Universidad de Valladolid

ESCUELA DE INGENIERÍA INFORMÁTICA

GRADO EN INGENIERÍA INFORMÁTICA
Mención en Ingeniería de Software

Project Master: aplicación para la visualización, edición y mejora de diagramas Mermaid

Alumno: Diego González Guerra

Tutores: Diego García Álvarez, Luis Ignacio Jiménez Gil

...

Agradecimientos

Quisiera expresar mi más sinceros agradecimientos a mis tutores académicos Diego y Nacho, por su constante orientación, apoyo y recomendaciones a lo largo de todo el desarrollo de este Trabajo Fin de Grado. Su experiencia e implicación en este proyecto han sido fundamentales para el desarrollo permitiendome afrontar las adversidades.

Asimismo, deseo agradecer a la empresa AVL Ibérica por brindarme la oportunidad de desarrollar Project Master en un entorno profesional que me ha permitido crecer tanto personal como profesionalmente. Su confianza y disposición para facilitar todos los recursos necesarios han contribuido de manera decisiva a la realización de este proyecto. Más concretamente al que fue mi tutor de prácticas Jose Villalón por ser la persona más implicada en el proyecto por parte de AVL, brindándome apoyo y tutela.

Creo que también es importante agradecer a todos los compañeros que me han acompañado en este largo camino, brindado un gran apoyo y han sido nuevas amistades que dudo mucho que se disuelvan. Sin ellos este camino no hubiese sido y el mismo, y echando la vista atrás me he llevado gente increíble. Así que quería agradecerles de todo corazón por formar parte de mi vida, sois unos tíos duros.

Por último quisiera agradecer a mi familia, no solo por su apoyo durante este Trabajo Fin de Grado, si no durante todo este periodo académico. Se podría decir que los años de universidad han sido mis mejores y peores años de vida y mi familia ha estado para mi tanto en los momentos buenos y en los no tan buenos, por eso y muchas más cosas que no corresponde decir en un TFG muchas gracias.

Resumen

En este documento se presenta una herramienta orientada a la visualización y edición de diagramas de flujo, como trabajo fin de grado. El propósito de esta herramienta es facilitar el seguimiento del desarrollo de proyectos y documentar de forma clara las decisiones tomadas a lo largo del ciclo de vida del mismo. Esta herramienta está especialmente pensada para entornos colaborativos, donde la trazabilidad de las decisiones y la comprensión visual de los procesos son elementos clave para la eficiencia y la transparencia.

La aplicación incorpora dos funcionalidades principales. por un lado ofrece una interfaz basada en la vista de diagramas, permitiendo seleccionar el diagrama al que se quiere acceder para que posteriormente el usuario puede explorar el diagrama generado y completar las tareas asociadas a cada nodo, facilitando así el seguimiento del progreso y la trazabilidad de los procesos representados. Por otro lado la aplicación incorpora un modo de edición dinámico que permite al usuario generar diagramas utilizando Mermaid, el cual es un lenguaje de marcado ligero que facilita la generación automática de distintos tipos de diagrama.Este editor dinámico se ha construido usando Monaco, el mismo editor utilizado en Visual Studio Code, que proporciona una experiencia avanzada de escritura con funcionalidades como resaltado de sintaxis, autocompletado y manejo eficiente del código.

El proyecto ha sido desarrollado como una aplicación web de página única, utilizando TypeScript como lenguaje de programación principal y el framework React para la construcción de la interfaz de usuario. Se ha optado por una arquitectura de software que combina elementos de metodologías ágiles,para adaptarse de forma rápida a los cambios de requisitos por parte del Product Owner y con un enfoque más clásico en cuanto al diseño del software, lo que ha permitido establecer una base sólida y mantenible para el sistema.

Este trabajo ha sido desarrollado en colaboración con la empresa AVL Ibérica, en el marco de un proyecto que partía de una base inicial mínima, tanto a nivel funcional como estructural. A partir de esta base, se ha diseñado y construido una solución completa, incorporando nuevas funcionalidades, refinando el diseño existente y aplicando buenas prácticas de Ingeniería del Software. Esta colaboración ha permitido aplicar los conocimientos adquiridos en el ámbito académico a un entorno profesional real, integrando el trabajo en equipo, la comunicación con *stakeholders* y la entrega de valor continuo.

Abstract

The main objective of this project is the development of an interactive web application designed to visualize and edit flowcharts. The tool aims to facilitate the tracking of project development and the documentation of key decisions made throughout its lifecycle. It is especially useful in collaborative environments, where visual clarity and traceability of decisions are essential for effective communication and transparency.

The application includes a dynamic editing mode that enables users to generate diagrams using Mermaid syntax, a text-based diagramming language widely adopted in technical documentation. To enhance the editing experience, the Monaco Editor has been integrated, providing features such as syntax highlighting, autocompletion, and efficient code handling.

The project was developed as a single-page application (SPA) using TypeScript as the primary programming language and React as the framework for building the user interface. A hybrid development approach was adopted that combined agile methodologies, to accommodate evolving requirements from the product owner, with more traditional software design practices to ensure a robust and maintainable architecture.

This work was carried out in collaboration with AVL Ibérica, as part of a project that began with a minimal functional and structural base. From this starting point, a complete and functional solution was developed, expanding features, refining the user experience, and applying software engineering best practices in a real-world professional context.

Índice general

Lista de Figuras

Lista de Tablas

Capítulo 1

Introducción

1.1. Contexto

Este documento corresponde con la memoria del Trabajo Fin de Grado (TFG) del Grado en Ingeniería en Informática de la Escuela de Ingeniería Informática de la Universidad de Valladolid [21].

En el ámbito de la ingeniería del conocimiento, como en el mundo laboral la representación visual de los procesos y su estructura juega un papel fundamental en la facilidad del desarrollo de las tareas. Para ello la idea era crear una página web capaz de mejorar el formato Mermaid de los diagramas de flujo, permitiendo añadir información adicional a los nodos como son descripciones , listas de tareas y documentación.

Mermaid es un lenguaje de marcado ligero que permite generar diagramas de forma declarativa y sencilla a partir de texto plano. Uno de los motivos principales por los que se usó Mermaid como base fue debido a sus herramientas de integración con React y la amplia documentación y soporte en plataformas colaborativas como puede ser Github o StackOverflow.

La idea de desarrollar esta aplicación fue promovida por AVL Ibérica, una empresa con la que se ha colaborado estrechamente, y más concretamente por José Villalón, integrante del departamento eléctrico. Su objetivo principal era mejorar la trazabilidad de procesos secundarios y repetitivos, como la toma de decisiones sobre los componentes a utilizar en diferentes escenarios de ensayo. A continuación, se detalla brevemente el contexto y actividad de esta empresa .

AVL Ibérica S.A. es una empresa del sector de la automoción enfocada en ingeniería especializada en el desarrollo, simulación, ensayo e integración de sistemas de propulsión y tecnologías de movilidad avanzada. Fundada en 1990 como filial del grupo austriaco AVL List GmbH, su sede principal en España se encuentra en el Paseo Arco de Ladrillo, 68, planta 5, en Valladolid. Con más de 30 años de experiencia, AVL Ibérica opera en los mercados

de España, México y Portugal, y cuenta con un equipo internacional de aproximadamente 200 empleados. Sus actividades son bastante variadas, pero todas están relacionadas con el sector automovilístico, por ejemplo, motores de combustión interna, conducción automática y cámaras de prueba de vehículos. La empresa ofrece soluciones a los problemas de automoción que cubren todo el ciclo de desarrollo, desde la fase de diseño hasta la producción en serie, incluyendo la calibración y pruebas unitarias. Además, AVL Ibérica tiene proyectos novedosos en el área tecnológica, sobre todo fusionando características del mundo del software con el sector de la automoción, como puede ser la implementación de un software y ciberseguridad a un coche.

1.2. Motivación

La idea de este proyecto surge como respuesta a la ineficiencia observada en el control y seguimiento de subtareas dentro de proyectos de gran envergadura en el ámbito empresarial. Aunque existen herramientas ampliamente utilizadas para la gestión de proyectos, como Jira o Trello, estas se centran principalmente en la organización general de tareas y el control del progreso a nivel macro. Sin embargo, no ofrecen una solución adecuada para descomponer y representar visualmente procesos secundarios o complementarios, como pueden ser checklists de validación, flujos de decisión, o series de pasos obligatorios que deben cumplirse dentro de una misma tarea.

Actualmente, la representación visual de estos flujos suele realizarse mediante diagramas estáticos, como los generados con Mermaid o plataformas de diagramación tipo Kanvas, los cuales sí bien permiten mostrar relaciones entre nodos y procesos. No están diseñados para reflejar dinámicamente el progreso o estado de cada parte del diagrama.

Por tanto, se detecta una carencia en el ecosistema de herramientas existentes: no hay aplicaciones que permitan al usuario construir sus propios diagramas de flujo interactivos y personalizados, que además integren la capacidad de marcar el avance o validación de cada paso del proceso. Este proyecto nace con el objetivo de cubrir ese vacío, aportando una solución que combine la libertad de diseño de diagramas con la trazabilidad activa del progreso, contribuyendo así a una gestión más precisa, visual y adaptada a las necesidades reales de los equipos de trabajo.

Tras una investigación sobre qué aplicaciones podrían cumplir una función similar, se han analizado algunos visualizadores y editores de diagramas de flujo que no requieren instalación, tales como:

- Lucidchart
- Draw.io
- Creately
- Whimsical

Todas estas alternativas presentan el mismo problema y es que no te permiten marcar una serie de hitos o subtareas correspondientes a cada nodo. Aunque haya alguna como whimsical que si te permita trazar el progreso cambiando de color los nodos completados.

1.3. Objetivos

1.3.1. Objetivos de desarrollo

El objetivo principal es desarrollar una aplicación web usando el framework de react. Los principales objetivos que debe cumplir esta de la aplicación web son:

- Visualización de diagramas mermaid.
- Permitir visualizar subtareas de los nodos.
- Permitir trazabilidad visual del avance del diagrama.
- Permitir cargar archivos con formato mermaid.
- Modo edición dinámico.

Estos serían los objetivos principales que deberá cumplir la aplicación, aunque estos lleven distintas historias de usuario asociadas. Hay otras funcionalidades que por estar fuera del alcance del proyecto no se han podido cumplir, que son las siguientes.

- Añadir tareas de tiempo
- Permitir tener diagramas anidados.

1.3.2. Objetivos Académicos

Dado que este proyecto pertenece al ámbito académico, existen distintos objetivos de formación tanto personal como académica:

- Práctica en el desarrollo de un proyecto de software real, empezando una aplicación desde zero, desde la fase de análisis, hasta diseño y pruebas unitarias.
- seguimiento de un marco de trabajo de una metodología agil.
- Profundización en un nuevo *framework* de trabajo como es react.
- Entendimiento en el uso de distintas APIS y librerías open source como son Monaco y Mermaid.
- Mejorar mis conocimientos de UX/UI.

1.4. Estructura de la memoria

Este documento se estructura de la siguiente forma:

Capítulo 2 Requisitos y planificación: En este capítulo se explica la metodología Scrum, el como se ha aplicado a este proyecto es decir su división en *sprints*,división en épicas, contingencia de riesgos y un plan de presupuestos.

Capítulo 3 Análisis: En este capítulo se presenta un análisis inicial de Ingeniería del Software sobre la aplicación ProjectMaster, donde se pueden ver las historias de usuario identificadas un modelo de dominio inicial y distintos diagramas de actividad que permiten entender la estructura inicial de la aplicación y la interacción con el usuario.

Capítulo 4 Tecnologías Utilizadas: En este capítulo se presentan las tecnologías usadas durante el desarrollo de este trabajo fin de grado, permitiendo conocer mejor el contexto de la utilización de las mismas.

Capítulo 4 Diseño: Es un capitulo centrado en el diseño de la aplicación Project Master permitiendo ver la arquitectura basada en componentes usada, como se comunican los componentes entre si y distintas decisiones de diseño como son el uso de tecnologías e interfaces gráficas.

Capítulo 5 Implementación y pruebas: Se detallan campos como los *data types* usados, los útiles y las distintas pruebas sobre el sistema que se han ejecutado para corroborar la solidez del *software*

Capítulo 6 Seguimiento del proyecto: Se explican de manera detallada los distintos *sprints* desarrollando las historias de usuario.

Capítulo 7 Conclusiones: Se explica de una manera detallada las conclusiones de este trabajo fin de grado y como se alinean con los objetivos.

Anexo A Manuales: Incluye manuales de mantenimiento, instalación, despliegue, y uso.

Anexo B Resumen de enlaces adicionales: Incluye enlaces de interés sobre el proyecto, como el repositorio de código y una webgrafia.

Capítulo 2

Requisitos y Planificación

2.1. SCRUM

La metodología escogida para la realización del TFG ha sido la metodología agil, usando el marco de trabajo Scrum. Como marca el Manifiesto *for Agile Software Development* [11] el marco de trabajo SCRUM cumple los cuatros requisitos fundamentales.

- **Individuos e interacciones** sobre *procesos y herramientas*.
- **Software funcionando** sobre *documentación exhaustiva*.
- **Colaboración con el cliente** sobre *negociación contractual*.
- **Respuesta ante el cambio** sobre *seguir un plan*.

Más concretramente el marco de trabajo Scrum está especializado en la organización de un único equipo de tamaño reducido, incluso de un solo integrante, que entrega y mantiene productos complejos a través de un sistema iterativo e incremental en bloques de tiempo cortos llamados sprints. Este enfoque trata controlar lo máximo posible la incertidumbre del proyecto a través de un sistema de control de riesgos.

El marco de trabajo Scrum esta basado en tres piladres fundamentales, los cuales son:

- **Transparencia**: El pilar de la transparencia en Scrum implica hacer visible la información de manera deliberada. Las decisiones importantes en Scrum se basan en la información disponible, para evitar riesgos, por lo que es crucial que esta información esté accesible y visible para todos los involucrados. Al visibilizar los artefactos y asegurar el acceso a ellos por parte del equipo y los interesados, se logra alcanzar la transparencia.

La transparencia va en ambas direcciones, no solo del equipo Scrum hacia afuera, sino también desde los interesados hacia el equipo Scrum. Al priorizar la transparencia, se facilita el flujo de información clave y se crea un ambiente propicio para la colaboración y la mejora continua.

- **Inspección:** Tanto la evolución del producto como las mejoras en el proceso de desarrollo deben inspeccionarse con frecuencia en Scrum. La evolución del producto puede acercar o alejar del objetivo, por lo que es fundamental detectar y corregir los desvíos oportunamente. En un contexto complejo e incierto, es necesario revisar y validar constantemente los supuestos en los que se basan las decisiones.

  En Scrum, el Incremento es el artefacto que permite inspeccionar el avance del producto, mientras que la *Sprint Review* es el evento destinado a esa evaluación al final de cada Sprint. Por su parte, la mejora continua del proceso se inspecciona en la Retrospectiva, donde se evalúan los cambios introducidos y su efectividad.

- **Adaptación:** Si se descubre que el producto se desvía considerablemente de su objetivo o el proceso se sale de los límites tolerables, será necesario tomar decisiones de adaptación en Scrum. Estas decisiones deben tomarse lo antes posible para evitar desviaciones mayores.

  En la Retrospectiva es donde se toman decisiones de adaptación del proceso, lo que implica cambios en la forma de trabajar. En el Sprint Planning, al comenzar un nuevo Sprint, también se toman decisiones de adaptación del producto. Estos eventos en Scrum están diseñados específicamente para realizar ajustes y adaptaciones necesarios en función de la evolución del producto y del proceso de creación [10].

Como se ha visto con los pilares fundamentales, podemos garantizar que para proyectos innovadores con una alta incertidumbre y un seguimiento constante el marco de trabajo SCRUM es uno de los más eficientes, debido a su capacidad de minimizar la incertidumbre y amoldarse a requisitos cambiantes [19, 17].

Para poder usar el marco de trabajo Scrum se necesitan tres componentes fundamentales: roles , eventos,artefactos

Un equipo SCRUM es un pequeño grupo auto organizado(internamente se gestiona la forma más eficiente de realización de tareas) y multifuncional(disponen de todas las habilidades necesarias para la gestión y ejecución del proyecto sin ayudas externas). Dentro de un equipo SCRUM hay tres roles inprescindibles.

- ***Product Owner***: Es el responsable de maximizar el valor del producto, actuando como enlace entre los interesados(stakeholders) y el equipo de desarrollo. Su principal tarea es gestionar el ***Product backlog*** el cual es un artefacto que se explicara más adelante. Asegurandose de que las funcionalidades implementadas corresponden a las necesidades del cliente.
- ***SCRUM Master***: Es el encargado de impulsar y facilitar el uso de prácticas de metodología agil. Manteniendo un ritmo de trabajo sostenible, permitiendo que el equipo sea autosuficiente y que este experimente una mejora constante.En definitiva es el encargado de la gestion del equipo y la metodología de trabajo.

- **Equipo de desarrolo**: Constituido por el resto de integrantes del equipo, encargados del desarrollo del producto y aportar valor en cada iteración del mismo.

En el marco de trabajo Scrum existe una diversidad de eventos, los cuales tienen como prioridad cumplir los pilares de la metodología scrum mencionados con anterioridad. Estos eventos son periodicos con una duración acotada. Es decir que son planificados al principio y son de un una duración fija e inamovible.[10]

- **Sprint**: Es el evento principal el cual contiene al resto de eventos.Los sprint son eventos periódicos de una duracion fija de entre una y cuatro semanas. Los cuales se producen de manera secuencial durante todo el proceso de desarrollo.Mientras se pruducen los sprints los requisitos son inamovibles, pero la experiencia adquirida puede servir para modificar los de sprint posteriores.El artefacto resultante de estos cambios se llama incremento.
- **Sprint planning**: Es el evento de iniciación del sprint en el cual se planifican las tareas y objetivos del sprint. Es una reunión en la cual participan todos los integrantes del equipo y en la que se asignan los items del Product Backlog, por grado de prioridad que se intentaran realizar ese sprint. Para ello se hace una estimación de trabajo(horas)El resultado de este evento es el Sprint Backlog.
- **Daily SCRUM**: Reunion diaria de corta duración que sirve para organizar al equipo e informar a los clientes de los objetivos del sprint que han sido cumplidos. Sirve para mejorar la comunicación del equipo y acabar de manera eficiente con problemas de conclictos de opiniones.
- **Sprint Review**: Reunión que ocurre al final de cada sprint, en la que el equipo completo y los clientes discuten de los objetivos cunplidos del sprint.Para poder decidir los del siguiente sprint.En caso de que fuese necesario se puede llegar a modificar el Product Backlog.

