SPRINT 6 – Generador Automático de Documentación para Proyectos Java

---

Alumno: Javier Manzano Oliveros
Curso: 2º DAW
Módulo: Entorno Cliente
Profesor: Ricardo Ruiz Anaya
Centro: ADA ITS
Año académico: 2025/2026

Índice

1. Introducción..... 2
2. Objetivo del Sistema..... 2
3. Arquitectura General del Proyecto..... 2
3.1 Diagrama de arquitectura general..... 3
4. Estructura del Repositorio..... 3
5. Backend – Diseño Técnico..... 4
5.1 Endpoints definidos..... 4
POST /api/analyze..... 4
GET /api/history..... 4
GET /api/docs/:id?format=pdf|md..... 4
5.2 Módulos internos del backend..... 5
5.3 Flujo principal del backend..... 5
6. Frontend – Diseño Técnico..... 6
6.1 Componentes previstos..... 6
6.2 Hooks utilizados..... 6
6.3 Flujo del usuario..... 7
7. Dockerización del Sistema..... 7
Contenedor del backend..... 7
Contenedor del frontend..... 7
docker-compose..... 7
8. Script de Automatización (Setup.ps1)..... 8
9. Mejoras Propuestas..... 8
10. Conclusión..... 8

1. Introducción

Este documento describe todo el diseño y planificación del sistema requerido en el Sprint 6 del módulo de Entorno Cliente.
El objetivo del sprint es **documentar** cómo funcionará un generador automático de documentación para proyectos Java, sin implementarlo todavía.

El sistema futuro será capaz de analizar código Java, generar diagramas UML, crear documentación en Markdown, enriquecer la información mediante IA y exportarlo a PDF, todo gestionado desde una interfaz web.

Este documento establece la estructura, arquitectura, endpoints, módulos y flujo completo para habilitar el desarrollo en sprints posteriores.

---

2. Objetivo del Sistema

Se debe diseñar un sistema que permita:

- Subir un proyecto Java (.zip).
- Analizar su contenido: clases, métodos, paquetes.
- Generar UML automáticamente mediante PlantUML.
- Crear una documentación en formato Markdown.
- Enriquecer la documentación mediante un modelo de IA local (por ejemplo, LMStudio).
- Convertir esa documentación a PDF.
  Mostrar al usuario un resultado descargable.
- Guardar un historial de ejecuciones.
- Ofrecer una interfaz web para gestionar todo el proceso.

  Importante: durante este sprint **no se implementa código**, solo se documenta el funcionamiento.

---

3. Arquitectura General del Proyecto

El sistema está compuesto por tres grandes módulos:

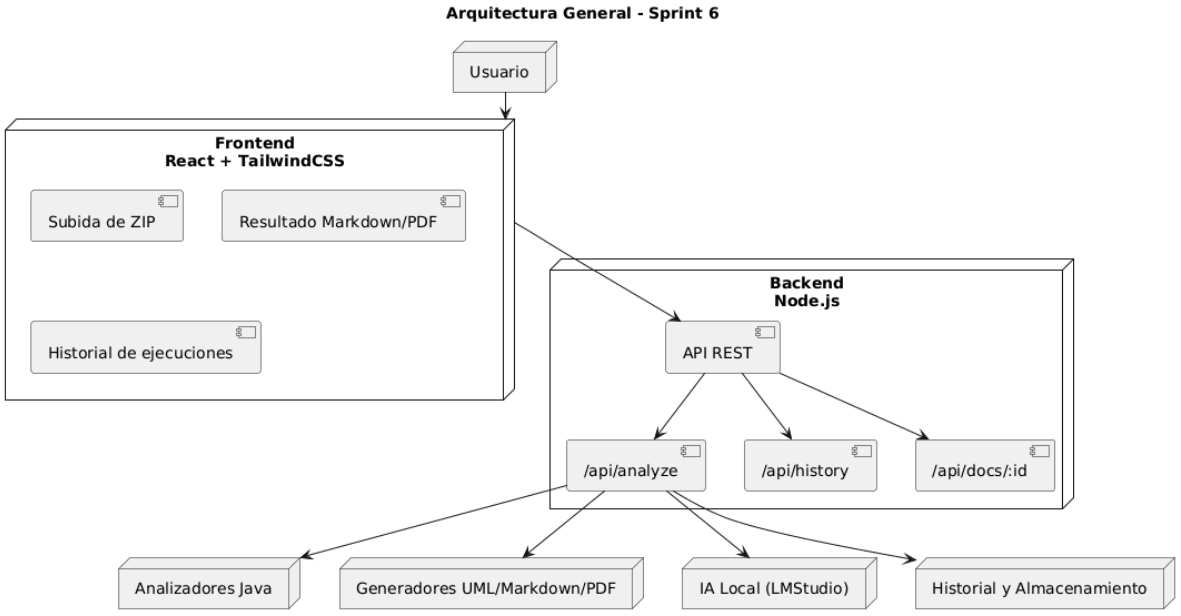
- **Frontend (React + TailwindCSS)**
  Interfaz que permite subir proyectos, ver el progreso del análisis, visualizar el resultado y consultar el historial.

