

Entorno Cliente

Generador Automático de Documentación para Proyectos Java

1. Objetivo

Desarrollar una aplicación capaz de analizar un proyecto Java y generar documentación técnica automáticamente.

- Finalizar la estructura básica del frontend en React con TailwindCSS.
- Implementar los endpoints principales del backend en Node.js.
- Preparar la integración con Docker para despliegue local.
- Configurar la base para la interacción con el modelo de IA (LMStudio u otro).
Garantizar que la interfaz permite la selección del proyecto Java, inicio de la generación y descarga de resultados.

DIAGRAMA COMPLETO (FRONTEND + BACKEND)

```
@startuml
title Diagrama de Componentes - Arquitectura General

node "Docker" {

    node "Contenedor Frontend" {
        component "React App\n(TailwindCSS)" as FE
        FE --> "http://backend:3000" : Peticiones REST
    }

    node "Contenedor Backend" {
        component "API Node.js" as API
        component "AnalyzerService" as AS
        component "UMLGenerator" as UML
        component "IAService" as IA
        component "MarkdownService" as MD
        component "HistorialService" as HS
    }
}
```

```

API --> AS
API --> UML
API --> IA
API --> MD
API --> HS
}

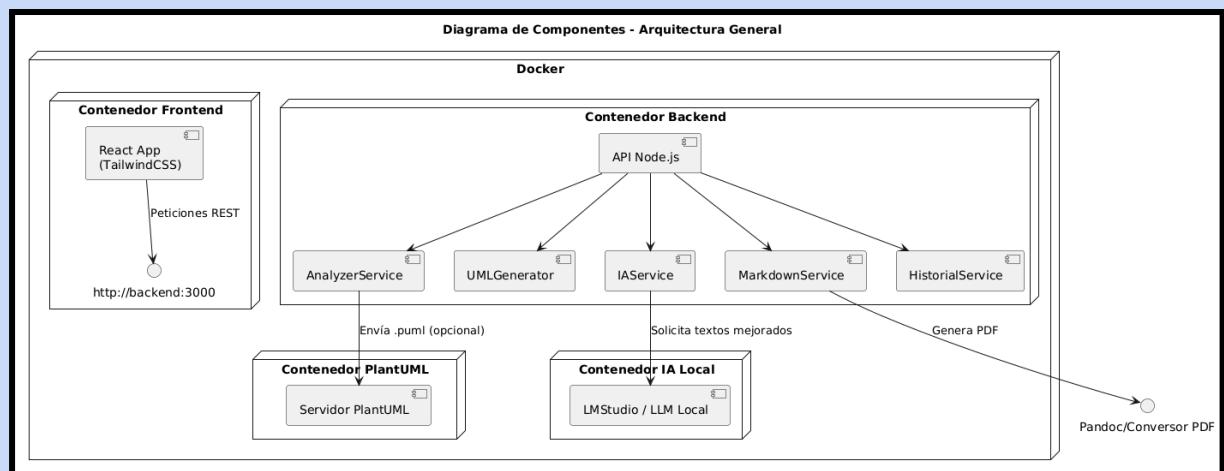
node "Contenedor PlantUML" {
    component "Servidor PlantUML" as PUML
}

node "Contenedor IA Local" {
    component "LMStudio / LLM Local" as LLM
}
}

AS --> PUML : Envía .puml (opcional)
IA --> LLM : Sigue textos mejorados
MD --> "Pandoc/Conversor PDF" : Genera PDF

@enduml

```



2. Frontend

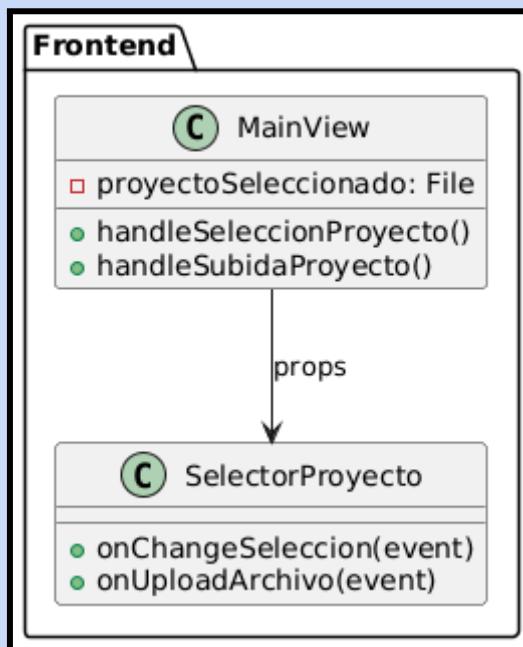
1. Una vista principal que permita seleccionar o subir el proyecto Java.

```
@startuml
package "Frontend" {
    class MainView {
        -proyectoSeleccionado: File
        +handleSeleccionProyecto()
        +handleSubidaProyecto()
    }

    class SelectorProyecto {
        +onChangeSeleccion(event)
        +onUploadArchivo(event)
    }
}

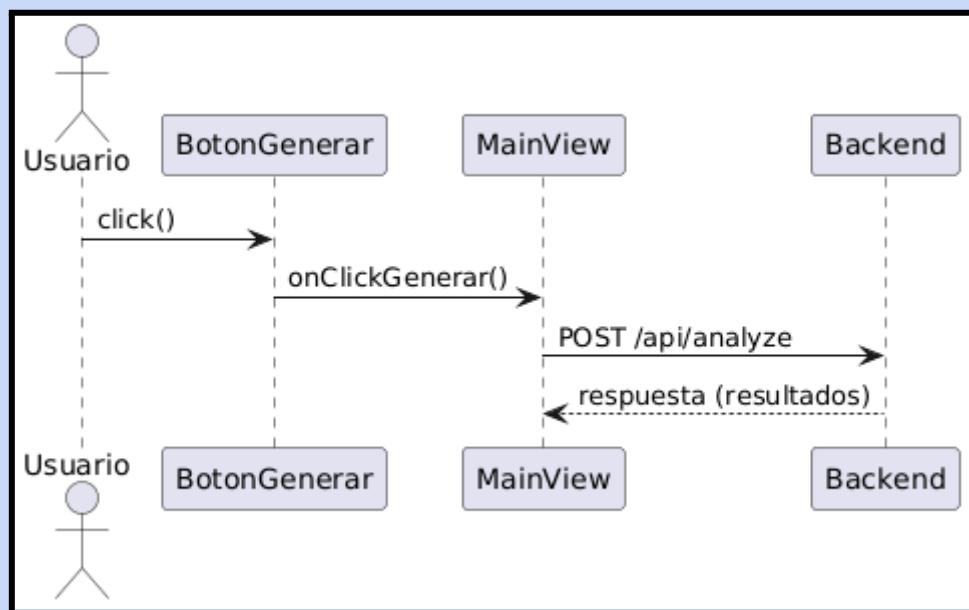
MainView --> SelectorProyecto : props

@enduml
```