En Scrum, los artefactos representan la materialización tangible del trabajo realizado y del valor entregado en cada iteración. Actúan como elementos clave que permiten visualizar el progreso, mantener la transparencia del proceso y asegurar la trazabilidad de las decisiones tomadas. A través de ellos, tanto el equipo como los stakeholders pueden comprender qué se ha hecho, qué se está haciendo y qué se planea hacer. Los artefactos principales son:

- **Product Backlog**:Documento principal que contiene la lista ordenada por prioridad de mejoras y requisitos que deben implementarse en el producto. Esta lista se estructura habitualmente en formato de historias de usuario, lo que facilita una comprensión clara y centrada en las necesidades del cliente. Su principal responsable es el Product Owner, quien se encarga de mantenerlo actualizado y alineado con el Product Goal(una meta futura que representa el estado deseado del producto y que guía el trabajo del equipo hacia la entrega de valor continuo)
- **Sprint Backlog**:Es la lista de historias de usuario petenecientes al product backlog que se quieren realizar en un sprint en concreto.Los responslables de este artefacto es

el equipo de desarrollo, su principal funcionan es la actualización de las daily Scrum.El objetivo del sprint backlog es alcanzar sprint goal, que es el estado idílico del proyecto al finalizar el sprint, aunque a su vez es flexible a la hora de la realización de trabajo.

- **Incremento**:Es el resultado del trabajo realizado durante un sprint, cada incremento como su propio nombre indica es acumulativo, es decir que la fusión de todos los incrementos da como resultado el producto final(product goal)Los responsables de este artefacto vuelven a ser el equipo de desarrollo, es importante que el incremento cumpla las condiciones mínimas de usabilidad ya que será probado por el cliente al final del sprint(normalmente durante el sprint review)

2.2. Aplicación de Scrum al proyecto

La decisión de usar el marco de trabajo Scrum fue tomado en base a principios de la metodología ágil ya que cumplía con bastantes exigencias de este proyecto, la primera exigencia era la adaptabilidad a los requisitos volátiles, ya que desde el principio la ampliación y cambio de requisitos fue una constante, por lo que la flexisivilidad que aporta Scrum en este aspecto fue fundamental para el correcto desarrollo del proyecto, por otro lado al ser la metodología de trabajo habitual de la empresa ayudaba a la integración y trazabilidad del proyecto de manera más cómoda y orgánica. Tambien otro punto importante para el uso de scrum fue el seguimiento semanal con el tutor de prácticas por parte de la empresa, facilitando la division en spritns semanales.

En cuanto al tema de los roles Scrum debido al ser un equipo de desarrollo de una única persona la division fue bastante sencilla. Los Scrum master fueron Jose Villalon Rodriguez(Responsable por parte de AVL), Diego García y Luis Ignacio Jiménez(tutores por parte de la UVA) y el alumno adopto los roles de equipo de desarrollo y *product owner*. De este modo estuve encargado del desarrollo y el *product backlog* mientras que mis tutores se encargaron de hacer cumplir los pricipios de la metodología ágil y asesorarme cuando fuese necesario.

Mientras tanto en la parte de eventos se optó por por sprints de una duración de una semana,por lo que los sprint planning y sprint review se producían de manera semanal. El dia escogido para estos eventos fue los viernes, mientras que los daily scrum se mantuvo el formato estandar de reunión diaría para la parte del desarrollo , mientras que para la memoria se modifico a una reunión cada tres días.

2.3. Planificación

La planificación inicial fue planteada para un periodo de tres meses, de lo cual se deriva en una una división en 13 sprints semanales con un trabajo medio de entre 25 y 30 horas. Aunque esto puede ser variable, quedando reflejado en el *sprint backlog*.El número de horas totales aproximadas sería entre unas 340, sabiendo que el trabajo fin de grado según la guia

docente corresponde a 12 créditos, las horas estimadas cumplen los requisitos de tiempo, incluso sobrepasándolos.

2.4. Calendario de Sprints (Semana 14 a 27 del año 2025)

Nº Sprint	Fecha inicio	Fecha fin	eventos
Sprint 1	31/03/2025	06/04/2025	Sprint Review, Sprint Retrospective & Sprint Planning 04/04/2025
Sprint 2	07/04/2025	13/04/2025	Sprint Review, Sprint Retrospective & Sprint Planning 11/04/2025
Sprint 3	14/04/2025	20/04/2025	Sprint Review, Sprint Retrospective & Sprint Planning 18/04/2025
Sprint 4	21/04/2025	27/04/2025	Sprint Review, Sprint Retrospective & Sprint Planning 25/04/2025
Sprint 5	28/04/2025	04/05/2025	Sprint Review, Sprint Retrospective & Sprint Planning 02/05/2025
Sprint 6	05/05/2025	11/05/2025	Sprint Review, Sprint Retrospective & Sprint Planning 9/05/2025
Sprint 7	12/05/2025	18/05/2025	Sprint Review, Sprint Retrospective & Sprint Planning 16/05/2025
Sprint 8	19/05/2025	25/05/2025	Sprint Review, Sprint Retrospective & Sprint Planning 23/05/2025
Sprint 9	26/05/2025	01/06/2025	Sprint Review, Sprint Retrospective & Sprint Planning 30/06/2025
Sprint 10	02/06/2025	08/06/2025	Sprint Review, Sprint Retrospective & Sprint Planning 06/06/2025
Sprint 11	09/06/2025	15/06/2025	Sprint Review, Sprint Retrospective & Sprint Planning 13/06/2025
Sprint 12	16/06/2025	22/06/2025	Sprint Review, Sprint Retrospective & Sprint Planning 20/06/2025
Sprint 13	23/06/2025	29/06/2025	Sprint Review, Sprint Retrospective & Sprint Planning 27/06/2025

2.5. Plan de Riesgos

Durante la planificación inicial del proyecto, también se realizó un plan de contención de riesgos. Ya que como todo proyecto, una vez hecha la planificación inicial aparece un componente de incertidumbre el cual hay que gestionar. Ya que nos estamos basando en suposiciones y conjeturas, pero en el ámbito real pueden ocurrir situaciones excepcionales que impidan el correcto desarrollo de plan inicial.

Un riesgo es un suceso futuro el cual puede comprometer el desarrollo del proyecto, por este motivo es importante realizar un plan de contención de riesgos para poder contener o minimizar los efectos de los mismos. Para ello se han tenido en cuenta 6 elementos de cada riesgo:

- **Titulo**: Frase autoexplicativa del riesgo
- **Descripción**: Breve explicación que detalla el riesgo.
- **Probabilidad**: Como su propio nombre indica es la probabilidad de ocurrencia de un riesgo, se valorará en alto,medio y bajo.
- **Impacto**: Es el daño que puede ejercer ese riesgo en el caso de que ocurra. Como ocurre con la probabilidad se valora con medio,alto y bajo.
- **Mitigación**: Son la serie de medidas que se toman para que un riesgo en concreto no llegue a manifestarse
- **Contingencia**: Serie de medidas que se toman cuando el riesgo ya se ha manifestado y se busca paliar sus efectos adversos.

De esta manera podemos clasificar por peligrosidad los riesgos y tener de manera sencilla y detallada los planes de mitigación y contingencia. A continuación se muestran los riesgos principales identificados en este trabajo fin de grado:

La Tabla 2.2 muestra el riesgo asociado a problemas de salud.

Nombre	Problemas de salud
Descripción	Enfermedad inesperada que impida trabajar durante un tiempo
Categoría	Salud
Vulnerabilidad	Imposibilidad de avanzar en el proyecto por razones de salud
Amenaza	Enfermedad o problemas de salud
Probabilidad	Media
Impacto	Bajo
Riesgo Total	Bajo
Estado del Riesgo	Abierto
Acciones de mitigación	Planificar los plazos con margen adicional
Acciones correctivas	Solicitar una extensión del plazo

Tabla 2.2: Riesgo: Problemas de salud

La Tabla 2.3 muestra el riesgo asociado a fallos tecnológicos.

Nombre	Fallos tecnológicos
Descripción	Problemas técnicos que puedan afectar al desarrollo
Categoría	Tecnología
Vulnerabilidad	Posibilidad de pérdida de datos o bloqueo del avance
Amenaza	Pérdida de información crítica
Probabilidad	Media
Impacto	Alto
Riesgo Total	Alto
Estado del Riesgo	Abierto
Acciones de mitigación	Realizar copias de seguridad frecuentes
Acciones correctivas	Restaurar desde backup o cambiar de equipo

Tabla 2.3: Riesgo: Fallos tecnológicos

La Tabla 2.4 muestra el riesgo derivado de la disponibilidad del tutor.

Nombre	Tutor ocupado
Descripción	El tutor no dispone de tiempo para reuniones
Categoría	Personal
Vulnerabilidad	Dependencia del tutor para avanzar y obtener feedback
Amenaza	Dificultad para coordinar reuniones con el tutor
Probabilidad	Media
Impacto	Bajo
Riesgo Total	Bajo
Estado del Riesgo	Abierto
Acciones de mitigación	Usar email u otros medios, y planificar reuniones con antelación
Acciones correctivas	Reajustar plazos del proyecto

Tabla 2.4: Riesgo: Tutor ocupado

La Tabla 2.5 muestra el riesgo de acceso limitado a recursos.

Nombre	Acceso limitado a recursos
Descripción	Dificultad para obtener bibliografía, software o datos necesarios
Categoría	Recursos
Vulnerabilidad	Dependencia de fuentes externas de información o herramientas
Amenaza	Falta de acceso a recursos clave
Probabilidad	Baja
Impacto	Alto
Riesgo Total	Medio
Estado del Riesgo	Abierto
Acciones de mitigación	Buscar alternativas de acceso libre
Acciones correctivas	Solicitar ayuda a la universidad o instituciones relacionadas

Tabla 2.5: Riesgo: Acceso limitado a recursos

La Tabla 2.6 muestra el riesgo por falta de experiencia con el software.

Nombre	Falta de experiencia con el software
Descripción	No se dominan las tecnologías necesarias para el desarrollo
Categoría	Personal
Vulnerabilidad	Proyecto depende de herramientas poco conocidas
Amenaza	Necesidad de aprender durante el desarrollo
Probabilidad	Media
Impacto	Alto
Riesgo Total	Alto
Estado del Riesgo	Identificado
Acciones de mitigación	Elegir tecnologías más conocidas si es posible
Acciones correctivas	Realizar formación específica sobre las tecnologías necesarias

Tabla 2.6: Riesgo: Falta de experiencia con el software

La Tabla 2.7 muestra el riesgo de pérdida del equipo.

Nombre	Pérdida de equipo
Descripción	No se dispone de equipo alternativo en caso de avería
Categoría	Tecnología
Vulnerabilidad	Ausencia de dispositivo de respaldo
Amenaza	Interrupción prolongada del trabajo
Probabilidad	Baja
Impacto	Alto
Riesgo Total	Medio
Estado del Riesgo	Abierto
Acciones de mitigación	Mantener el equipo en buen estado mediante revisiones
Acciones correctivas	Reparar o sustituir el equipo; notificar al tutor la situación

Tabla 2.7: Riesgo: Pérdida de equipo

La Tabla 2.8 muestra el riesgo de documentación ineficiente.

Nombre	Documentación ineficiente
Descripción	Falta de detalle o calidad en la documentación del proyecto
Categoría	Artefactos
Vulnerabilidad	Documentación incompleta o poco clara
Amenaza	Penalización en la evaluación o suspenso del TFG
Probabilidad	Media
Impacto	Medio
Riesgo Total	Medio
Estado del Riesgo	Abierto
Acciones de mitigación	Documentar cada fase con modelos y diagramas adecuados
Acciones correctivas	Solicitar nueva fecha de entrega y mejorar la documentación

Tabla 2.8: Riesgo: Documentación ineficiente

2.6. Plan de Presupuestos

El plan de presupuestos, al estar condicionado por un contexto académico, se ha dividido en dos partes diferenciadas: un **presupuesto ficticio**, que contempla el desarrollo de la aplicación en un entorno profesional real; y un **presupuesto real**, correspondiente al desarrollo como Trabajo de Fin de Grado en colaboración con la empresa AVL Ibérica, sin financiación económica.

2.6.1. Presupuesto Ficticio

Para el cálculo del presupuesto ficticio se han tenido en cuenta diversos parámetros que se detalla a continuación:

- **Mano de obra**: Se estima que el desarrollo ha requerido unas 240 horas de trabajo. Suponiendo una tarifa media de 11€/hora para un desarrollador junior, el coste directo asciende a 2640€. Sumando un 33,4 % correspondiente a la Seguridad Social, el coste total asciende a 3521,76€.
- **Equipo informático**: Se ha utilizado un portátil MSI Thin GF63 12UC-688XES valorado en 980€, con una vida útil de 5 años. Se prorratea su coste a 4 meses, resultando un coste aproximado de 78,4€.
- **Licencias de software**: Se considera la utilización de una licencia de Astah a 12€/mes.
- **Espacio de trabajo**: Se estima un coste de oficina de 160€/mes.

La Tabla 2.9 muestra el presupuesto estimado del proyecto.

Concepto	**Unidad**	**Coste Total (€)**
Sueldo desarrollador junior	240 h x 11€/h	2640,00
Seguridad Social (33,4 %)	—	881,76
Portátil (amortización 4 meses)	980€/60 meses × 4	65,33
Licencia Astah (4 meses)	12€/mes × 4	48,00
Espacio de trabajo (4 meses)	160€/mes × 4	640,00
Total estimado		**4275,09**

Tabla 2.9: Presupuesto ficticio estimado para el desarrollo de la aplicación en entorno profesional.

2.6.2. Presupuesto Real

El desarrollo de este proyecto como TFG no ha requerido una inversión directa por parte del alumno ni de la universidad, más allá de los recursos ya disponibles. Por tanto, el presupuesto real contempla únicamente los siguientes costes:

- **Equipo informático**: El mismo equipo portátil, prorrateado por 4 meses.
- **Consumo eléctrico**: Estimado en 20€/mes durante 4 meses.

La Tabla 2.10 muestra el presupuesto real del proyecto.

Concepto	**Unidad**	**Coste Total (€)**
Portátil (amortización 4 meses)	980€/60 meses × 4	65,33
Consumo eléctrico (4 meses)	20€/mes × 4	80,00
Total estimado		**145,33**

Tabla 2.10: Presupuesto real del desarrollo como Trabajo de Fin de Grado.

2.7. Product Backlog Inicial

El product backlog inicial en vez de haber sido dividido por historias de usuario, hemos optado por agruparlo en épicas, que mas tarde en los *sprints* serán subdivididos en historias de usuario.Como se ve en la tabla 2.11

Épica	**Descripción**
E1	Gestión de diagramas
E2	Edición avanzada del editor de texto
E3	Organización por departamentos
E4	Soporte visual y documentación contextual

Tabla 2.11: Épicas del sistema propuesto

Capítulo 3

Análisis

En este capítulo se recoge el análisis detallado de la aplicación *Project Master*, que constituye la base sobre la que se ha construido el diseño e implementación del sistema.

En primer lugar, se presentan las épicas y las historias de usuario, las cuales describen de manera estructurada las funcionalidades esperadas desde el punto de vista del usuario final. Estas historias permiten identificar los objetivos principales de la herramienta, como la gestión y visualización de diagramas, la edición dinámica de contenido y la organización de los mismos por departamentos.

A continuación, se enumeran los requisitos no funcionales, que establecen criterios de calidad, restricciones técnicas y características adicionales que la aplicación debe cumplir. Entre estos requisitos se incluyen aspectos relacionados con la usabilidad, el rendimiento, la mantenibilidad y la adecuación visual al entorno corporativo.

Posteriormente, se detalla el modelo de dominio, que define las entidades más relevantes del sistema y sus relaciones, proporcionando una representación conceptual de los elementos que intervienen en la solución propuesta.

Por último, se describen los modelos de análisis mediante diagramas de actividades, que ilustran el comportamiento del sistema y la interacción del usuario en los principales flujos de trabajo. Estos diagramas abarcan tanto el modo de visualización de diagramas y gestión de tareas como el modo de edición dinámica de contenidos.

3.1. Historias de usuario

La división en requisitos, realmente no se ha hecho ya que al ser un proyecto con marco de trabajo Scrum se estructura por épicas descompuestas en historias de usuario, es decir los requisitos propuestos por los usuarios con el formato de Çomo usuario me gustaría X",

lo único que se ha tratado como requisitos ha sido los no funcionales , ya que era mas esclarecedor en este formato que en el de historias.

La tabla 3.1 se muestran las historias de usuario relacionas con la épica 1.

ID	**Épica E1: Gestión de diagramas**
HU1	Como usuario, me gustaría poder visualizar los diagramas.
HU2	Como usuario, me gustaría poder hacer y deshacer el progreso de un diagrama.
HU3	Como usuario, me gustaría poder reiniciar el progreso del diagrama.
HU4	Como usuario, me gustaría poder generar mis propios diagramas.
HU5	Como usuario, me gustaría poder renderizar diagramas propios.
HU6	Como usuario, me gustaría tener diagramas prerenderizados por departamento.

Tabla 3.1: Historias de usuario para la épica E1: Gestión de diagramas.

La tabla 3.2 se muestran las historias de usuario relacionas con la épica 2.

ID	**Épica E2: Edición avanzada del editor de texto**
HU7	Como usuario, me gustaría que los cambios en la generación del diagrama se visualicen de manera dinámica.
HU8	Como usuario, me gustaría que el editor de texto tuviese opciones de edición.
HU9	Como usuario, me gustaría que el editor tuviese autocompletado.
HU10	Como usuario, me gustaría que el modo editor me permita importar los diagramas.
HU11	Como usuario, me gustaría que el modo editor me permitiera exportar los diagramas.
HU12	Como usuario, me gustaría que el modo editor me permitiera reiniciar el diagrama.

Tabla 3.2: Historias de usuario para la épica E2: Edición avanzada del editor.

La tabla 3.3 se muestran las historias de usuario relacionas con la épica 3.

ID	**Épica E3: Organización por departamentos**
HU13	Como usuario, me gustaría tener una división en carpetas por departamentos.
HU14	Como usuario, me gustaría tener diagramas prerenderizados por departamento.

Tabla 3.3: Historias de usuario para la épica E3: Organización por departamentos.

La tabla 3.4 se muestran las historias de usuario relacionas con la épica 4.

ID	Épica E4: Soporte visual y documentación contextual
HU15	Como usuario, me gustaría tener una serie de tareas por nodo.
HU16	Como usuario, me gustaría tener una ayuda visual para saber cuál es el nodo actual.
HU17	Como usuario, me gustaría poder acceder a una documentación para cada nodo.

Tabla 3.4: Historias de usuario para la épica E4: Soporte visual y documentación contextual.

3.2. Requisitos no funcionales

En la tabla 3.5 se muestran los requisitos no funcionales recopilados.

ID	Requisito No Funcional	Categoría
RNF-01	La aplicación debe usar los colores corporativos de AVL.	Estilo visual
RNF-02	El modo visualización no debe mostrar efectos de "pop-ping"ni recargas innecesarias.	Rendimiento / UX
RNF-03	Debe obtener al menos un 8 en un test de usabilidad con miembros del departamento.	Usabilidad
RNF-04	El editor debe actualizarse de forma dinámica al modificar el código.	Interacción / Rendimiento
RNF-05	El desarrollo debe hacerse íntegramente con TypeScript.	Tecnología / Mantenibilidad
RNF-06	La aplicación debe incluir un modo oscuro.	Estilo visual / Accesibilidad
RNF-07	Los tiempos de respuesta deben ser menores a 0,5 segundos.	Rendimiento

Tabla 3.5: Requisitos no funcionales del sistema.

