UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya (UPC) – BarcelonaTech
Facultat d'Informàtica de Barcelona (FIB)
Centre Internacional de Metodes Numerics en Enginyeria (CIMNE)
Departament de GiD

DESARROLLO DE ENTORNO DE PROGRAMACIÓN ONLINE DENTRO DE APP ANGULAR

Trabajo Fin de Grado
Grado en Ingeniería Informática - Computación

Laura Santos López

Director: Adrià Melendo Ribera
Ponente: Mercè Mora Giné

15 de enero del 2022

Resumen

El proyecto está dirigido a desarrollar un conjunto de funcionalidades complementarias en la aplicación web que la empresa CIMNE está implementando. Esta aplicación se utilizará para ejecutar simulaciones online con interfaz completamente personalizable tanto en aspecto como en funcionalidad.

Una de las funcionalidades complementarias consiste en crear un sistema que permita programar de manera online, generando funciones y componentes HTML5 personalizados que las activarían. A este sistema se le han añadido ayudas adicionales para facilitar la programación al usuario. Algunas de estas herramientas consisten en métodos para poder gestionar los ficheros, poder ver las versiones anteriores y ayudas de autocompletado.

La otra funcionalidad permitirá ejecutar todo aquello que se ha programado y ver que funciona como el usuario espera antes de guardarlo e incorporarlo definitivamente a la aplicación.

Dado que este proyecto ampliaría una aplicación web, el lenguaje de programación que se utilizará es JavaScript en el framework de Angular con algunos módulos adicionales.

Abstract

This project is focused on developing a set of new complementary functionalities in the web application that CIMNE is implementing. This application will be used to run online simulations with a fully customizable UI(User Interface) in both looks and functionality.

One of the featured additions consists of creating a system that allows users to program their custom functions online and the HTML5 components that would apply them. Into this system, additional tools have been added to make development easier for users. Some of these features consist of methods that allow file management, version control or autocompletion.

Another functionality allows the execution of everything that has been developed. This way, users can check if everything works accordingly before saving or incorporating the files into the final project.

Given that this project is an extension of a web application, the programming language that is used during development is Javascript using an Angular Framework with some additional modules.

Índice

Índice de Tablas

Índice de Figuras

1. Contextualización del proyecto

1.1. Introducción

Siempre ha habido debates en torno a las aplicaciones de escritorio o las aplicaciones web, ¿cuál es mejor? Hasta ahora se podían encontrar argumentos para todo lo que podía aportar una aplicación web que no tenía una aplicación de escritorio y viceversa.

Hoy se puede ver que algunas de estas diferencias que favorecen a las aplicaciones de escritorio, como la velocidad de ejecución, se están recortando gracias a las nuevas tecnologías. Esto, unido a la situación excepcional que se ha vivido, ha hecho que la portabilidad de las aplicaciones sea más valorada, lo que permite que un software concreto y las librerías específicas dejen de ser tan imprescindibles. Por este motivo, muchas empresas han decidido dar el paso y migrar algunas de sus aplicaciones de escritorio a un formato nuevo y online que favorece la conexión desde diferentes lugares.

Este trabajo de fin de carrera corresponde al Grado en Ingeniería Informática de la Facultad de Informática de Barcelona (FIB). Dentro de este Grado podemos encontrar cinco especialidades, este trabajo corresponde a la mención de Computación, dentro del campo del diseño de soluciones informáticas a problemas, adaptándose a las herramientas del momento, utilizando los fundamentos aprendidos en la carrera.

Con el avance de la tecnología, las herramientas están cambiando y lo que existe ahora, quizás dentro de 10 años, estará desactualizado, pero los fundamentos seguirán siendo los mismos.

El TFG puede tener diferentes modalidades, dependiendo de si se realiza en el centro, en empresas o en el extranjero. En este caso estamos hablando de la modalidad B (trabajo realizado en una empresa). La empresa que tutorizará el

trabajo es CIMNE[1] (Centre Internacional de Mètodes Numèrics en Enginyeria) [1] concretamente el departamento de GiD [2], este nombre proviene de las palabras Geometria i Dades. Hay que aclarar que cuando se menciona GiD no solo se hace referencia al departamento de CIMNE, comentado anteriormente, sino también al programa creado por este departamento que recibe su mismo nombre, GiD.
Se quiere migrar la propia aplicación GiD (que es un programa que permite a sus usuarios ejecutar simulaciones numéricas) de escritorio a un formato web y añadir nuevas funcionalidades. Algunas de estas funcionalidades son las que corresponden a este trabajo de final de Grado.

