", descripcion:"Perfume mixto 100ml", precio:100, cantidad:10);
    Producto otroProducto = new Producto(id:12L, nombre:"Antonio Teteras Premium", descripcion:"Perfume hombre 75ML",precio:50000,cantidad:5);
    when (productoserviceimpl.save(any(Producto.class))).thenReturn(otroProducto);
    mockmvc.perform(post("/api/productos")
        .contentType(MediaType.APPLICATION_JSON)
        .content(objectMapper.writeValueAsString(unProducto)))
        .andExpect(status().isCreated());
}

// MODIFICAR PRODUCTO EXISTENTE
@Test
public void modificarProductoTest() throws Exception {
    Producto original = new Producto(id:5L, nombre:"Old Perfume", descripcion:"Descripción vieja", precio:30000, cantidad:15);
    Producto modificado = new Producto(id:5L, nombre:"New Perfume", descripcion:"Descripción nueva", precio:35000, cantidad:20);

    when(productoserviceimpl.findById(id:5L)).thenReturn(Optional.of(original));
    when(productoserviceimpl.save(any(Producto.class))).thenReturn(modificado);

    mockmvc.perform(put("/api/productos/5")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificado)))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(modificado)));
}
```

```java
// INTENTAR MODIFICAR PRODUCTO INEXISTENTE
@Test
public void modificarProductoNoExisteTest() throws Exception {
    Producto modificado = new Producto(id:99L, nombre:"Nombre", descripcion:"Desc", precio:10000, cantidad:1);

    when(productoserviceimpl.findById(id:99L)).thenReturn(Optional.empty());

    mockmvc.perform(put("/api/productos/99")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificado)))
            .andExpect(status().isNotFound());
}

// ELIMINAR PRODUCTO EXISTENTE
@Test
public void eliminarProductoTest() throws Exception {
    Producto productoEliminar = new Producto(id:7L, nombre:"Eliminar", descripcion:"Desc", precio:12345, cantidad:2);

    when(productoserviceimpl.delete(any(Producto.class))).thenReturn(Optional.of(productoEliminar));

    mockmvc.perform(delete("/api/productos/7")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(productoEliminar)));
}

// INTENTAR ELIMINAR PRODUCTO INEXISTENTE
@Test
public void eliminarProductoNoExisteTest() throws Exception {
    when(productoserviceimpl.delete(any(Producto.class))).thenReturn(Optional.empty());

    mockmvc.perform(delete("/api/productos/999")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
}
```

- RestControllerTest de Usuario

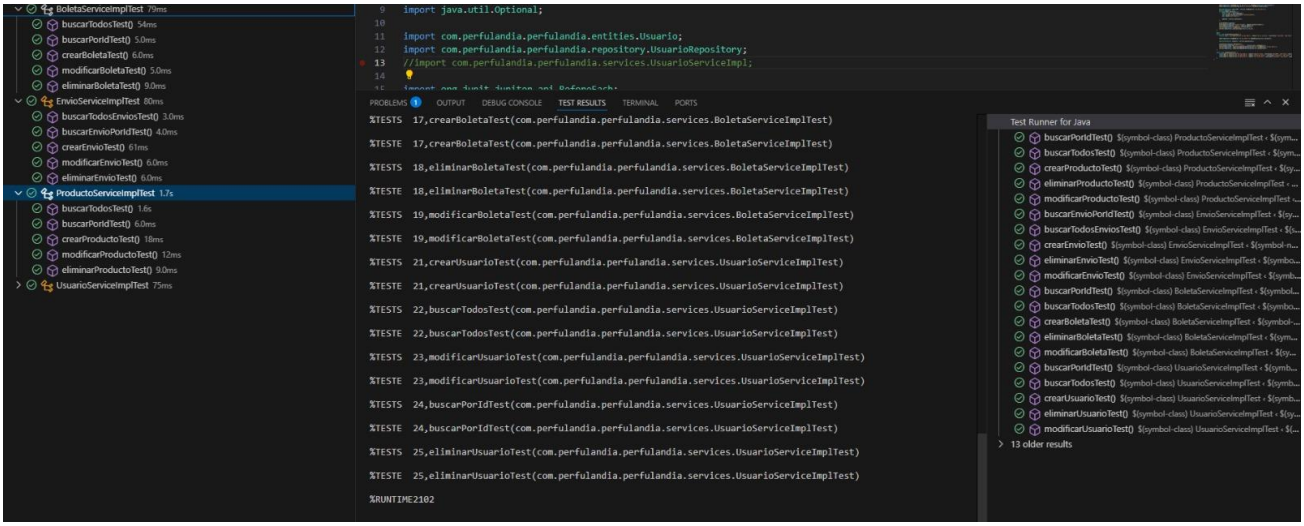
```java
@SpringBootTest
@AutoConfigureMockMvc
public class UsuarioControllerTest {

    @Autowired
    private MockMvc mockmvc;

    @Autowired
    private ObjectMapper objectMapper;

    @MockitoBean
    private UsuarioServiceImpl usuarioServiceImpl;

    private List<Usuario> usuariosLista;

    // Ver todos los usuarios
    @Test
    public void verUsuariosTest() throws Exception {
        when(usuarioServiceImpl.findByAll()).thenReturn(usuariosLista);
        mockmvc.perform(get("/api/usuarios")
                .contentType(MediaType.APPLICATION_JSON))
                .andExpect(status().isOk());
    }