3.3. Modelo de dominio

Teniendo en cuenta las historias de usuario se procedió a crear un modelo de dominio inicial, como se ve en la figura 3.1:

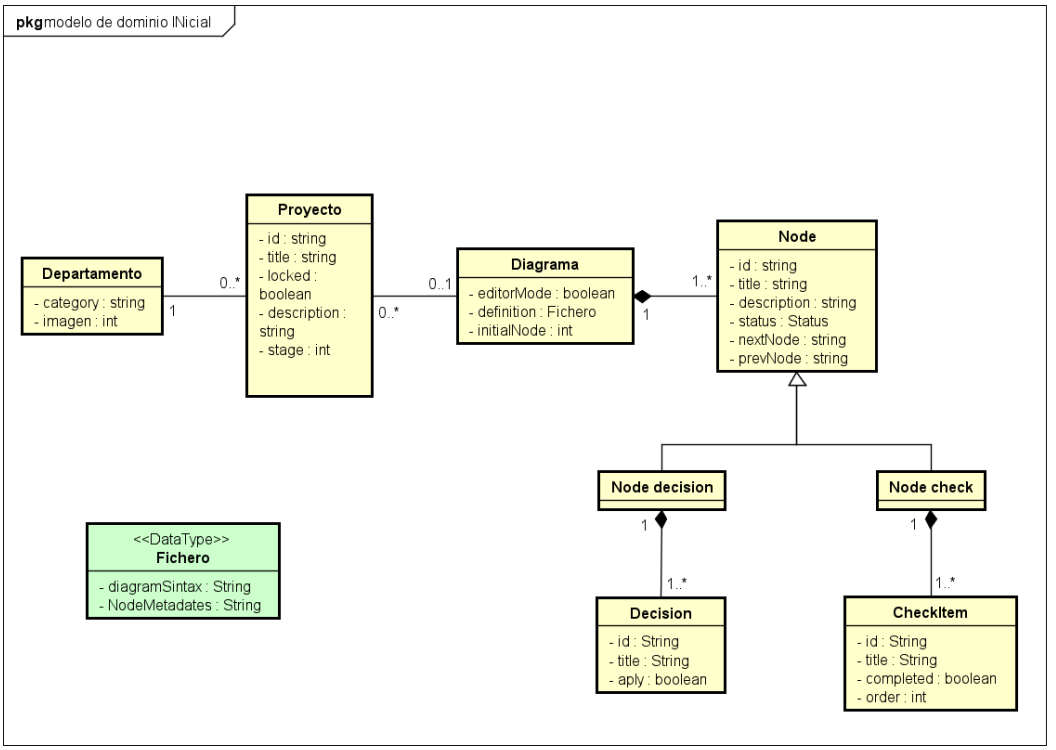

Figura 3.1: Modelo de dominio de Project Master. Se representan las clases principales del sistema y sus relaciones: Diagrama, Nodo,Departamento,ChecklistItem, entre otras. Este modelo sirvió como base conceptual para el diseño estructural de la aplicación.

En este diagrama se pueden identificar ocho clases y un *dataType*. La clase principal de este modelo de dominio es **Diagrama**, la cual es la abstracción del diagrama que se quiere representar, cuenta con un atributo de tipo booleano el cual permite saber si se esta en modo edición o en modo visualización, también cuanta con un atributo tipo *String* con la utilidad de saber cual es el nodo actual que marca el progreso del usuario. Como último atributo se tiene al datatype fichero, que permite guardar la sintaxis Mermaid y la *String* de metadatos de los nodos. La siguiente clase de análisis mas importante es Node,permite almacenar todos los datos relevantes de los nodos, siendo entre otros el titulo, la descripción y el estado. Esta clase genera una herencia los dos tipos de nodos que hay en los diagramas de flujo Mermaid, siendo estos los nodosCheck (nodos normales modificados para poder tener una lista de tareas) y nodos decisión, los cuales permiten la toma de decisiones para poder continuar por una ramificación u otra del diagrama de flujo en cuestión.Cave resaltar que los nodos también guardan como datos cuales son los nodos padre y cuales son sus nodos hijos. Luego las dos clases que son mas auxiliares que son departamento y proyecto, siendo clases que ayudan al archivado de los diagramas de una manera cómoda y orgánica. Permitiendo a un departamento tener distintos proyectos asociados con sus respectivo diagrama.

3.4. Modelo de Análisis

En esta sección se presentan dos diagramas de actividades que reflejan el comportamiento del sistema y la interacción del usuario durante el uso de la aplicación *Project Master*. Estos diagramas tienen como objetivo cubrir el mayor número de historias de usuario definidas en el análisis previo, proporcionando una visión general del flujo de trabajo sin entrar en detalles técnicos de implementación.

La figura 3.2 muestra el diagrama de actividad que sigue la historia de usuario de completar un diagrama.

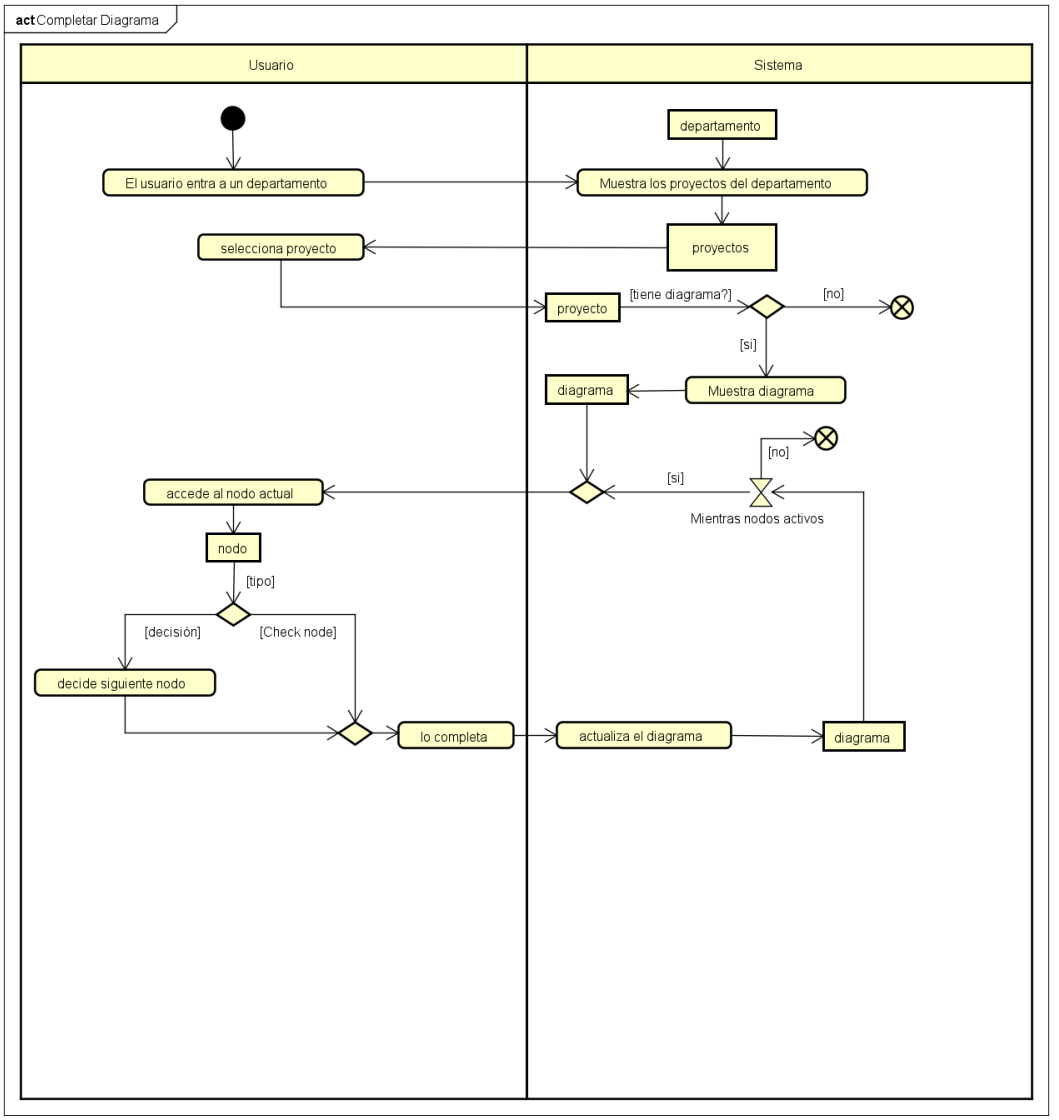

Figura 3.2: Modo visualización: el usuario completa subtareas asociadas a nodos y avanza por el flujo del diagrama.

En el primer diagrama (Figura 3.2), se muestra cómo un usuario interactúa con el sistema al completar un diagrama previamente generado. Este flujo representa una de las funcionalidades principales de la aplicación: la visualización activa de procesos a través de nodos interactivos. El usuario accede a un diagrama perteneciente a un departamento determinado y comienza a completar nodos tipo *check*, cada uno con subtareas asociadas.

El flujo es mayoritariamente lineal: una vez completadas todas las tareas de un nodo,

este se marca como finalizado y el sistema activa automáticamente el siguiente nodo en el recorrido. En el caso de nodos de tipo *decisión*, el usuario debe seleccionar una de las ramas disponibles para continuar. Este comportamiento refleja la lógica de avance y validación de flujos de trabajo internos dentro de un entorno de ingeniería, aportando trazabilidad y seguimiento del progreso. Además, se utilizan colores e iconos para representar visualmente el estado de cada nodo (pendiente, actual, completado), lo que mejora la experiencia de usuario y la comprensión del flujo.

La figura 3.3 muestra la carga y edición de un diagrama.

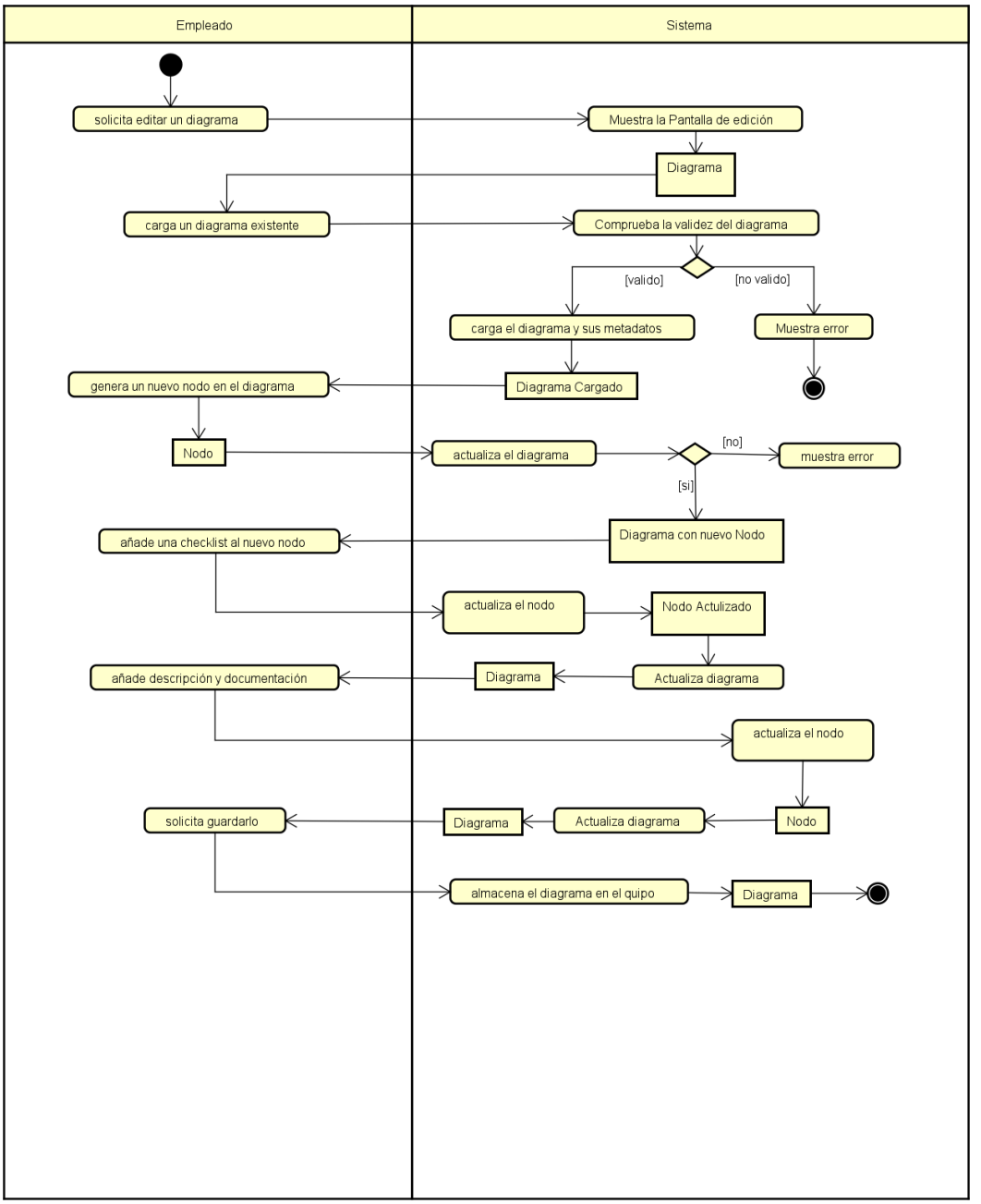

Figura 3.3: Interfaz del modo edición. El usuario puede escribir código Mermaid y visualizar el resultado dinámicamente.

El segundo diagrama (Figura 3.3) representa un flujo de uso más complejo, correspondien-

te al modo edición de la aplicación. Aquí el usuario comienza cargando un archivo existente o creando uno nuevo. A continuación, puede editar directamente el código Mermaid utilizando el editor embebido llamado Monaco Editor, añadir nuevos nodos, configurar subtareas, modificar descripciones o actualizar metadatos del diagrama.

Una vez realizado el trabajo de edición, el usuario puede guardar el contenido, ya sea localmente o exportarlo en un formato estructurado compatible con el sistema. Este diagrama abarca funcionalidades como la edición dinámica del grafo, la validación visual del resultado, la gestión de archivos Mermaid y la sincronización entre el código fuente y su representación gráfica. Este flujo también contempla la posibilidad de rehacer y deshacer acciones, reiniciar el progreso y centrar el grafo, lo que proporciona mayor control y flexibilidad en entornos colaborativos o de documentación técnica.

Ambos diagramas ponen de manifiesto la importancia de ofrecer una interfaz intuitiva y funcionalidades alineadas con las necesidades reales del usuario, especialmente en entornos donde la visualización de flujos y la trazabilidad de decisiones son clave.

Capítulo 4

Tecnologías utilizadas

En este capítulo se detallaran en profundidad las distintas tecnologías usadas para el desarrollo, explicandose en este apartado las siguientes tecnologías: Git, GitHub, React, JavaScript, viteJS, Monaco Editor, Mermaid , Node.js, ChatGPT, Latex, TypeScript.

4.1. Git

Git [12] es un sistema de control de versiones, distribuido y popular en el ámbito de la ingeniería de software. Su creador fue Linus Torvalds (tan bien es el creador del sistema operativo linux) en 2005 con el objetivo de gestionar el desarrollo del núcleo de Linux, luego de una ruptura con el sistema propietario BitKeeper.

A diferencia de otros sistemas de control de versiones más antiguos, Git es distribuido, lo que significa que cada desarrollador posee una copia completa del repositorio, incluidas todas sus versiones y ramas. Esta arquitectura permite un trabajo más seguro y flexible, incluso sin conexión a Internet. Sus principales características son:

- Sistema distribuido completo: cada copia del repositorio es completa e independiente del resto.
- Integridad de datos: esto es debido al uso de funciones hash SHA-1 que permiten identificar cada cambio de manera única e inequívoca.
- Velocidad en las operaciones remotas como commit ,push o merge.
- Ramas ligeras que permiten un cambio entre las mismas agil y sencillo
- Facil seguimiento debido a los metadatos y el sistema de etiquetas en los commits

En el caso particular de este trabajo fin de grado Git ha sido la herramienta de control de versiones utilizada, garantizando en todo momento poder tener una copia de seguridad distribuida, lo que permitio mejorar la trazabilidad del proyecto debido al sistema de commits.

4.2. GitHub

GitHub [1] por su parte es una plataforma web lanzada en el 2008, la cual ofrece alojamineto a repositorios Git, combinado con herramientas colaborativas y un formato de red social. En la actualidad GitHub es una patente de Microsoft, lo que ha ampliado su alcance e integración con distintas tecnologías de esta misma empresa. Hay que destacar que GitHub y Git no son lo mismo. GitHub es una plataforma basada en Git para brindar una experiencia enriquecida del sistema Git.

Las funcionalidades clave que han llevado a su exito a GitHub son:

- Division de repositorios en publicos y privados: permitiendo la privacidad o compartir código completo de manera libre y sencilla.
- *Issues y project board*s: sistema de gestión de tareas dentro del mismo entorno del repositorio.
- Colaboración global: permite que miles de desarrolladores trabajen de forma coordinada en distintos proyectos.

4.3. React

React es un *frameWork* open source basado en JavaScript [7], especializado en el desarrollo web, mas concretamente en la construcción de inerfaces de usuario a través de un sistema de componentes. Fue desarrollado por la compañía Meta en el 2013 con el objetivo de mejorar el rendimiento y escalabilidad de páginas web complejas, especialmente las que requieren de una actualización dinámica de la interfaz gráfica sin necesidad de recargar la página. Desde su lanzamiento fue una de las tecnologías *frontend* más usada en el ámbito del desarrollo software, llegando en la actualidad a ser la más utilizada. Las principales características de React son

- Programación declarativa: el desarrollador define qué aspecto debe tener la interfaz en un determinado estado, y React se encargará de reflejarlo.
- Arquitectura basada en componentes: La interfaz en react se basa en una estructura de componentes reutilizables, donde cada uno representa una unidad funcional independinte del resto.Esto ayuda a la mantenibilidad, legilibilidad y escalabilidad del código.
- Uso de un Document Object Model o más conocido por sus siglas DOM virtual: React utiliza una representación en memoria del DOM real, denominada *Virtual* DOM, que

permite identificar los cambios mínimos necesarios y aplicarlos de forma eficiente, mejorando así el rendimiento.Ya que no tendría que recompilar el DOM real entero , solo los cambios mínimos efectuados.