1.2. Definición de conceptos

Hay algunos conceptos que aparecerán más adelante en la memoria cuando se hable en profundidad de la aplicación. A continuación, se definirán estos conceptos para comprender mejor las explicaciones del proyecto.

El primer concepto es "**simulación numérica**" [3], que utiliza herramientas matemáticas para simular, modelar o predecir comportamientos en las cosas. Se utilizan normalmente en fenómenos mecánicos o estructurales, termodinámica, acústica, estructuras que tienen interacción con fluido, etc.

También es necesario saber que en la aplicación GiD hay dos fases [4], el "**pre-procesado**", que consiste en el tratamiento que se hace a los datos antes de hacer la simulación numérica, y el "**post-procesado**", que consiste en la visualización

[1] CIMNE: organización de investigación creada en 1987 en el seno de la prestigiosa Universidad Politécnica de Cataluña (UPC) como colaboración entre la Generalitat de Cataluña y la UPC, en colaboración con la UNESCO. El objetivo de CIMNE es el desarrollo de métodos numéricos y técnicas computacionales para el avance del conocimiento y la tecnología en ingeniería y ciencias aplicadas.

de los resultados. En esta segunda fase se pueden visualizar los datos por colores, curvas de nivel, etiquetas, vectores, gráficos, animaciones, etc.

En el ámbito de la aplicación GiD aparecen los conceptos de "**ProblemType**" y "**WorkingMode**". El ProblemType es un conjunto de archivos creados por un desarrollador externo para preparar o procesar los datos para su análisis de una manera específica [5]. El WorkingMode, que son los diferentes tipos de trabajo que tiene la aplicación.

Otro concepto que se mencionará es el "**Framework**", que es una estructura base utilizada como punto de partida para crear proyectos, una especie de plantilla. En este caso se necesita un framework para aplicaciones web y se ha decidido utilizar "**Angular**" [6], que utiliza el lenguaje de programación TypeScript.

Se hablará de "**Depurar**", que consiste en ejecutar un código en busca de errores, como representa la figura 1. Existen herramientas para facilitar este proceso, como los puntos de parada con los que se agiliza la localización de errores.

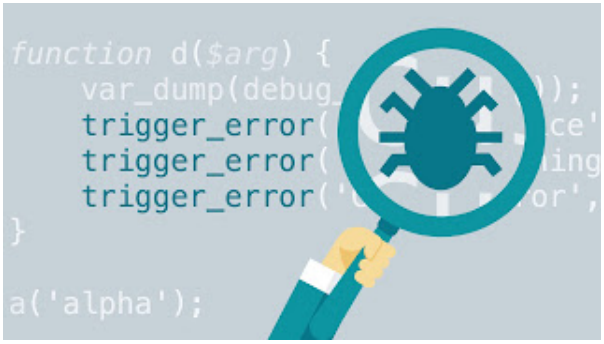


Figura 1 : Representa un código en el que se están buscando errores. La imagen se ha extraído de [7]

En el área de la programación las siglas "**IDE**" corresponden a "**Entorno de desarrollo integrado**" [8]. Son aplicaciones que proporcionan servicios para

facilitar el desarrollo de software. Normalmente están formadas por un editor de código fuente, herramientas de construcción automáticas y un depurador.

Finalmente, "**Commit**" corresponde a un paso dentro del ciclo básico del trabajo con Git[2]. Es equivalente a la acción de guardar, como la de cualquier programa informático.

1.3. Actores implicados

Tratándose de un proyecto relacionado con una empresa, los principales implicados son los propios miembros del departamento de GiD junto con el alumno que realizará el TFG. Dentro de la empresa se hace mención especial a Adrià Melendo Ribera que es el director del trabajo, supervisa, monitoriza y guía al estudiante. El estudiante que desarrolla el trabajo es Laura Santos López.
También están implicados los profesores de la universidad que guiarán el TFG y asesorarán al director para que el trabajo cumpla con los requisitos necesarios, especialmente la profesora Mercè Mora Giné que es la ponente del trabajo.

