Documento de Requisitos del Proyecto URL Manager

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. 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 y clientes.
- Cliente: Puede gestionar entornos y 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.
- POST /admin: Registro de administradores.
- Inicio de sesión (login):
- POST /login: Autenticación de usuarios.
- Edición de datos personales:
- PUT /cliente: Actualización de datos de clientes.

- PUT /admin: Actualización de datos de administradores.
- Eliminación de usuarios:
- DELETE /cliente: Eliminación de clientes.
- DELETE /admin: Eliminación de administradores.

3.3 Gestión de Entidades

Entornos:

- Crear entorno: POST /entorno/create
- Actualizar entorno: PUT /entorno
- Eliminar entorno: DELETE /entorno/{id}
- Obtener todos los entornos: GET /entorno
- Obtener entorno por ID: GET /entorno/{id}

URLs:

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

3.4 Lógica dependiente de usuarios

Privilegios y Restricciones:

- Los administradores pueden gestionar a otros administradores y clientes.
- Los clientes pueden gestionar sus propios entornos y URLs.
- 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 debe ser capaz de manejar múltiples solicitudes concurrentes sin degradar el rendimiento.

4.3 Mantenibilidad

Código Limpio:

- El código debe seguir buenas prácticas de programación y estar bien documentado.
- Uso de anotaciones y validaciones para asegurar la integridad de los datos.

4.4 Usabilidad

Documentación:

- La API debe estar 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.