Uno de los puntos más interesantes de React es su sintaxis, ya que a diferencia de otras tecnologías *frontend* como angular esta tiene una sintaxis JSX. Esto quiere decir que no mantiene una sintaxis JavaScript habitual, si no que la combina con sintaxis HTML. Esto aunque no sea obligatorio es buena práctica ya que se asemeja más a la estructura DOM real.

Otra de las funcionalidades más importantes de React son los ganchos o más conocidos por su nombre en ingles *hooks*. En la versión 16.8, React introdujo los *hooks*, una forma de incorporar estado y funcionalidades de ciclo de vida a los componentes funcionales. Entre los más utilizados se encuentran:

- *useState*: para gestionar el estado local de un componente.
- *useEffect*: para gestionar efectos secundarios como peticiones HTTP o suscripciones.

Estos *hooks* han reemplazado en gran parte a las clases tradicionales, simplificando el código y favoreciendo patrones más declarativos y funcionales. Incluso obligando a otros *frameworks* a adoptarlos.

El motivo principal del uso de React fue que es una página web que depende de una actualización dinámica constante , ya que al ser un visualizador y editor dinámico era imprescindible este aspecto, unido al sistema de componentes que mejoraba la claridad y legibilidad del código haciendo más fácil su escalamiento. Se llegó a la conclusión de que era el mejor *frameWork* en el cual se podía desarrollar Project Master. Cabe destacar que el uso de *hooks* facilitó mucho el desarrollo de funcionalidades complejas como puede ser el control de cámara o actualización dinámica de la lista de nodos y que al usar el dom Virtual la parte de testeo y arreglo de bugs fue menos tediosa debido a la rapidez a la hora de recompilar los cambios.

4.4. ViteJS

En el ecosistema del desarrollo *frontend* moderno, las herramientas de construcción (*build tools*) desempeñan un papel esencial en la optimización del rendimiento y la experiencia de desarrollo. Durante muchos años, soluciones como Webpack, Parcel o Rollup han sido las más utilizadas, debido a su flexibilidad y potencia. No obstante, estas herramientas presentan ciertas limitaciones en términos de velocidad y complejidad de configuración, especialmente en proyectos de gran tamaño. Ante esta problemática, surge **ViteJS**[9], una herramienta de nueva generación que propone un enfoque radicalmente distinto.

ViteJS es una herramienta de desarrollo creada por *Evan You* que permite iniciar rápidamente proyectos *frontend* con tiempos de arranque extremadamente bajos y una experiencia

de desarrollo ágil. Su nombre proviene del francés *vite*, que significa rápido, lo que refleja su principal objetivo: acelerar el proceso de desarrollo web mediante un sistema basado en módulos nativos de JavaScript y una construcción dividida entre desarrollo y producción .

Durante el desarrollo, Vite utiliza los módulos nativos de JavaScript (ESM) soportados por los navegadores modernos para servir los archivos directamente al navegador sin necesidad de realizar un *bundle* completo. Esto contrasta con herramientas como Webpack, que deben empaquetar toda la aplicación incluso para realizar pequeños cambios.

Cuando un archivo es solicitado, Vite lo transpila (por ejemplo, usando *esbuild* para convertir código TypeScript o JSX a JavaScript) y lo sirve al navegador. Gracias a este enfoque, los tiempos de arranque son prácticamente instantáneos.

Para entornos de producción, Vite utiliza *Rollup*, una herramienta madura y optimizada para la creación de paquetes eficientes. Esto permite generar archivos minificados y segmentados, adecuados para entornos reales. Vite separa de forma clara los procesos de desarrollo y producción, permitiendo optimizar cada uno de manera independiente.

Algunas de las características a destacar de vite son:

- **Inicio instantáneo del servidor**: gracias al uso de ES Modules, el arranque es casi inmediato.
- **Actualización rápida de módulos (HMR)**: permite una retroalimentación muy veloz.
- **Compatibilidad con TypeScript, JSX y otros preprocesadores**: mediante el uso de *esbuild*, Vite transpila con gran rapidez.
- **Integración con frameworks modernos**: Vue, React, Svelte, Preact, entre otros.
- **Sistema de plugins basado en Rollup**: reutiliza el ecosistema existente.
- **Configuración sencilla y moderna**: apta para desarrolladores principiantes y avanzados.

Vite también presenta ciertas limitaciones. Su enfoque basado en tecnologías modernas del navegador implica que su compatibilidad con navegadores antiguos no está garantizada sin configuraciones adicionales. Además, aunque su popularidad está en aumento, herramientas como Webpack siguen siendo preferidas en entornos corporativos debido a su madurez y adopción masiva.

Vite representa una evolución importante en la construcción de aplicaciones web modernas. Su arquitectura basada en ES Modules para desarrollo y Rollup para producción permite una experiencia de desarrollo más fluida y veloz. A medida que el ecosistema JavaScript avanza, Vite se posiciona como una herramienta clave para proyectos web eficientes y modernos. Por lo que finalmente se opto por ella para poder lanzar la aplicación Project mMster.

4.5. JavaScript

JavaScript [2] es un lenguaje de programación de alto nivel, interpretado y orientado a objetos, diseñado originalmente para ser ejecutado en el navegador y permitir la creación de sitios web interactivos. Desde su creación en 1995 por Brendan Eich, JavaScript en la actualidad es la tecnología web principal junto con HTML y CSS

Actualmente, JavaScript no solo se utiliza en el *frontend*, sino también en el *backend* gracias a entornos como Node.js. Su versatilidad y la amplia cantidad de librerías y frameworks hacen de JavaScript una herramienta fundamental en el desarrollo web.

JavaScript es un lenguaje:

- **Multiparadigma**: admite programación imperativa, funcional y orientada a objetos.
- **Débilmente tipado**: las variables no tienen tipos estrictos, lo que facilita la escritura de código, aunque también puede generar errores sutiles.
- **Interpretado**: el código se ejecuta directamente en el navegador o en el motor correspondiente sin necesidad de compilación previa.
- **Basado en eventos**: especialmente útil en entornos de interfaz gráfica de usuario.
- **Asíncrono**: incorpora mecanismos como promesas, `async/await` y callbacks para el manejo de tareas concurrentes.

Aunque inicialmente fue diseñado para ejecutarse en navegadores (con motores como V8 de Google Chrome o SpiderMonkey de Firefox), hoy en día JavaScript también se ejecuta en servidores y otros entornos. Entre los más importantes destacan:

- **Node.js**: permite ejecutar JavaScript en el servidor, habilitando el desarrollo de aplicaciones completas usando un solo lenguaje.
- **Deno**: una alternativa moderna a Node.js, desarrollada por el propio creador de este último, con un enfoque más seguro y soporte nativo para TypeScript.

JavaScript al ser el lenguaje más utilizado cuenta con una extensa cantidad de bibliotecas y *frameworks*. Algunas de las librerías y *frameworks* más destacados incluyen:

- **React.js**: biblioteca para construir interfaces de usuario basada en componentes, desarrollada por Facebook.
- **Vue.js**:*frameworks* progresivo para construir interfaces, con una curva de aprendizaje amigable.
- **Angular**: *frameworks* completo para aplicaciones *Single Page Application* o más conocido como SPA, respaldado por Google.

- **Express.js**: framework minimalista para crear servidores y APIs con Node.js.

Las principales ventajas del uso de JavaScript en el desarrollo de aplicaciones web son su amplio soporte en navegadores y plataformas, así como un ecosistema muy activo y con abundantes recursos que facilitan el aprendizaje y la integración de nuevas tecnologías. También su flexibilidad permite desarrollar tanto el *frontend* como el *backend* utilizándose como único lenguaje, lo que simplifica considerablemente el tiempo de vida de los proyectos. No obstante, también presenta ciertas desventajas, entre las que destacan la falta de tipado, que puede provocar errores difíciles de detectar en tiempo de ejecución, y la rápida evolución del lenguaje, que en ocasiones genera problemas de compatibilidad con versiones anteriores o dependencias de bibliotecas externas.

JavaScript ha pasado de ser un lenguaje limitado a uno de los pilares del desarrollo de software moderno. Su continua evolución, junto con el soporte de los navegadores y el auge de tecnologías como Node.js, le ha permitido posicionarse como una herramienta esencial en el desarrollo web *full stack*. Por lo que resultó ser la opción mas competente para el desarrollo de Project Master, descartando la idea de desarrollarlo con Python y el frameworks de Django.

4.6. Node.js

Node.js [6] es un entorno de ejecución para JavaScript. Fue desarrollado por Ryan Dahl en 2009 con la finalidad de poder ejecutar JavaScript fuera del navegador, especialmente en el servidor. Desde entonces, ha ganado una gran popularidaddebido a su arquitectura basada en eventos y su modelo de entrada/salida no bloqueante.

Node.js permite a los desarrolladores utilizar JavaScript para el *frontend* como para el *backend*, lo que facilita la creación de aplicaciones completas utilizando un único lenguaje. Además, su diseño orientado a eventos y su eficiencia en la gestión de conexiones concurrentes lo convierten en una opción ideal para aplicaciones en tiempo real.

Las características más destacadas de Node.js son:

- **Eficiencia y escalabilidad**: gracias a su modelo de ejecución basado en un solo hilo y el uso de un bucle de eventos o más conocido como *event loop*, puede manejar múltiples conexiones concurrentes de forma eficiente.
- **Entrada/salida no bloqueante**: todas las operaciones de entrada salida o E/S se ejecutan de manera asíncrona, lo que mejora el rendimiento en aplicaciones que realizan muchas operaciones con archivos o redes.
- **Gran ecosistema de módulos**: Node.js incluye un gestor de paquetes llamado npm (Node Package Manager), que permite acceder a una vasta colección de módulos reutilizables.
- **Compatibilidad con JavaScript moderno**: permite utilizar las características más recientes del lenguaje gracias al motor V8.

Node.js se utiliza ampliamente en diversos contextos de desarrollo, entre ellos:

- **APIs RESTful y GraphQL**: para gestionar peticiones y respuestas de clientes de forma eficiente.
- **Aplicaciones en tiempo real**: como chats, notificaciones *push* o videojuegos multijugador.
- **Herramientas de desarrollo**: muchas herramientas modernas como Webpack, ViteJS o ESLint están desarrolladas en Node.js.
- **Microservicios**: gracias a su ligereza y modularidad, es común usar Node.js para construir arquitecturas basadas en servicios independientes.

Node.js presenta diversas ventajas que lo convierten en una tecnología ampliamente adoptada en el desarrollo web. Permite unificar el desarrollo frontend y backend utilizando JavaScript, lo que facilita la coordinación de proyectos y equipos. Ofrece un excelente rendimiento en aplicaciones que requieren gestionar muchas conexiones simultáneas, gracias a su arquitectura orientada a eventos. Además, cuenta con una amplia comunidad y un soporte activo que proporcionan recursos, así como una rápida instalación y despliegue que aceleran el inicio de nuevos proyectos. Sin embargo, también tiene una serie de desventajas importantes:como no ser una opción ideal para tareas intensivas en CPU debido a su modelo de ejecución de un solo hilo, o el el manejo operaciones asíncronas puede resultar complejo para desarrolladores sin experiencia y su fuerte dependencia de paquetes de terceros puede introducir riesgos de seguridad o de estabilidad.

Como conclusión Node.js ha revolucionado el desarrollo web al extender el uso de JavaScript al *backend*. Su arquitectura ligera, su eficiencia y variedad de módulos lo han convertido en una herramienta fundamental para el desarrollo web, especialmente en aplicaciones en tiempo real y arquitecturas distribuidas.

Esto unido a lo explicado con anterioridad, y la posibilidad de desarrollar enteramente Project Master con javaScript fue el motivo por el cual se opto por el uso de Node.js como entorno de ejecución.

4.7. Mermaid

Mermaid [4]es un lenguaje de marcado ligero, basado en texto plano, su concepción fue con el objetivo de facilitar la creación de diagramas de manera sencilla y programática. Su finalidad principal era facilitar la generación de diagramas en archivos de texto o documentos markdown, permitiendo integrar diagramas en documentación sin ayuda de herramientas externas. El auge de su popularidad viene dado por su facilidad de uso e integración con sistemas de documentación como Markdown, su compatibilidad con plataformas como GitHub, GitLab y con distintos lenguajes de programación y *frameworks* como son JavaScript y react. Su filosofía se basa en la automatización de diagramas a partir de texto plano, lo cual mejora la mantenibilidad y reproducibilidad de la documentación técnica.

Mermaid permite generar una amplia variedad de diagramas, entre los que se destacan:

- **Diagramas de flujo (*flowcharts*)**: Representan procesos o flujos de trabajo mediante nodos y conexiones direccionales(este es en el que nos basaremos para el desarrollo de project master).
- **Diagramas de Gantt**: Utilizados para representar cronogramas de proyectos y gestión del tiempo.
- **Diagramas de secuencia (*sequence diagrams*)**: Representan interacciones entre entidades en el tiempo, especialmente en el análisis de sistemas.
- **Diagramas de clases UML**: Permiten representar la estructura de clases y relaciones en sistemas orientados a objetos.
- **Diagramas de estado (*state diagrams*)**: Muestran los distintos estados de un sistema y las transiciones entre ellos.
- **Diagramas de entidad-relación (ERD)**: Utilizados para modelar bases de datos relacionales.

Todos estos diagramas se definen mediante bloques de código que siguen una sintaxis específica. A continuación en la figura 4.1 se muestra la definición de un diagrama de flujo sencillo como ejemplo.

```
graph TD
A[Inicio] --> B[Proceso 1]
B --> C{¿Condición?}
C -- Sí --> D[Proceso 2]
C -- No --> E[Fin]
```

Figura 4.1: Ejemplo de definición de diagrama Mermaid en formato texto

Este bloque será interpretado por el motor de Mermaid para generar el diagrama correspondiente en tiempo de renderizado, ya sea en un navegador o una plataforma compatible.

Entre las principales ventajas del uso de Mermaid se encuentran:

- **Simplicidad**: La sintaxis es clara, accesible y fácil de aprender.
- **Portabilidad**: Al estar basada en texto plano, puede incluirse fácilmente en sistemas de control de versiones como Git.
- **Automatización**: Los diagramas se generan automáticamente sin necesidad de herramientas gráficas.

- **Actualización sencilla**: Modificar un diagrama es tan simple como editar unas pocas líneas de texto.

No obstante, también presenta ciertas limitaciones:

- **Estética limitada**: Aunque configurable, el estilo visual de los diagramas es relativamente básico en comparación con herramientas gráficas profesionales.
- **Curva de aprendizaje para diagramas complejos**: Diagramas muy detallados pueden resultar difíciles de gestionar únicamente mediante texto.
- **Dependencia de renderizado**: Requiere un entorno compatible que interprete correctamente el código Mermaid, aunque este error no es importante ya que para nuestro entorno si es compatible.

Mermaid representa una solución eficiente y moderna para la generación de diagramas dinámicos en entornos técnicos. Su enfoque centrado en el texto plano lo hace simple y acesible, unido a su compatibilidad con React era la mejor manera de representar y editar diagramas de flujo para Project Master ya que permite exportar los diagramas en formato SVG, mejorando así la calidad de visualización y la integración con la propia aplicación.

4.8. Monaco Editor

Una de las tecnologías claves en el desarrollo del modo editor de Project Master fue Monaco Editor [5]. Monaco editor es un editor de texto enriquecido basado en tecnologías web, desarrollado por Microsoft. Este editor es el núcleo funcional de Visual Studio Code, está diseñado especificamente para tener una integración total en aplicaciones web, ofreciendo una experiencia de edición muy similar a la de entornos de desarrollo de escritorio.

Monaco Editor destaca por una serie de herramientas, que lo transforman en una herramienta idónea para la edición de código en navegador, entre las cuales destacan:

- **Autocompletado inteligente:** sugiere funciones, variables y estructuras sintácticas conforme al lenguaje activo.
- **Resaltado de sintaxis:** compatible con una amplia variedad de lenguajes de programación, incluyendo JavaScript, TypeScript, JSON, Python, Mermaid entre otros.
- **Plegado de código:** permite ocultar secciones del código para facilitar la navegación y organización.
- **Multicursor:** edición simultánea en múltiples ubicaciones del documento.
- **Soporte para análisis de errores:** gracias a su arquitectura, permite integrar linters o analizadores estáticos para ofrecer retroalimentación inmediata.

- **Alta personalización:** desde el esquema de colores hasta la configuración de accesibilidad, Monaco Editor puede adaptarse fácilmente al diseño general de la aplicación.

Aunque varias de estas funcionalidades no las hemos implementado en project master creo que era importante destacarlas.

Monaco Editor esta desarrollado en TypeScript,el cual a su vez esta desarrollado en javaScript y se ejecuta sobre el navegador utilizando tecnologías estándar como HTML5, CSS3 y JavaScript. Internamente, hace uso de un modelo basado en *Web Workers* que permite ejecutar tareas complejas, como el análisis de sintaxis o la validación de código, en paralelo al hilo principal de la interfaz de usuario, lo que garantiza un rendimiento óptimo incluso en operaciones pesadas.

La utilización de Monaco Editor ha aportado una solución robusta y moderna para la edición de código embebida en el navegador. Su integración ha sido fundamental para permitir una edición dinámica del diagrama en tiempo real, lo que ha incrementado significativamente la interactividad y utilidad de la aplicación desarrollada en este Trabajo de Fin de Grado.

4.9. TypeScript

Realmente el desarrollo de Project Master no ha sido con JavaScript, si no con TypeScript [8] el cual es un lenguaje de programación basado enteramente en JavaScript, pero añadiendo tipado estático y otras series de caracteríscticas de lenguajes más modernos.Su objetivo principal es mejorar la escalabilidad y robustez respecto a las aplicaciones web desarrolladas en JavaScript

Aunque javaScript es el lenguaje principal y un estanadar en el desarrollo web, su tipado dinámico suele dar lugar a errores y bugs muy dificiles de detectar. Typescript por su parte soluciona este problema incluyendo un tipado opcional, permitiendo detectar errores en tiempo de compilación, a su vez facilitando el uso de herramientas de desarrollo como los linters que son programas capaces de analizar automáticamente el código fuente para identificar errores potenciales.

Las principales ventajas de TypeScript frente a JavaScript incluyen:

- **Tipado estático:** permite declarar tipos para variables, funciones, parámetros y estructuras, reduciendo la probabilidad de errores en tiempo de ejecución.
- **Interfaces y tipos personalizados:** facilita la definición de contratos de datos y estructuras complejas.
- **Compilación anticipada:** el código TypeScript se transpila a JavaScript estándar, permitiendo detectar errores antes de su ejecución.
- **Compatibilidad total:** todo código JavaScript válido es también válido en TypeScript, lo cual permite una migración progresiva.

- **Integración con editores modernos:** herramientas como Visual Studio Code o Monaco Editor ofrecen una experiencia mejorada al trabajar con TypeScript, gracias a su análisis semántico en tiempo real.

Realmente el codigo TypeScript no se ejecuta directamente en el navegador, si no que se transpila a Javascript mediante el compilador propio de TypeScript (TSC). Esto como ya se menciono antes permite una compatibilidad total entre TypeScript y JavaScript.