- **Backend (Node.js)**
  API REST que recibe el proyecto, realiza el análisis, genera documentación y guarda resultados.
- **Infraestructura (Docker + PowerShell)**
  Encargada de contenerizar backend y frontend, además de integrar herramientas como PlantUML, Pandoc y el modelo IA local.

3.1 Diagrama de arquitectura general

El siguiente diagrama representa la arquitectura global del sistema.

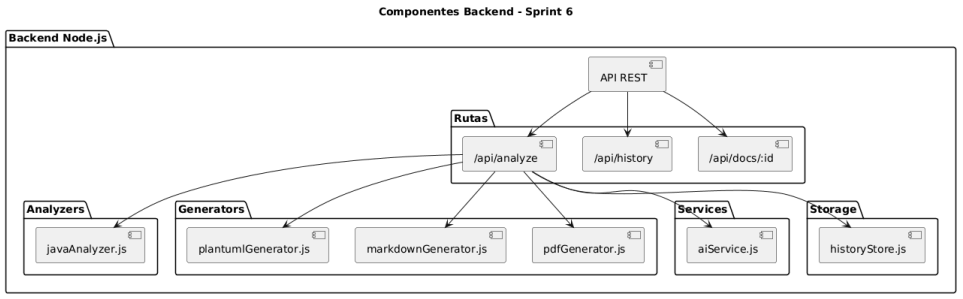
Se observa la interacción entre el usuario, el frontend desarrollado en React, el backend en Node.js y los servicios auxiliares encargados del análisis, la generación de documentación y el enriquecimiento mediante IA. También se incluye la infraestructura basada en Docker, que permite que todos los componentes funcionen en contenedores aislados y reproducibles. Esta vista global permite comprender cómo se comunican los distintos módulos y qué papel desempeña cada uno dentro del flujo completo del sistema.



3.2 Diagrama de Componentes del Backend

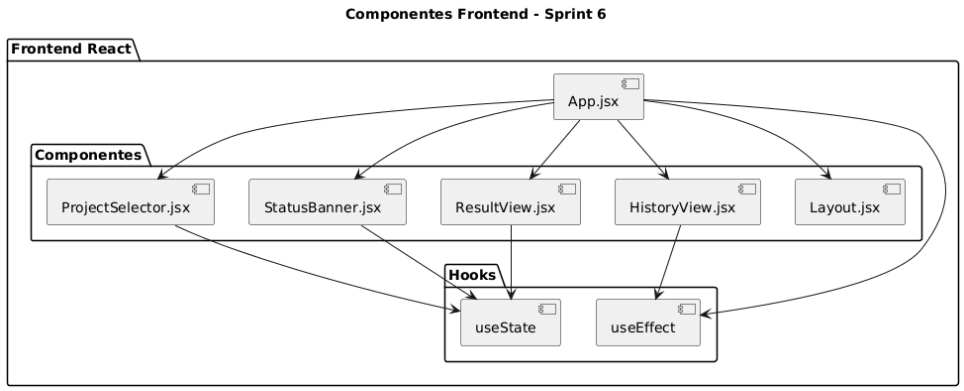
Este diagrama muestra los componentes principales del backend y cómo se organizan internamente.
El backend se divide en diferentes capas:
- rutas (endpoints expuestos),
- controladores,
- módulos de análisis,
- generadores (UML, Markdown y PDF),
- servicios (IA, almacenamiento e historial).