Las funcionalidades que se quieren desarrollar van dirigidas a todas las personas que utilicen el programa de GiD, especialmente a los usuarios actuales como estudiantes, ingenieros y científicos, pero con este proceso también se intenta expandir los horizontes, proporcionando innovaciones en la aplicación para poder llegar a un abanico de posibles usuarios más amplio. Dentro de los usuarios se pueden identificar los que ejecutan simulaciones estándar, y los que crean sus propios ProblemTypes con "solvers", que son programas que resuelven ecuaciones específicas del propio usuario. La parte en que se centra este trabajo favorece al

[2] Git [9] es un sistema de control de versiones de código abierto y gratuito diseñado para manejar todo, tipo de proyectos pequeños y grandes, con velocidad y eficiencia.

segundo tipo de usuarios, siendo capaz de unir su solver con la parte gráfica de la nueva aplicación.

Si esta nueva web es un éxito, los primeros beneficiados serán el departamento de GiD y a su vez la empresa de CIMNE, dado que se comprarán más licencias. Pero estos no serian los únicos beneficiados, ya que si los usuarios pagan por un servicio que necesitan, en cierta manera también están sacando provecho de todas las opciones que la aplicación les proporciona.

2. Alcance del proyecto

El alcance es el conjunto de requisitos que se tienen que cumplir en el proyecto para que se considere terminado. Esto lo convierte en un aspecto muy importante que hay que tener bien definido para poder cumplir todos los plazos y que no se alargue en exceso por profundizar demasiado en algunos aspectos y quedarse corto en otros.

2.1. Identificación del problema

Dentro del ámbito de la Ciencia y de la Ingeniería es necesario hacer simulaciones numéricas. La aplicación de escritorio de GiD intenta cubrir todas las necesidades comunes de estas simulaciones desde el punto de vista de pre y post procesado. Algunas de estas necesidades son el modelado geométrico, la definición efectiva de datos de análisis, mallados, transferencia de datos a softwares de análisis y la visualización y análisis de estos resultados numéricos.
GiD ofrece muchas opciones de trabajo, entre otras permite crear entornos dentro de la aplicación con funcionalidades propias y cálculos específicos para el trabajo que se está realizando. Esto es conocido como ProblemType.

La empresa ha decidido migrar esta aplicación a un entorno web para facilitar a los usuarios conectarse desde donde lo deseen sin tener que preocuparse del espacio físico en el que tienen guardado el trabajo. Al dar este paso surge el problema o necesidad de pensar una fórmula en la que se puedan crear los ProblemTypes de manera fácil, cómoda y eficiente. En la aplicación actual de escritorio, el usuario crea un directorio especifico donde introduce todos los ficheros que ha programado anteriormente en su ordenador y que la aplicación GiD necesita para personalizar el espacio de trabajo. Esta dinámica se intenta agilizar y facilitar con la nueva aplicación. El trabajo pretende resolver esta necesidad dedicando un espacio dentro de la nueva aplicación web, para crear un editor. Una posibilidad para este nuevo apartado es que tenga dos funcionalidades principales, una sería la edición y la otra la ejecución.

La parte de edición quiere conseguir un entorno cómodo, con ayudas y con un prototipo inicial de gestión de versiones, donde el usuario pueda crear las funciones de sus ProblemTypes. Además, se quiere incorporar una segunda funcionalidad complementaria a la primera, permitiendo que el usuario pueda ejecutar parte del trabajo que está haciendo para comprobar si funciona correctamente.

2.2. Justificación

Anteriormente, se ha explicado el problema a resolver, como se gestiona actualmente y cómo se afrontará, pero lo que no se ha explicado son los motivos que han llevado a esa solución. Para poder entenderlo hay que situarse en el área de las simulaciones numéricas y en el mercado actual.