2. Un botón de generación que dispare una petición al backend.

```
@startuml  
actor Usuario  
participant BotonGenerar  
participant MainView  
participant Backend  
  
Usuario -> BotonGenerar : click()  
BotonGenerar -> MainView : onClickGenerar()  
MainView -> Backend : POST /api/analyze  
Backend --> MainView : respuesta (resultados)  
  
@enduml
```



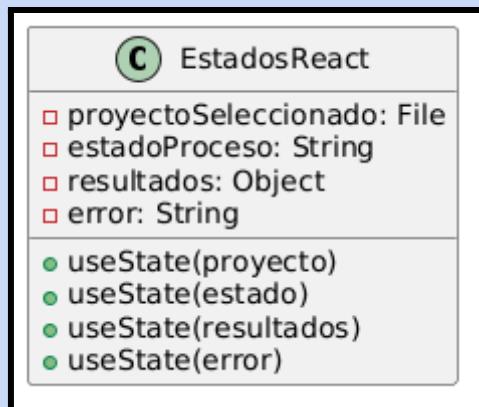
3. Gestión de datos mediante useState para:

- Proyecto seleccionado
- Estado del proceso
- Resultados
- Errores

```
@startuml
class EstadosReact {
    -proyectoSeleccionado: File
    -estadoProceso: String
    -resultados: Object
    -error: String
}

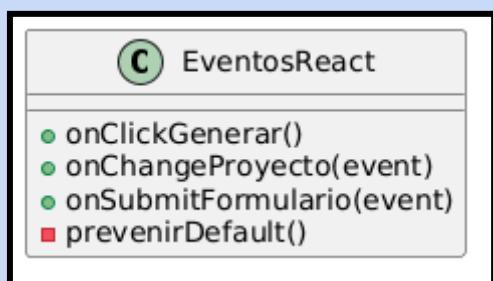
EstadosReact : +useState(proyecto)
EstadosReact : +useState(estado)
EstadosReact : +useState(resultados)
EstadosReact : +useState(error)

@enduml
```



4. Manejo de eventos: onClick, onChange y onSubmit.

```
@startuml  
  
class EventosReact {  
  
    +onClickGenerar()  
  
    +onChangeProyecto(event)  
  
    +onSubmitFormulario(event)  
  
}  
  
EventosReact : -prevenirDefault()  
  
@enduml
```



5. Comunicación entre componentes mediante props (lifting state up).

```
@startuml

class MainView {
    -proyectoSeleccionado
    +setProyecto()

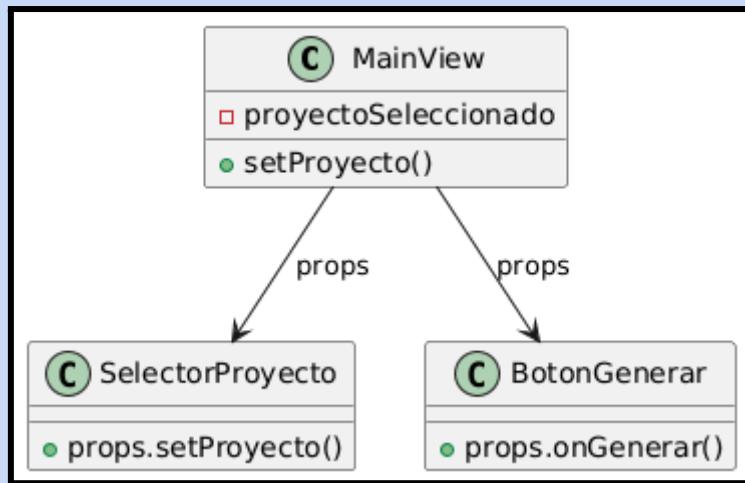
}

class SelectorProyecto {
    +props.setProyecto()
}

class BotonGenerar {
    +props.onGenerar()
}

MainView --> SelectorProyecto : props
MainView --> BotonGenerar : props

@enduml
```



6. Uso obligatorio de useEffect para cargar el historial o reaccionar a cambios de estado.

```

@startuml

class HistorialView {

    -historial: List

    +useEffect(() => cargarHistorial(), [])

    +cargarHistorial()

}

class Backend {

    +GET /api/history

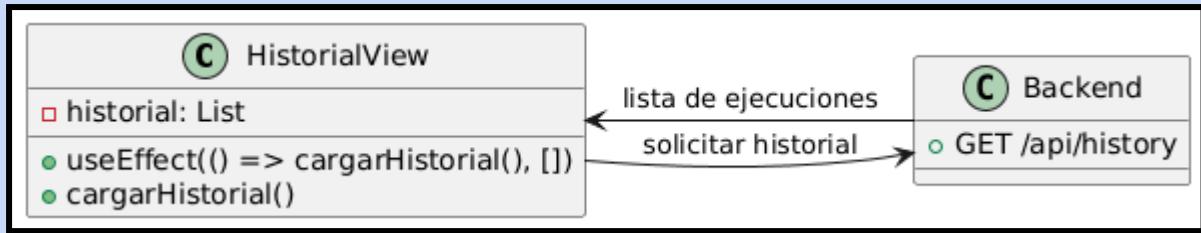
}

HistorialView -> Backend : solicitar historial

Backend --> HistorialView : lista de ejecuciones

@enduml