El sistema de tipos de TypeScript es estructural y no nominal, lo que significa que se basa en la forma de los datos en lugar de en sus nombres para verificar la compatibilidad. Esto proporciona una gran flexibilidad a la hora de trabajar con diferentes estructuras y librerías.

La elección de TypeScript para el desarrollo de project master, como ya se ha podido intuir por lo expuesto anterioriormente persigue garantizar la fiabilidad y claridad del código, permitiendo detectar errores de forma más sencilla debido al sistema de tipado que este mismo aporta.

4.10. LaTeX

LaTeX [3] es un sistema de composición y creacción de documentos, orientado especialmente a la creación de textos científicos, técnicos y académicos. Fue desarrollado inicialmente por Leslie Lamport como una colección de macros sobre el sistema de composición tipográfica TeX, creado por Donald Knuth. En la actualidad LaTeX se ha convertido en la herramienta principal para la redacción de artículos científicos, tesis doctorales, libros técnicos y otros documentos que requieren de una estructura y presentación del contenido específica y acotado.

Las características principales que han posicionado a LaTeX como la herramienta principal de redacción de textos científicos son:

- **Calidad tipográfica:** produce documentos con una presentación profesional, especialmente en lo referente a la composición de fórmulas matemáticas y manejo de tablas.
- **Separación entre contenido y forma:** el autor se centra en la estructura lógica del documento (secciones, figuras, tablas, referencias), mientras que el estilo visual se gestiona a través de plantillas y clases de documento.
- **Gestión automática de referencias:** permite la inclusión de bibliografía y citas cruzadas mediante herramientas como BibTeX o biber.
- **Gran soporte para matemáticas:** es considerado el estándar más potente y flexible para la escritura de expresiones matemáticas complejas.
- **Extensibilidad:** dispone de miles de paquetes (*packages*) que permiten incorporar funcionalidades avanzadas como generación de índices, gestión de glosarios, inclusión de gráficos vectoriales, diagramas, pseudocódigo, y más, un ejemplo de esto es que tiene integración con Mermaid.

- **Compatibilidad multiplataforma:** puede ejecutarse en cualquier sistema operativo (Linux, Windows, macOS) mediante distribuciones como TeX Live o MiKTeX, tambien cuenta con distintos editores online como pueden ser Overleaf o papeeria.

A diferencia de otros editores de texto convencionales como puede ser word, LaTeX se basa en un modelo de compilación. Es decir el autor escribe el contenido en distintos archivos .tex, usando una sintaxis que podría considerarse casi programática. Estos archivos luego serán procesados por un compilador y generara como salida un documento en formato PDF. Durante este proceso de compilación, LaTeX analiza la estructura del documento, resuelve referencias cruzadas, compone ecuaciones, formatea tablas, y construye los elementos auxiliares (índice, bibliografía, figuras, etc.).

Esta memoria de trabajo fin de grado ha sido redactada en su totalidad usando LaTeX, lo cual ha permitido estructurar de manera clara y coherente los contenidos.

4.11. ChatGPT

ChatGPT [18] es un modelo de lenguaje desarrollado por OpenAI, basado en la arquitectura *Transformer*, que ha sido entrenado con grandes volúmenes de texto para comprender y generar lenguaje natural de forma coherente y contextualizada. Esta tecnología forma parte de la familia de modelos GPT *Generative Pre-trained Transformer*, y se ha consolidado como una herramienta versátil para tareas de asistencia en redacción, programación, generación de documentación técnica, entre otras aplicaciones.

En el contexto del presente Trabajo de Fin de Grado, ChatGPT ha sido empleado como herramienta de apoyo fundamental durante el desarrollo de la aplicación web y la redacción de esta memoria.

Las principales características que han motivado el uso de ChatGPT en este proyecto son:

- **Asistencia en programación:** permite obtener sugerencias de código, resolver dudas sobre errores, y generar estructuras funcionales en diversos lenguajes, como JavaScript, HTML o React, facilitando el flujo de trabajo durante el desarrollo de la aplicación.
- **Redacción técnica:** ayuda a redactar explicaciones claras y estructuradas sobre tecnologías utilizadas, conceptos teóricos y documentación técnica, garantizando precisión y coherencia en los textos a través de una explicación inicial.
- **Generación de contenido en LaTeX:** ha sido especialmente útil para formatear y estructurar secciones del documento en lenguaje LaTeX, así como para crear tablas, ecuaciones y otros elementos complejos de forma rápida y precisa.
- **Interacción contextual:** su capacidad para mantener el contexto de una conversación o una serie de instrucciones ha permitido una interacción fluida y progresiva, adaptándose a las necesidades cambiantes durante el desarrollo del proyecto.

- **Multidisciplinariedad:** ofrece soporte tanto en temas técnicos como lingüísticos, facilitando la revisión gramatical y la formulación adecuada de conceptos.

La utilización de ChatGPT ha representado una ayuda a lo largo de este Trabajo de Fin de Grado. Su versatilidad y capacidad para generar contenido técnico y académico han contribuido tanto al desarrollo de la aplicación web como a la redacción clara y estructurada de la memoria. En particular, ha sido una herramienta clave para la creación de tablas y estructuras complejas en LaTeX, acelerando el proceso de documentación y mejorando la calidad del resultado final.

Capítulo 5

Diseño

5.1. Decisiones de diseño

A lo largo de este proyecto se han tomado una serie de decisiones que han permitido el correcto desarrollo de Project Master, en esta sección se mostrarán las decisiones de diseño más importantes y como se alinean con las historias de usuario [20] .

- Uso de React: al tener como requisito el uso de TypeScript propuesto por la empresa, el *framework* más popular y completo para aplicaciones web dinámicas es React.
- Uso de Node.js: al hacer la aplicación completamente con TypeScript se necesitaba un entorno de ejecución de código JavaScript y se optó por Node.js al ser el más completo y popular para el desarrollo web.
- Uso de Vite.js: para permitir un desarrollo más fácil debido a la simpleza y eficiencia desplegando aplicaciones web.
- para clarificar el código se ha optado por los estilos arquitectónicos más comunes en el desarrollo con el framework React (desarrollo basado en componentes).

5.2. Arquitectura Lógica

Introducción

La arquitectura lógica del sistema se ha estructurado siguiendo una arquitectura basada en componentes. La arquitectura basada en componentes es un paradigma de diseño software centrado en la descomposición del sistema en unidades modulares, llamadas *componentes*,

que encapsulan funcionalidad y estado de tal manera que son unidades independientes y completas. Este enfoque promueve la reutilización, mantenibilidad y escalabilidad del producto.

En el contexto de este trabajo, se ha optado por aplicar una arquitectura basada en componentes utilizando el *framework* React, dada su orientación natural hacia este modelo y su amplia adopción en la industria.

La arquitectura basada en componentes se rige por una serie de principios clave:

- **Encapsulamiento**: Cada componente representa una unidad funcional independiente que mantiene su propio estado interno y lógica.
- **Reutilización**: Los componentes están diseñados para ser reutilizables en diferentes partes de la aplicación, reduciendo la duplicación de código.
- **Composición**: Componentes más complejos se construyen a partir de componentes más simples, siguiendo un enfoque jerárquico.
- **Separación de responsabilidades**: Se distingue entre componentes de presentación (responsables de mostrar datos) y componentes contenedores (encargados de la lógica de negocio o interacción con fuentes de datos).

En la práctica, la arquitectura se organiza en varios tipos de componentes, cada uno con una función específica:

- **Componentes de presentación** (*Presentational components*): Se encargan exclusivamente de la visualización de datos y no contienen lógica de negocio. Reciben los datos a través de propiedades (*props*) y emiten eventos hacia el componente superior.
- **Componentes contenedores** (*Container components*): Gestionan la lógica de negocio, el estado de la aplicación y las interacciones con servicios externos (por ejemplo, llamadas a APIs). Se encargan de preparar los datos y pasarlos a los componentes de presentación.
- **Componentes funcionales y de clase**: Aunque React permite ambas formas, actualmente se favorece el uso de componentes funcionales con *hooks* para manejar el estado y los efectos secundarios, mejorando la legibilidad y simplicidad del código.
- **Componentes de orden superior (HOCs) y *render props***: Son patrones avanzados que permiten reutilizar lógica entre múltiples componentes. Su uso depende de las necesidades de abstracción del proyecto.

La elección de esta arquitectura proporciona una serie de beneficios técnicos alineados con los objetivos del diseño de software de calidad:

- **Modularidad**: El sistema se construye como un conjunto de piezas intercambiables, lo cual facilita la localización y corrección de errores.

- **Escalabilidad**: Es posible extender el sistema incorporando nuevos componentes sin alterar significativamente los existentes.
- **Facilidad de prueba (testabilidad)**: Cada componente puede ser probado de forma aislada, lo cual simplifica la aplicación de pruebas unitarias.
- **Facilidad de mantenimiento**: Al aislar la funcionalidad en componentes pequeños y específicos, se facilita la comprensión y evolución del código.

5.3. Explicación detallada de componentes

La arquitectura lógica del sistema se muestra el diagrama de componentes de la Figura 5.1.

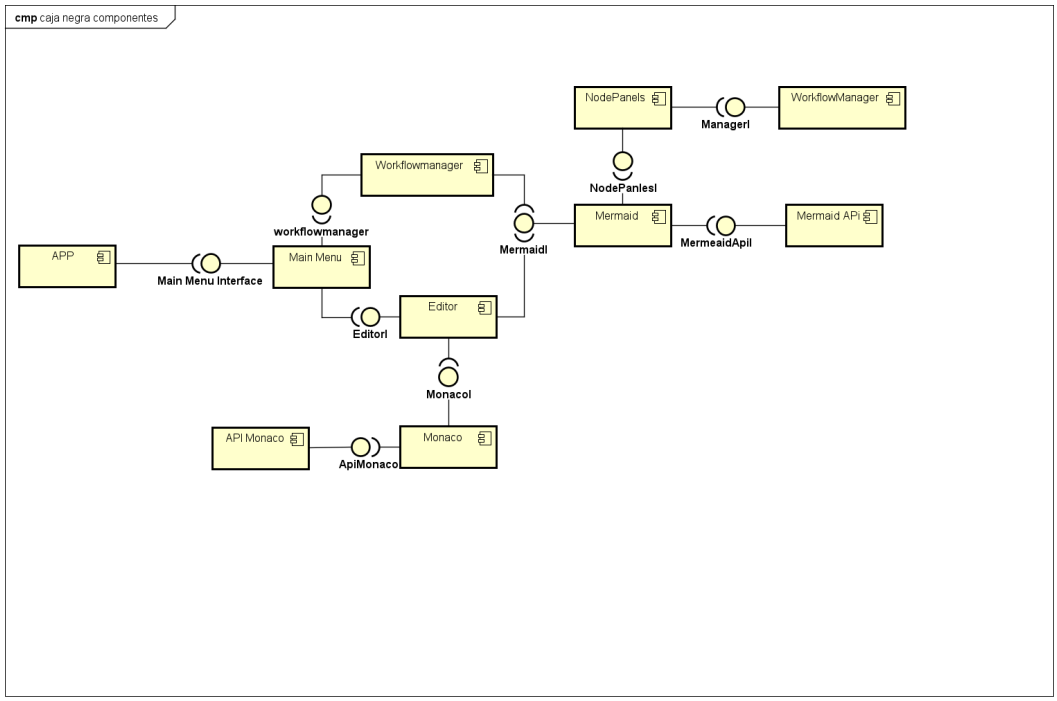

Figura 5.1: Diagrama de componentes (caja negra) de *Project Master*. Representa las principales conexiones entre módulos y las interfaces ofrecidas por cada uno. Se incluyen componentes clave como `Editor`, `Mermaid`, `Monaco`, así como los módulos de gestión y visualización del flujo.

5.3.1. Componente `MainMenu`

El componente `MainMenu` es la vista inicial de la aplicación. Su función principal es ofrecer una interfaz gráfica centralizada desde la cual el usuario puede seleccionar distintas categorías de proyectos de ingeniería (como *Electrical Engineering*, *Mechanical Engineering*, *Software Engineering* o *Custom Project*),y acceder a los distintos proyectos pertenecientes de estas categorías.

Cada categoría contiene una serie de proyectos representadas visualmente mediante tarjetas (*cards*), que pueden estar bloqueadas o desbloqueadas en función del progreso o configuración de este mismo. El usuario solo puede interactuar con las etapas desbloqueadas. Al seleccionar una etapa válida, se invoca la función `onWorkflowSelect`, llamando así al componente `workflowManager` que funciona como un *bootstraper* del componente Mermaid, adicionalmente el MainMenu presenta otra función de llamada que es (`onEditProject`, el cual llamaría al componente editor permitiendo modificar los diagramas.

El componente gestiona internamente el estado de la categoría seleccionada mediante el *hook* `useState`, lo que permite alternar dinámicamente entre la vista de selección de categorías y la visualización de proyectos correspondientes. Por otro lado, el sistema de imágenes está diseñado para asociar dinámicamente una imagen con cada etapa o categoría, utilizando una imagen por defecto cuando no se encuentra una específica. En términos de comunicación, `MainMenu` se relaciona directamente con el componente padre a través de las funciones `onWorkflowSelect` y `onEditProject`. Esta arquitectura permite mantener el componente desacoplado, facilitando su reutilización y mantenimiento dentro del flujo general de la aplicación.

5.3.2. Componente `Editor`

El componente `Editor` ofrece una interfaz avanzada para la edición visual y textual de diagramas de flujo en notación *Mermaid*. Su principal funcionalidad consiste en permitir al usuario modificar el contenido del grafo mediante un editor de texto basado en `MonacoEditor`, al tiempo que proporciona una vista previa interactiva del mismo, facilitada por el componente `Mermaid`.

`Editor` mantiene un estado interno del flujo de trabajo (`workflowState`) que se sincroniza con el estado generado por el componente `Mermaid`, permitiendo así capturar información semántica adicional (como descripciones y nodos activos). Esta integración se logra a través de la función `onWorkflowStateChange`, que comunica al editor cualquier cambio estructural relevante. Esto último es una aplicación del patrón observador.

Entre sus funcionalidades clave se incluyen:

- Guardado de diagramas como archivos de configuración, incluyendo metadatos como el título del diagrama y el nodo actual.
- Carga de archivos mediante expresiones regulares para extraer tanto el contenido del grafo como su título.

- Reinicio del contenido al estado por defecto.
- Ocultación automática de comentarios en la notación Mermaid (líneas iniciadas por `%%`) mediante las capacidades del editor Monaco.

A nivel estructural, `Editor` se comunica con el componente `MonacoEditorView` para la edición de texto, con `Mermaid` para la visualización dinámica, y con el componente padre a través del *callback* `onGoBack`. Esta modularidad permite su integración flexible dentro del sistema general, favoreciendo su reutilización en diferentes etapas del flujo de trabajo.

5.3.3. Componente `Mermaid`

El componente `Mermaid` se encarga de la visualización y gestión interactiva de diagramas de flujo definidos mediante el lenguaje *Mermaid.js*. Este componente actúa como puente entre la representación textual del flujo de trabajo y su interpretación gráfica, permitiendo tanto la visualización dinámica como la interacción directa con los nodos del diagrama, esto se ha logrado usando un patrón decorador pudiendo ampliar la funcionalidad principal que tendría la biblioteca Mermaid.

A nivel funcional, este componente interpreta la definición textual del diagrama, genera su representación SVG mediante la biblioteca Mermaid, y aplica lógica personalizada para reflejar el estado de cada nodo (pendiente, actual o completado). Esta información se gestiona internamente a través del estado `workflowState`, y puede compartirse con componentes externos mediante la propiedad `onWorkflowStateChange` (usado como implementación del patrón observador).

Además, `Mermaid` ofrece funcionalidades interactivas avanzadas, como:

- Selección de nodos mediante clic.
- Control de cámara: desplazamiento (*drag*), zoom y centrado automático en nodos activos.
- Panel lateral con pestañas para visualizar y editar la descripción, la lista de tareas (*checklist*) y documentación técnica asociada a cada nodo.
- Evaluación de decisiones tipo *choice*, gestionadas automáticamente a partir de los nodos hijos conectados.

Este componente se comunica directamente, editor y con `nodePanel`, siendo `nodePanel` el encargado de mostrar toda la información adicional de los nodos Mermaid.

5.3.4. Componente `MonacoEditorView`

El componente `MonacoEditorView` proporciona una interfaz de edición de texto basada en el editor `Monaco`, el mismo motor utilizado en *Visual Studio Code*. Este editor permite

al usuario escribir, modificar y visualizar definiciones de diagramas en lenguaje `Mermaid`, ofreciendo una experiencia avanzada con temas visuales y soporte para gramáticas personalizadas.

Desde el punto de vista funcional, el componente recibe el contenido del código a través de la propiedad `code` y notifica los cambios al componente padre mediante la función `onCodeChange`. También expone el objeto del editor a través de la función `onInit`, lo cual permite a otros componentes interactuar con la instancia del editor (por ejemplo, para ocultar líneas específicas o modificar dinámicamente el contenido).

Entre sus características destacadas se encuentran:

- Carga dinámica de gramáticas `TextMate` específicas para el lenguaje Mermaid.
- Aplicación de temas visuales personalizados, convertidos desde definiciones de VS Code mediante la librería `@estruyf/vscode-theme-converter`.
- Sincronización automática entre el estado externo del código y el contenido visualizado en el editor.
- Gestión de un selector de temas que permite al usuario cambiar la apariencia del editor de forma interactiva.

Este componente se comunica directamente con:

- El componente `Editor`, al cual informa sobre los cambios de contenido del código Mermaid.
- El propio sistema `Monaco Editor`, sobre el cual se monta la interfaz de edición y se aplican configuraciones avanzadas como el resaltado de sintaxis y los temas.

De esta manera este componente funciona realmente como un adaptador del sistema Monaco Editor, facilitando su integración en esta aplicación web.

5.3.5. Componente `NodePanel`

El componente `NodePanel` proporciona una vista simplificada de la lista de tareas asociadas a un nodo específico dentro de un flujo de trabajo. Su funcionalidad principal es permitir al usuario visualizar y gestionar los elementos de la *checklist* correspondiente a un nodo, así como marcarlo como completado una vez que todas las tareas hayan sido finalizadas.

A nivel técnico, `NodePanel` recibe como propiedades:

- El nodo actual (`node`) con sus metadatos, incluyendo el estado y la checklist.
- Una función `onChecklistItemToggle`, que permite alternar el estado de cada ítem de la checklist.

- Una función `onNodeComplete`, que se invoca al completar todas las tareas y confirmar la finalización del nodo.
- Una bandera opcional `disableInteraction`, que controla si el usuario puede interactuar con los elementos del panel.

El componente evalúa si todas las tareas han sido completadas mediante una verificación booleana y, en caso afirmativo, muestra un botón que permite marcar el nodo como *completo*. También incorpora controles de validación y depuración para asegurar que los elementos de la checklist se estructuren correctamente como objetos, no como cadenas.