En internet se pueden encontrar algunos programas de escritorio que proporcionan al usuario una interfaz interactiva para el pre-procesado y/o post-procesado de modelos. La mayoría de estas aplicaciones vienen con su propio solver o motor de procesado, estos suelen estar centrados en los ámbitos de la ciencia que estudian, con funciones específicas para calcular y analizar datos.

También se puede encontrar la situación inversa, empresas o paginas web que proporcionan los solvers, pero sin incluir la parte gráfica del pre-procesado y/o post-procesado.

Algunos de los competidores de GiD son los programas de **Tecplot** [10] y **ParaView** [11] que solo proporcionan el post-procesado. Otros programas más parecidos a GiD son **Femap** [12] y **FEMGV** [13], ambos proporcionan pre y post-procesado de modelos de análisis. Femap permite combinarlo con sistemas CAD (Diseño Asistido por Ordenador), hacer simulaciones y combinar solvers ya existentes.

Lo que diferencia a la aplicación de GiD de los programas anteriores, es que ofrece a sus usuarios un entorno gráfico con un solver y la posibilidad de crear tu propio solver e integrarlo con la interfaz de GiD para visualizar los datos y soluciones de

las simulaciones. Además, proporciona plena libertad a la hora de decidir el tipo de entrada y salida de los datos, el programa se adapta totalmente a las necesidades de formato de los datos del usuario.

También se ha evaluado el mercado al que se quiere acceder con la aplicación web, en este caso no se han encontrado tantas aplicaciones, solo se mencionará **OnShape** [14]**.** Proporciona un entorno CAD avanzado, pero la parte de simulaciones se tiene que añadir con un módulo. La aplicación web tiene una tienda en la que se pueden comprar y descargar solvers para complementar la aplicación, **Simscale** [15] sería el más parecido a GiD. Al ejecutar la simulación se abre una pestaña aparte, esto hace que la simulación no se vea muy integrada en la aplicación. Esto con la nueva aplicación web GiD no pasará, todo se verá muy unificado y el conjunto parecerá una aplicación personalizada.

Alguna de las preguntas claves que se tienen que responder antes de empezar el proyecto son las siguientes. ¿Está el problema ya resuelto?, ¿Se puede utilizar o adaptar una solución existente o hay que diseñar una solución nueva?
Para resolver estas preguntas se dedicó tiempo a investigar las soluciones existentes y programas parecidos para ver como afrontaban el problema.
Se ha llegado a la conclusión de que el departamento de GiD quiere una solución muy especifica para complementar su programa. Se ha dividido el alcance del proyecto en requisitos más comunes que pueden ser solicitados por cualquier empresa o usuario. De esta manera se han encontrado diferentes soluciones ya existentes para cumplir con los requisitos propuestos que denominaremos módulos.

Uno de los módulos es el editor. Después de unos días de búsqueda se encontraron tres opciones que se podían adaptar a nuestras necesidades. Estos editores son "Runkit" [16], "ACE" [17] y "Monaco" [18]. El primero está muy centrado en la ejecución y es complicado adaptarlo a nuestras necesidades. Con el segundo no ocurre lo mismo, es fácil de adaptar, pero no es compatible con un editor que

resalta las diferencias de código en distintas versiones, por tanto se ha decidido utilizar "Monaco Editor".

El siguiente módulo a comentar es el que corresponde al gestor de versiones. Una de las opciones es crear una API propia que gestione los datos de los ficheros con sus versiones, pero este proceso consume demasiado tiempo y se ha decidido buscar una ya existente, como las que proporcionan GitHub [19] o GitLab [20]. En un futuro es posible que se cree una API propia para CIMNE, pero de momento se ha decidido utilizar la API de GitLab [21], ya que la propia empresa lo tiene instalado y ejecutándose en uno de sus servidores.

La última decisión que se ha tomado hace referencia a la parte de ejecución y visualización. Se comprobará que el código introducido por el usuario no tiene errores y, en caso de que los tuviera, la propia aplicación los destacará para que sean corregidos. La parte de visualización se ha hecho con la misma dinámica que se utiliza en los "WorkingModes". Esta dinámica consiste en utilizar unas clases que crean diferentes Objetos JSON que se transformarán a componentes de Angular para su visualización.

