Documento de Requisitos del Proyecto URL Manager

Índice

1. Introducción

Este documento describe los requisitos de información, funcionales y no funcionales del proyecto URL Manager. El objetivo del proyecto es desarrollar una API utilizando el framework Spring Boot para gestionar URLs de proyectos web, con diferentes niveles de acceso y funcionalidades según el rol del usuario.

2. Requisitos de Información

2.1 Entidades principales

1. Actor: Representa a un usuario del sistema. Puede ser de dos tipos: Admin y Cliente.

- Atributos:
  - username: Nombre de usuario.
  - password: Contraseña.
  - email: Correo electrónico.
  - rol: Rol del usuario (ADMIN o CLIENTE).

2. Admin: Usuario con privilegios administrativos.

- Hereda de Actor.

3. Cliente: Usuario con privilegios para gestionar sus entornos y URLs.

- Hereda de actor
- Atributos adicionales:
  - entornos: Conjunto de entornos creados por el cliente.

4. Entorno: Representa un entorno de trabajo dentro de un proyecto.

- Atributos:
  - name: Nombre del entorno.

- cliente: Cliente al que pertenece el entorno.
- urls: Conjunto de URLs asociadas al entorno.

5. SolicitudEntorno: Es la solicitud que realiza un cliente para crear un entorno en el que guardar sus urls

- Atributos:
  - estado: Estado en el que se encuentra la solicitud, puede ser "PENDIENTE", "ACEPTADO" o "RECHAZADO"
  - nombreEntorno: Nombre que recibirá el entorno creado si se acepta la solicitud
  - cliente: Cliente al que pertenece la solicitud

6. Url: Representa una URL dentro de un entorno.

- Atributos:
  - url: Dirección de la URL.
  - descripcion: Descripción de la URL.
  - fechaCreacion: Fecha de creación de la URL.
  - entorno: Entorno al que pertenece la URL.

3. Requisitos Funcionales

3.1 Sistema de Usuarios y Roles

Roles de usuario:

- Admin: Tiene mayores privilegios, puede gestionar otros administradores, clientes y las solicitudes de creación de entornos.
- Cliente: Puede solicitar crear un entorno y crear URLs dentro de sus entornos.

3.2 Autenticación y Funcionalidades Básicas

Autenticación:

- Implementación de un sistema de autenticación basado en tokens JWT.
- Endpoints de Autenticación:

Registro de usuarios:

- POST /cliente: Registro de clientes - Sin autorización
- POST /admin: Registro de administradores - Admin

Inicio de sesión (login):

- POST /login: Autenticación de usuarios. - Sin autorización

Edición de datos personales:

- PUT /cliente: Actualización de datos de clientes - Cliente
- PUT /admin: Actualización de datos de administradores - Admin

Eliminación de usuarios:

- DELETE /cliente: Eliminación de cliente logueado - Cliente
- DELETE /cliente/{id}: Eliminación de cliente por ID - Admin
- DELETE /admin: Eliminación de administradores - Admin

3.3 Gestión de Entidades

Entornos:

- Actualizar entorno: PUT /entorno - Cliente
- Eliminar entorno: DELETE /entorno/{id} - Cliente
- Obtener todos los entornos: GET /entorno - Admin
- Obtener entornos de un cliente: GET /entorno/{id} - Cliente, Admin
- Obtener entorno por ID: GET /entorno/{id} - Admin, Cliente

URLs:

- Añadir URL a entorno: POST /entorno/{codEnt}/anadirUrl - Cliente
- Actualizar URL: POST /entorno/actualizarUrl/{codEnt} - Cliente
- Eliminar URL: GET /entorno/eliminarUrl/{idUrl} - Cliente
- Obtener las URLs de un entorno: GET /entorno/{entornoCod}/urls - Cliente

Solicitud de entornos:

- Aceptar solicitud: GET /solicitudEntorno/aceptar/{id} - Admin

- Rechazar solicitud: GET /solicitudEntorno/rechazar/{id} - Admin
- Obtener solicitud por ID: GET /solicitudEntorno/{id} - Admin, Cliente
- Obtener todas las solicitudes: GET /solicitudEntorno - Admin
- Realizar una solicitud: POST /solicitudEntorno - Cliente

3.4 Lógica dependiente de usuarios

Privilegios y Restricciones:

- Los administradores pueden gestionar a otros administradores, clientes y las solicitudes de creación de entornos.
- Los clientes pueden gestionar sus URLs y solicitar la creación de un entorno.
- Los endpoints deben estar protegidos según los roles de los usuarios.

4. Requisitos No Funcionales

4.1 Seguridad

Autenticación:

- Uso de tokens JWT para la autenticación de usuarios.
- Encriptación de contraseñas utilizando BCrypt.

4.2 Rendimiento

Escalabilidad:

- La API es capaz de manejar múltiples solicitudes concurrentes sin degradar el rendimiento.

4.3 Mantenibilidad

Código Limpio:

- El código sigue prácticas recomendables de programación y está bien documentado con la api de Swagger.
- Uso de anotaciones y validaciones para asegurar la integridad de los datos.

4.4 Usabilidad

Documentación:

- La API está documentada utilizando OpenAPI (Swagger) para facilitar su uso y comprensión.

4.5 Compatibilidad

Base de Datos:

- Uso de PostgreSQL como base de datos relacional.
- Configuración de Hibernate para la gestión de entidades y relaciones.