```



7. Renderizado condicional para mostrar estados de carga, error o éxito.

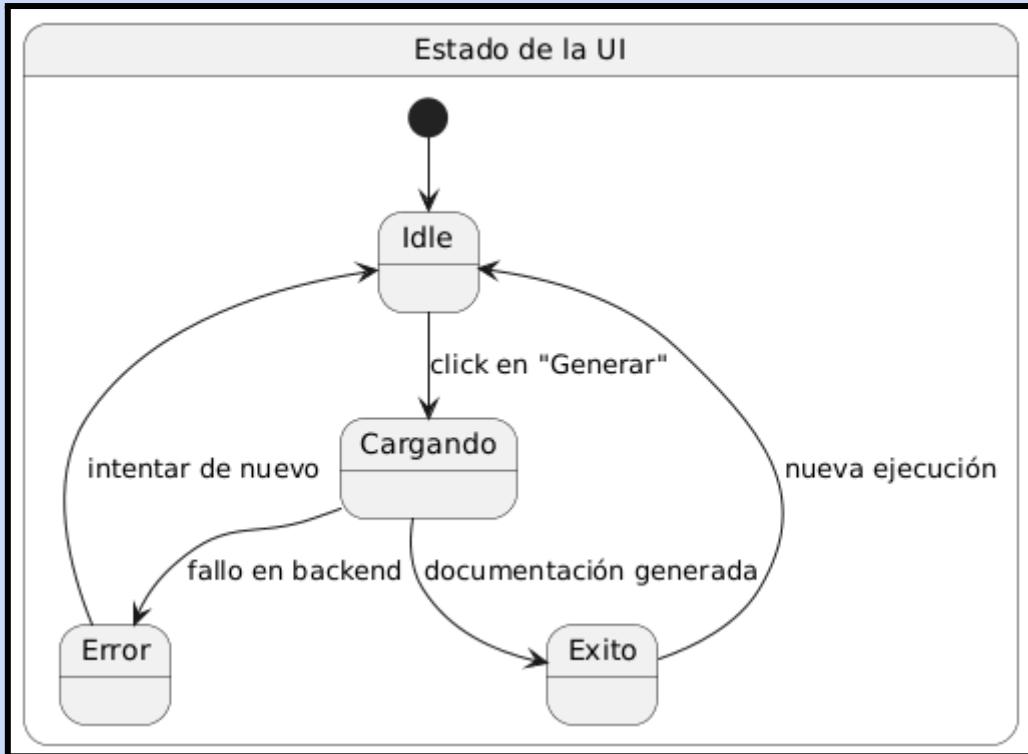
```

@startuml

state "Estado de la UI" as UI {
    [*] --> Idle
    Idle --> Cargando : click en "Generar"
    Cargando --> Error : fallo en backend
    Cargando --> Exito : documentación generada
    Error --> Idle : intentar de nuevo
    Exito --> Idle : nueva ejecución
}

@enduml

```



8. Una vista de resultado que renderice el Markdown generado y permita descargar el PDF.

```

@startuml

class ResultadosView {

    -markdown: String
    -pdfURL: String

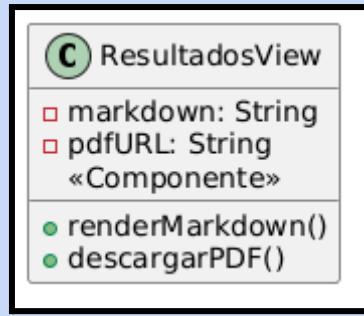
    +renderMarkdown()
    +descargarPDF()

}

ResultadosView : << Componente >>

@enduml

```



9. Una vista de historial con ejecuciones previas.

```

@startuml

class HistorialView {

    -itemsHistorial: List

    +useEffect(cargarHistorial)

    +mostrarHistorial()

}

class Backend {

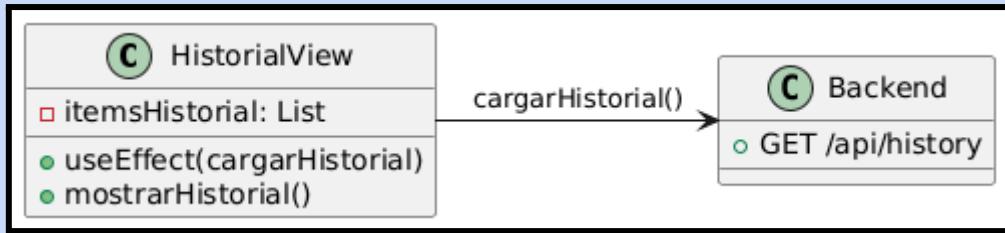
    +GET /api/history

}

HistorialView -> Backend : cargarHistorial()

@enduml

```



10. Estructura limpia de componentes:

- Selección
- Carga
- Resultado
- Historial

```

@startuml

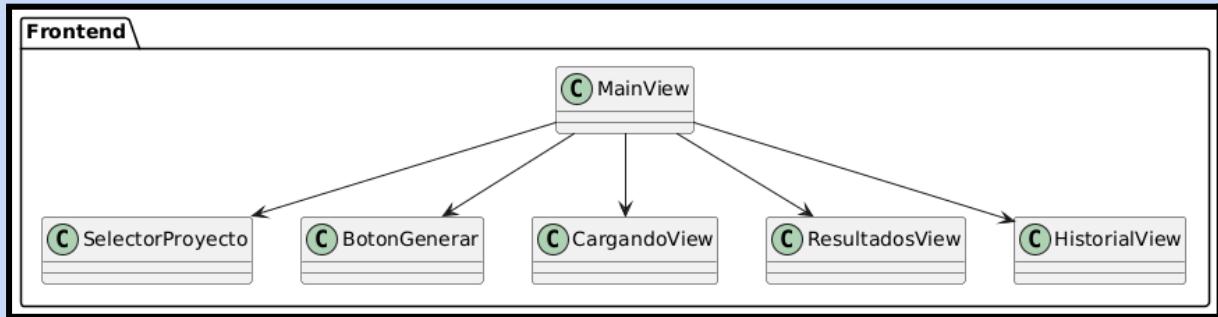
package "Frontend" {

    class MainView
    class SelectorProyecto
    class BotonGenerar
    class CargandoView
    class ResultadosView
    class HistorialView
}

MainView --> SelectorProyecto
MainView --> BotonGenerar
MainView --> CargandoView
MainView --> ResultadosView
MainView --> HistorialView

@enduml

```



3. Backend

El backend debe estar implementado en Node.js y exponer, como mínimo, los siguientes endpoints:

- POST /api/analyze: recibe el proyecto Java, lo analiza y genera la documentación.
- GET /api/history: devuelve el historial de ejecuciones anteriores.
- GET /api/docs/id devuelve los archivos generados (PDF o MD).