2.3. Definir los objetivos

Al ser un trabajo realizado en colaboración con la empresa CIMNE en consorcio con la UPC, el objetivo final a conseguir está muy definido desde el principio. Consiste en que los usuarios que utilizarán la nueva aplicación web de GiD, que está en desarrollo, consigan crear funcionalidades en sus ProblemTypes.
Como el proyecto es de gran envergadura, se empezará definiendo los requisitos básicos que se identificarán con objetivos secundarios.

Uno de los objetivos secundarios más destacados, será conseguir un editor funcional, tarea que también puede resultar compleja ya que un editor implica muchas funcionalidades a desarrollar, por lo que también pondremos ciertos limites. La versión principal del proyecto debe contar con una parte donde escribir código, ha de tener la capacidad de deshacer y de rehacer, debe facilitar las ayudas de autocompletado y también ha de contar con botones con las opciones de abrir, guardar y eliminar ficheros.

Otro objetivo secundario es tener un entorno en el que el usuario sea capaz de gestionar las diferentes versiones. Con este objetivo pasa lo mismo que en el anterior, al hablar de un gestor de versiones, directamente se piensa en Git y cómo trabajar simultáneamente en un mismo proyecto. La idea de futuro es conseguir que GiD web permita trabajar a varios usuarios en un mismo fichero de ProblemTypes igual que Git.
Al finalizar este trabajo se quiere conseguir una primera versión que muestre un listado de las modificaciones anteriores de un fichero y que permita compararlas.

Por último, se ha de verificar la ejecución del código del usuario, comprobando que no tenga errores y realmente funciona como el usuario espera. Si el código tiene errores se mostrarán en la aplicación lo que permitirá una rápida corrección.
Este trabajo podrá ser ampliado en un futuro añadiendo puntos de parada y una forma de poder depurar el código, pero este apartado ya no forma parte del TFG.

Una vez comentados el objetivo principal y los objetivos secundarios que son la mayor parte de los requisitos funcionales, falta comentar otro requisito funcional que para la empresa son muy importantes. Concretamente, que el trabajo realizado por el alumno se pueda integrar bien en la aplicación web que el departamento de GiD está creando.

Otros requisitos no funcionales que han sido impartidos a lo largo de la carrera serian, que no solo es importante que la aplicación funcione y tenga todo lo que el usuario necesita o lo que la empresa pide, sino conseguir también la eficiencia, usabilidad y re-usabilidad.

Si una aplicación tiene muchas funcionalidades, pero es muy compleja de utilizar, o va lenta, o los diferentes menús están muy escondidos y no son intuitivos, es muy posible que los usuarios busquen otra aplicación que, a pesar de no ofrecer tantas funcionalidades, sea más fácil de usar y que se ejecute más rápida. Por este motivo el trabajo también se centrará en que el usuario tenga una buena experiencia cuando utiliza la aplicación centrándose en la eficiencia y usabilidad.
La reusabilidad y modularización de los diferentes componentes que serán creados a lo largo del trabajo, también son factores muy importantes, ya que es posible que en un futuro se quiera incorporar el componente de gestión de versiones para los modelos y simulaciones, y resultaría muy costoso si no se han modularizado bien las diferentes partes a la vez que generaría mucho código duplicado.

2.4. Posibles riesgos y obstáculos

Uno de los principales riesgos en todo proyecto que suele aparecer en la fase inicial, es fijarse más objetivos de los que se pueden cumplir dentro del plazo estimado. El impacto de este riesgo es bajo dado que se podría reducir el tiempo de verificación si fuera necesario, pero para prevenir esta situación se ha hecho una planificación temporal cuidadosa y detallada.

Para este proyecto se han tomado diferentes decisiones técnicas explicadas anteriormente respecto a utilizar algunos paquetes ya existentes. Si durante el curso del trabajo resulta que no se pueden adaptar o limitan demasiado para el fin esperado, se tendrían que buscar otros paquetes o simplemente implementar los componentes uno mismo, y eso conllevaría una pérdida de tiempo muy grande. El

impacto de fallar en alguna de estas decisiones es alto, dado que cambiaría el rumbo del proyecto demasiado cerca de la fecha de entrega. Para prevenirlo se estudiarán detenidamente las diferentes opciones para poder tomar las decisiones con criterio y acierto. También se ha dejado un poco más de tiempo en las tareas de programación para solucionar imprevistos.