Esta separación clara de responsabilidades permitirá implementar un backend modular, extensible y mantenible. Además, refleja cómo los controladores se apoyan en módulos especializados para ejecutar tareas complejas como el análisis de código Java, la comunicación con la IA local o la conversión final a PDF.



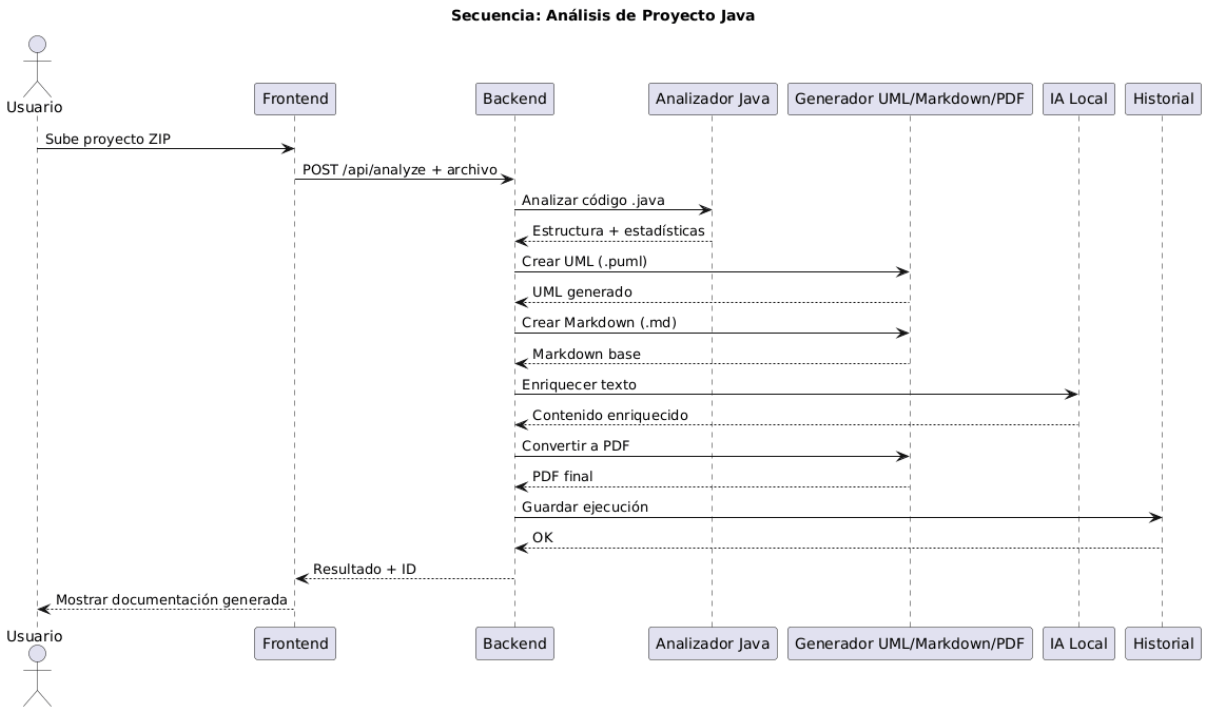
3.3 Diagrama de Componentes del Frontend

En este diagrama se detalla la estructura prevista del frontend.
La aplicación está organizada en componentes reutilizables (selector de proyecto, vista de resultados, historial, etc.) que se integran dentro de una disposición general definida por el componente Layout.
La gestión del estado se centraliza en App.jsx y se comunica a los componentes mediante props, siguiendo el patrón de lifting state up.
Asimismo, se muestra la relación entre las vistas principales (Home, Resultados e Historial) y los servicios encargados de realizar peticiones al backend. Esta arquitectura facilita el mantenimiento, la escalabilidad y la separación entre interfaz, lógica y comunicación con la API.