El backend debe:

1. **Analizar el código Java (parseo básico o modularizado en carpetas analyzers/).**

```

@startuml
title Análisis del Código Java

package "analyzers" {
    interface IJavaAnalyzer {
        +analyze(projectPath: String): AnalysisResult
    }

    class BasicAnalyzer {
        +analyze(projectPath: String): AnalysisResult
    }

    class ModularAnalyzer {
        +parseClasses(path: String)
        +parseMethods(path: String)
        +parseDependencies(path: String)
        +analyze(projectPath: String): AnalysisResult
    }
}

```

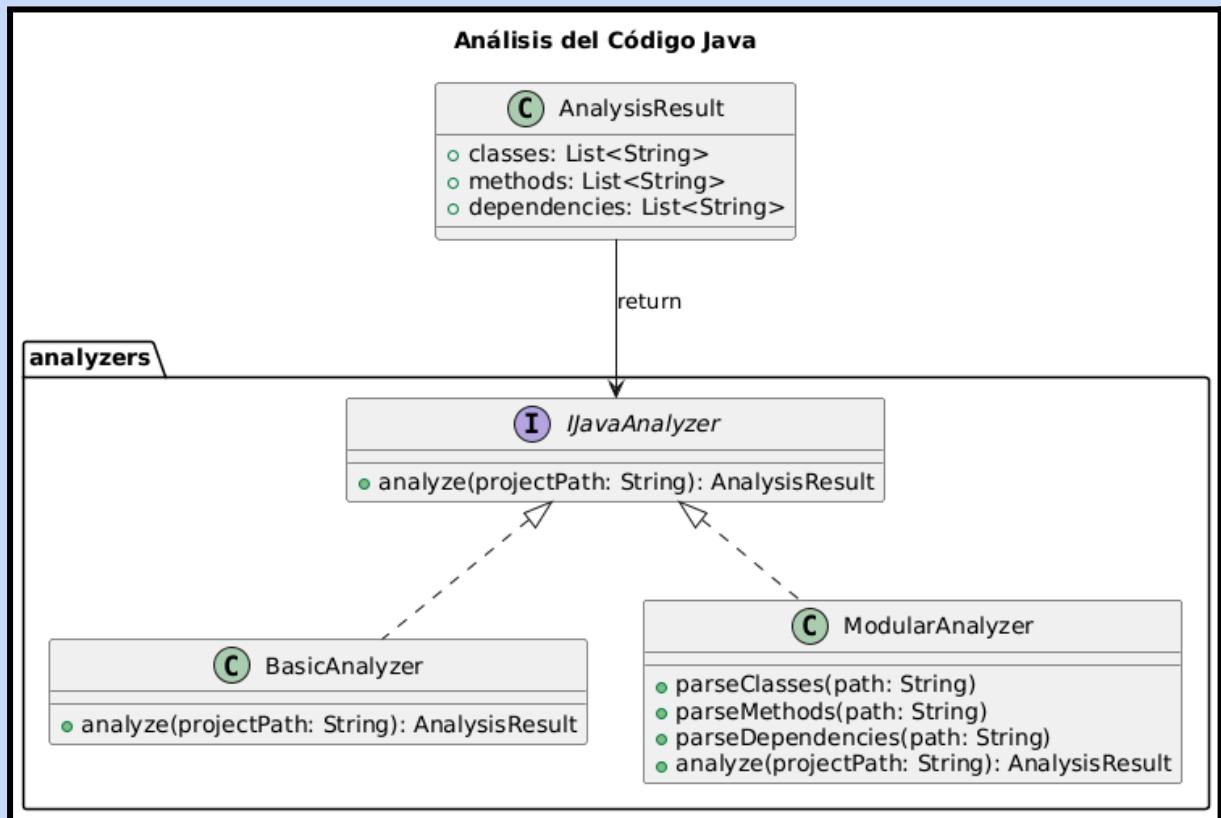
```

class AnalysisResult {
    +classes: List<String>
    +methods: List<String>
    +dependencies: List<String>
}

IJavaAnalyzer <|.. BasicAnalyzer
IJavaAnalyzer <|.. ModularAnalyzer
AnalysisResult --> IJavaAnalyzer : «return»

@enduml

```



2. Generar archivos PlantUML (.puml) en generators/.

```
@startuml
title Generación de Diagramas PlantUML

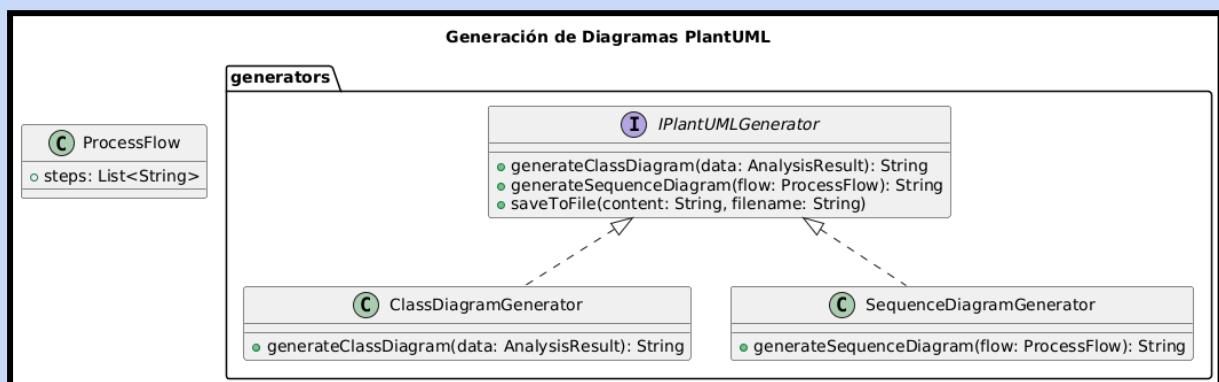
package "generators" {
    interface IPlantUMLGenerator {
        +generateClassDiagram(data: AnalysisResult): String
        +generateSequenceDiagram(flow: ProcessFlow): String
        +saveToFile(content: String, filename: String)
    }

    class ClassDiagramGenerator {
        +generateClassDiagram(data: AnalysisResult): String
    }

    class SequenceDiagramGenerator {
        +generateSequenceDiagram(flow: ProcessFlow): String
    }
}

class ProcessFlow {
    +steps: List<String>
}

IPlantUMLGenerator <|.. ClassDiagramGenerator
IPlantUMLGenerator <|.. SequenceDiagramGenerator
@enduml
```