No solo existen esos riesgos, sino que también hay un par de obstáculos, el desconocimiento y la inexperiencia. Los dos van cogidos de la mano, porque el estudiante no había tratado hasta ahora con el Framework de Angular y había tocado cómo parte complementaria de alguna asignatura el diseño de páginas web y los lenguajes de programación TypeScript y JavaScript. Esto hace que el estudiante tenga que dedicar un pequeño esfuerzo extra para que la curva de aprendizaje sea lo más pronunciada y rápida posible.
Se considera que estos obstáculos conllevan un impacto mediano-bajo dado que en la planificación se ha tenido en cuanta este tiempo de aprendizaje. Gracias a la asignatura de Lenguajes de Programación impartida en la universidad, son obstáculos alcanzables a pesar de que requieren un esfuerzo extra. Por si acaso el tiempo reservado no fuera suficiente se ha decidido añadir algo más de tiempo extra para prevenir el desconocimiento y la inexperiencia de las herramientas.

3. Metodología y rigor

Para realizar este trabajo, el estudiante ha tenido que adaptarse a las metodologías que utilizaba la empresa. Primero hay una fase de autoaprendizaje y adaptación al Framework de Angular y al lenguaje de programación web.

Cómo estamos hablando de un trabajo de creación que también contendrá una pequeña fase de investigación para encontrar la mejor manera de hacer las cosas, se necesitará una metodología flexible. La empresa ya trabaja con una metodología ágil que permite adaptar la forma de trabajo a las condiciones del proyecto y gestionar la realización de las tareas. Este tipo de metodología es el kanban [22], que es un método visual formado por un tablero continuo que se divide en columnas. Cada columna representa una actividad específica que, en conjunto, conforman un "flujo de trabajo". Las tarjetas van moviéndose por el flujo de trabajo hasta que este termina. En este caso los flujos de trabajo corresponden a "next in line", "in progress", "in review" y "done". Cada una de las tareas a realizar se representa en tarjetas. Esta distribución se puede ver en la siguiente imagen.

Figura 2 : Representa el tablero de kanban del 27 de septiembre del 2021. Elaboración propia.

Esto permite dividir el trabajo en partes, teniendo muy claro que conlleva cada una de las tareas y se visualizarán todas a la vez para que se prioricen las que están en curso. Al principio del trabajo se ha hecho una reunión con el director del TFG en

la cual se acordaron y definieron las diferentes tareas que se tendrían que realizar. Una vez definidas las tareas el estudiante creó una tarjeta por cada tarea y se colocaron en la columna de "next in line". Poco a poco se fueron empezando las tareas que se movían a la columna de "in progress". Una vez terminada la tarea se situaba en la columna de "in review" y si el trabajo realzado era correcto, se pasaba a la columna de "done" y se empezaba con la siguiente tarea de la cola.

Esto va de la mano de unas reuniones semanales de seguimiento para ver si los objetivos se van cumplido y resolver dudas que vayan surgiendo. La comunicación con el tutor del trabajo es constante, ya que se ha trabajado en la misma empresa y departamento de manera presencial con el tutor durante la ejecución del proyecto. Dentro de lo que las medidas Covid permiten en cada momento, así la comunicación se realiza en el día a día del trabajo.

3.1. Herramientas

Para poder llevar a cabo la metodología anterior se ha decidido utilizar las herramientas que la empresa Atlassian [23] proporciona.
Las herramientas que se utilizarán son "Jira" que permiten crear tableros y gestionar las tareas, "Confluence" que se utiliza para compartir documentos y redactarlos de manera online colaborativa y la aplicación de "Bitbucket" sirve para crear repositorios y almacenar el código que junto a "Sourcetree" [24] se obtiene una manera más simple de control de versiones. Gracias a esta interfaz se evita tener que aprender todas las órdenes de gestión de versiones Git que se tendrían que introducir por terminal.