    // Ver un usuario
    @Test
    public void verUnUsuarioTest() {
        Usuario user = new Usuario(rut:"10.111.222-3", nombre:"Juan", contrasena:"correo@gmail.com", rol:"clave123", email:"Admin");
        try {
            when(usuarioServiceImpl.findById(rut:"10.111.222-3")).thenReturn(Optional.of(user));
            mockmvc.perform(get("/api/usuarios/10.111.222-3")
                    .contentType(MediaType.APPLICATION_JSON))
                    .andExpect(status().isOk());
        } catch (Exception ex) {
            fail("El testing arrojó un error " + ex.getMessage());
        }
    }
```

```java
// Usuario no existe
@Test
public void usuarioNoExisteTest() throws Exception {
    when(usuarioServiceImpl.findById(rut:"99.999.999-9")).thenReturn(Optional.empty());
    mockmvc.perform(get("/api/usuarios/99.999.999-9")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
}

// Crear usuario
@Test
public void crearUsuarioTest() throws Exception {
    Usuario nuevoUsuario = new Usuario(rut:null, nombre:"Ana", contrasena:"ana@email.com", rol:"clave123", email:"Cliente");
    Usuario usuarioGuardado = new Usuario(rut:"12.345.678-9", nombre:"Ana", contrasena:"ana@email.com", rol:"clave123", email:"Cliente");
    when(usuarioServiceImpl.save(any(Usuario.class))).thenReturn(usuarioGuardado);

    mockmvc.perform(post("/api/usuarios")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(nuevoUsuario)))
            .andExpect(status().isCreated());
}


// MODIFICAR USUARIO EXISTENTE
@Test
public void modificarUsuarioTest() throws Exception {
    Usuario original = new Usuario(rut:"10.111.222-3", nombre:"Juan", contrasena:"correo@gmail.com", rol:"clave123", email:"Admin");
    Usuario modificado = new Usuario(rut:"10.111.222-3", nombre:"Juan Pérez", contrasena:"nuevoCorreo@gmail.com", rol:"nuevaClave", email:"Cliente");

    when(usuarioServiceImpl.findById(rut:"10.111.222-3")).thenReturn(Optional.of(original));
    when(usuarioServiceImpl.save(any(Usuario.class))).thenReturn(modificado);

    mockmvc.perform(put("/api/usuarios/10.111.222-3")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificado)))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(modificado)));
}
```

```java
// INTENTAR MODIFICAR USUARIO INEXISTENTE
@Test
public void modificarUsuarioNoExisteTest() throws Exception {
    Usuario modificado = new Usuario(rut:"11.111.111-1", nombre:"Nombre", contrasena:"correo@ejemplo.com", rol:"clave", email:"Cliente");

    when(usuarioServiceImpl.findById(rut:"11.111.111-1")).thenReturn(Optional.empty());

    mockmvc.perform(put("/api/usuarios/11.111.111-1")
            .contentType(MediaType.APPLICATION_JSON)
            .content(objectMapper.writeValueAsString(modificado)))
            .andExpect(status().isNotFound());
}

// ELIMINAR USUARIO EXISTENTE
@Test
public void eliminarUsuarioTest() throws Exception {
    Usuario usuarioEliminar = new Usuario(rut:"12.345.678-9", nombre:"Ana", contrasena:"ana@email.com", rol:"clave123", email:"Cliente");

    when(usuarioServiceImpl.delete(any(Usuario.class))).thenReturn(Optional.of(usuarioEliminar));

    mockmvc.perform(delete("/api/usuarios/12.345.678-9")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isOk())
            .andExpect(content().json(objectMapper.writeValueAsString(usuarioEliminar)));
}

// INTENTAR ELIMINAR USUARIO INEXISTENTE
@Test
public void eliminarUsuarioNoExisteTest() throws Exception {
    when(usuarioServiceImpl.delete(any(Usuario.class))).thenReturn(Optional.empty());

