“PomPizza”

Aplicación Web para una pizzería con Spring Boot y MongoDB

Objetivos del Proyecto

El objetivo de esta práctica es desarrollar una **aplicación web completa** que incluya:

✅ Una **API REST** en **Spring Boot** con **MongoDB** como base de datos.

✅ Un **sistema de autenticación con JWT**, que permita gestionar permisos de acceso.

✅ Una **interfaz web en HTML, CSS y JavaScript (Fetch API)** para consumir la API.

✅ Implementación de **roles (`CLIENTE` y `ADMIN`)** con acceso restringido a ciertos endpoints.

Parte 1: API REST (Backend con Spring Boot y MongoDB)

🔷 Requisitos de la API

Se debe desarrollar una **API REST** con los siguientes módulos:

1️⃣ Módulo de Usuarios (Autenticación con JWT)

- **Registro de usuarios (`POST /auth/register`)**
- **Inicio de sesión (`POST /auth/login`)**
- La API debe generar un **JWT** al iniciar sesión, que se usará en todas las peticiones protegidas.
- Existen dos tipos de usuarios:
  - **ADMIN**: Puede gestionar pizzas y pedidos.
  - **CLIENTE**: Puede consultar el menú y realizar pedidos.

2️⃣ Módulo de Pizzas

- **Ver todas las pizzas (`GET /api/pizzas`)** → Público.
- **Crear una pizza (`POST /api/pizzas`)** → Solo ADMIN.
- **Editar una pizza (`PUT /api/pizzas/{id}`)** → Solo ADMIN.
- **Eliminar una pizza (`DELETE /api/pizzas/{id}`)** → Solo ADMIN.

3️⃣ Módulo de Pedidos

- **Crear un pedido (`POST /api/pedidos`)** → Solo CLIENTE.
- **Ver mis pedidos (`GET /api/pedidos/misPedidos`)** → Solo CLIENTE.
- **Ver todos los pedidos (`GET /api/pedidos`)** → Solo ADMIN.
- **Actualizar estado del pedido (`PUT /api/pedidos/{id}`)** → Solo ADMIN.

Parte 2: Desarrollo Web (Frontend con HTML, CSS y JavaScript)

🔷 Requisitos de la Interfaz Web

Se deben desarrollar **páginas web** para interactuar con la API.

1️⃣ Login y Registro (`login.html` y `register.html`)

- Formulario para **registrarse** (`POST /auth/register`).
- Formulario para **iniciar sesión** (`POST /auth/login`).
- Guardar el **token JWT** en `localStorage` tras el login.

2️⃣ Catálogo de Pizzas (`pizzas.html`)

- Consultar las pizzas (`GET /api/pizzas`).
- Mostrar en una tabla o tarjetas.
- Si el usuario es **ADMIN**, agregar opciones para **editar y eliminar** pizzas.

3️⃣ Gestión de Pedidos (`pedidos.html`)

- Como **CLIENTE**:
    - Elegir pizzas y enviar un **pedido (`POST /api/pedidos`)**.
    - Consultar **mis pedidos (`GET /api/pedidos/misPedidos`)**.
- Como **ADMIN**:
    - Ver todos los **pedidos (`GET /api/pedidos`)**.
    - Actualizar su estado (`PUT /api/pedidos/{id}`).

4️⃣ Panel de Administración (`admin.html`)

- Permite a los **ADMIN** gestionar pizzas y pedidos.
- Accesible solo si el usuario tiene rol `ADMIN`.

📂 Estructura de la Base de Datos (MongoDB)

MongoDB es una base de datos **NoSQL**, lo que significa que no utiliza tablas relacionales, sino documentos en formato **JSON/BSON** almacenados en colecciones.

El proyecto tendrá **tres colecciones principales**:

- **`usuarios`** → Almacena los datos de los usuarios y sus roles.
- **`pizzas`** → Contiene la información de las pizzas del menú.
- **`pedidos`** → Registra los pedidos realizados por los clientes.

🔷 Colección Usuarios

📌 Propósito:
Almacenar la información de los usuarios, incluyendo credenciales y roles de acceso (`CLIENTE` o `ADMIN`).

📌 Campos:

Campo	Tipo	Descripción
`_id`	`ObjectId`	Identificador único generado automáticamente.
`username`	`String`	Nombre de usuario único.
`password`	`String`	Contraseña encriptada con BCrypt.
`email`	`String`	Correo electrónico del usuario.
`role`	`String`	Rol del usuario (`CLIENTE` o `ADMIN`).

📌 Notas:

- **Las contraseñas deben estar encriptadas** antes de guardarlas en la base de datos.
- El **ADMIN** podrá gestionar el menú y los pedidos.
- Los **CLIENTES** solo pueden realizar pedidos y ver el menú.

🔷 Colección Pizzas

📌 Propósito:

Almacenar la información de las pizzas disponibles en la pizzería.

📌 **Campos:**

Campo	Tipo	Descripción
`_id`	`ObjectId`	Identificador único generado automáticamente.
`nombre`	`String`	Nombre de la pizza.
`descripcion`	`String`	Breve descripción de la pizza.
`ingredientes`	`Array`	Lista de ingredientes de la pizza.
`precio`	`Double`	Precio en euros.
`imagenUrl`	`String`	URL de la imagen de la pizza.
`disponible`	`Boolean`	Indica si la pizza está disponible en el menú.

📌 **Notas:**

- Un **ADMIN** puede agregar, editar y eliminar pizzas.
- Un **CLIENTE** solo puede ver las pizzas disponibles.

🔷 Colección Pedidos

📌 **Propósito:**
Almacenar los pedidos realizados por los clientes.

📌 **Campos:**

Campo	Tipo	Descripción
`_id`	`ObjectId`	Identificador único del pedido.
`cliente`	`String`	Nombre de usuario que hizo el pedido.
`pizzas`	`Array`	Lista de pizzas en el pedido (con nombre y precio).
`total`	`Double`	Precio total del pedido.
`fecha`	`Date`	Fecha y hora en que se realizó el pedido.
`estado`	`String`	Estado del pedido (`Pendiente`, `En preparación`, `Listo`, `Entregado`).

📌 **Notas:**

- Los **CLIENTES** solo pueden ver **sus propios pedidos**.
- Los **ADMIN** pueden ver **todos los pedidos** y cambiar su estado.

📌 Relación entre Colecciones

MongoDB es una base de datos **NoSQL**, por lo que no hay relaciones estrictas como en una base de datos relacional. Sin embargo, la relación entre colecciones se maneja de la siguiente manera:

1️⃣ **Cada `pedido` está vinculado a un `usuario` a través del campo `cliente` (nombre de usuario).**

2️⃣ **Las `pizzas` dentro de un `pedido` se almacenan como subdocumentos** (en lugar de usar `ObjectId`).

3️⃣ **El `role` en `usuarios` define los permisos sobre las colecciones.**