Camyo

Grupo 5

ALM y Stack Tecnológico

Devising a Project (#DP)

Miembros del grupo:

- José Ramón Baños Botón
- Isabel X. Cantero Corchero
- Sheng Chen
- Carlos García Martínez
- Carlos García Ortiz
- Raúl Heras Pérez
- Pedro Jiménez Guerrero
- Claudia Meana Iturri
- Rubén Pérez Garrido
- Lucía Pérez Gutiérrez
- Francisco Pérez Manzano
- Diego José Pérez Vargas
- María C. Rodríguez Millán
- Sonia María Rus Morales
- Adriana Vento Conesa
- Jun Yao

Ingeniería del Software y Práctica Profesional • Curso 2024-25

Índice

Application Lifecycle Management (ALM)

El ALM (Application Lifecycle Management) del proyecto estará estructurado en varias fases clave del ciclo de vida del software. Cada una de estas fases estará soportada por herramientas específicas que facilitarán el desarrollo, la colaboración y la gestión del proyecto.

Fases del Ciclo de Vida y Herramientas Asociadas

1. **Planificación y Gestión del Proyecto**
    a. **Herramienta: GitHub Projects**
    b. GitHub Projects se utilizará para la gestión de tareas y sprints mediante tableros Kanban o Scrum.
    c. Permitirá definir hitos, asignar tareas y dar visibilidad al equipo sobre el estado del proyecto.
    d. Microsoft Teams facilitará la comunicación interna del equipo, permitiendo la coordinación en tiempo real mediante chats, videollamadas y reuniones.
    e. Se utilizará Teams para compartir documentación, archivos y recursos clave para el desarrollo del proyecto.
2. **Desarrollo e Implementación**
    a. **Herramientas: GitHub, Clockify**
    b. GitHub será el repositorio de código donde los desarrolladores trabajarán con ramas y pull requests.
    c. Clockify permitirá medir el tiempo de dedicación de cada miembro para evaluar la distribución del trabajo.
3. **Integración y Pruebas**
    a. **Herramienta: GitHub Actions**
    b. Automatizará la ejecución de pruebas unitarias, pruebas de integración y otros chequeos de calidad del código en cada push o pull request.
4. **Despliegue Continuo**
    a. **Herramienta: GitHub Actions**
    b. Se encargará de la integración y despliegue continuo, asegurando que cada versión nueva pueda ser puesta en producción de manera eficiente y sin errores.
5. **Mantenimiento y Soporte**
    a. **Herramientas: GitHub Issues**
    b. Se continuará utilizando GitHub Issues para reportar errores y gestionar mejoras futuras.

Stack Tecnológico

Antes de definir el stack tecnológico, se realizó un análisis interno mediante encuestas para evaluar las habilidades y preferencias del equipo. Con base en estos resultados, cada grupo tomó decisiones alineadas con sus fortalezas y conocimientos, en coordinación con el resto del equipo. El análisis completo se encuentra disponible en el documento **Análisis de Equipo**.

Frontend

Se ha elegido **React Native** para el desarrollo del frontend debido a las siguientes razones:

1. **Experiencia del equipo**: La mayoría del equipo de frontend tiene conocimientos previos en React, lo que permite un desarrollo más eficiente.
2. **Multiplataforma**: React Native permite desarrollar una aplicación PWA que funciona tanto en web como en dispositivos móviles sin necesidad de escribir código separado.
3. **Amplia comunidad y soporte**: React Native cuenta con una gran comunidad de desarrolladores, lo que facilita encontrar soluciones a problemas y disponer de múltiples librerías.
4. **Rendimiento optimizado**: Permite un buen equilibrio entre rendimiento y facilidad de desarrollo gracias a su arquitectura basada en componentes reutilizables.

Otras opciones consideradas y descartadas:

- **Flutter**: Se consideró debido a su rendimiento optimizado y desarrollo multiplataforma, pero el equipo no tenía experiencia con Dart, lo que hubiera generado una curva de aprendizaje pronunciada.
- **PWA con React tradicional**: Se descartó porque requería más trabajo de optimización para adaptarse a dispositivos móviles en comparación con React Native.

Backend

Para el desarrollo del backend se ha optado por **Spring Boot** con **Java** por las siguientes razones:

1. **Experiencia del equipo**: Los desarrolladores del backend están más familiarizados con Java y Spring Boot en comparación con otros frameworks.
2. **Escalabilidad y rendimiento**: Spring Boot proporciona una arquitectura robusta y eficiente para manejar grandes volúmenes de tráfico y procesos concurrentes.
3. **Ecosistema sólido**: Cuenta con una gran cantidad de herramientas y bibliotecas que facilitan la implementación de funcionalidades avanzadas.
4. **Seguridad**: Incluye herramientas integradas para la autenticación y autorización, lo que facilita la implementación de seguridad en el proyecto.

Otras opciones consideradas y descartadas:

- **Django (Python)**: Se consideró por su rapidez en el desarrollo y su ORM integrado, pero se descartó porque el equipo tenía más experiencia en Java y Spring Boot ofrece un mejor rendimiento en entornos empresariales.
- **Flask (Python)**: Aunque es más ligero y flexible, no proporciona tantas funcionalidades listas para producción como Spring Boot, lo que aumentaría la carga de trabajo del equipo.

Almacenamiento de Datos

Para la base de datos se ha elegido **MySQL** debido a:

1. **Compatibilidad con Spring Boot**: Se integra de manera sencilla con Hibernate y JPA, facilitando la gestión de datos en el backend.
2. **Rendimiento y estabilidad**: Es una base de datos relacional altamente optimizada para proyectos que requieren consistencia y escalabilidad.
3. **Facilidad de uso**: La experiencia previa del equipo con MySQL permitirá una implementación más ágil y sin una curva de aprendizaje pronunciada.
4. **Soporte y documentación**: MySQL es ampliamente utilizado en la industria, lo que asegura una gran cantidad de documentación y recursos disponibles para resolver problemas.

Otras opciones consideradas y descartadas:

- **PostgreSQL**: Se evaluó por sus capacidades avanzadas y mejor manejo de datos geoespaciales, pero MySQL era más familiar para el equipo y su rendimiento era suficiente para los requerimientos del proyecto.
- **MongoDB**: Se consideró por su flexibilidad en modelos NoSQL, pero se descartó porque el proyecto requería estructuras de datos relacionales, y SQL era la mejor opción para la integridad referencial.

- **SQLite:** Se valoró por su facilidad de configuración y bajo consumo de recursos, lo que la hace ideal para aplicaciones ligeras. Sin embargo, al tratarse de un proyecto que requiere concurrencia y escalabilidad, MySQL resultó una mejor opción.