    mockmvc.perform(delete("/api/usuarios/99.999.999-9")
            .contentType(MediaType.APPLICATION_JSON))
            .andExpect(status().isNotFound());
}
```

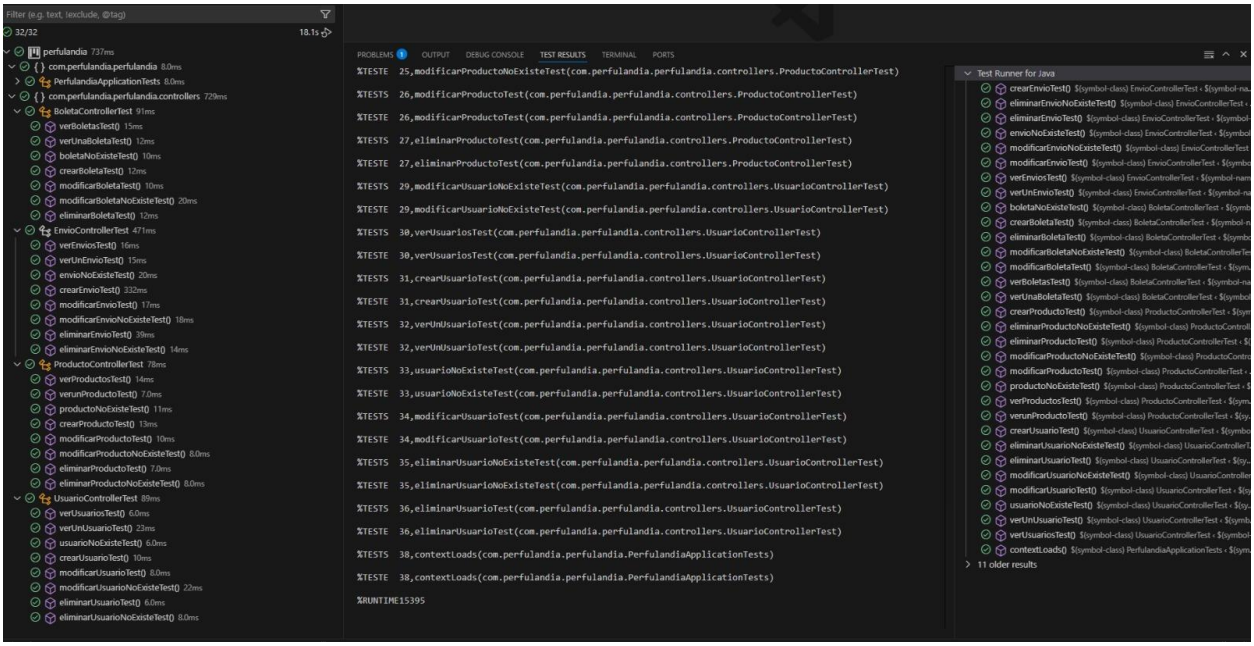
4. Ejecución de Pruebas

4.1 Pruebas Unitarias



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

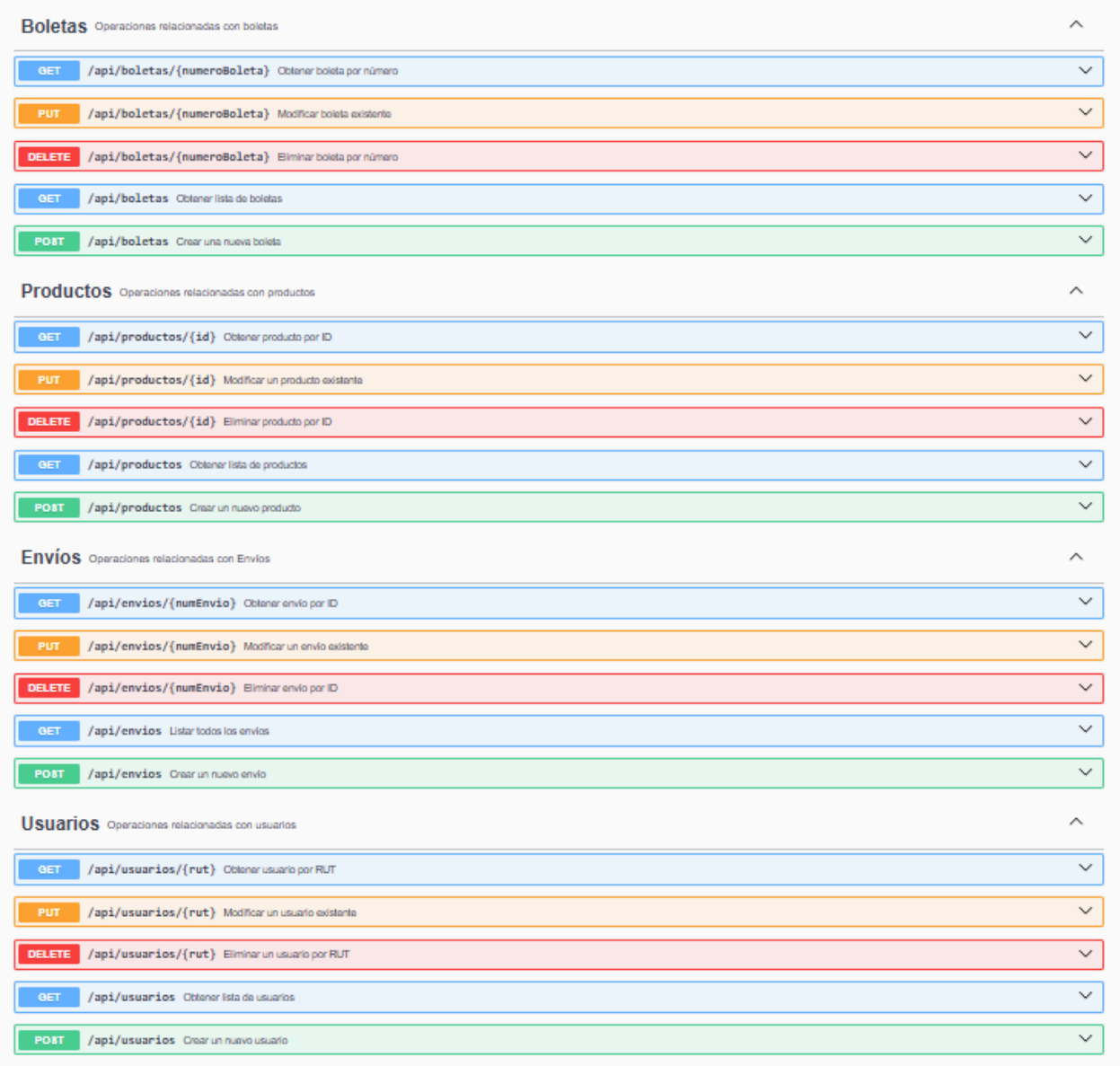
