Documentación del proyecto Java (Backend E-Commerce)

Cuando empecé este proyecto, mi idea era montar una aplicación de e-commerce muy básica para aprender cómo se conectan las piezas en un backend real. Lo primero que decidí fue hacerlo con **Java** usando **Spring Boot**, porque es un framework bastante estándar y me permite tener control de todo: seguridad, conexión a base de datos, endpoints REST, etc.

1. Arrancando el proyecto

El proyecto lo monté con **Spring Initializr**. Ahí seleccioné dependencias como:

- Spring Web (para crear la API REST),
- Spring Data JPA (para trabajar con base de datos de forma más sencilla),
- MySQL Driver (porque la base de datos que uso es MySQL),
- Spring Security (para la parte de login y control de acceso).

Una vez generado, me descargué el proyecto y lo abrí en IntelliJ (aunque también se podría en Eclipse o VS Code).

2. Configuración de la base de datos

En el archivo `application.properties` configuré la conexión con la base de datos MySQL que tengo en local. Ahí puse la URL, usuario y contraseña. También añadí configuraciones para que Hibernate cree las tablas automáticamente a partir de mis entidades.

```
spring.datasource.url=jdbc:mysql://localhost:3306/basic_ecommerce
spring.datasource.username=basic_user
spring.datasource.password=password123

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

3. Creación de las entidades

Definí las entidades principales que iba a tener la aplicación:

- **Usuario** → para registrar clientes, con datos como nombre, email, contraseña, rol (para distinguir si es admin o usuario normal).
- **Producto** → para gestionar el catálogo de productos (nombre, descripción, precio, stock).
- **Pedido** → para representar las compras que hacen los usuarios.

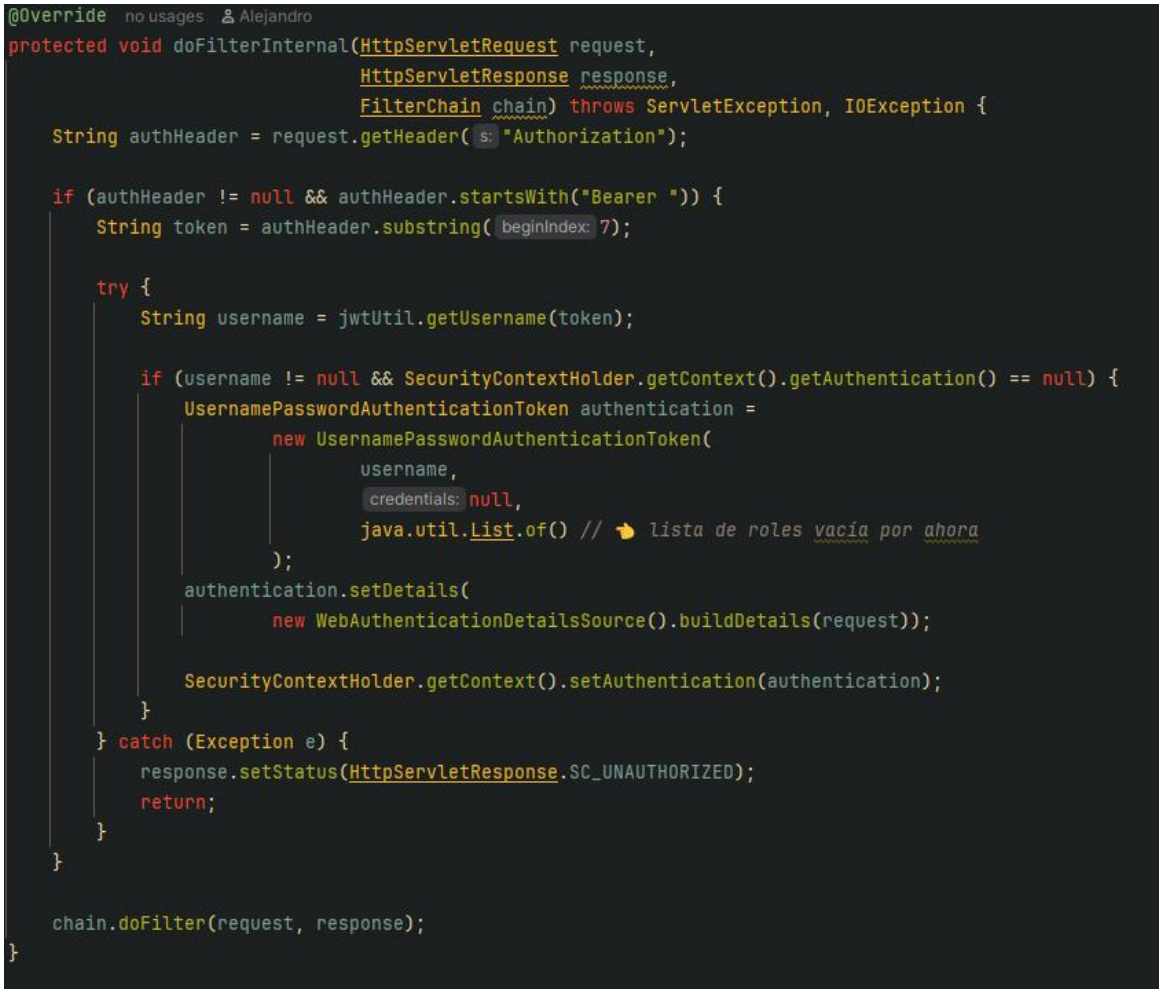
Cada entidad está mapeada como una clase Java con anotaciones de JPA, y tienen relaciones entre sí (por ejemplo, un pedido pertenece a un usuario, y un pedido tiene productos).

