📘 Proyecto Biblioteca — Documentación General

Esta aplicación es un sistema completo de gestión de biblioteca desarrollado siguiendo la **Arquitectura Hexagonal** e implementado con Spring Boot, Thymeleaf, JPA, JWT, Docker y MySQL. Aquí se explica cómo está organizado el proyecto, las tecnologías utilizadas, la API REST, su seguridad, y cómo interactúan todos los módulos.

🧩 Tecnologías Utilizadas

- **Spring Boot 3** — Motor principal del backend.
- **Spring Web** — Controladores REST.
- **Spring Security 6** — Protección de rutas y autenticación.
- **JJWT** — Generación y validación de tokens JWT.
- **Spring Data JPA** — Acceso a base de datos y persistencia.
- **MySQL 8** — Base de datos relacional.
- **Docker** — Contenedor para MySQL.
- **Thymeleaf** — Vistas HTML para la web pública y privada.
- **MapStruct** — Conversión entre entidades, DTOs y modelos de dominio.
- **SpringDoc / Swagger UI** — Documentación interactiva de la API.

🏛 Arquitectura Hexagonal

El proyecto adopta la Arquitectura Hexagonal (Ports & Adapters). Esto permite una separación clara entre:

- **Dominio**: reglas de negocio puras.
- **Aplicación (Use Cases)**: define qué se puede hacer.
- **Infraestructura**: controladores REST, JPA, seguridad, DTOs, mappers…

📁 Estructura real del proyecto

```
src/main/java/onion/whnazv/biblioteca/
    ├── application
    │     ├── port (in/out)
    │     └── usecase (lógica de negocio)
    ├── domain
    │     ├── model (Book, User, Sale…)
    │     └── exception
    ├── infrastructure
    │     ├── adapter
    │     │     ├── in (REST y Web MVC)
    │     │     └── out (Persistencia JPA)
    │     ├── mapper (DTO y Entity)
```

```
    │      └── security (JWT + filtros)
    └── BibliotecaApplication.java
```

🧠 Principio fundamental

El dominio NO depende de infraestructura. Esto significa que la lógica de negocio puede sobrevivir incluso si cambias la base de datos, los controladores REST o la interfaz web.

🐬 Base de Datos y Docker

Docker Compose utilizado:

```
services:
  mysql:
    image: mysql:8.0
    container_name: mysql_proyecto
    environment:
      MYSQL_DATABASE: proyecto_11
      MYSQL_USER: nombre_1
      MYSQL_PASSWORD: contra_1
      MYSQL_ROOT_PASSWORD: contra_1
    ports:
      - "3306:3306"
    volumes:
      - ./mysql_data:/var/lib/mysql
```

La aplicación se conecta mediante:

```
spring.datasource.url=jdbc:mysql://localhost:3306/proyecto_11
spring.datasource.username=nombre_1
spring.datasource.password=contra_1
```

El esquema se genera automáticamente gracias a:

```
spring.jpa.hibernate.ddl-auto=update
```

🔐 Seguridad y Autenticación (JWT)

La autenticación se realiza mediante JSON Web Tokens. El flujo es el siguiente:

1. El usuario envía sus credenciales a **/api/v1/public/auth/login**.
2. Spring Security verifica email + contraseña.
3. Se genera un JWT firmado.
4. El usuario lo envía en cada petición privada:

   ```
   Authorization: Bearer <token>
   ```

5. Un filtro (JwtAuthenticationFilter) valida el token.

Ejemplo de login:

```
POST /api/v1/public/auth/login
Content-Type: application/json

{
  "email": "admin@example.com",
  "password": "1234"
```

```
}
```

Respuesta (token JWT):

```
{
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI..."
}
```

🌍 API REST

La API sigue el estilo REST y devuelve siempre respuestas JSON. Se divide en endpoints públicos, de cliente y de administrador.

🌐 Endpoints Públicos

- **POST /api/v1/public/auth/login** — Devuelve token JWT.

🛡️ Endpoints Administrador (requieren ROLE_ADMIN)

📄 Ventas

- **GET /api/v1/private/admin/sales** — Lista todas las ventas.
- **GET /api/v1/private/admin/sales/{id}** — Detalles de una venta.
- **POST /api/v1/private/admin/sales?userId=ID** — Crea venta.
- **PUT /api/v1/private/admin/sales/{id}?userId=ID** — Actualiza venta.
- **DELETE /api/v1/private/admin/sales/{id}** — Elimina venta.

📚 Libros

- **GET /api/books**
- **GET /api/books/{id}**
- **POST /api/books**
- **PUT /api/books/{id}**
- **DELETE /api/books/{id}**

👤 Endpoints Cliente (requieren ROLE_CLIENT)

(Si implementados en tu proyecto)

- **GET /api/v1/private/client/sales** — Ventas propias.
- **GET /api/v1/private/client/sales/{id}**

📦 Ejemplo de respuesta JSON

```
{
  "id": 1,
  "title": "El Quijote",
  "author": "Miguel de Cervantes",
  "price": 19.99,
  "stock": 12
}
```

📑 Swagger UI

Puedes explorar y probar la API de forma interactiva aquí:

👉 Abrir documentación Swagger

🔄 Flujo completo de una operación

1. El cliente envía una petición REST al controlador.
2. El controlador (adapter IN) convierte DTO → dominio.
3. El caso de uso ejecuta la lógica de negocio.
4. El puerto OUT llama al adapter de persistencia.
5. El adapter usa JPA para guardar en MySQL.
6. Se devuelve la respuesta en formato JSON.

📝 Notas finales

Esta documentación resume la estructura interna, API y funcionamiento del proyecto. Su arquitectura modular permite escalar, añadir nuevas funcionalidades y modificar la base de datos o la interfaz sin afectar al núcleo de negocio.

© 2025 Mi Biblioteca. Todos los derechos reservados.