4.2 Pruebas de Integración



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

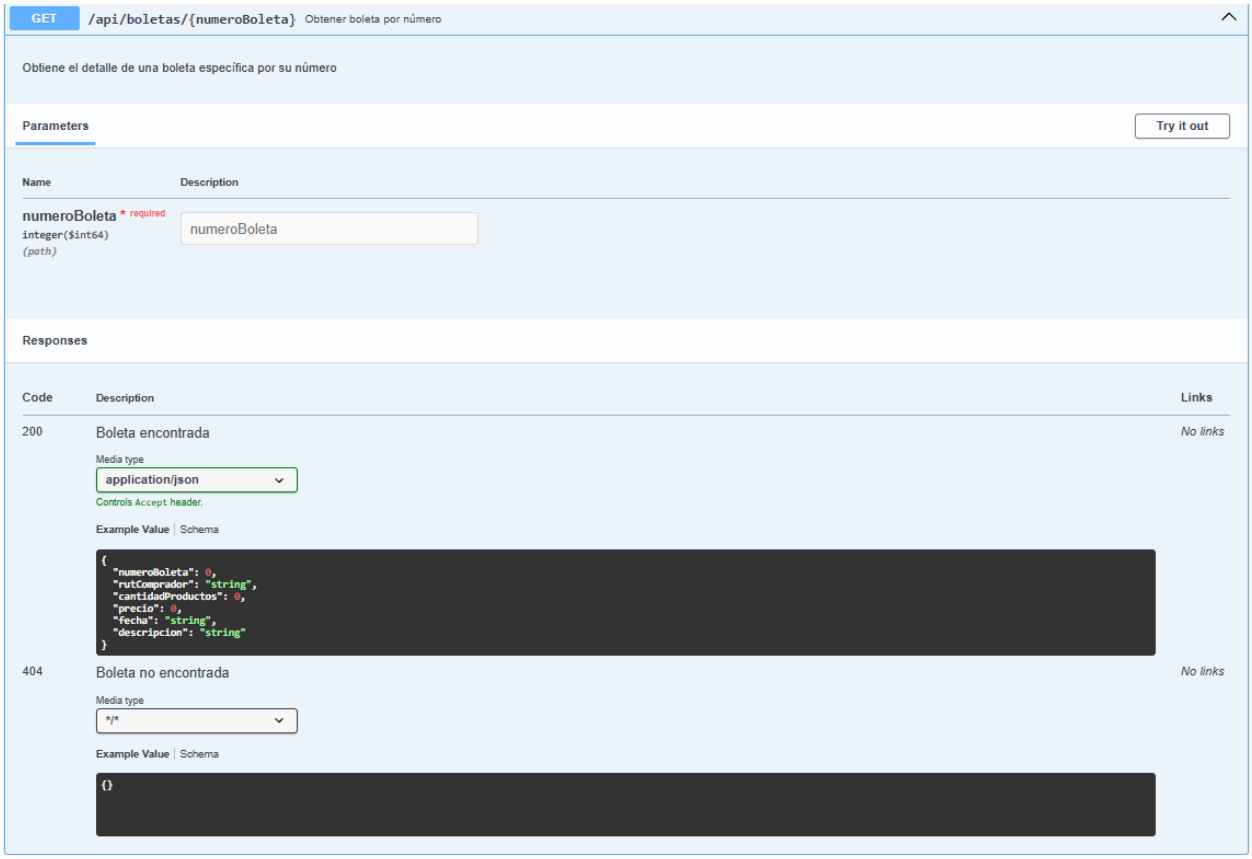
4.3 Documentación con Open Api

Documentación de todos los métodos creados para todos los microservicios

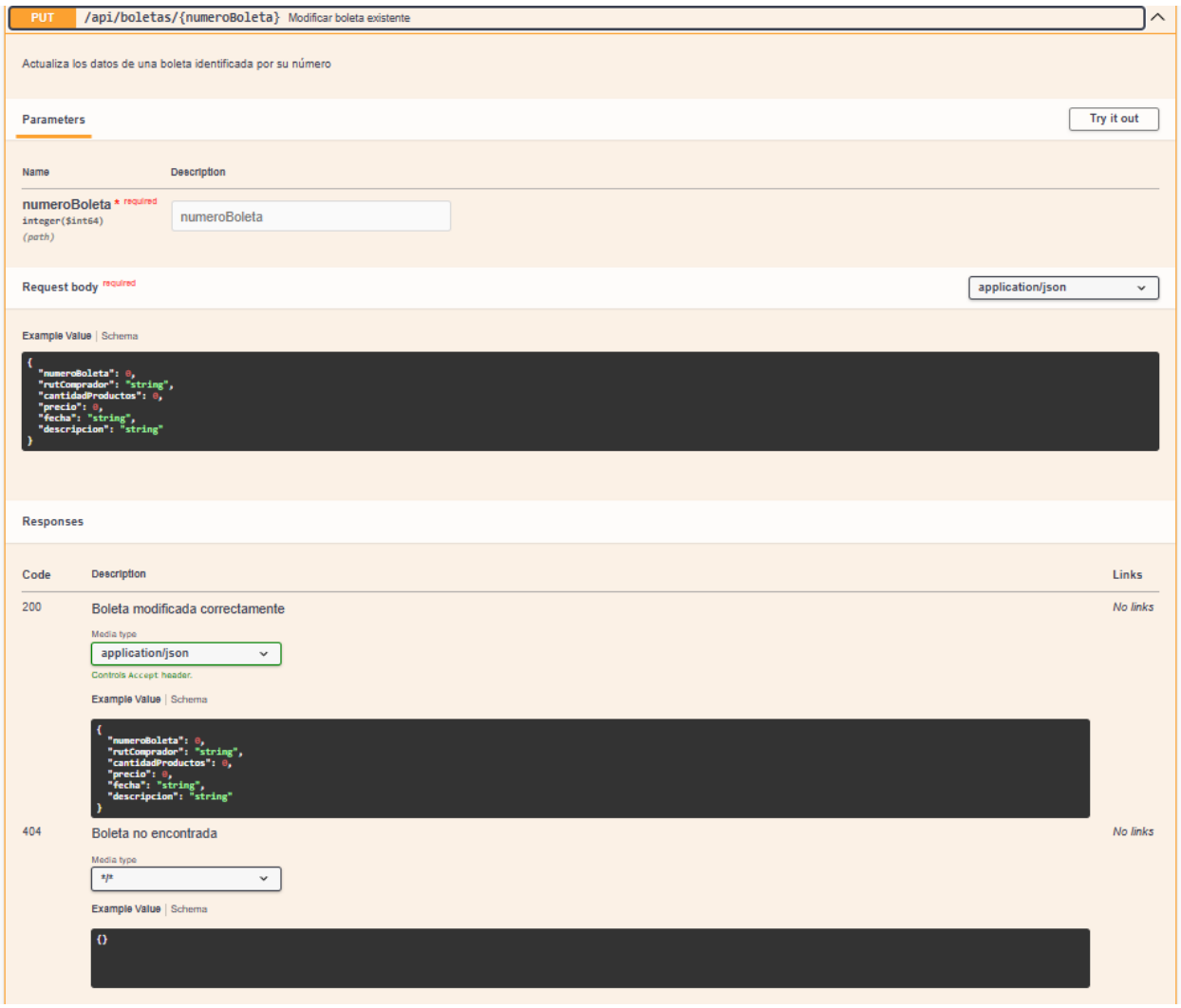


La documentación de métodos es similar para todos los microservicios, por lo que solo se adjuntan imágenes de los métodos del microservicio de boletas.