3. Integrarse con un modelo de IA encapsulado en services/.

```
@startuml
title Integración con el Servicio de IA

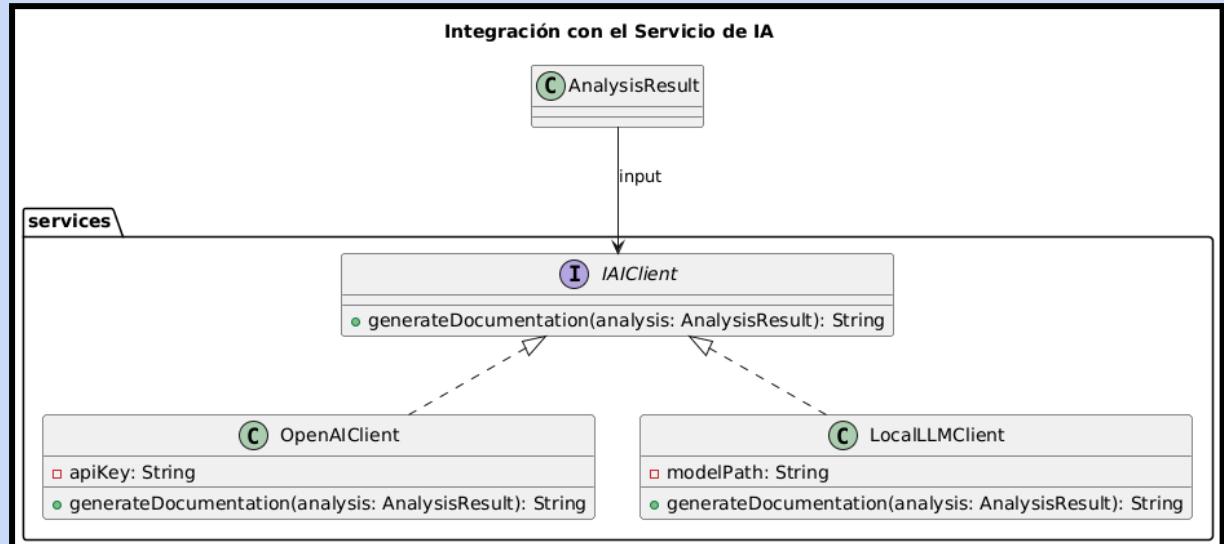
package "services" {
    interface IAIclient {
        +generateDocumentation(analysis: AnalysisResult): String
    }

    class OpenAIClient {
        +generateDocumentation(analysis: AnalysisResult): String
        -apiKey: String
    }

    class LocalLLMClient {
        +generateDocumentation(analysis: AnalysisResult): String
        -modelPath: String
    }
}

AnalysisResult --> IAIclient : input
IAIclient <|.. OpenAIClient
IAIclient <|.. LocalLLMClient

@enduml
```



4. Generar Markdown y convertirlo a PDF.

```
@startuml
title Generación de Markdown y PDF

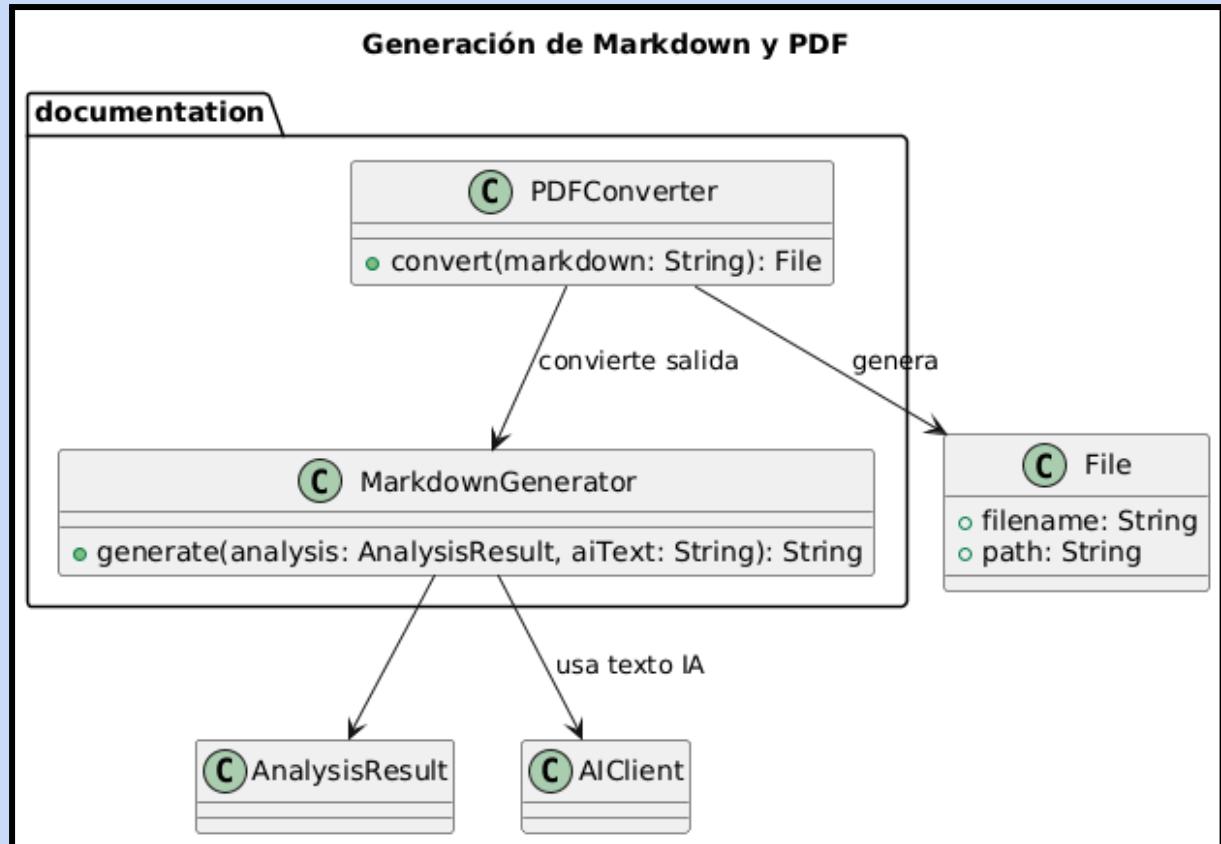
package "documentation" {
    class MarkdownGenerator {
        +generate(analysis: AnalysisResult, aiText: String): String
    }

    class PDFConverter {
        +convert(markdown: String): File
    }
}

class File {
    +filename: String
    +path: String
}

MarkdownGenerator --> AnalysisResult
MarkdownGenerator --> AIclient : usa texto IA
PDFConverter --> MarkdownGenerator : convierte salida
PDFConverter --> File : genera

@enduml
```