Con la situación actual también ha sido necesario buscar nuevas herramientas para poder mantener la comunicación entre los miembros del equipo y esto ha hecho que se utilicen otras aplicaciones cómo "Slack" [25], los "Meets de google" [26], "AnyDesk" [27] y otras opciones de escritorios remotos.

No hay que olvidar otros programas que son necesarios para realizar el TFG. Algunos de estos son "Word" [28], "PowerPoint" que se utilizarán para redactar la memoria y preparar la exposición. También se utilizará el "Visual Studio Code" [29] para programar y la aplicación "Gantter" [30] para planificar y hacer el diagrama de Gantt.

3.2. Validación

Para verificar que se cumplen con los requisitos funcionales y no funcionales, se realizarán diferentes tipos "de test" o pruebas.

Los tests se diferenciarán en dos fases, una de ellas es la corrección y la otra es la usabilidad. Para la parte de corrección se emplean tests e2e y unitarios. Los tests e2e [31] comprueban que la aplicación funciona correctamente generando unos flujos de datos que se moverán por la plataforma y se comprobará que el estado y la ejecución de la web corresponda a las acciones realizadas. La otra forma de testear es la unitaria, que comprueba si un componente en particular actúa cómo debería por sí solo, sin tener en cuenta el entorno. Con estos test se podrá ver si los requisitos funcionales han sido satisfactoriamente resueltos, para ello se ha tenido que crear la funcionalidad relacionada a ese requisito y han tenido que pasar los test.

La otra fase cuenta con ayuda externa. Se enseñará el trabajo realizado a miembros de la empresa para que lo utilicen y de esta manera se espera que se proporcione información útil, tanto de errores cómo posibles mejoras de cara al usuario. Esta es la manera en la que se evaluará si el proyecto cumple con los requisitos no funcionales, en base a la opinión de los usuarios.

4. Planificación inicial

Todo proyecto ha de tener una buena planificación para poder conseguir todos los objetivos que se proponen antes de la fecha límite. Por este motivo, a continuación se entrará más en profundidad sobre las tareas que tiene el trabajo, se explicará en qué consiste cada una de las tareas teniendo en cuenta los materiales necesarios y el tiempo asignado para terminar.

El tiempo está calculado en horas, pero en el diagrama de Gantt aparece puesto por días, porque la jornada laboral dentro de la empresa es de 5 horas diarias sin tener en cuenta el fin de semana que se centrará en escribir la memoria

4.1. Descripción de las tareas

La idea del proyecto surgió hacia el mes de junio. El tema fue propuesto por parte de la empresa, se buscó un tutor dentro de la facultad y entonces empezaron los trámites del convenio cooperativo educativo. Por este motivo se considera que el trabajo ha empezado el día 1 de julio del 2021, día en que se hizo efectivo el convenio que ha terminado el 16 de enero del 2022, dado que el turno de lectura ha de ser el siguiente al de finalizar el convenio.

A continuación se explican todas las tareas, que se especifican en la Tabla 1 del apartado "Recursos". Esta tabla contiene además la estimación de tiempo y el material necesario para cada tarea. Las tareas de la Figura 29, 30 y 31 corresponden a la planificación inicial del proyecto que aparece en el **Anexo A**.

Todas las tareas se han dividido en seis ámbitos diferentes, el de **Documentación** (D), el de **Reuniones de seguimiento** (S) y por último los ámbitos relacionados con la creación que son la **Preparación previa** (CP), el **Diseño** (CD), la **Implementación** (CI) y finalmente las **Pruebas** (CT).

D – Documentación

Dentro del módulo de la documentación se pueden detectar tres tareas diferentes, las entregas de la asignatura gestión de proyectos, la documentación del TFG y la preparación de la presentación final.

D1 - Entregas de GEP

Al principio del cuatrimestre se imparte una asignatura que ayuda a Gestionar la elaboración del TFG. Esta asignatura dura un mes, durante el cual hay que presentar cuatro entregables. En la primera entrega hay que hablar del contexto y el alcance del proyecto, la segunda es la planificación temporal, la tercera el presupuesto y sostenibilidad y por último hay que elaborar un documento que contenga todo lo anterior, aplicando las correcciones que los profesores han hecho a lo largo de la asignatura. La estimación en horas de esta asignatura, según la guía docente de la misma, es de 90 horas.