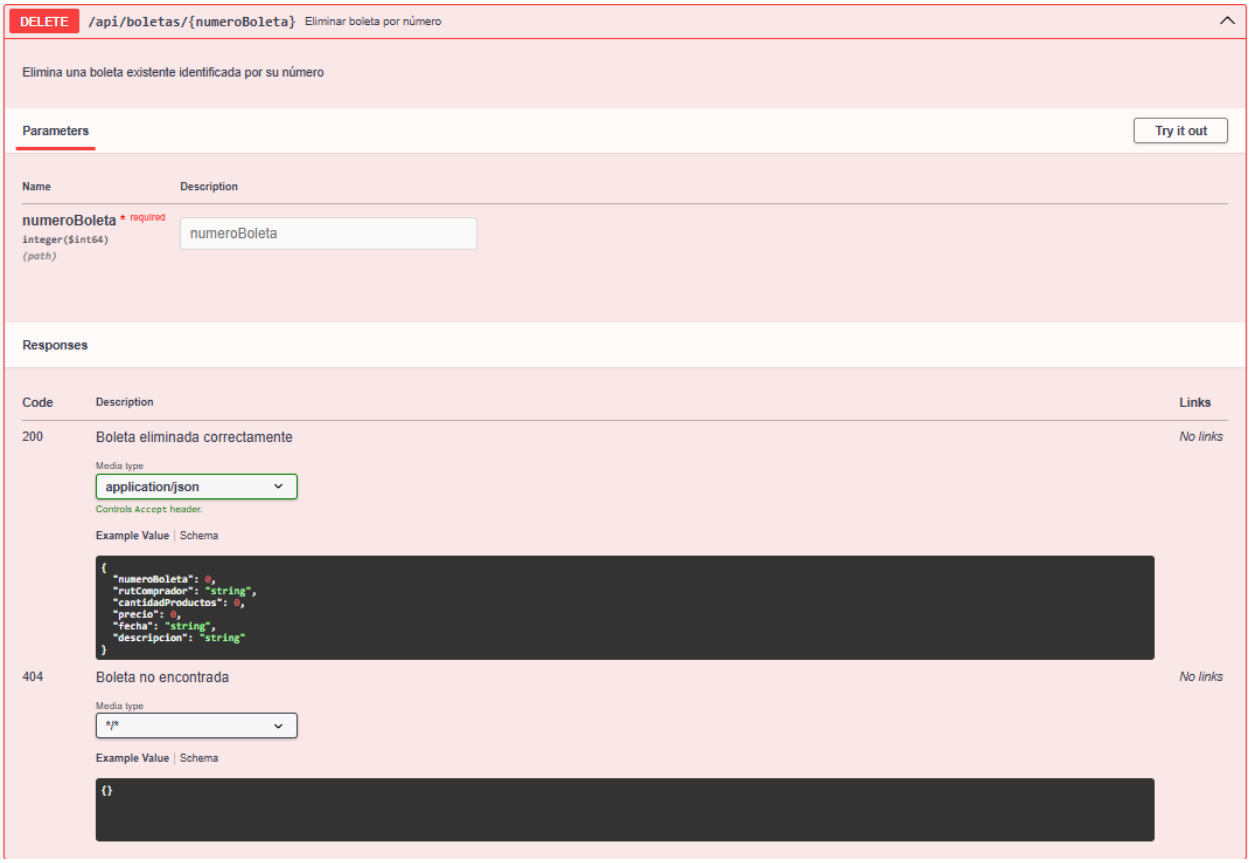
Documentación método verDetalle() (findById)



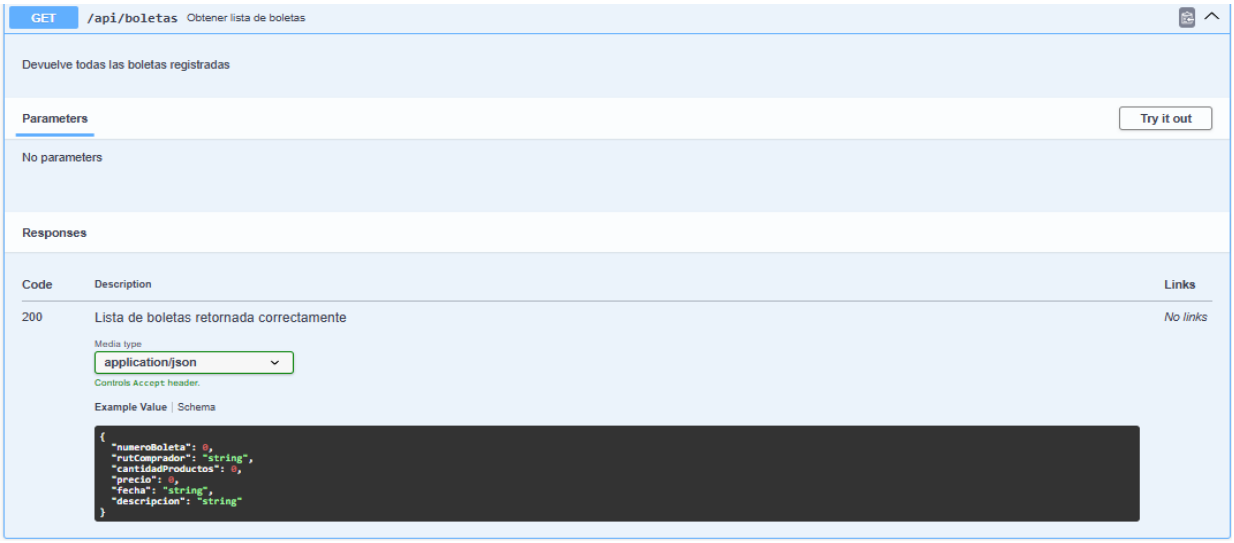
Documentación metodo modificar()



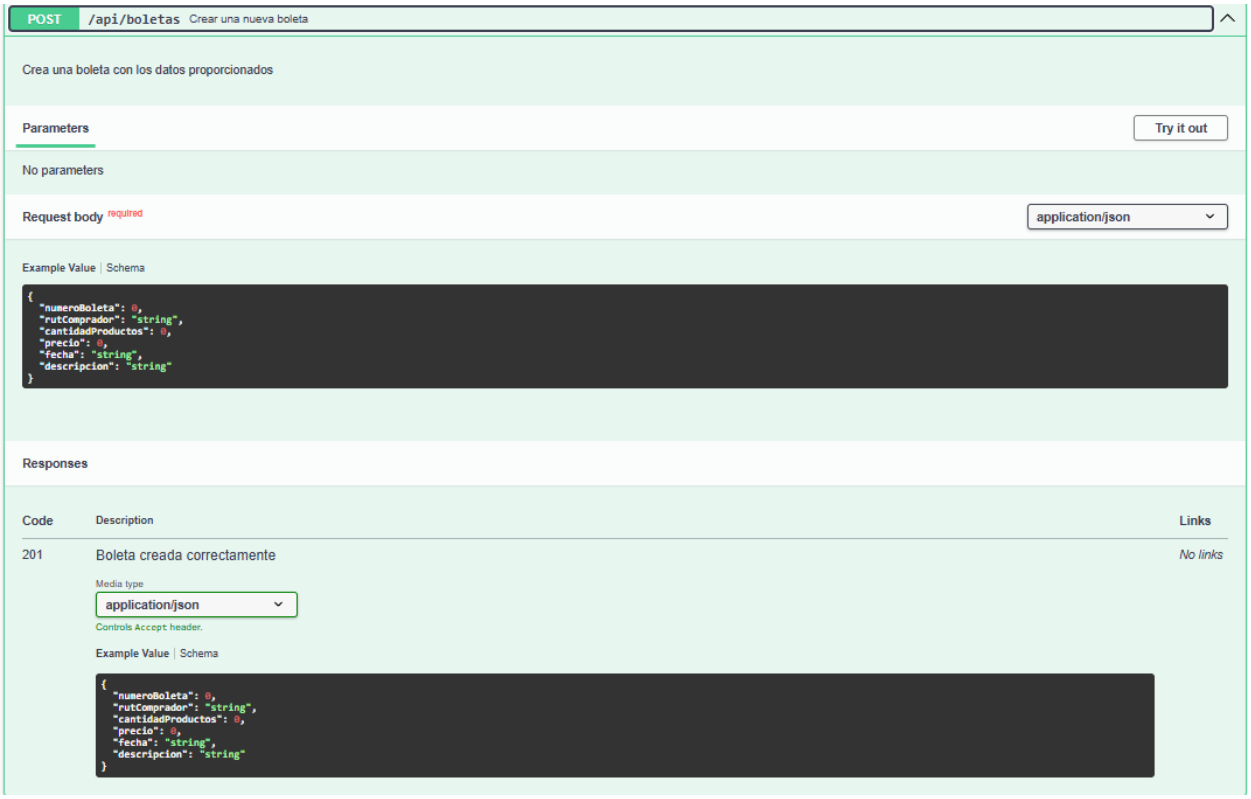
Documentación metodo eliminar()



Documentación método list() (findByAll)



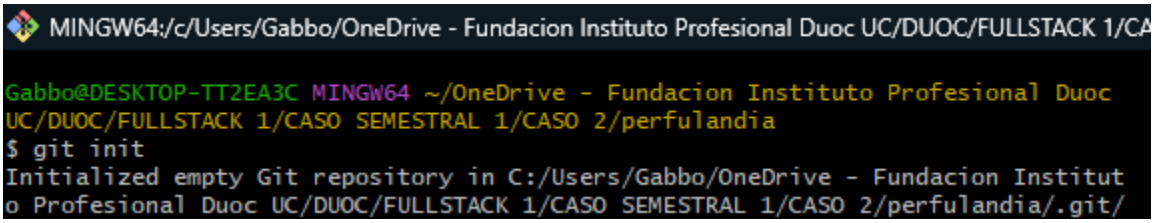
