Flavio Melián Santana **@flaviomelian**

Índice

Introducción

En el ámbito del desarrollo de software, la gestión eficiente de proyectos es un factor clave para el éxito de cualquier iniciativa tecnológica. Herramientas como GitHub han demostrado ser pilares fundamentales en la organización, colaboración y control de versiones, convirtiéndose en referentes dentro del ecosistema de desarrollo. Inspirado por esta necesidad y con la inquietud de explorar nuevas tecnologías, el presente proyecto tiene como objetivo el diseño y desarrollo de un sistema de gestión de proyectos de software que, si bien toma como referencia funcionalidades de plataformas consolidadas, busca ofrecer una alternativa moderna y adaptable a necesidades específicas.

Este proyecto se ha planteado también como un ejercicio de aprendizaje, orientado a la integración del stack tecnológico React-Spring, combinando la potencia declarativa de React con las capacidades de animación fluida y natural que ofrece Spring. Esta elección tecnológica no solo responde al interés académico de profundizar en su uso conjunto, sino también a la intención de crear una interfaz atractiva, dinámica y altamente interactiva para los usuarios.

La memoria que aquí se presenta detalla el proceso de diseño, implementación y validación del sistema, abordando tanto los aspectos técnicos como las decisiones de diseño que han guiado su desarrollo. A lo largo del documento, se analiza cómo se ha buscado equilibrar la funcionalidad con la experiencia de usuario, y cómo este enfoque puede traducirse en una alternativa viable y flexible frente a otras soluciones del mercado.

Arquitectura del Sistema

El sistema de gestión de proyectos se ha diseñado utilizando una arquitectura cliente-servidor multicapa, con una clara separación de responsabilidades entre el frontend (cliente), el backend (servidor de aplicaciones) y la capa de persistencia (base de datos). Esta decisión arquitectónica facilita el mantenimiento, la escalabilidad y el desarrollo paralelo de componentes.

Vista general

El sistema se compone de tres capas principales:

- **Frontend (Cliente React):** Responsable de la interfaz de usuario y la interacción con el usuario final. Implementado con React y tecnologías web modernas (HTML5, CSS3, JavaScript, Chart.js).

- **Backend (API REST Spring Boot):** Encargado de la lógica de negocio, el control de acceso y la exposición de servicios mediante una API RESTful.

- **Base de datos (PostgreSQL):** Almacena la información persistente, incluyendo proyectos, usuarios, tareas y mensajes.

Estas capas se comunican mediante peticiones HTTP, siguiendo el estándar REST para mantener la interoperabilidad entre componentes.

Descripción por capas

a) Capa de presentación (React)

Esta capa ofrece una GUI que permite:

- Visualizar proyectos y su estado mediante dashboards interactivos.

- Gestionar tareas, usuarios y mensajes.

- Realizar autenticación y control de sesión.
  La comunicación con el backend se realiza mediante `Axios` y peticiones `fetch`.

b) Capa de negocio (Spring Boot)

El backend se estructura en distintos módulos:

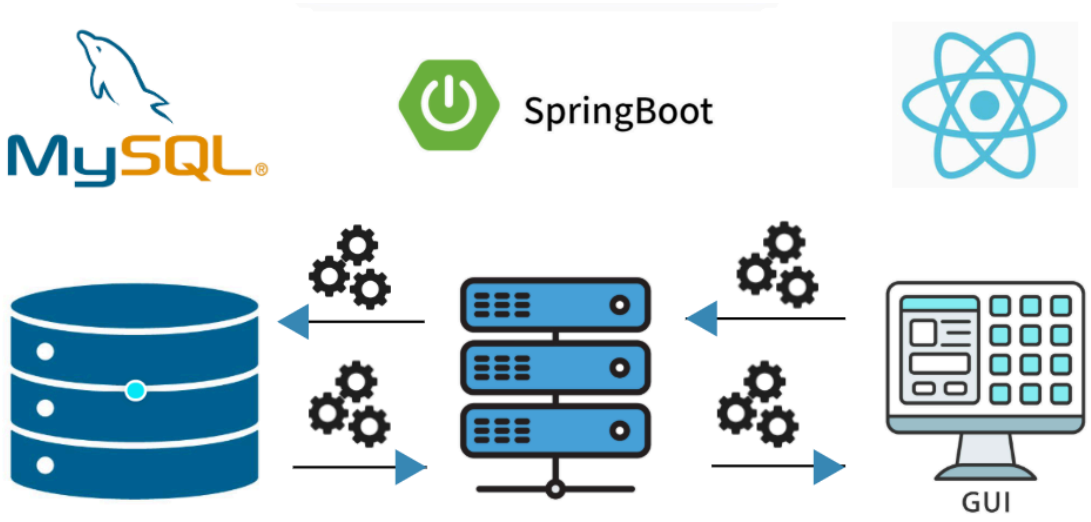
- **Controladores (`@RestController`):** gestionan las peticiones externas.
- **Servicios (`@Service`):** contienen la lógica del negocio.
- **Repositorios (`@Repository`):** proporcionan el acceso a la base de datos a través de JPA.

