**Documentación del Parcial3**

**1. Introducción**

Este documento detalla el proceso de desarrollo de una API reactiva con Spring WebFlux, incluyendo la configuración de seguridad, internacionalización y pruebas automatizadas. Se documenta cada componente desarrollado y su función dentro del sistema.

**2. Tecnologías Utilizadas**

* **Java 17**
* **Spring Boot** (WebFlux, Security, i18n)
* **Reactor (Flux, Mono)**
* **Mockito y JUnit 5** (Pruebas automatizadas)
* **Maven**

**3. Paso a Paso del Desarrollo**

**3.1. Configuración del Proyecto**

Se creó un proyecto Spring Boot con WebFlux y seguridad deshabilitada para pruebas.

**Archivos relevantes:**

* SecurityConfig.java: Configura la seguridad permitiendo todas las solicitudes sin autenticación.
* LocaleConfig.java: Configura la internacionalización para mensajes en distintos idiomas.

**3.2. Desarrollo de Entidades**

Se crearon las clases DTO con anotaciones Lombok para simplificar la generación de getters, setters y constructores.

**Ejemplo:**

@Data

@AllArgsConstructor

@NoArgsConstructor

public class Pedido {

private String id;

private String producto;

private int cantidad;

private double precioTotal;

private LocalDateTime fecha;

}

**3.3. Implementación de Servicios y Controladores**

**Servicio PedidoService**

* Permite la creación y consulta de pedidos.
* Utiliza listas en memoria para simular una base de datos.

**Controlador PedidoController**

* Expone los endpoints /api/pedidos y /api/pedidos/mensaje.
* Responde en distintos idiomas según el parámetro lang.

**Servicio ProductoService**

* Gestiona una lista de productos de manera reactiva.

**Controlador ProductoController**

* Expone el endpoint /api/productos para listar productos.

**Controlador SaludoController**

* Devuelve un mensaje según el idioma solicitado en /api/saludo?lang=es.

**3.4. Pruebas Automatizadas**

Se implementaron pruebas con **Mockito y StepVerifier** para validar el comportamiento de los servicios de forma reactiva.

**Ejemplo:**

@Test

void testCrearPedido() {

Pedido nuevoPedido = new Pedido("1", "Monitor", 1, 200.0, LocalDateTime.now());

PedidoService pedidoService = new PedidoService();

Mono<Pedido> resultado = pedidoService.crearPedido(nuevoPedido);

StepVerifier.create(resultado)

.expectNextMatches(pedido -> pedido.getProducto().equals("Monitor"))

.verifyComplete();

}

**4. Internacionalización (i18n)**

Los mensajes de la API se almacenan en archivos messages\_xx.properties.

Ejemplo de messages\_es.properties:

mensaje.lista = Lista de pedidos

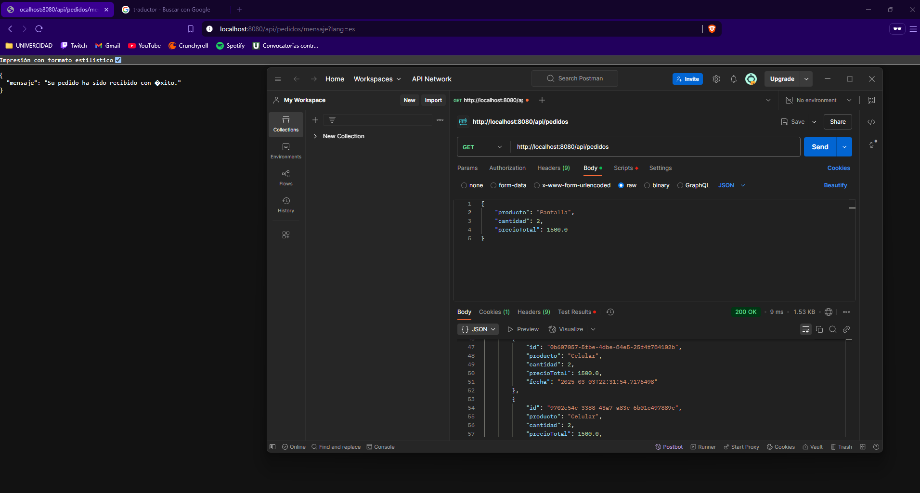
mensaje.creado = Pedido creado exitosamente

mensaje.pedido = Bienvenido a la API de pedidos

**5. Conclusiones**

Se logró desarrollar una API reactiva con Spring WebFlux, permitiendo consultas y creación de pedidos y productos, con soporte para internacionalización y pruebas automatizadas.

**6. Capturas de Pantalla**

**

Captura de pantalla de computadora

El contenido generado por IA puede ser incorrecto.