Documentación metodo crear() (save)



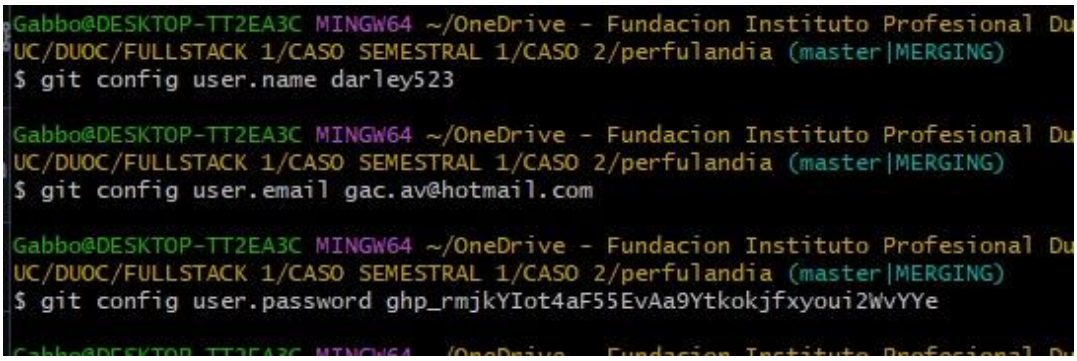
5. Comandos Git-GitHub

5.1 Anteriormente usados

Comando para inicializar un repositorio de git en local.



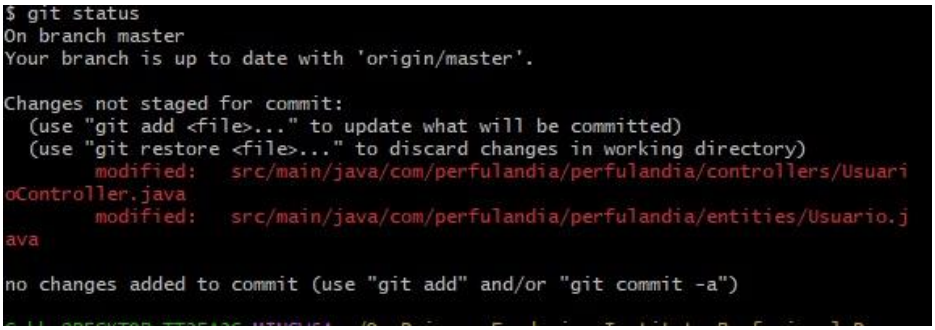
Comando para identificarse al momento de hacer los commit, agregando una contraseña para el repositorio online.



