SPRINT 6 – Generador Automático de Documentación para Proyectos Java

1. Introducción

Este documento forma parte del Sprint 6 del módulo de Entorno Cliente. El objetivo no es implementar un sistema real, sino **documentar la arquitectura, diseño técnico y procesos clave** del proyecto propuesto: un generador automático de documentación para proyectos Java. El sistema final, una vez implementado en futuros sprints, permitirá analizar proyectos Java, extraer su estructura interna, generar diagramas UML, producir documentación Markdown, integrarse con un modelo de IA y exportar la documentación resultante a formato PDF.

2. Arquitectura General del Sistema

La arquitectura propuesta se basa en una estructura modular donde cada componente cumple un rol específico, facilitando su mantenimiento, escalabilidad y sustitución en caso de necesidad. **Componentes principales:** - **Frontend (React + TailwindCSS):** Interfaz del usuario. - **Backend (Node.js):** API REST encargada del análisis, generación y comunicación. - **Infraestructura Docker:** Contenedores independientes que garantizan entornos reproducibles. - **Servicios externos:** Modelo IA local (LMStudio), PlantUML, Pandoc. El flujo del sistema sería: Usuario → Frontend → Backend → Analizadores / IA / Generadores → Documentación generada

3. Estructura del Proyecto

La estructura del repositorio se define para mantener orden y claridad: SPRINT6/ frontend/ → Código React (solo documentación en este sprint) backend/ → API Node.js (solo documentación) docker/ → Configuración de contenedores Docker Setup.ps1 → Script de configuración Documentación.pdf → Documento final entregado Cada carpeta está planificada para un desarrollo modular posterior.

4. Backend – Diseño Técnico en Profundidad

El backend es el corazón del sistema y se encargará de: 1. Recibir y procesar el proyecto Java subido por el usuario. 2. Analizar su contenido (clases, métodos, paquetes). 3. Generar los diagramas PlantUML mediante scripts internos. 4. Crear la documentación base en Markdown. 5. Integrarse con un modelo de IA para enriquecer los textos. 6. Convertir el Markdown final a PDF mediante Pandoc. 7. Registrar y devolver el historial de ejecuciones previas. ### 4.1 Endpoints definidos - **POST /api/analyze** → Recibe el proyecto Java y comienza el análisis. - **GET /api/history** → Obtiene el historial de ejecuciones previas. - **GET /api/docs/:id** → Devuelve los documentos generados (PDF o MD). ### 4.2 Submódulos del backend - **analyzers/** → Scripts

dedicados a analizar código Java. - **generators/** → Creación de UML, Markdown y PDF. - **services/** → Comunicación con IA. - **storage/** → Gestión del historial y almacenamiento de resultados. ### 4.3 Flujo del endpoint principal 1. Recibir archivo ZIP. 2. Extraerlo en un entorno temporal. 3. Analizar archivos .java. 4. Crear estadísticas. 5. Generar UML. 6. Generar Markdown. 7. Llamar al modelo IA para enriquecer la documentación. 8. Exportar PDF. 9. Guardar ejecución en historial. 10. Responder al frontend.

5. Frontend – Diseño Completo

El frontend estará construido con React en combinación con TailwindCSS. ### 5.1 Requisitos obligatorios - Gestión de estado con **useState** - Carga de historial con **useEffect** - Manejo de eventos: **onClick**, **onChange**, **onSubmit** - Componentes estructurados: selección, carga, historial, resultados - Renderizado condicional para mostrar: - estado “cargando” - errores - éxito - Vista de resultado que permita: - visualizar Markdown - descargar PDF - ver estadísticas - Vista de historial con ejecuciones previas ### 5.2 Componentes previstos - **ProjectSelector.jsx** - **UploadForm.jsx** - **ResultView.jsx** - **HistoryView.jsx** - **StatusBanner.jsx** - **Layout.jsx** Cada componente intercambiará información mediante props.

6. Dockerización del Proyecto

La aplicación se ejecutará en Docker mediante dos contenedores: 1. **Contenedor del Backend** - Node.js - Java Runtime - PlantUML - Pandoc - Dependencias necesarias para análisis 2. **Contenedor del Frontend** - React + Vite - Servido en el puerto 8978 El archivo `docker-compose.yml` se encargará de levantar ambos contenedores y conectarlos. Beneficios: - Reproducibilidad total del entorno - Compatibilidad entre sistemas - Aislamiento

7. Script Setup.ps1

El script Setup.ps1 automatizará: 1. Instalación de dependencias frontend 2. Instalación de dependencias backend 3. Construcción de contenedores Docker 4. Levantamiento del proyecto con un solo comando Además, permitirá preparar el entorno de pruebas sin necesidad de conocimientos avanzados del sistema.

8. Mejoras Adicionales Documentadas

Los siguientes apartados representan mejoras opcionales que el proyecto puede desarrollar: ### 8.1 Estadísticas Avanzadas del Proyecto - Conteo de líneas de código - Detección de clases grandes o complejas - Media de métodos por clase ### 8.2 Filtros por paquete o clase El usuario podría elegir generar documentación solo de ciertas partes del proyecto. ### 8.3 Comparación de versiones Permite comparar dos carpetas ZIP del mismo proyecto. ### 8.4 Soporte a múltiples lenguajes Extender el sistema para analizar: - Kotlin - Python - JavaScript - TypeScript ### 8.5 Recomendaciones de IA La IA podría: - Sugerir refactorizaciones - Detectar code smells - Reorganizar paquetes

9. Conclusión Final

Este Sprint se centra exclusivamente en la **documentación completa del futuro sistema**, sin desarrollar ninguna de sus partes. El resultado es un documento detallado, técnico y estructurado, que permite comprender cómo funcionará el proyecto una vez implementado. La información incluida en este PDF permitirá iniciar el desarrollo en sprints posteriores sin ambigüedades.