DESARROLLO DE APLICACIONES OPEN SOURCE (SI729)

Ejercicio 2
2024-1

Caso LibraryManagement

LibraryManagement Inc. (https://www.librarymanagement.com) desea desarrollar una plataforma para gestionar los libros y los usuarios de una biblioteca moderna. El objetivo es proporcionar una solución que permita a los administradores de la biblioteca controlar y monitorear el inventario de libros y a los usuarios gestionar sus préstamos de manera eficiente utilizando un backend robusto y una aplicación web intuitiva.

Objetivo:

El equipo de desarrollo debe crear un RESTful API que soporte las operaciones de LibraryManagement, incluyendo la gestión de libros y usuarios, así como la autenticación y autorización mediante IAM (Identity and Access Management) con roles específicos y la configuración de Swagger con JWT para autenticación.

Código Base:

Se proporciona un código base que contiene los bounded contexts **Shared** e **IAM** avanzados. **Shared** contiene campos de auditoría created_at y updated_at. **IAM** incluye funcionalidades de registro, inicio de sesión y gestión de roles. El estudiante debe implementar el bounded context **Inventory** y modificar el IAM de acuerdo con lo solicitado**.**

REQUISITOS:

1. **Gestión de Libros (Books):**
    - Cada libro (Book) debe tener los siguientes atributos:
        - id (Long Primary Key Autogenerado)
        - isbn (String Obligatorio Único máximo 13 caracteres)
        - title (String Obligatorio máximo 100 caracteres)
        - author (String Obligatorio máximo 50 caracteres)
        - publishedDate (Date Obligatorio) - Debe ser menor o igual a la fecha actual del sistema
        - status (String Obligatorio) - Posibles valores: AVAILABLE, BORROWED máximo 10 caracteres
        - genre (Genre Obligatorio) - Debe estar asociado a la tabla pre-poblada genres
        - created_at (Timestamp) - Autogenerado por Spring Boot
        - updated_at (Timestamp) - Autogenerado por Spring Boot

- **Reglas de negocio:**
  - isbn debe ser único.
  - status debe ser uno de los valores válidos.
  - publishedDate no puede ser una fecha futura.

2. **Tabla de Géneros (Genres):**
   - Se debe tener una tabla pre-poblada genres que contenga los posibles valores de Genre. Esta tabla debe ser verificada y poblada al inicio de la aplicación utilizando un evento dentro del bounded context **Inventory**.
   - **Tabla Genres:**
     - id (Long Primary Key Autogenerado)
     - genre (String Obligatorio Único máximo 20 caracteres)
   - **Los valores iniciales deben ser:**
     - 1 "FICTION"
     - 2 "NON-FICTION"
     - 3 "SCIENCE"
     - 4 "FANTASY"
     - 5 "MYSTERY"
3. **Gestión de Usuarios (Users):**
   - Implementar las funcionalidades de gestión de usuarios (registro e inicio de sesión) en el contexto IAM.
   - Utilizar roles ADMIN, LIBRARIAN y MEMBER para la autorización de usuarios.
4. **Población de Tabla Inicial:**
   - Se debe tener una tabla pre-poblada user_roles que contenga los posibles valores de UserRole. Esta tabla debe ser verificada y poblada al inicio de la aplicación utilizando un evento de ApplicationReady.

Endpoints:

1. **Books Endpoint:**
   - Agregar un Book (POST): /api/v1/books
     - Al agregar un nuevo libro se debe retornar el status HTTP 201 (Created) y el objeto creado incluyendo su id generado.
   - Actualizar un Book (PUT): /api/v1/books/{id}
     - Debe permitir actualizar un libro existente. Se debe retornar el status HTTP 200 (OK) si la actualización es exitosa y el objeto actualizado. Si el id no existe retornar el status HTTP 404 (Not Found).
   - Listar todos los Books (GET): /api/v1/books
     - Debe permitir obtener una lista de todos los libros. Se debe retornar el status HTTP 200 (OK) y la lista de libros.
2. **IAM Endpoint:**
   - Registro de usuario (POST): /api/v1/auth/signup
     - Al registrarse, se debe retornar el status HTTP 201 (Created) y el objeto creado incluyendo su id generado.
   - Inicio de sesión (POST): /api/v1/auth/signin
     - Al iniciar sesión, se debe retornar un token JWT si las credenciales son correctas.

Bounded Context:

1. **Inventory:** Gestiona los libros (Book).
2. **IAM (Identity and Access Management):** Gestiona los usuarios (User), roles y autenticación.
3. **Shared:** Contiene campos de auditoría created_at y updated_at.

Configuración de Seguridad:

1. **Roles y Permisos:**
   - Los usuarios con rol ADMIN deben tener permisos para agregar, actualizar y listar libros, y gestionar usuarios.
   - Los usuarios con rol LIBRARIAN deben tener permisos para agregar, actualizar y listar libros.
   - Los usuarios con rol MEMBER solo deben tener permisos para listar libros.
2. **Autenticación y Autorización:**
   - Utilice JWT para la autenticación.
   - Configure Swagger para que requiera JWT para acceder a los endpoints protegidos.
   - Utilice el siguiente *secret* para JWT: *8Zz5tw0Ionm3XPZZfN0NOml3z9FMfmpgXwovR9fp6ryDIoGRM8EPHAB6iHsc0fb*.
   - La expiración del token JWT debe ser de 3 días.

Technical Constraints:

- Elabore la solución con Java 22 y Spring Boot Framework 3.
- La información debe ser persistente en una base de datos relacional (MySQL) en un esquema library.
- Los packages deben tener como nombre raíz com.library.platform.upc.
- Incluya documentación de los Endpoints con OpenAPI y asegúrese de que Swagger esté configurado para utilizar JWT.
- Gestione las excepciones en la aplicación.

Consideraciones adicionales:

- La tabla user_roles y la tabla genres deben ser verificadas y pobladas al inicio de la aplicación usando un evento ApplicationReady dentro de sus respectivos bounded contexts.
- Utilice minúsculas para los nombres de URL y términos compuestos separados por guión medio (-) para todos los endpoints.
- Utilice la biblioteca Lombok para el manejo de métodos constructores y de acceso en las clases POJO.
- Utilice records en vez de clases para almacenamiento de valores inmutables.
- Para Book, incluya atributos de auditoría createdAt y updatedAt con valores poblados de forma automática por Spring Boot al momento de la creación.
- Utilice el patrón Assembler para el Object Mapping en la sección transform en la interface layer.

- Documente su código con JavaDoc colocando información de propósito para principales objetos de programación, así como propósito, parámetros y valor retornado en clases y métodos relevantes. Incluya como parte de la documentación sus nombres y apellidos como valor para @author.
- Aplique buenas prácticas de Arquitectura de Software, enfoque de Domain-Driven Design, separación en bounded contexts, layered architecture (domain, application, interfaces, infrastructure), patrones de strategic y tactical Domain-Driven Design, patrón CQRS, principios y patrones de diseño de software orientado a objetos, convenciones de nomenclatura en inglés, así como buenas prácticas de nomenclatura en Java (entre ellas Upper-Camel-Case para Clases, Lower-Camel-Case para atributos y métodos) y buenas prácticas para nomenclatura de objetos de Base de Datos (entre ellas snake case, tablas en plural, sin mnemónicos).

No incluido en el alcance:

- Soporte de CORS.
- Testing.