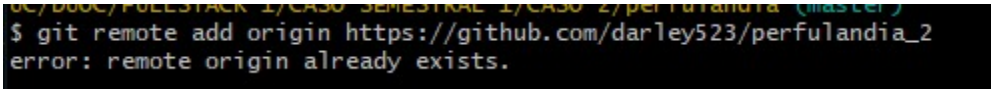
Comando para añadir todos los archivos creados o modificados al repositorio.

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

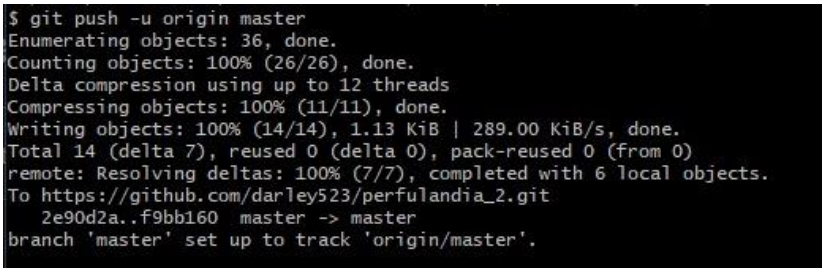
Comando para comprobar que cosas no han sido guardadas en el repositorio.



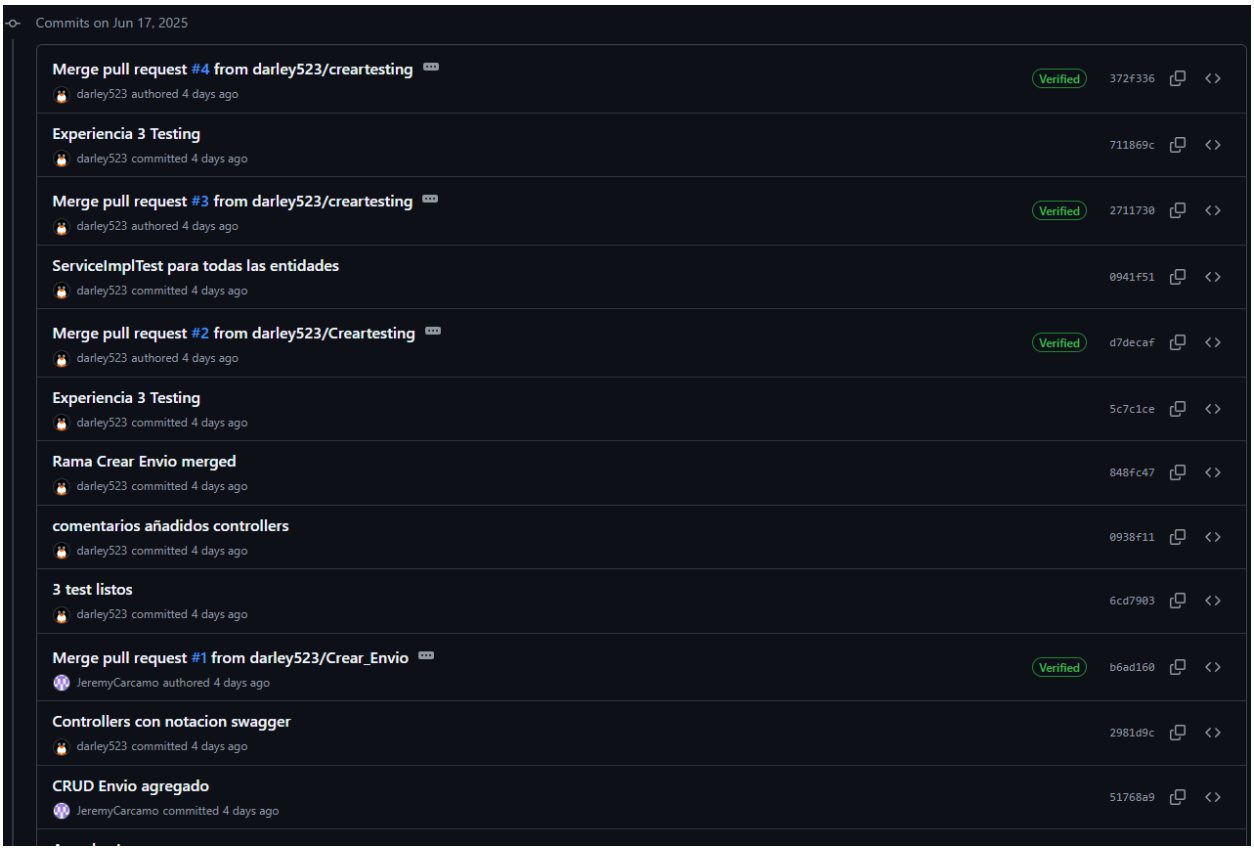
Comando para vincular el repositorio local con uno online.