`NodePanel` se comunica principalmente con el componente `Mermaid`, que lo incorpora como parte de su panel lateral, delegando en él la interacción con las tareas individuales de cada nodo. Esta separación de responsabilidades facilita la reutilización del componente, aumentando la modularidad y escalabilidad.

5.3.6. Componente `WorkflowManager`

El componente `WorkflowManager` actúa como intermediario entre el sistema de navegación de flujos de trabajo y el componente visual `Mermaid`. Su objetivo principal es cargar, interpretar y gestionar un archivo de configuración que representa un flujo de trabajo definido en notación Mermaid enriquecida con metadatos.

Este componente permite:

- Cargar flujos predefinidos desde archivos locales o integrados en la aplicación.
- Procesar y extraer definiciones Mermaid, títulos y nodos activos mediante expresiones regulares.
- Mantener el estado del nodo actual del flujo mediante `localStorage`, garantizando persistencia entre sesiones.
- Implementar funcionalidades de navegación temporal como *undo* y *redo*, basadas en un historial de nodos visitados.

La configuración cargada se representa mediante una estructura `WorkflowConfig`, que contiene:

- El título del diagrama.
- La definición Mermaid del grafo.
- El identificador del nodo actualmente activo.

Cuando se selecciona un flujo de trabajo personalizado (`open-custom`), el componente permite al usuario subir un archivo `.config`, que es analizado y visualizado de manera interactiva.

`WorkflowManager` se comunica principalmente con el componente `Mermaid`, al cual le transfiere la definición del grafo y el nodo inicial, y del que recibe notificaciones de cambio en el nodo activo mediante la función `onCurrentNodeChange`. También interactúa con el almacenamiento local del navegador (`localStorage`) para guardar el estado del nodo actual.

En conjunto, este componente facilita la carga dinámica, persistencia e interacción temporal con diagramas de flujo enriquecidos, proporcionando una experiencia robusta y flexible para la gestión de flujos de trabajo en entornos de ingeniería.

5.4. Realización del historias de usuario

Para completar la explicación de la comunicación entre componentes, las Figuras 5.2 y 5.3 presentan dos diagramas de secuencia que explican el flujo de interacción entre los componentes de las principales historias de usuario.

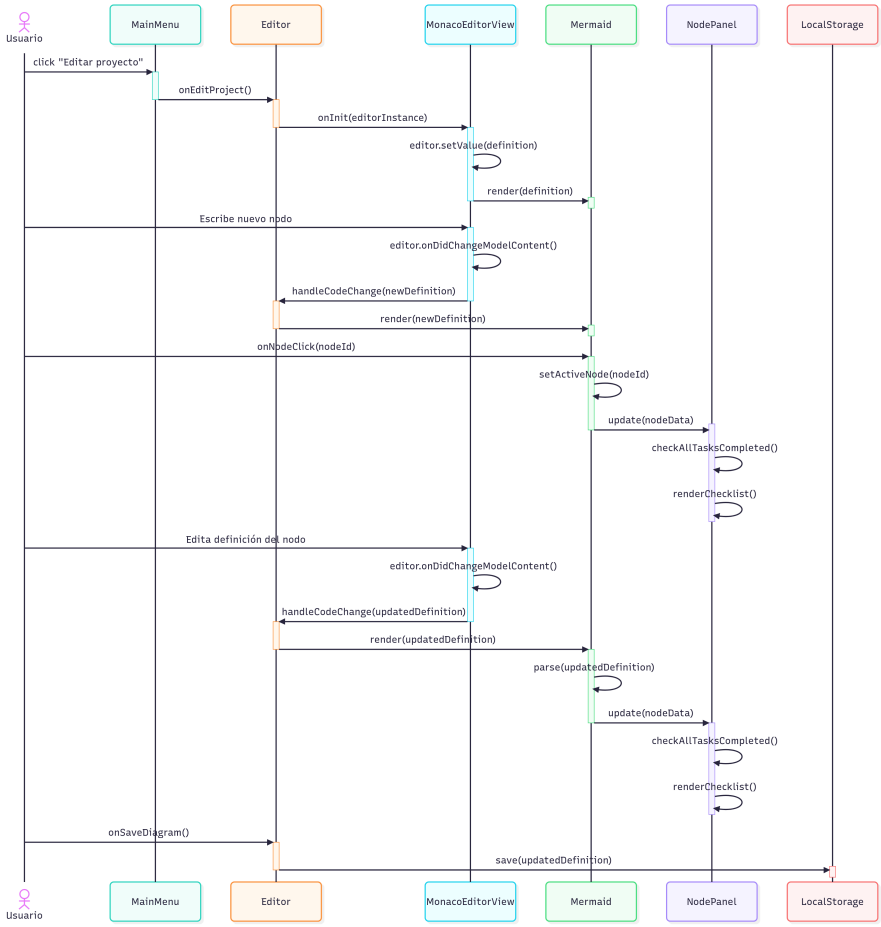

Figura 5.2: Diagrama de secuencia del flujo de edición de un proyecto. Representa la interacción entre el usuario, el editor de código (Monaco), el motor de renderizado Mermaid y el sistema de almacenamiento local durante la edición y guardado de un diagrama.

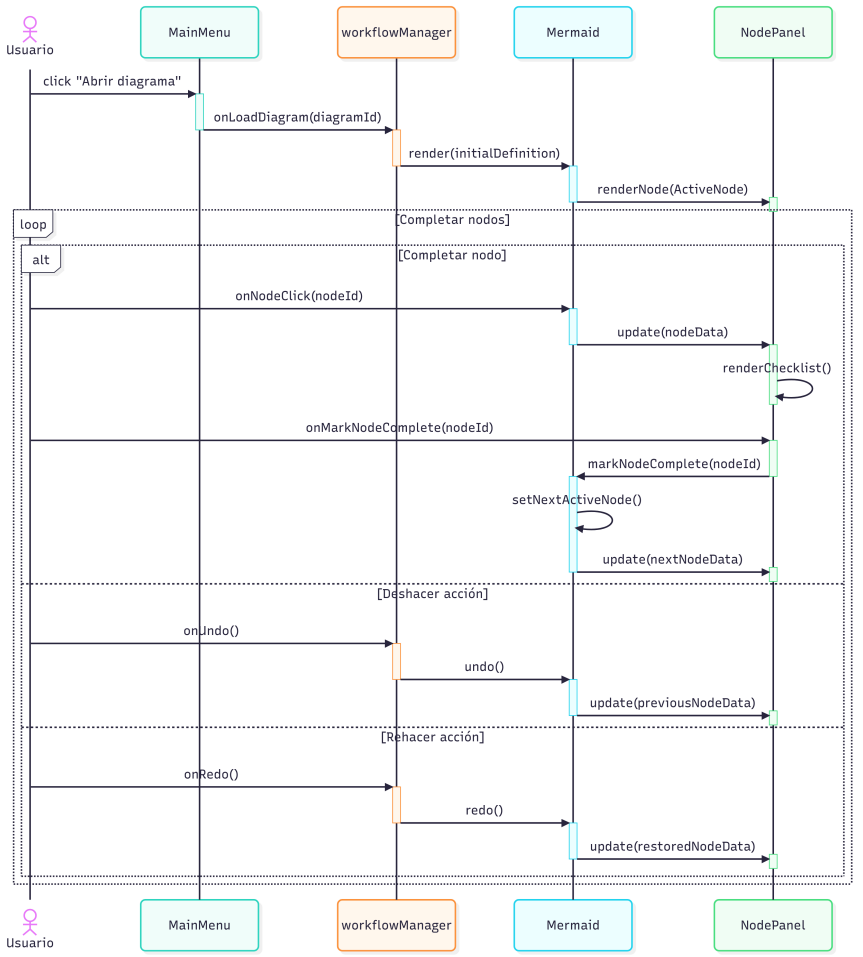

Figura 5.3: Diagrama de secuencia del flujo de completar un diagrama. Muestra la carga de un diagrama existente, la interacción del usuario al completar nodos, así como las acciones de deshacer y rehacer sobre el progreso del diagrama.

5.5. Patrones usados

Para el desarrollo de Project Master se han utilizado patrones de diseño que permitiesen facilitar el mantenimiento del código y su escalabilidad, más concretamente se usaron tres patrones Observador, Adaptador y Decorador [15], [16, 14].

5.5.1. Patrón Observador (Observer)

El patrón **Observador** es un patrón de diseño de comportamiento que permite definir una dependencia uno-a-muchos entre objetos, de modo que cuando un objeto cambia su estado, todos sus dependientes son notificados y actualizados automáticamente. Este patrón es especialmente útil cuando un cambio en un objeto requiere cambios en otros objetos, pero no se conoce de antemano cuántos objetos deben ser actualizados.

En este trabajo fin de grado el componente Editor funciona como un observador que notifica los cambios ocurridos en el componente Monaco al componente Mermaid. Nodepanel es un observador del componente Mermaid, permitiendo cambiar el estado de los nodos.

5.5.2. Patrón Adaptador (Adapter)

El patrón **Adaptador** es un patrón de diseño estructural que permite que interfaces incompatibles colaboren. Se utiliza para convertir la interfaz de una clase en otra interfaz que los clientes esperan, permitiendo que clases que de otro modo no podrían trabajar juntas lo hagan sin modificar su código original.

Este patrón se utiliza en el componente MonacoEditor para poder utilizar la biblioteca Monaco.

5.5.3. Patrón Decorador (Decorator)

El patrón **Decorador** es un patrón de diseño estructural que permite añadir responsabilidades adicionales a un objeto de manera dinámica. Proporciona una alternativa flexible a la especialización para extender funcionalidades, al envolver objetos dentro de otros objetos decoradores que implementan la misma interfaz.

En concreto este patrón ha sido utilizado en el componente Mermaid, permitiendo tener la funcionalidad principal de dibujar diagramas SVG pero añadiendo funcionalidad extra como un cambio de aspecto dinámico, y opciones de hacer y deshacer, entre otras funcionalidades explicadas en la sección 5.3.3.

5.6. Diseño de interfaz de usuario

Durante el transcurso de primer *sprint* se realizaron una serie de bocetos y esquemas de como serían las vistas y como interactuarían entre si. La objetivo principal era poder tener una vista previa de como sería y se vería la aplicación y poder mostrarla a los usuarios finales. Una vez validada, guió el diseño de la aplicación web. Las Figuras 5.4, 5.5 y 5.6 muestran los bocetos de las vistas diseñadas.

La Figura 5.4 muestra la vista del modo editor, se han seguido los principios de diseño UX/UI, manteniendo un buen formato de tamaño de botones y un espacio amplio para la escritura del código Mermaid, pero sin restringir el espacio a la visualización del diagrama, la parte de información de los nodos ocupa un tercio de pantalla con una serie de botones para cambiar entre la vista de descripción, documentación y tareas de una manera cómoda. Se ha tratado de mantener la coherencia de los botones, dejando los que tienen una utilidad similar cercanos, como puede ser cargar un fichero o descargarlo.

Figura 5.4: Vista del modo editor. Se han seguido principios UX/UI, con un área amplia para escribir el código Mermaid, y una zona lateral para gestionar descripciones, documentación y subtareas asociadas a cada nodo. Los botones están agrupados según funcionalidad (importar, exportar, reiniciar, etc.).

La Figura 5.5, por otra parte, es bastante mas simple a nivel conceptual ya que lo que se busca es darle el máximo espacio posible a la renderizado del diagrama, aunque sin perder la coherencia entre botones, manteniendo los que tienen una utilidad similar próximos, mientras que la parte del panel de nodos se mantiene siendo un tercio de la pantalla

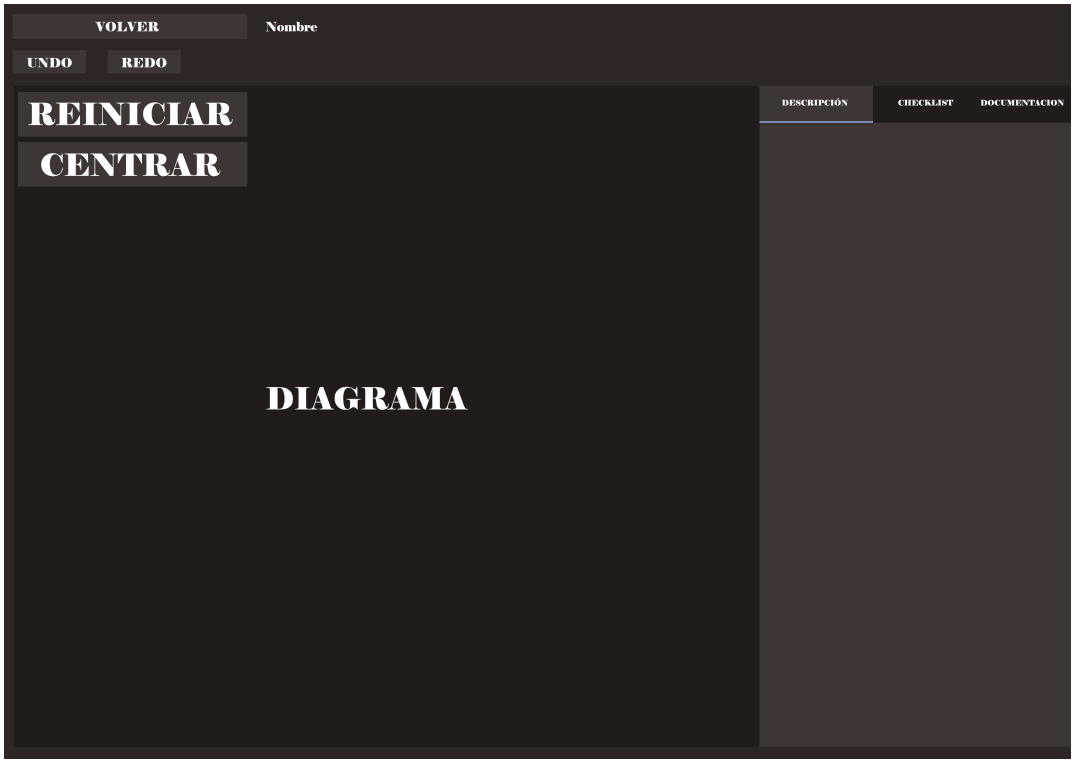

Figura 5.5: Vista del modo visualización. Se maximiza el espacio para renderizar el grafo y se mantiene un panel lateral para interactuar con el nodo activo.

La Figura 5.6 muestra la vista inicial de Project Master, permite una visualización de los departamentos dividida por tarjetas con una imagen identificativa de cada departamento, con un tamaño ajustable según la cantidad de departamentos introducidos. Se muestra uan tarjeta extra que permite cargar un fichero propio en modo visualización, manteniendo la coherencia de la vista como una funcionalidad útil y con un fácil acceso.

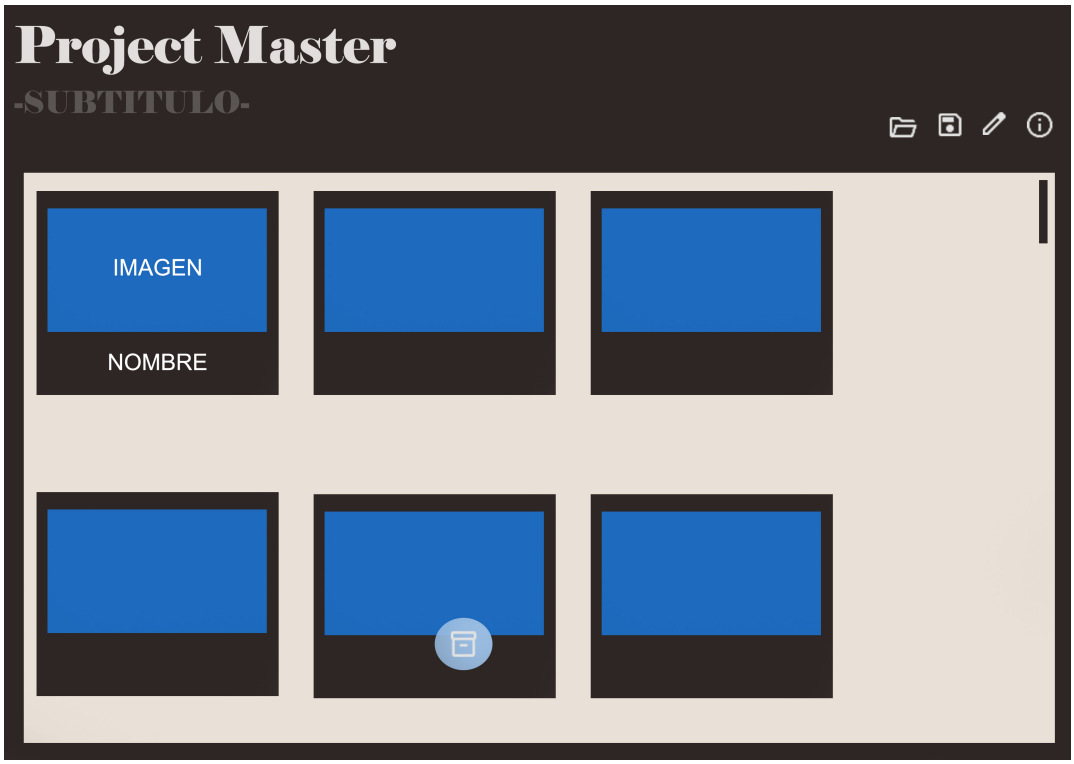

Figura 5.6: Vista inicial de la aplicación. Muestra los departamentos en formato de tarjetas visuales, adaptando el diseño dinámicamente al número de elementos.

5.7. Arquitectura física del sistema

La Figura 5.7 muestra el diagrama de despliegue de la arquitectura física del sistema. Se ha optado por un diseño en dos niveles para la aplicación, debido a su sencillez y la ausencia de una conexión a una base de datos.