5. Gestionar un historial de ejecuciones.

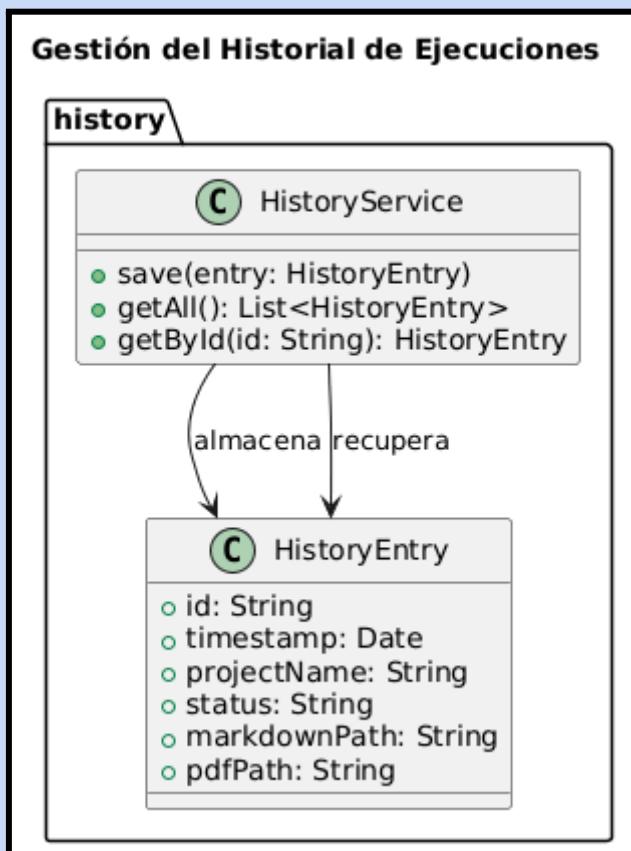
```

@startuml
title Gestión del Historial de Ejecuciones

package "history" {
    class HistoryService {
        +save(entry: HistoryEntry)
        +getAll(): List<HistoryEntry>
        +getById(id: String): HistoryEntry
    }

    class HistoryEntry {
        +id: String
        +timestamp: Date
        +projectName: String
        +status: String
        +markdownPath: String
        +pdfPath: String
    }
}
  
```

```
HistoryService --> HistoryEntry : almacena  
HistoryService --> HistoryEntry : recupera  
  
@enduml
```



6. DIAGRAMA UML DE SECUENCIA – POST /api/analyze

```
@startuml
title Secuencia: POST /api/analyze

actor Usuario
participant "Frontend React" as FE
participant "Backend Node.js" as BE
participant "AnalyzerService" as AS
participant "UMLGenerator" as UML
participant "IAService" as IA
participant "MarkdownService" as MD
participant "HistorialService" as HS

Usuario -> FE: Selecciona proyecto y pulsa "Generar"
FE -> BE: POST /api/analyze (archivo .zip / carpeta)

BE -> AS: analizarProyecto(rutaProyecto)
AS --> BE: estructuraCódigo

BE -> UML: generarDiagramas(estructuraCódigo)
UML --> BE: diagramas .puml

BE -> IA: enriquecerDocumentación(estructuraCódigo)
IA --> BE: descripcionesMejoradas

BE -> MD: generarMarkdown(estructuraCódigo, diagramas, IA)
MD --> BE: archivo .md

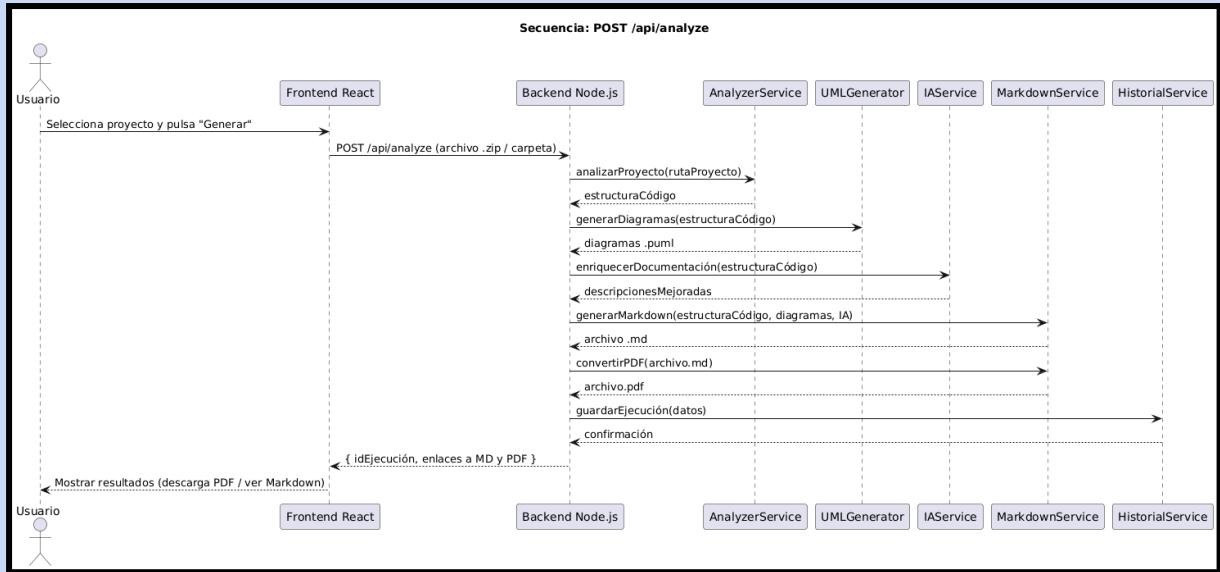
BE -> MD: convertirPDF(archivo.md)
MD --> BE: archivo.pdf

BE -> HS: guardarEjecución(datos)
HS --> BE: confirmación

BE --> FE: { idEjecución, enlaces a MD y PDF }

FE --> Usuario: Mostrar resultados (descarga PDF / ver Markdown)

@enduml
```