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

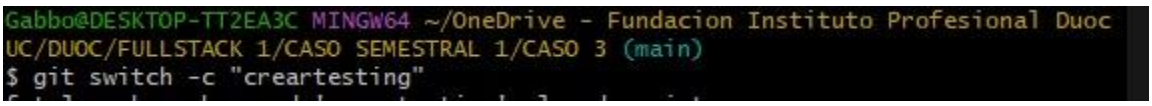


Historial de commits colaborativos durante el desarrollo del proyecto.

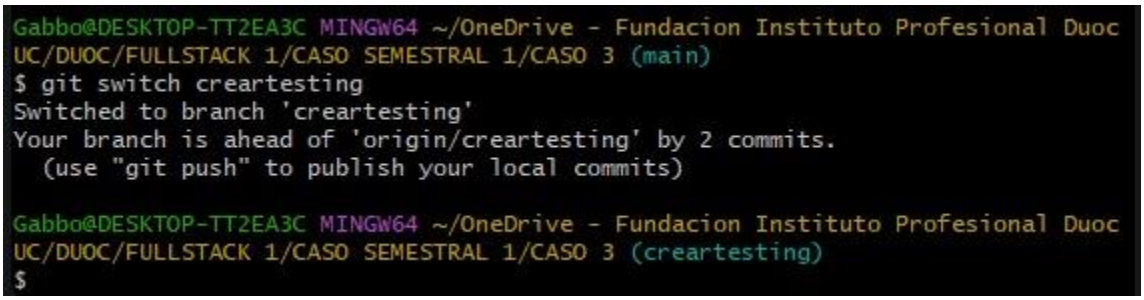


5.2 Comandos nuevos

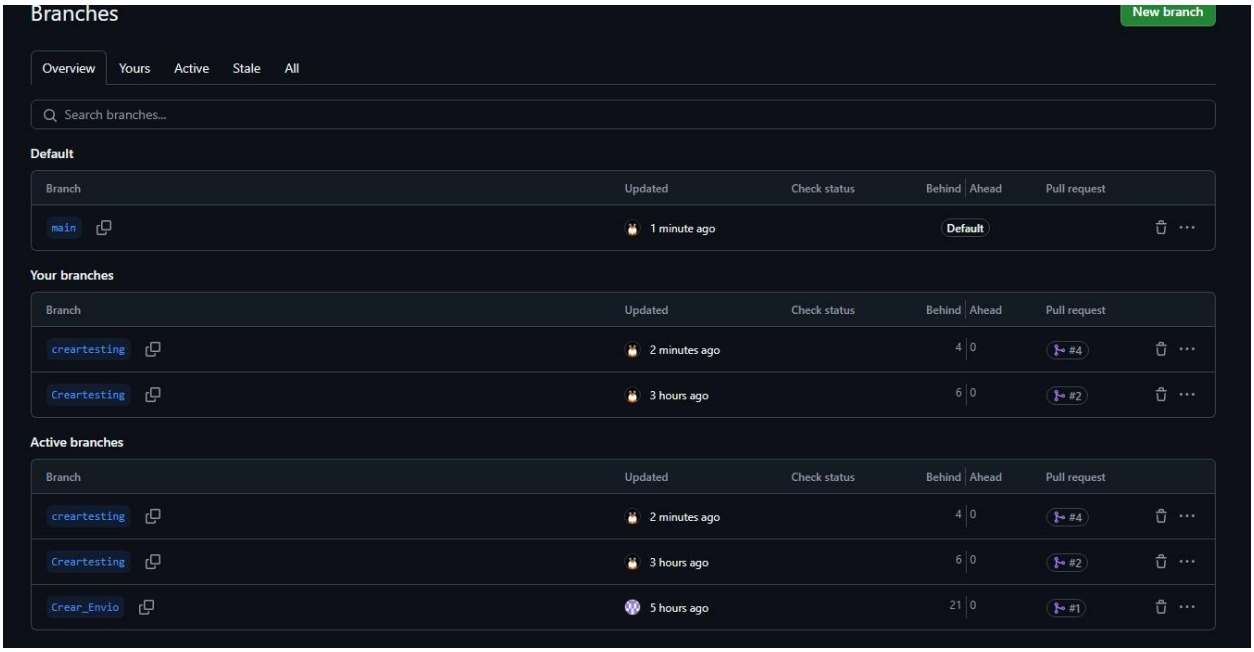
Para crear ramas.



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.