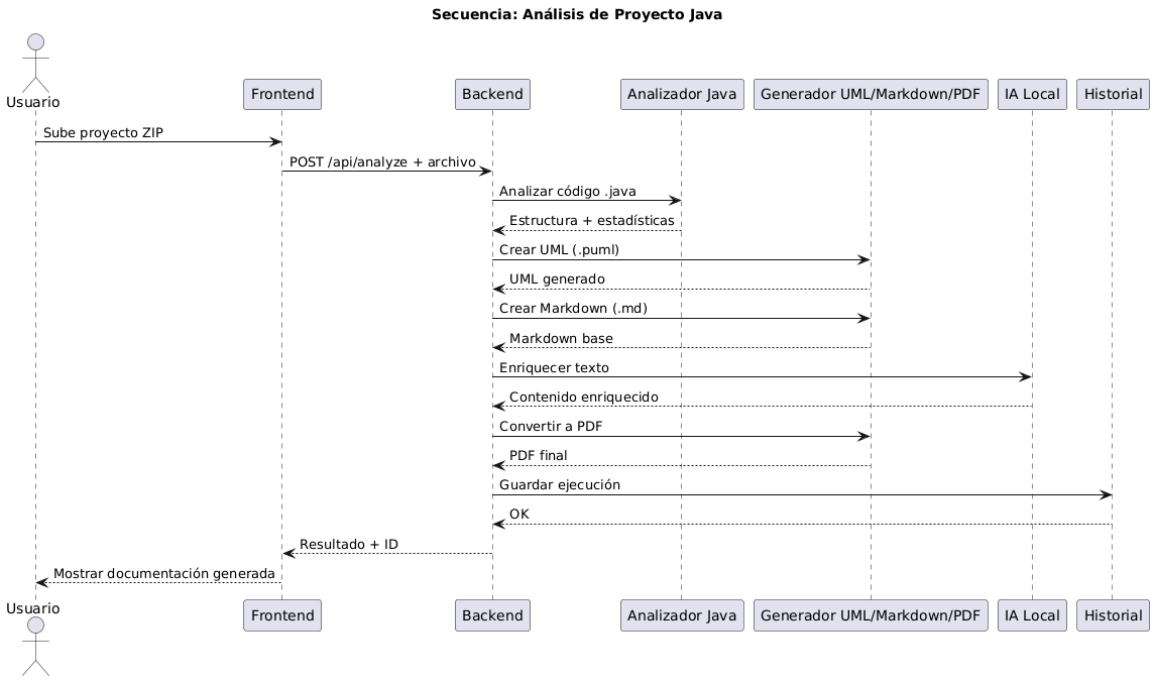
3.4 Diagrama del Flujo General del Sistema (Resumen Global / Actividad)

Este diagrama resume el flujo completo del sistema, desde que el usuario selecciona un archivo ZIP hasta que descarga la documentación generada.
Se muestran todas las etapas clave: análisis del código Java, generación de diagramas UML, creación y enriquecimiento del Markdown mediante IA, conversión a PDF y almacenamiento del resultado en el historial.
También se incluye el flujo secundario donde el usuario puede consultar ejecuciones anteriores.
Este diagrama proporciona una visión completa, lineal y detallada del comportamiento del sistema y cómo interactúan sus módulos principales.



3.5 Diagrama de Secuencia del Análisis

El diagrama de secuencia del análisis detalla la comunicación paso a paso entre el frontend, el backend y los módulos internos encargados del proceso.
Permite visualizar el orden exacto en el que se ejecutan las operaciones: recepción del ZIP, análisis de clases y paquetes, creación de UML, envío de información a la IA y generación final del PDF.
Este diagrama es especialmente útil para comprender la interacción entre los distintos componentes y asegura que la implementación futura mantenga el flujo definido en esta fase de documentación.



---

4. Estructura del Repositorio

El repositorio final del sprint debe seguir la siguiente organización:

```
SPRINT6/
  ├── diagrams/                    → Archivos .puml
  ├── diagrams_png/                → Exportaciones PNG/SVG
  ├── backend/                     → Estructura planificada
  ├── frontend/                    → Estructura planificada
  ├── docker/                      → Configuración planificada
  ├── Setup.ps1                    → Script documentado
  └── README.md                    → Documento principal
```

Este sprint no requiere código funcional, pero sí la **planificación completa** de la estructura de carpetas.

---

5. Backend – Diseño Técnico

El backend se implementará con Node.js y expondrá una API REST con tres endpoints principales.

5.1 Endpoints definidos

POST `/api/analyze`

- Recibe el archivo ZIP del proyecto Java.
- Extrae el contenido.
- Ejecuta el análisis de código.
- Genera UML.
- Genera Markdown.
- Envía el contenido al modelo IA.
- Convierte el Markdown a PDF.
- Guarda los archivos generados.
- Devuelve un ID y estadísticas.

