**Documentación del Sistema Construido e Integrado (FERREMAS)**

**Nombre del Proyecto: Sistema Web Ferremas**

**Integrantes del grupo:**

* Matías Rain
* Benjamín Celis
* Matías Sáez

**1. Descripción General del Sistema**

El sistema Ferremas fue construido como una API RESTful utilizando el framework **Spring Boot**, con conexión a una base de datos **MySQL** gestionada desde **phpMyAdmin**. El sistema tiene como objetivo simular una ferretería digital capaz de gestionar productos, precios, mensajes de contacto, conversiones de moneda, y pagos simulados mediante WebPay.

**2. Tecnologías Utilizadas**

* Java 21
* Spring Boot 3.4.7
* Maven
* MySQL 8
* phpMyAdmin
* Swagger UI
* Postman
* Jakarta Persistence (JPA)
* Hibernate

**3. Configuración del Proyecto**

**Nombre de la base de datos:**

nginx

CopiarEditar

miprimeraapi\_db

⚠️ Es obligatorio usar este nombre exacto para que la aplicación funcione correctamente. Debe crearse antes de ejecutar la aplicación Spring Boot.

**Archivos importantes del proyecto:**

* application.properties: contiene la configuración de conexión a MySQL.
* Paquetes:
  + model: contiene todas las entidades (Producto, Usuario, MensajeContacto, etc.).
  + controller: define todos los controladores REST.
  + repository: interfaces JPA para acceder a la base de datos.

**4. Estructura de Tablas en MySQL**

Al iniciar la aplicación con una base de datos vacía llamada miprimeraapi\_db, se crearán automáticamente las siguientes tablas gracias a JPA:

* producto
* usuario
* mensaje\_contacto
* valor\_dolar
* transaccion\_dolar
* transaccion\_pago
* historial\_precio

Estas tablas se crean según las entidades Java anotadas con @Entity.

**5. Flujo para Ejecutar el Proyecto**

1. Abrir **phpMyAdmin** y crear la base de datos:

sql

CopiarEditar

CREATE DATABASE miprimeraapi\_db;

1. Asegurarse que MySQL está corriendo en localhost puerto 3306.
2. Verificar credenciales en application.properties:

properties

CopiarEditar

spring.datasource.url=jdbc:mysql://localhost:3306/miprimeraapi\_db

spring.datasource.username=root

spring.datasource.password=TU\_PASSWORD\_AQUI

1. Ejecutar el proyecto en Spring Boot desde el archivo:

CopiarEditar

MiprimeraapiApplication.java

1. Acceder a Swagger UI:

bash

CopiarEditar

http://localhost:8080/swagger-ui/index.html

**6. Explicación Detallada de los Módulos y Código**

**6.1 Módulo Producto**

**Entidad:** Producto.java  
Campos clave: codigoProducto, marca, nombre, precio, stock, categoria, fechaPrecio.  
Validaciones:

* Código único obligatorio.
* Precio y stock deben ser positivos.
* No se permite crear productos sin nombre o sin categoría.

**Controlador:** ProductoController.java  
Rutas implementadas:

* POST /productos: Crear producto.
* GET /productos/{codigo}: Buscar por código.
* GET /productos/buscarNombre?nombre=: Buscar por nombre.
* GET /productos/categoria?categoria=: Buscar por categoría.
* GET /productos/bajoStock?stock=: Buscar productos con stock bajo.

**Repositorio:** ProductoRepository.java  
Extiende JpaRepository, incluye métodos como:

java

CopiarEditar

Producto findByCodigoProducto(String codigoProducto);

List<Producto> findByNombreContainingIgnoreCase(String nombre);

**6.2 Módulo Usuario**

**Entidad:** Usuario.java  
Campos: id, nombre, direccion, rol.  
Validaciones:

* El campo rol solo puede ser "interno" o "externo".

**Controlador:** UsuarioController.java  
Rutas:

* POST /usuarios: Crear usuario.
* GET /usuarios/{id}: Obtener por ID.

**Repositorio:** UsuarioRepository.java

**6.3 Módulo Mensaje de Contacto**

**Entidad:** MensajeContacto.java  
Campos: nombre, email, mensaje, fecha.  
Validaciones:

* No se permiten campos vacíos.
* Email debe tener un formato válido.

**Controlador:** MensajeContactoController.java  
Rutas:

* POST /mensajes: Enviar mensaje.
* GET /mensajes: Obtener todos los mensajes (sin restricción).
* GET /mensajes/interno/{idUsuario}: Solo usuarios con rol interno pueden acceder.

**6.4 Módulo WebPay Simulado**

**Controlador:** TransaccionPagoController.java  
Rutas:

* POST /webpay/pagar: simula un pago.
* GET /webpay/transacciones: lista todos los pagos simulados.

Valida que los datos del pago estén completos. Si falta monto, nombre o método de pago, se devuelve error.

**6.5 Módulo Banco Central (Simulación de Conversión)**

**Controlador:** BancoCentralController.java  
Rutas:

* POST /bancocentral/convertir: convierte montoUSD a CLP.
* GET /bancocentral/transacciones: ver historial de conversiones.

Valida que el monto sea positivo. Si no hay monto, la conversión no se realiza.

**6.6 Historial de Precios**

**Entidad:** HistorialPrecio.java  
Registra la evolución de precios de productos.

**Controlador:** HistorialPrecioController.java  
Rutas:

* POST /historial-precio: agrega historial si el producto existe.
* GET /historial-precio/{codigoProducto}: busca por código.

**7. Pruebas y Validaciones**

Durante el desarrollo, el sistema fue probado con:

* **Swagger UI** para pruebas unitarias.
* **Postman** para pruebas de integración y simulación externa.
* Se detectaron y corrigieron múltiples errores de validación.
* Se reforzaron restricciones para evitar registros inválidos.

**8. Resultados**

* Todos los módulos se encuentran completamente funcionales.
* El sistema interactúa correctamente con la base de datos.
* Se agregaron validaciones necesarias para evitar inconsistencias.
* Todas las pruebas realizadas tuvieron éxito, incluyendo campos obligatorios, validaciones por rol y flujos funcionales.

**9. Conclusión**

El sistema Ferremas ha sido construido e integrado exitosamente. Se han cumplido los requisitos funcionales definidos en la pauta, incluyendo:

* Manejo de productos con validaciones.
* Registro de usuarios internos y externos.
* Restricción de acceso por rol.
* Módulo de contacto, conversión y simulación de pagos.
* Base de datos funcional y bien estructurada.

Gracias a Spring Boot, MySQL y las herramientas de prueba utilizadas, el sistema es estable, seguro y preparado para su despliegue.