4. Mejoras a documentar

En el desarrollo del proyecto se han empleado las siguientes tecnologías obligatorias, cada una con un rol específico dentro del flujo de trabajo:

Node.js (Backend)

Node.js se utiliza como entorno de ejecución para implementar toda la lógica del backend. Permite:

- Exponer los endpoints REST.
- Ejecutar procesos de análisis del proyecto Java.
- Generar archivos Markdown y PDF.
- Comunicar el backend con servicios externos como modelos IA locales.

Su estructura se divide en carpetas como `routes/`, `controllers/`, `services/`, `analyzers/`, `generators/` y `history/`.

React + Tailwind (Frontend)

El frontend se ejecuta en el puerto **8978**.

React se utiliza para crear la interfaz modular basada en componentes, y TailwindCSS para acelerar el diseño mediante clases utilitarias.

Incluye:

- Vista principal
- Vista de resultados

- Vista de historial
- Gestión de estados mediante useState
- Efectos con useEffect
- Comunicación entre componentes por props

Docker

Todo el proyecto debe ejecutarse mediante **contenedores Docker**, incluyendo:

- Backend
- Frontend
- PlantUML
- Modelo de IA local (opcional)

Powershell

Se implementa un script `Setup.ps1` capaz de:

- Construir los contenedores
- Levantar el entorno completo con `docker compose`
- Comprobar versiones necesarias (Node, Docker, etc.)
- Crear carpetas iniciales del proyecto
- Descargar dependencias automáticamente

PlantUML

PlantUML se utiliza para la generación de diagramas UML en formato `.puml`, imprescindibles para documentar el backend y el frontend.

Todo se ejecuta dentro de un contenedor **plantuml**.

LMStudio

Se emplea como herramienta para enriquecer la documentación generada, especialmente:

- Interpretación del código Java
- Redacción de documentación
- Generación de descripciones y textos explicativos

El backend debe integrarse con este modelo a través de una capa `services/IAClient`.

UML A – Arquitectura General del Sistema (Frontend + Backend + Docker + IA)

```
@startuml

title Arquitectura General del Proyecto

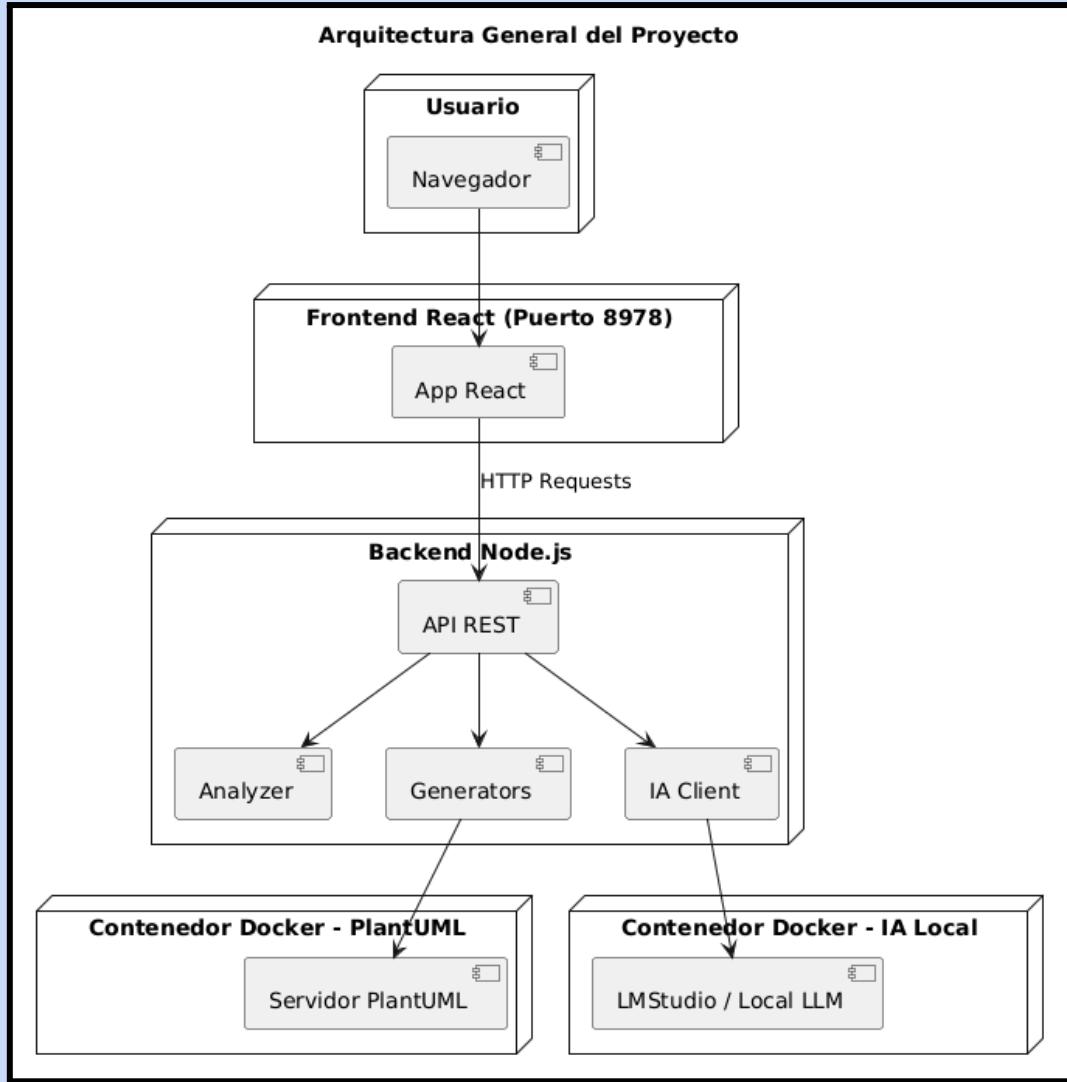
node "Usuario" {
    component "Navegador" as browser
}

node "Frontend React (Puerto 8978)" {
    component "App React"
}

node "Backend Node.js" {
    component "API REST"
    component "Analyzer"
    component "Generators"
    component "IA Client"
}

node "Contenedor Docker - PlantUML" {
    component "Servidor PlantUML"
}
```

```
node "Contenedor Docker - IA Local" {  
    component "LMStudio / Local LLM"  
}  
  
browser --> "App React"  
  
"App React" --> "API REST" : HTTP Requests  
  
"API REST" --> "Analyzer"  
  
"API REST" --> "Generators"  
  
"API REST" --> "IA Client"  
  
"Generators" --> "Servidor PlantUML"  
  
"IA Client" --> "LMStudio / Local LLM"  
  
@enduml
```