GET `/api/history`

- Devuelve un historial de todas las ejecuciones realizadas.
- Incluye: id, nombre del proyecto, fecha, estadísticas del análisis.

GET `/api/docs/:id?format=pdf|md`

- Permite descargar el archivo Markdown o PDF generado.

5.2 Módulos internos del backend

Carpeta	Función
`analyzers/`	Analiza código Java (.java), detecta clases/métodos/paquetes
`generators/`	Genera UML (.puml), Markdown (.md) y PDF (.pdf)
`services/`	Comunicación con modelo IA local (LMStudio)
`storage/`	Guarda historial y localización de archivos generados
`routes/`	Encapsula endpoints REST
`config/`	Configuración

5.3 Flujo principal del backend

```
1. Recibir archivo ZIP desde /api/analyze
2. Descomprimir y analizar archivos .java
3. Generar estadísticas del proyecto
4. Crear diagramas UML
5. Generar documento Markdown
6. Enviar texto al modelo IA para ampliarlo
7. Insertar el texto IA dentro del Markdown
8. Convertir Markdown a PDF con Pandoc
9. Guardar todo en el historial
10. Devolver ID + estadísticas al frontend
```

---

6. Frontend – Diseño Técnico

El frontend se construirá usando **React + TailwindCSS**, priorizando:

- Estados controlados con hooks.
- Comunicación con el backend mediante fetch.
- Renderizado condicional.
- Componentes limpios y reutilizables.
- Vistas independientes para:
    - selección de proyecto,
    - carga,
    - resultado,
    - historial.

6.1 Componentes previstos

Componente	Función
`ProjectSelector.jsx`	Selección y subida del proyecto Java
`StatusBanner.jsx`	Estados: cargando, éxito, error
`ResultView.jsx`	Mostrar estadísticas + descargas
`HistoryView.jsx`	Mostrar ejecuciones pasadas
`Layout.jsx`	Estructura general

6.2 Hooks utilizados

- `useState`
- `useEffect`
- Eventos:
    - `onChange`
    - `onSubmit`
    - `onClick`
- Comunicación por props (lifting state up)

6.3 Flujo del usuario

```
1. El usuario selecciona un ZIP
2. Presiona "Generar documentación"
3. El frontend llama POST /api/analyze
4. El backend procesa el análisis
5. El frontend recibe el ID y estadísticas
6. Se muestra el resultado:
     - estadísticas
     - botones de descarga
7. El historial se actualiza automáticamente
```

---

7. Dockerización del Sistema

El sistema se ejecutará sobre contenedores Docker.

Contenedor del backend

Incluye:

- Node.js
- Java Runtime
- PlantUML
- Pandoc
- Dependencias necesarias para análisis

Contenedor del frontend

Incluye:

- React
- Vite
- TailwindCSS
  Se expone en el puerto 8978

docker-compose

- Construye ambos contenedores
- Los levanta en conjunto
- Enlaza frontend ↔ backend

---

8. Historial de ejecuciones — Explicación detallada

El sistema mantiene un historial de todas las ejecuciones realizadas por el usuario.
Este historial permite consultar fácilmente documentaciones generadas anteriormente sin necesidad de volver a subir el mismo proyecto.

Cada entrada del historial contiene:

- **id** → Identificador único de la ejecución
- **projectName** → Nombre del proyecto Java analizado
- **date** → Fecha y hora en la que se realizó la generación
- **status** → Estado final de la ejecución (OK / ERROR)
- **stats** → Datos útiles sobre el proyecto, por ejemplo:
    - número de clases encontradas
    - número de paquetes
    - líneas de código aproximadas
    - número de métodos
- **files** → Rutas a los archivos generados (Markdown, PDF, diagramas UML)

¿Para qué sirve el historial?

- Permite **descargar documentación anterior** sin regenerarla.
- Permite **consultar la evolución del proyecto** si se suben distintas versiones.
- Evita procesos largos e innecesarios.
- Permite crear **comparaciones** entre versiones en próximos sprints.

¿Cómo funciona internamente?