```
@Getter
@Setter
@OneToMany(mappedBy = "pedido", cascade = CascadeType.ALL, orphanRemoval = true)
private List<PedidoItem> items;
```

4. Seguridad y autenticación

Aquí fue donde más tiempo invertí. Configuré **Spring Security** para tener:

- **Registro (register)**: un endpoint público donde un usuario puede crear su cuenta. Al registrarse, el sistema guarda la contraseña encriptada en la base de datos y le devuelve un **token JWT**.
- **Login**: un endpoint que valida el usuario y contraseña y genera un JWT válido.
- **Protección de rutas**: hay endpoints que solo se pueden usar si mandas el token en el header `Authorization: Bearer <token>`. Por ejemplo, para borrar productos o ver pedidos necesitas estar logueado y tener los permisos correctos.



```java
@Override  no usages  Alejandro
protected void doFilterInternal(HttpServletRequest request,
                                HttpServletResponse response,
                                FilterChain chain) throws ServletException, IOException {
    String authHeader = request.getHeader( s: "Authorization");

    if (authHeader != null && authHeader.startsWith("Bearer ")) {
        String token = authHeader.substring( beginIndex: 7);

        try {
            String username = jwtUtil.getUsername(token);

            if (username != null && SecurityContextHolder.getContext().getAuthentication() == null) {
                UsernamePasswordAuthenticationToken authentication =
                        new UsernamePasswordAuthenticationToken(
                                username,
                                credentials: null,
                                java.util.List.of() // 👈 lista de roles vacía por ahora
                        );
                authentication.setDetails(
                        new WebAuthenticationDetailsSource().buildDetails(request));

                SecurityContextHolder.getContext().setAuthentication(authentication);
            }
        } catch (Exception e) {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            return;
        }
    }

    chain.doFilter(request, response);
}
```

5. Creación de los endpoints

Fui creando controladores (`@RestController`) para manejar las peticiones. De momento tengo estos grupos:

- **AuthController**
  - `POST /auth/register` → registro de usuario
  - `POST /auth/login` → login y obtención de token
- **ProductoController**
  - `GET /productos` → listar productos
  - `GET /productos/{id}` → ver un producto concreto
  - `POST /productos` → crear producto (solo admin)
  - `PUT /productos/{id}` → actualizar producto (solo admin)
  - `DELETE /productos/{id}` → eliminar producto (solo admin)
- **PedidoController**
  - `POST /pedidos` → crear un pedido como usuario autenticado
  - `GET /pedidos` → ver mis pedidos

- `GET /pedidos/{id}` → ver un pedido concreto (solo si eres el dueño o admin)

```java
@RestController  no usages  Alejandro
@RequestMapping("/api/usuarios")
public class UsuarioController {
    private final UsuarioRepository repo;  3 usages

    public UsuarioController(UsuarioRepository repo) {  no usages  Alejandro
        this.repo = repo;
    }

    @GetMapping  no usages  Alejandro
    public List<Usuario> listar() {
        return repo.findAll();
    }

    @GetMapping("/{id}")  no usages  Alejandro
    public Usuario obtener(@PathVariable Long id) {
        return repo.findById(id).orElseThrow(() -> new RuntimeException("Usuario no encontrado"));
    }
}
```

6. Cómo probar la API

Ahora mismo no hay frontend. Eso significa que la aplicación solo tiene backend. Para probarla utilizo **Postman**, donde puedo enviar las peticiones y ver las respuestas en formato JSON.
El flujo típico es este:

1. Hago un `POST /auth/register` con los datos de un usuario.
2. El backend me devuelve un token JWT.
3. Copio ese token y lo pongo en Postman, en el apartado de Headers (`Authorization: Bearer <token>`).
4. Con eso ya puedo probar endpoints protegidos como crear pedidos, ver productos, borrar productos, etc.

Cuando un endpoint me devuelve *Forbidden*, normalmente es porque no he puesto bien el token o porque el usuario no tiene permisos para esa acción (por ejemplo, borrar productos siendo un usuario normal).

7. Próximos pasos

De momento este proyecto no tiene frontend. La idea es que más adelante voy a hacer un cliente en **Flutter**, que se conecte a esta API para mostrar los productos, permitir loguearse, hacer pedidos, etc.

Por ahora, todo lo que puedo hacer es desde Postman, pero la API ya está lista para usarse.

Conclusión

- Crear una API REST en Java con Spring Boot.
- Manejar una base de datos con JPA/Hibernate.
- Implementar autenticación y autorización con JWT.