Además, se implementa seguridad mediante Spring Security, gestionando autenticación JWT y autorización basada en roles.

c) Capa de persistencia (MySQL + JPA)

La base de datos relacional MySQL almacena todas las entidades del sistema: usuarios, proyectos, tareas, mensajes, etc. El acceso a los datos se realiza mediante Spring Data JPA, lo cual permite abstraer la lógica SQL en interfaces declarativas.

Diagrama de arquitectura



Justificación de la arquitectura

Esta arquitectura se ha elegido por ser ampliamente adoptada en entornos de desarrollo modernos, favoreciendo la independencia de tecnologías, la reutilización de componentes y el desarrollo modular. La separación clara entre capas también mejora la capacidad de prueba, el control de versiones y la incorporación de mejoras futuras.

Librerias externas

BACKEND:

Spring Boot Starter Web

- `org.springframework.boot:spring-boot-starter-web`

Lombok

- `org.projectlombok:lombok` (opcional)

Spring Boot Starter Test

- `org.springframework.boot:spring-boot-starter-test` (scope: test)

Spring Boot Starter JPA

- `org.springframework.boot:spring-boot-starter-data-jpa`

Jakarta Persistence API

- `jakarta.persistence:jakarta.persistence-api:3.1.0`

H2 Database

- `com.h2database:h2` (scope: runtime)

MySQL Connector

- `com.mysql:mysql-connector-j:8.0.32`

Swagger (SpringDoc OpenAPI)

- `org.springdoc:springdoc-openapi-starter-webmvc-ui:2.8.6`

FRONTEND:

`axios` – ^1.8.4
`chart.js` – ^4.4.9

html2canvas – ^1.4.1
jspdf – ^3.0.1
react – ^19.0.0
react-chartjs-2 – ^5.3.0
react-dnd – ^16.0.1
react-dnd-html5-backend – ^16.0.1
react-dom – ^19.0.0
react-hot-toast – ^2.5.2
react-router-dom – ^7.4.1
chromatic-com/storybook – ^3.2.6
@eslint/js – ^9.21.0
@storybook/addon-docs – ^8.6.12
@storybook/addon-essentials – ^8.6.12
@storybook/addon-onboarding – ^8.6.12
@storybook/blocks – ^8.6.12
@storybook/experimental-addon-test – ^8.6.12
@storybook/react – ^8.6.12
@storybook/react-vite – ^8.6.12
@storybook/test – ^8.6.12
@types/react – ^19.0.10
@types/react-dom – ^19.0.4
@vitejs/plugin-react – ^4.3.4
@vitest/browser – ^3.1.3
@vitest/coverage-v8 – ^3.1.3
eslint – ^9.21.0
eslint-plugin-react-hooks – ^5.1.0
eslint-plugin-react-refresh – ^0.4.19
eslint-plugin-storybook – ^0.12.0
globals – ^15.15.0
playwright – ^1.52.0
prop-types – ^15.8.1
storybook – ^8.6.12
vite – ^6.2.0
vitest – ^3.1.3