1. Cada vez que se llama a `POST /api/analyze`, el backend:
    - genera un nuevo ID
      crea una carpeta `/history/<id>/`
      guarda:
        - el Markdown generado
        - el PDF
        - los `.puml`
        - las estadísticas del análisis
2. Luego escribe/actuliza un archivo del tipo:

```
/history/history.json
```

Con algo así:

```
[
  {
    "id": "2025-02-10_001",
    "projectName": "alquiler-coches",
    "date": "2025-02-10T16:23:14",
    "status": "OK",
    "stats": {
      "packages": 6,
      "classes": 23,
      "methods": 142
    },
    "files": {
      "markdown": "history/2025-02-10_001/doc.md",
      "pdf": "history/2025-02-10_001/doc.pdf",
      "uml": "history/2025-02-10_001/diagrams/"
    }
  }
]
```

3. El frontend, al abrirse, llama a:

```
GET /api/history
```

Y muestra la información en una tabla con:

- proyecto
- fecha
- estado
- botón **"Ver resultados"**

9. Script de Automatización (Setup.ps1)

El script Setup.ps1 se encargará de:

1. Instalar dependencias del backend (`npm install`).
2. Instalar dependencias del frontend.
3. Construir imágenes Docker.
4. Levantar el proyecto completo con `docker-compose up --build`.

Permite, con un solo comando, iniciar todo el entorno.

---

10. Justificación de las Tecnologías Utilizadas

La elección de tecnologías se ha realizado siguiendo criterios de compatibilidad, facilidad de desarrollo, posibilidades de automatización y adecuación a los requisitos del sprint.

Permite construir una API REST ligera, modular y escalable utilizando JavaScript tanto en servidor como en cliente.
Su ecosistema facilita tareas como procesar archivos, lanzar procesos externos (PlantUML, Pandoc) y comunicarse con servicios de IA locales.
Es una opción óptima para construir un backend rápido y flexible.

React facilita la construcción de interfaces reactivas, organizadas en componentes reutilizables y con un control claro del estado mediante hooks.
TailwindCSS permite aplicar estilos de manera rápida y consistente, eliminando la necesidad de escribir CSS extenso y permitiendo avanzar más ágilmente en el diseño del frontend.
Juntas, estas tecnologías permiten crear una interfaz moderna, accesible y perfectamente mantenible.

Docker garantiza que todo el sistema (frontend, backend, PlantUML, Pandoc e IA local) se ejecute en entornos aislados y reproducibles.
Evita problemas de configuración entre máquinas diferentes y permite desplegar el proyecto completo con un solo comando (`docker compose up`).

PlantUML permite generar diagramas UML mediante archivos de texto `.puml`, lo cual facilita la automatización desde el backend.
No requiere herramientas gráficas, se integra fácilmente con scripts y produce diagramas profesionales de forma automática.

El uso de un modelo de IA ejecutado en local permite enriquecer las descripciones técnicas del proyecto sin depender de servicios externos.
Asegura privacidad, rapidez y autonomía, especialmente necesaria en entornos educativos donde el código analizado puede ser sensible.

Pandoc es una herramienta robusta capaz de convertir el Markdown generado automáticamente en un PDF final listo para entregar.
Permite unificar la documentación en un único flujo automatizado sin intervención manual.

El uso de PowerShell permite automatizar tareas como la instalación de dependencias, la construcción de imágenes Docker y el levantamiento de todos los servicios.
Facilita que cualquier usuario pueda iniciar el entorno completo de trabajo con un único comando.

En conjunto, estas tecnologías ofrecen un sistema moderno, modular, automatizable y completamente alineado con los requisitos técnicos del Sprint 6.

11. Mejoras Propuestas

El sistema podrá ampliarse fácilmente incorporando:

- Estadísticas avanzadas (LOC, métodos por clase, complejidad).
- Filtros por paquetes y clases.
- Comparación entre diferentes versiones del mismo proyecto.
- Soporte adicional para lenguajes como Kotlin, Python o JavaScript.
- Recomendaciones automáticas mediante IA:
    - refactorización,
    - reorganización de paquetes,
    - detección de code smells,
    - sugerencias de patrones de diseño.

---

12. Conclusión

El Sprint 6 establece **la documentación completa del sistema**, dejando listos todos los planos técnicos para comenzar la implementación real en los siguientes sprints.

El resultado de este sprint incluye:

- Documento PDF completo (este contenido).
- README.md detallado.
- Diagramas UML en `.puml` + imágenes.
- Planificación del frontend, backend y Docker.
- Estructura del repositorio definida.

Este sprint garantiza que el sistema pueda desarrollarse sin dudas ni improvisaciones.