UML B – Proceso de Despliegue con Docker y Setup.ps1

```
@startuml
title Proceso de Deployment con Setup.ps1
```

```
actor Usuario
participant "Setup.ps1"
participant "Docker"
participant "Frontend React"
participant "Backend Node.js"

Usuario -> "Setup.ps1" : Ejecutar script
```

```
"Setup.ps1" -> Docker : docker compose build
```

```
Docker --> "Setup.ps1" : Imágenes construidas
```

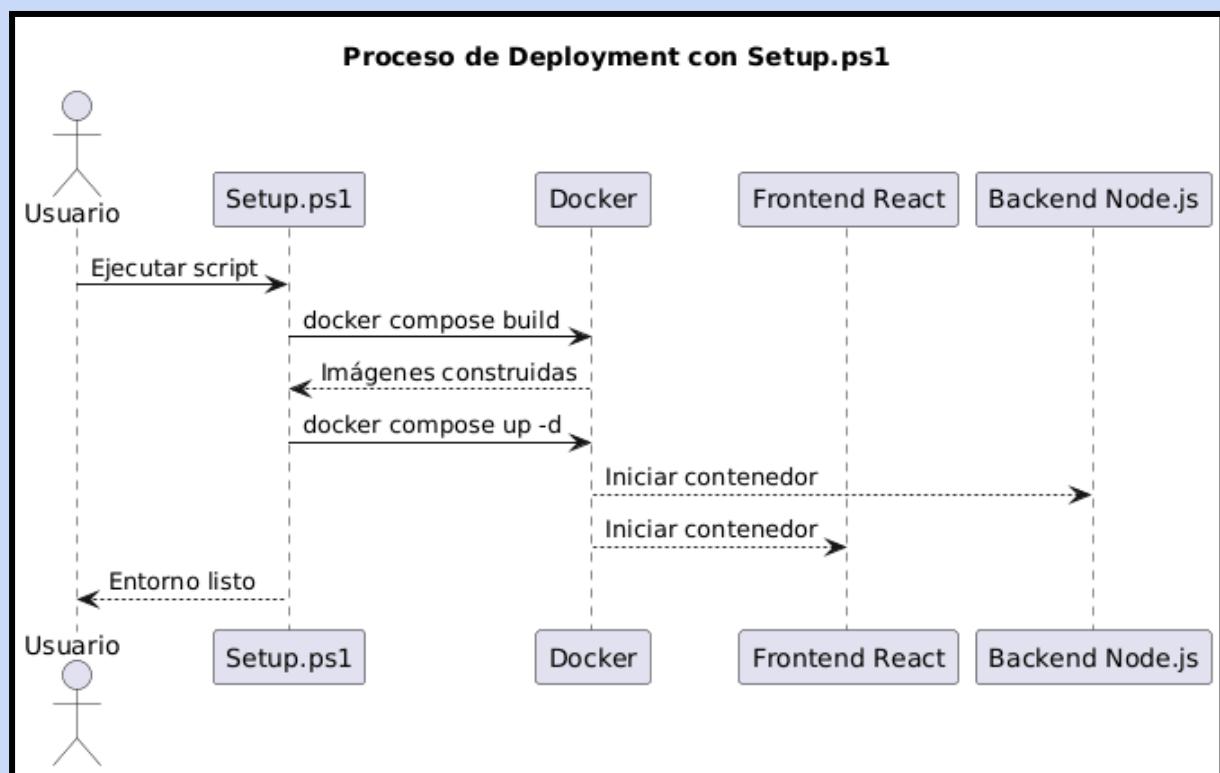
```
"Setup.ps1" -> Docker : docker compose up -d
```

```
Docker --> "Backend Node.js" : Iniciar contenedor
```

```
Docker --> "Frontend React" : Iniciar contenedor
```

```
Usuario <-- "Setup.ps1" : Entorno listo
```

```
@enduml
```



UML C – Flujo Markdown → PDF (Pandoc)

```
@startuml

title Conversión de Markdown a PDF

start

:Backend genera Markdown;

if (Markdown válido?) then (Sí)

    :Ejecutar Pandoc;

    :Generar archivo PDF;

else (No)

    :Registrar error;

endif

:Guardar PDF en historial;

stop

@enduml
```