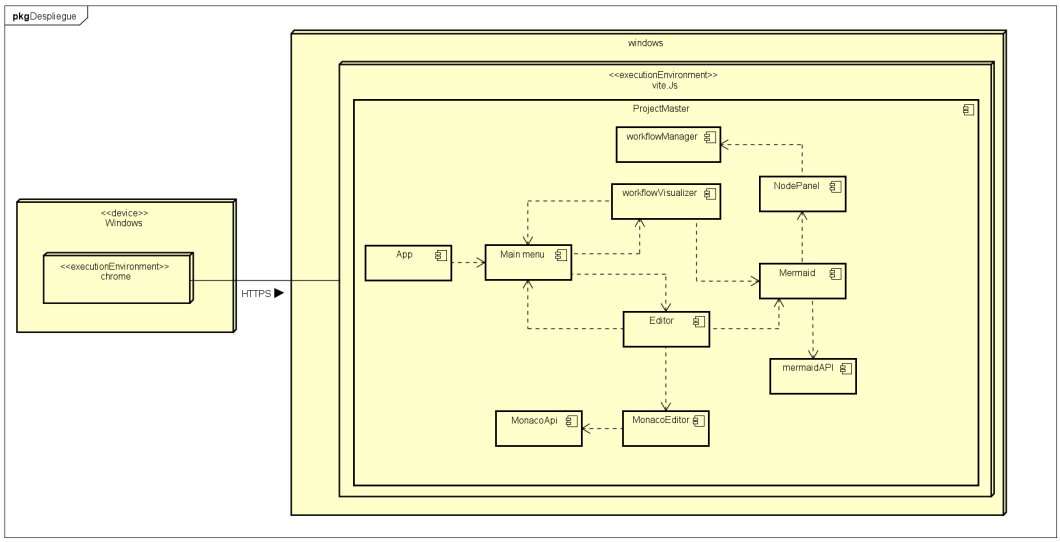

Figura 5.7: Diagrama de despliegue de la aplicación *Project Master*. Representa la distribución lógica de los componentes en tiempo de ejecución. La aplicación se ejecuta en un navegador (Chrome) dentro de un entorno Windows y se comunica vía HTTPS con un entorno de desarrollo construido con Vite.js que aloja todos los módulos principales.

Como se puede ver en la Figura 5.7 se ha seguido una arquitectura física centrada en dos niveles, permitiendo conectarse de manera remota vía HTTPS desde cualquier navegador a Project Master, el cual se ejecuta en un entorno web a través del lanzador ViteJS.

Capítulo 6

Implementación y pruebas

6.1. Implementación de la Aplicación

En esta sección se detalla la implementación de los componentes que conforman la interfaz interactiva de este Trabajo Fin de Grado. El desarrollo se ha realizado utilizando React como biblioteca principal para la construcción de interfaces de usuario, complementado con tecnologías especializadas como Monaco Editor para la edición de código, Mermaid.js para la representación visual de diagramas, y diversas herramientas auxiliares para la gestión del estado y la persistencia, apoyandose de distintos patrones de diseño para mejorar su claridad y modularidad.

Cada componente ha sido diseñado siguiendo principios de modularidad, reutilización y separación de responsabilidades, lo que permite una evolución escalable del sistema y una mayor facilidad en las tareas de mantenimiento [13] . La lógica se distribuye entre componentes encargados de la navegación (MainMenu), edición interactiva (Editor, Mermaid, MonacoEditorView) y visualización o gestión de flujos (WorkflowManager), entre otros.

Asimismo, se ha prestado especial atención a la comunicación entre componentes, tanto mediante *props*(datos que un componente recibe de su componente padre) como a través de funciones, permitiendo una sincronización coherente del estado global de la aplicación. También se ha desarrollado un sistema de almacenamiento persistente local, mediante el uso de *localStorage*, que asegura la continuidad del trabajo del usuario a lo largo del tiempo, simplificando mucho el diseño de la aplicación en si mismo.

Antes de abordar las pruebas realizadas sobre la aplicación, y con el fin de facilitar la comprensión de los fragmentos de código presentados, se describen a continuación los principales tipos de datos utilizados, así como la estructura interna de los componentes clave del sistema.

6.1.1. Estructura del Proyecto

La estructura del directorio `src/` se ha organizado siguiendo criterios de modularidad y claridad. A continuación se describe brevemente el propósito de las principales carpetas del proyecto:

- **components/**: contiene los distintos componentes reutilizables de la interfaz, organizados por funcionalidad para facilitar su mantenimiento y localización.
- **utils/**: almacena funciones auxiliares relacionadas con la conversión de definiciones Mermaid y el manejo de datos internos del sistema.
- **assets/**: incluye imágenes y temas visuales personalizados utilizados en el editor.
- **workflows/**: contiene ejemplos de flujos predefinidos y documentación asociada.
- **Archivos raíz**: `App.tsx`, `main.tsx` y archivos de estilo definen la estructura principal de la aplicación y su punto de entrada.

6.1.2. Definición de Tipos de Datos

Con el objetivo de garantizar una estructura coherente y fuertemente tipada en la aplicación, se han definido diversos tipos TypeScript que representan las entidades principales del sistema. Estos tipos permiten formalizar la información asociada a los nodos del flujo de trabajo, así como la estructura completa del grafo y su estado en tiempo real.

A continuación, se describen los tipos más relevantes utilizados durante el desarrollo:

- **ChecklistItem**: representa un ítem de tarea dentro de un nodo, que puede estar completado o pendiente, y cuya posición puede estar ordenada mediante un campo opcional `order`.
- **WorkflowNodeMetadata**: almacena metadatos de un nodo dentro del flujo de trabajo, incluyendo su identificador, título, descripción, estado (*pending*, *current* o *complete*), tipo de nodo (normal, decisión o elección) y su lista de tareas.
- **WorkflowState**: estructura que agrupa el estado general del diagrama de flujo, incluyendo la lista de nodos enriquecidos, los nodos visuales (`flowNodes`) y las aristas (`flowEdges`) del grafo, en formato compatible con la biblioteca `React Flow`.
- **WorkflowConfig**: representa la configuración de un flujo cargado desde archivo, conteniendo el título del flujo, su definición en Mermaid y el nodo activo.
- **WorkflowNode** y **Workflow**: tipos utilizados para representar flujos de trabajo alternativos, estructurados como una colección de nodos con referencias explícitas a nodos anteriores y siguientes, así como un nodo activo.

- **NodePanelProps**: tipo utilizado para tipar las propiedades del componente `NodePanel`, incluyendo las funciones de interacción con ítems de la checklist.
- **WorkflowNodeType**: extensión del tipo WorkflowNode que incorpora una documentación adicional en formato Markdown, utilizada para enriquecer el contenido técnico de los nodos.

Estos tipos de datos constituyen la base para la validación estructural y la comunicación entre componentes dentro del sistema, facilitando una programación robusta durante la construcción de los distintos módulos funcionales.

6.2. Casos de prueba

A continuación se detallan los principales casos de prueba realizados sobre la aplicación *Project Master*. Cada caso se documenta siguiendo el formato clásico con trazabilidad hacia las historias de usuario definidas.En las tablas 6.1, 6.2, 6.3, 6.4, 6.5, 6.6, 6.7, 6.8, 6.9 y 6.10 se pueden ver todos los casos de prueba.

CP01 – Carga de diagrama desde archivo

Identificador	CP01
Precondiciones	El usuario ha accedido al modo editor.
Datos de entrada	Archivo `.config` con sintaxis Mermaid válida.
Salida Esperada	Diagrama y nodos renderizados correctamente.
Escenario	Pulsar “Cargar diagrama”, seleccionar archivo y visualizar.
Trazabilidad	HU10, HU7
Ejecución	20/06/2025
Defectos encontrados	Ninguno.

Tabla 6.1: Caso de prueba CP01 – Verifica que la carga de un archivo Mermaid se realiza correctamente y el diagrama se muestra en pantalla.

CP02 – Marcado de nodo como completado

Identificador	CP02
Precondiciones	Diagrama cargado en modo visualización.
Datos de entrada	Subtareas completadas del nodo actual.
Salida Esperada	Nodo completado y avance automático.
Escenario	Completar checklist y pulsar “Finalizar nodo”.
Trazabilidad	HU15, HU16
Ejecución	21/06/2025
Defectos encontrados	Error menor de sincronización (resuelto).

Tabla 6.2: Caso de prueba CP02 – Comprueba que el nodo se marca como completado y el flujo avanza automáticamente al siguiente nodo.

CP03 – Funcionalidad de deshacer/rehacer

Identificador	CP03
Precondiciones	Usuario ha realizado acciones sobre nodos.
Datos de entrada	Pulsar los botones “Undo” o “Redo”.
Salida Esperada	Estado del grafo actualizado correctamente.
Escenario	Hacer cambios, luego aplicar undo/redo.
Trazabilidad	HU2
Ejecución	21/06/2025
Defectos encontrados	Ninguno.

Tabla 6.3: Caso de prueba CP03 – Verifica que los cambios realizados pueden deshacerse o rehacerse correctamente en el diagrama.

CP04 – Edición en tiempo real del diagrama

Identificador	CP04
Precondiciones	Modo editor activo.
Datos de entrada	Código Mermaid modificado en Monaco Editor.
Salida Esperada	Diagrama actualizado en tiempo real.
Escenario	Editar código y observar vista gráfica.
Trazabilidad	HU7, HU9
Ejecución	22/06/2025
Defectos encontrados	Ligeros retardos en equipos lentos.

Tabla 6.4: Caso de prueba CP04 – Comprueba que las modificaciones en el editor actualizan dinámicamente el diagrama mostrado.

CP05 – División por departamentos

Identificador	CP05
Precondiciones	Aplicación en pantalla de inicio.
Datos de entrada	Diagrama por defecto o archivo cargado.
Salida Esperada	Vista categorizada por tipo de proyecto.
Escenario	Elegir departamento → cargar diagrama.
Trazabilidad	HU13, HU14
Ejecución	20/06/2025
Defectos encontrados	Ninguno.

Tabla 6.5: Caso de prueba CP05 – Verifica que la aplicación categoriza correctamente los diagramas por departamento o tipo de proyecto.

CP06 – Guardado automático en localStorage

Identificador	CP06
Precondiciones	Diagrama con progreso actualizado.
Datos de entrada	Cambios de estado en los nodos.
Salida Esperada	Progreso persistente tras cerrar y reabrir la app.
Escenario	Marcar nodo → cerrar navegador → abrir app.
Trazabilidad	HU3, HU15
Ejecución	21/06/2025
Defectos encontrados	Si se borran las cookies, se pierde el progreso.

Tabla 6.6: Caso de prueba CP06 – Verifica que el progreso de los diagramas se guarda automáticamente en localStorage y se recupera al reiniciar la aplicación.

CP07 – Ayuda contextual y documentación por nodo

Identificador	CP07
Precondiciones	Nodo activo seleccionado en el panel lateral.
Datos de entrada	Descripción o documentación Markdown cargada.
Salida Esperada	Visualización contextual de la documentación.
Escenario	Seleccionar nodo → mostrar ayuda lateral.
Trazabilidad	HU17
Ejecución	22/06/2025
Defectos encontrados	Markdown no se renderiza si el formato es incorrecto.

Tabla 6.7: Caso de prueba CP07 – Comprueba que se muestra la documentación contextual en Markdown asociada al nodo activo.

CP08 – Reinicio de progreso del diagrama

Identificador	CP08
Precondiciones	Diagrama parcialmente completado.
Datos de entrada	Pulsación del botón “Reiniciar progreso”.
Salida Esperada	Todos los nodos marcados como pendientes.
Escenario	Ejecutar tareas → pulsar reiniciar → comprobar estado.
Trazabilidad	HU3
Ejecución	23/06/2025
Defectos encontrados	Ninguno.

Tabla 6.8: Caso de prueba CP08 – Verifica que el progreso del diagrama puede reiniciarse y todos los nodos vuelven a estado pendiente.

CP09 – Exportación de diagrama

Identificador	CP09
Precondiciones	Diagrama modificado en modo editor.
Datos de entrada	Pulsación del botón “Exportar diagrama”.
Salida Esperada	Archivo descargado con extensión `.config` y contenido válido.
Escenario	Editar contenido → exportar → abrir archivo descargado.
Trazabilidad	HU11
Ejecución	23/06/2025
Defectos encontrados	Ninguno.

Tabla 6.9: Caso de prueba CP09 – Verifica que el sistema exporta el diagrama en un archivo externo con el formato correcto.

CP10 – Cambio de tema en el editor (modo oscuro)

Identificador	CP10
Precondiciones	Modo editor abierto.
Datos de entrada	Selección del tema “modo oscuro”.
Salida Esperada	Cambio visual inmediato en el editor.
Escenario	Abrir editor → cambiar tema → confirmar que se aplica.
Trazabilidad	RNF-06
Ejecución	23/06/2025
Defectos encontrados	Ninguno.

Tabla 6.10: Caso de prueba CP10 – Verifica que el cambio de tema se aplica correctamente y se muestra el modo oscuro en el editor.

Cabe destacar que durante el desarrollo de este Trabajo Fin de Grado se han llevado a cabo una serie de pruebas de aceptación, que aunque no estén documentadas fueran llevadas a cabo en formato de entrevista personal con el cliente, dándose de manera semanal coincidiendo con el Sprint Review.

Capítulo 7

Seguimiento del proyecto

7.1. Asignación de actividades por Sprint y su Desarrollo detallad

A continuación se describe el desarrollo del sistema, desglosado por sprints semanales, desde la semana 14 hasta la 27 del año 2025. Las tareas se organizaron en base a la prioridad técnica y funcional, combinando la ejecución iterativa de funcionalidades, refactorización progresiva y pruebas de validación continuas. Se siguió el marco de trabajo Scrum, con reuniones semanales de planificación y revisión, cumpliendo con los eventos Scrum. Como se puede ver en la tabla 7.1 se muestran los sprints y sus actividades.

Sprint	**Actividades asignadas (Épicas, HU y RNF)**
Sprint 1	- Inicio del proyecto *Project Master*. - Configuración del entorno con TypeScript (RNF-05). - Desarrollo de la lógica inicial para la construcción del grafo (Épica E1, HU1). - Preparación y pruebas iniciales con Monaco Editor (Épica E2, HU8).
Sprint 2	- Implementación de la lógica de cámara para desplazamiento básico en el grafo (RNF-02, HU16). - Primeros controles para centrar la cámara sobre nodos (Épica E4, HU16).
Sprint 3	- Continuación y refinamiento de la lógica de cámara al pulsar nodos (HU16, HU15). - Avance en la organización de diagramas por departamentos (Épica E3, HU13, HU6). - Uso de colores corporativos AVL en la interfaz (RNF-01).
Sprint 4	- Mejora en la lógica de completado de tareas en nodos (HU15). - Permitir centrar y mover el grafo manualmente (HU16). - Inicio del desarrollo del modo editor (Épica E2, HU7).
Sprint 5	- Desarrollo de la lógica del modo editor y ajustes visuales (HU7, HU8). - Mejoras en el centrado de cámara para mayor precisión (HU16).
Sprint 6	- Implementación del zoom en la visualización del grafo (HU7). - Ajustes para mantener actualización dinámica en el editor (RNF-04).

Sprint 7	- Reorganización y mejora del menú principal, división por departamentos (Épica E3, HU13). - Inicio de la integración completa de modo editor con visualización (HU7, HU8).
Sprint 8	- Unión del modo editor y modo visualización en una sola aplicación (HU7, HU10, HU11). - Eliminación de bugs del modo editor. - Preparación para carga de archivos externos (HU10).
Sprint 9	- Permitir la carga de archivos y actualización dinámica de la vista (HU10, HU7). - Inicio del sistema de guardado de progreso en diagramas (HU3, HU15).
Sprint 10	- Implementación de botones de *undo* y *redo* (HU2). - Mejoras visuales y refinamiento del modo oscuro (RNF-06).
Sprint 11	- Continuación del desarrollo y eliminación de bugs. - Mejoras en la actualización y comportamiento de la cámara (HU16). - Optimización del sistema de carga de ficheros. - Validación de tiempos de respuesta (RNF-07).
Sprint 12	- Pruebas con usuarios reales para evaluación de usabilidad (RNF-03). - Ajustes basados en feedback y validación de accesibilidad (RNF-06).
Sprint 13	- Sprint de estabilización y cierre de bugs. - Revisión y documentación final para entrega del proyecto y memoria del TFG.

Tabla 7.1: Planificación de sprints y actividades realizadas en el desarrollo del proyecto.

Sprint 1 (31/03/2025 – 06/04/2025): Inicio del proyecto y lógica base

Este primer sprint marcó el inicio del proyecto *Project Master*, configurando el entorno de desarrollo (basado en TypeScript y Vite) e integrando Monaco Editor como componente principal de edición (HU8). Se comenzó con la lógica fundamental para la construcción del grafo visual, y se definieron los primeros diagramas para pruebas. Además, se empezaron a identificar los requisitos no funcionales y a modelar las primeras épicas y HU para organizar el trabajo.

Sprint 2 (07/04/2025 – 13/04/2025): Cámara y navegación inicial

En este sprint se implementó la lógica de cámara básica, centrada en permitir el desplazamiento dentro del lienzo de grafo (RNF-02). Esta funcionalidad fue clave para permitir que el usuario explore el grafo libremente, respetando principios de UX. Se conectó la lógica de cámara con los nodos del diagrama, lo que sentó las bases para el centrado automático.

Sprint 3 (14/04/2025 – 20/04/2025): Completado de tareas y nodos activos

Se amplió la lógica de cámara y se comenzó la implementación del sistema de tareas por nodo (HU15). Al hacer clic en un nodo, se actualiza la vista de cámara para centrarse en él, y se activa el sistema de avance visual según tareas completadas. Esta dinámica es parte central de la experiencia interactiva y conecta directamente con la épica E4.

Sprint 4 (21/04/2025 – 27/04/2025): Mejora del sistema de tareas

Se mejoró la lógica de completado de tareas, haciéndola más robusta y permitiendo diferenciar estados de avance. Además, se permitió centrar y mover el grafo manualmente, integrando controles de navegación más accesibles (HU16). Este sprint concluyó con los primeros avances en la implementación de un modo editor (HU7), en el cual los usuarios pueden escribir contenido que afecta al diagrama.

Sprint 5 (28/04/2025 – 04/05/2025): Modo editor y refinamientos visuales

Se desarrolló la lógica principal del modo editor (HU8, HU9), incluyendo los primeros ajustes visuales como el diseño de botones y estilo de interfaz (RNF-01). También se revisó el comportamiento de la cámara para hacer más preciso el centrado tras acciones de usuario. Se integró el editor con el motor de visualización.

Sprint 6 (05/05/2025 – 11/05/2025): Zoom y navegación mejorada

Este sprint se centró en la implementación del zoom interactivo (HU7), añadiendo una funcionalidad esencial para explorar diagramas complejos. Se refinaron los controles de centrado y desplazamiento, y se añadieron animaciones suaves para mejorar la experiencia visual (RNF-02, RNF-04).

Sprint 7 (12/05/2025 – 18/05/2025): Reorganización de menús y experiencia de usuario

Se reestructuró el menú principal (HU13), clasificando el contenido por departamentos e integrando accesos más intuitivos a las diferentes funcionalidades. Se mejoró la estética y coherencia visual, incluyendo una estructura más clara del flujo de navegación entre el modo visualización y el modo editor (E3).

Sprint 8 (19/05/2025 – 25/05/2025): Unificación de modos y pruebas de integración

Se unificaron el modo editor y el modo visualización en una sola aplicación cohesiva, facilitando la alternancia entre edición y vista previa (HU7, HU10, HU11). Este cambio estructural mejoró la usabilidad y simplificó la lógica interna de la app. Además, se corrigieron errores relacionados con el modo editor y se preparó el sistema para recibir archivos externos (HU10).

