Memoria del Proyecto: API Segura con Sequelize, JWT y Redis

1. Introducción

Este proyecto consiste en una API RESTful desarrollada con Node.js que implementa un sistema completo de seguridad. Incluye autenticación de usuarios con JWT, almacenamiento seguro de contraseñas con bcrypt, y validación de entradas. La API interactúa con una base de datos SQL (SQLite) mediante Sequelize y utiliza Redis.

2. Objetivos del Proyecto

- Desarrollar una API RESTful completa con autenticación.
- Implementar control de acceso con tokens JWT.
- Gestionar usuarios y recursos mediante Sequelize y SQLite.
- Cachear peticiones GET para mejorar el rendimiento usando Redis.
- Incluir validación y sanitización de datos de entrada.

3. Tecnologías Utilizadas

- Node.js
- Express
- Sequelize
- SQLite
- Redis
- JWT
- bcrypt
- dotenv
- helmet

4. Estructura del Proyecto

```
api_segura_completa/
├── controllers/
├── middlewares/
├── models/
├── routes/
├── .env
├── db.sqlite
├── index.js
├── package.json
├── README.md
```

5. Endpoints de la API

POST /register → Registro de nuevos usuarios.
POST /login → Login y obtención de token JWT.
GET /recursos → Lista todos los recursos (con caché).
GET /recursos/:id → Obtiene un recurso por ID (con caché).
POST /recursos → Crea un nuevo recurso (requiere JWT, invalida caché).
PUT /recursos/:id → Actualiza un recurso (requiere JWT, invalida caché).
DELETE /recursos/:id → Elimina un recurso (requiere JWT, invalida caché).

6. Seguridad Implementada

La API implementa un sistema de autenticación mediante JWT. Las contraseñas se almacenan cifradas usando bcrypt. Las rutas sensibles requieren un token válido en la cabecera Authorization. Además, se aplica Helmet para protección básica contra ataques comunes y express-validator para evitar entradas maliciosas.

7. Implementación del Caché con Redis

- Redis se usa para cachear las respuestas de GET /recursos y GET /recursos/:id.
- Se configura un TTL de 10 minutos para las entradas.
- Tras POST, PUT o DELETE, se invalidan las claves correspondientes para mantener la coherencia.
- Redis se conecta mediante redis://localhost:6379 y se inicializa desde un middleware separado.

8. Pruebas y Validación

Las pruebas se realizaron con Postman:
- Registro de usuario y verificación de hash en la base de datos.
- Login correcto y obtención de token JWT.
- Acceso a rutas protegidas solo con token válido.
- Pruebas de validaciones (inputs inválidos, tokens inválidos).
- Verificación de funcionamiento del caché y su invalidación.
- Uso de redis-cli para inspeccionar las claves almacenadas.

9. Resultados y Observaciones

- La API funciona de forma segura y eficiente.
- Redis mejora significativamente los tiempos de respuesta en peticiones GET repetidas.
- Las validaciones y el uso de bcrypt protegen la integridad de los datos.
- El control de acceso basado en JWT restringe correctamente las operaciones protegidas.

10. Conclusiones y Mejoras Futuras

Este proyecto demuestra cómo integrar múltiples tecnologías clave de backend moderno en una sola solución segura. Las mejoras futuras podrían incluir:
- Tests automatizados con Jest y Supertest.
- Control de roles y permisos.
- Despliegue en la nube (Render, Railway o Heroku).

11. Repositorio GitHub

https://github.com/RaulMatas/Seguridad-pruebas-y-despliegue.git