Sprint 9 (26/05/2025 – 01/06/2025): Carga de archivos y actualización reactiva

Se completó la funcionalidad de carga de ficheros personalizados y se mejoró el sistema de actualización del grafo tras cada cambio (HU10, HU7). Además, se implementó el guardado del progreso del usuario en los diagramas, asociando la información al estado de cada nodo (HU3, HU15).

Sprint 10 (02/06/2025 – 08/06/2025): Undo, redo y mejoras visuales

Se implementaron los botones de deshacer y rehacer (HU2), permitiendo al usuario explorar y modificar su progreso de manera más segura. También se introdujeron mejoras visuales en la interfaz del editor, enfocadas en facilitar el trabajo prolongado (RNF-06, modo oscuro).

Sprint 11 (09/06/2025 – 15/06/2025): Refactorización y validación de requisitos

Este sprint se centró en la limpieza y reorganización del código, preparando la base para futuras extensiones. Se revisaron todos los requisitos no funcionales, asegurando tiempos de respuesta aceptables (RNF-07) y consistencia visual (RNF-01). Además, se inició la evaluación interna de usabilidad (RNF-03).

Sprint 12 (16/06/2025 – 22/06/2025): Evaluación y resultados de usabilidad

Se ejecutaron pruebas de usabilidad con usuarios del departamento de ingeniería, obteniendo una calificación superior a 8 en efectividad, estética y facilidad de uso (RNF-03). También se ajustaron detalles menores según el feedback recibido, garantizando una experiencia pulida.

Sprint 13 (23/06/2025 – 29/06/2025): Cierre de proyecto y documentación

Se corrigieron errores menores, se validó el comportamiento final de todos los flujos de trabajo y se completó la documentación técnica del sistema. Se consolidaron las funcionalidades principales, se empaquetó la versión estable y se redactó la memoria técnica para la defensa del TFG.

Esta planificación permitió implementar gradualmente las funcionalidades más importantes del sistema, asegurando una evolución coherente, centrada en el usuario y basada en requisitos medibles.

Capítulo 8

Conclusiones

En conclusión el desarrollo de este Trabajo de Fin de Grado ha permitido alcanzar la mayoría de los objetivos propuestos, tanto a nivel técnico como académico. A lo largo del proyecto se ha logrado diseñar y construir una aplicación web funcional utilizando el *framework* React, integrando diversas tecnologías modernas como Mermaid para la generación de diagramas y Monaco Editor para la edición dinámica del contenido.

Desde el punto de vista del desarrollo, la aplicación permite actualmente visualizar y editar diagramas con una lista de subtareas, gestionar el progreso visual, cargar y modificar archivos en formato Mermaid, así como interactuar con los nodos mediante funciones como la edición de descripciones, subtareas y nombres de los diagramas. También se ha implementado funcionalidad avanzada como el guardado en *LocalStorage*, el movimiento dinámico de cámara y la capacidad de deshacer o rehacer cambios.

Cabe destacar que, de todas las historias de usuario planteados, el único que no se ha logrado completar ha sido la inclusión de documentación en formato Markdown dentro del modo editor. Esta funcionalidad, si bien útil, fue pospuesta por cuestiones de alcance y tiempo, sin comprometer la coherencia ni la utilidad general del proyecto.

En cuanto a los objetivos académicos, el proyecto ha sido una experiencia enriquecedora que ha contribuido significativamente al desarrollo de habilidades en ingeniería de software. Se ha trabajado con metodologías ágiles, desde la fase de análisis hasta la implementación y prueba del sistema. Además, se ha adquirido un conocimiento sólido en el uso de bibliotecas y tecnologías modernas como React, Monaco y Mermaid, así como una mejora notable en aspectos relacionados con la experiencia de usuario (UX) y el diseño de interfaces (UI).

El proyecto ha supuesto un ejercicio completo de diseño, implementación y documentación técnica, permitiendo no solo el cumplimiento de los objetivos académicos planteados, sino también la entrega de una herramienta funcional y con posibilidades reales de expansión futura.

En cuanto a las líneas de trabajo futuras se centran, en primer lugar, en completar

los objetivos que no pudieron ser finalizados. En el caso concreto de este TFG, hubo un único objetivo que no pudo implementarse de forma satisfactoria debido a limitaciones de tiempo: la historia de usuario que planteaba la posibilidad de añadir una documentación en formato Markdown a cada nodo desde el modo editor. Aunque sí se pudo desarrollar un lector funcional que permite visualizar el contenido si el nodo ya contiene dicho metadato, no se llegó a implementar la funcionalidad de edición o carga directa del texto Markdown desde la interfaz. Completar esta funcionalidad permitiría a los usuarios documentar los nodos de forma más detallada y estructurada, lo cual enriquecería significativamente la comprensión y trazabilidad de los diagramas generados.

Además de finalizar dicha tarea pendiente, se proponen nuevas líneas de trabajo para ampliar las capacidades del sistema. Entre ellas destaca la incorporación de un nuevo tipo de nodo denominado nodo diagrama, cuya función sería encapsular diagramas más pequeños dentro de un nodo principal. Esta funcionalidad facilitaría la organización jerárquica de la información y permitiría trabajar con sistemas complejos de manera más modular y escalable.

Otra posible mejora consistiría en implementar una funcionalidad de gestión de tareas, asociadas a los nodos del diagrama. Estas tareas podrían incluir campos como fechas de inicio y fin, plazos o estados, ofreciendo así una integración más estrecha entre la visualización de procesos y la planificación operativa de los mismos. Este enfoque transformaría la herramienta en una solución más completa y útil para entornos colaborativos o de gestión de proyectos. Tan bien se pretende eliminar elementos estéticos sobrantes en las vistas. En conjunto, estas líneas de trabajo futuras no solo resolverían limitaciones actuales, sino que también abrirían nuevas posibilidades de uso que podrían beneficiar a usuarios con necesidades más avanzadas.

Bibliografía

[1] Github docs. Documentación oficial de GitHub, consultado en 2025.

[2] Javascript - mdn web docs. Documentación técnica de JavaScript, consultado en 2025.

[3] Latex project - official website. Sitio oficial de LaTeX, consultado en 2025.

[4] Mermaid - markdownish syntax for generating diagrams. Documentación oficial de Mermaid.js, consultado en 2025.

[5] Monaco editor. Repositorio oficial y documentación del editor Monaco, consultado en 2025.

[6] Node.js documentation. Documentación oficial de Node.js, consultado en 2025.

[7] React – a javascript library for building user interfaces. Documentación oficial de React, consultado en 2025.

[8] Typescript: Javascript with syntax for types. Documentación oficial de TypeScript, consultado en 2025.

[9] Vite – next generation frontend tooling. Documentación oficial de Vite, consultado en 2025.

[10] Alaimo Labs. Scrum pilares y artefactos, 2025. Accedido el 2 de junio de 2025.

[11] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, and Dave Thomas. Manifesto for agile software development, 2001. Accedido el 30 de mayo de 2025.

[12] Scott Chacon and Ben Straub. *Pro Git*. Apress, 2nd edition, 2014. Accedido en 2025.

[13] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 2nd edition, 2018.

[14] Eric Freeman, Elisabeth Robson, Bert Bates, and Kathy Sierra. *Head First Design Patterns*. O'Reilly Media, 2nd edition, 2020.

[15] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[16] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[17] Craig Larman. *Agile and Iterative Development: A Manager's Guide*. Addison-Wesley, 2003.

[18] OpenAI. Chatgpt, 2025. Modelo de lenguaje usado como asistente durante el TFG.

[19] Ken Schwaber and Mike Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.

[20] Ian Sommerville. *Software Engineering*. Pearson, 10th edition, 2015.

[21] Universidad de Valladolid. Proyecto docente del trabajo de fin de grado 2019-2020 (mención ingeniería de software). Accedido el 6 de julio de 2019.

Apéndice A

Manuales

A.1. Manual de despliegue e instalación

Para ejecutar correctamente la aplicación **Project Master**, se requiere tener instalado `Node` (versión 18 o superior) y `npm` en el sistema. A continuación, se detallan los pasos necesarios para el despliegue local:

1. **Clonar el repositorio:**

   ```
   git clone https://gitlab.inf.uva.es/diegonz0/tfg_diego_gonzalez_guerra_project
   cd project-master
   ```

2. **Instalar dependencias:**

   ```
   npm install
   ```

3. **Ejecutar en modo desarrollo:**

   ```
   npm run dev
   ```

   Esto lanzará la aplicación en `http://localhost:5173` utilizando Vite como servidor de desarrollo.

4. **Compilación para producción (opcional):**

   ```
   npm run build
   ```

   Este comando genera una versión optimizada del proyecto en la carpeta `dist/`.

A.2. Manual de mantenimiento

El mantenimiento de la aplicación se centra en tres aspectos principales: actualización de dependencias, control de errores y evolución funcional.

- **Actualización de dependencias:** Se recomienda ejecutar periódicamente:

  ```
  npm outdated
  npm update
  ```

  para mantener actualizadas bibliotecas como React, TypeScript, Monaco Editor o Mermaid.js.
- **Gestión de errores:** La aplicación registra errores de compilación y ejecución en la consola del navegador. Se recomienda el uso de herramientas como ESLint para detectar errores de sintaxis, y añadir pruebas automáticas en versiones futuras.
- **Extensibilidad:** El código está estructurado en componentes y hooks, lo que facilita la adición de nuevas funcionalidades (por ejemplo, exportación a PDF o compatibilidad con más tipos de diagrama).
- **Documentación interna:** Se recomienda mantener actualizada la documentación técnica de los componentes y registrar cambios relevantes en el archivo `CHANGELOG.md`.

A.3. Manual de usuario

Project Master es una aplicación SPA (Single Page Application) que permite gestionar flujos de trabajo mediante diagramas interactivos Mermaid. Está compuesta por tres vistas principales:

1. Menú principal de selección de proyecto

La pantalla de inicio permite seleccionar un proyecto dentro de un departamento (Electrical, Mechanical, Software) o bien cargar uno personalizado.

- Cada proyecto aparece representado como una tarjeta visual.
- Los iconos superiores permiten acceder a funciones como editar, cargar o consultar ayuda.
- El botón inferior permite cargar un archivo Mermaid propio.

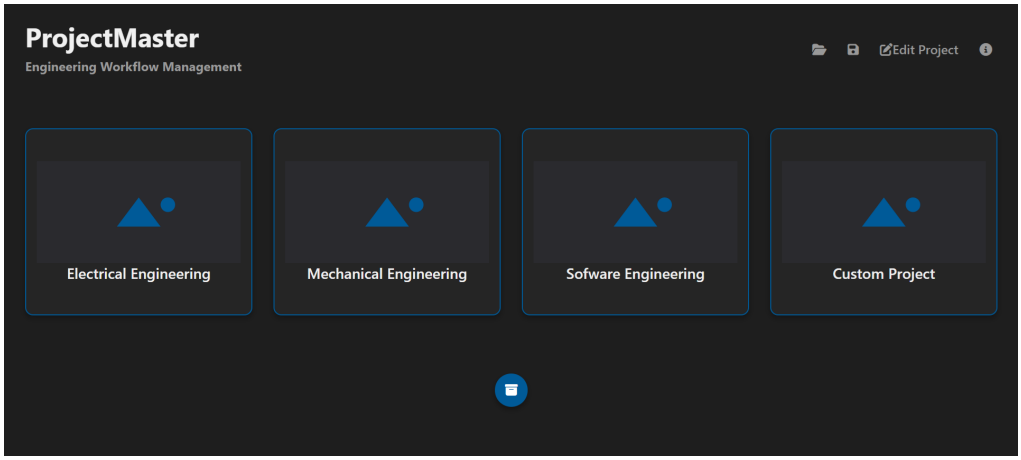

Figura A.1: Vista inicial de selección de proyecto

2. Modo visualización

Esta vista permite navegar por el diagrama, completar tareas de los nodos y consultar la documentación asociada.

- Visualización del flujo Mermaid.
- Panel lateral con tres pestañas: **Description**, **Checklist** y **Documentation**.
- Botones `Undo`, `Redo`, `Reset Progress` y `Center the Diagram`.

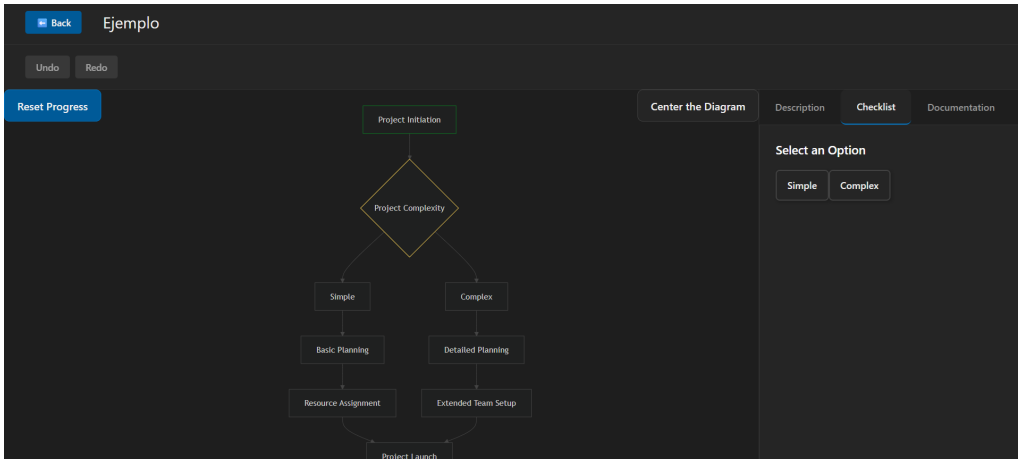

Figura A.2: Modo visualización con flujo y panel lateral de documentación

3. Modo edición

El editor permite crear y modificar diagramas Mermaid con una vista previa en tiempo real.

- Área izquierda con **Monaco Editor** y selector de tema visual.
- Área derecha con renderizado interactivo del grafo.
- Panel lateral con pestañas para editar descripción, checklist o documentación por nodo.
- Botones para guardar, cargar y reiniciar el contenido.

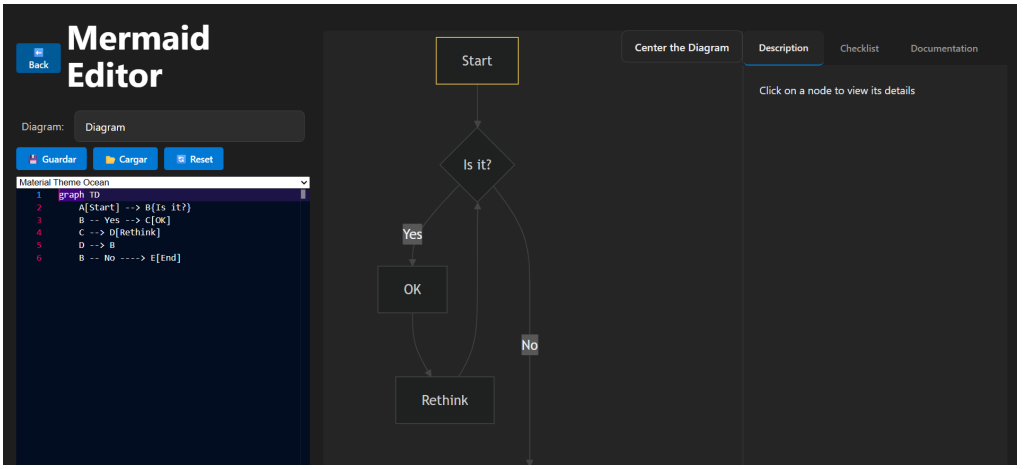

Figura A.3: Modo edición con código Mermaid y vista previa

Flujo de uso recomendado

1. Seleccionar o cargar un diagrama desde el menú principal.
2. Navegar por el flujo visualizando los nodos y completando tareas.
3. Acceder al editor para modificar o crear nuevos diagramas.
4. Exportar, guardar o documentar los resultados para su uso posterior.

Apéndice B

Resumen de enlaces adicionales

Los enlaces útiles de interés en este Trabajo Fin de Grado son:

- Repositorio del código: `https://gitlab.inf.uva.es/diegonz0/tfg_diego_gonzalez_guerra_project_maste.git`.
- descarga de Node.js para poder ejecutar el programa: `https://nodejs.org/es/download`

Webgrafía

- `https://git-scm.com/`
  Página oficial de Git, sistema de control de versiones distribuido.
- `https://github.com/`
  Plataforma de alojamiento de repositorios Git y herramientas colaborativas.
- `https://react.dev/`
  Documentación oficial de React, biblioteca para construir interfaces de usuario.
- `https://vitejs.dev/`
  Sitio oficial de Vite, herramienta de desarrollo frontend de nueva generación.
- `https://developer.mozilla.org/es/docs/Web/JavaScript`
  Documentación de JavaScript en MDN Web Docs.
- `https://nodejs.org/es/`
  Página oficial de Node.js, entorno de ejecución de JavaScript en servidor.
- `https://mermaid.js.org/`
  Documentación de Mermaid, lenguaje de marcado para generación de diagramas.

- `https://microsoft.github.io/monaco-editor/`
  Repositorio oficial de Monaco Editor.
- `https://www.typescriptlang.org/`
  Página oficial de TypeScript.
- `https://www.latex-project.org/`
  Página oficial del proyecto LaTeX.
- `https://astah.net/`
  Página oficial de Astah, herramienta de modelado UML.
- `https://www.avl.com/`
  Página corporativa de AVL, empresa colaboradora en el proyecto.

Glosario

Sigla	Descripción
AVL	Anstalt für Verbrennungskraftmaschinen List. Empresa de ingeniería especializada en automoción, colaboradora en este TFG.
TFG	Trabajo Fin de Grado.
SPA	Single Page Application. Aplicación web de página única.
UX	User Experience. Experiencia de usuario.
UI	User Interface. Interfaz de usuario.
API	Application Programming Interface. Interfaz de programación de aplicaciones.
DOM	Document Object Model. Modelo de objetos del documento.
ESM	ECMAScript Modules. Módulos estándar de JavaScript.
ESLint	Herramienta de análisis de código para identificar problemas en JavaScript/TypeScript.
HMR	Hot Module Replacement. Reemplazo en caliente de módulos.
JSON	JavaScript Object Notation. Formato de intercambio de datos.
SVG	Scalable Vector Graphics. Formato vectorial de gráficos escalables.
RAM	Random Access Memory. Memoria de acceso aleatorio.
CPU	Central Processing Unit. Unidad Central de Procesamiento.
VSC o VS Code	Visual Studio Code. Editor de código de Microsoft.
RNF	Requisito No Funcional.
HU	Historia de Usuario.
JSX	JavaScript XML. Extensión de sintaxis que permite escribir HTML en JavaScript.
JSON	JavaScript Object Notation. Formato de intercambio de datos basado en texto.
ViteJS	Herramienta moderna de construcción y desarrollo frontend.
npm	Node Package Manager. Gestor de paquetes de Node.js.
Git	Sistema de control de versiones distribuido.
GitHub	Plataforma de alojamiento de repositorios Git.
HOC	Higher Order Component. Patrón avanzado en React.