D2 - Escribir memoria

Esta tarea consiste en aplicar las correcciones de la entrega 4 de GEP y seguir complementando a lo largo del desarrollo del proyecto documentando las distintas fases. Esta tarea se hará de manera paralela al proyecto aprovechando los fines de semana para documentar todo lo que se ha hecho durante la semana. Se estima que esta tarea durará unas 60 horas.

D3 - Preparar presentación

En esta tarea se tendrá que crear un PowerPoint que se utilizará durante la exposición y se practicará la exposición con el tutor de la empresa y el de la universidad. Se estima que la duración será de unas 20 horas.

S - Reuniones de seguimiento

S1 - Reunión inicial

Esta reunión se hace antes de inscribir el TFG, es donde el tutor de CIMNE una vez presentado el tema y aceptado por parte del alumno, explica claramente y con detalle las tareas y objetivos del trabajo. Se estima que serán unas 3 horas.

S2 - Reuniones seguimiento semanales

Cada semana se hace una reunión de seguimiento para evaluar si se han alcanzado los objetivos de esa semana y poner nuevos para la siguiente. Se estima que cada reunión durará unos 30 minutos, que a lo largo de todo el trabajo serán unas 15 horas.

S3 - Reunión hito intermedio

Durante el TFG hay que hacer una reunión para evaluar el estado de los objetivos cuando se estime que estará hecho la mitad del trabajo. Se considera que se tardarán 2 horas.

S4 - Reunión final

Un par de semanas antes de la fecha límite de entrega de la memoria se hace esta reunión para validar que se han cumplido todos los objetivos y corregir cualquier posible error. Se estiman 2 horas.

CP - Preparación previa

Todo trabajo tiene una parte previa de preparación en la que se definen las tareas, se planifica el trabajo, se hace una labor de adaptación a todas las herramientas nuevas que se utilizarán durante el trabajo y se prepara el entorno de trabajo.

CP1 - Definir alcance

Una vez ha tenido lugar la tarea S1, el alumno tiene que pensar en la información necesaria para inscribir el TFG. Para eso se debe tener los objetivos bien definidos para relacionarlos con las competencias técnicas, hacer una buena descripción del trabajo y buscar un título. Se estiman unas 10 horas.

CP2 - Planificación temporal

Se divide el objetivo principal en objetivos secundarios y estos en las tareas, una vez hecho esto se decide cuanto tiempo se le dedicará a cada una de las tareas. Para poder realizar CP2, primero se ha tenido que hacer CP1. Se estiman unas 10 horas.

CP3 - Presupuesto y análisis de sostenibilidad

Se hará un presupuesto para cuantificar el coste del proyecto y un análisis de sostenibilidad. Con esta tarea se quiere prever qué recursos serán necesarios y que roles de personas estarán implicados, a pesar de necesitar diferentes roles el estudiante actuará cómo la mayoría de ellos dado que es un trabajo individual. Se estiman unas 15 horas.

CP4 - Aprendizaje Angular y TypeScript

Buscar información de Angular y realizar algún curso en Udemy [32], que es una plataforma de cursos online, sobre el funcionamiento de Angular y cómo programar en TypeScript. Se estiman unas 40 horas.

CP5 – Preparar el entorno

En esta tarea se descargarán todos los programas y paquetes necesarios para la ejecución y compilación del código. Bajar el proyecto del repositorio y preparar todo para poder trabajar con el gestor de versiones. Se estiman unas 10 horas.

CD – Diseño

En todo proyecto hay una fase de investigación, se expone el problema a resolver y se valoran y buscan diferentes soluciones. En este caso también se busca la mejor manera de poder unificar y conectar los siguientes módulos.

CD1 - Diseño del editor

Dentro de los paquetes de posibles editores, buscar información de cada uno y compararlos para encontrar la mejor solución. Se estiman unas 12 horas.

CD2 - Diseño del gestor de versiones

Investigar las diferentes maneras de hacer el gestor de versiones, decidir la mejor y definir cómo será la interacción con el resto de los elementos de la aplicación. Se estiman unas 16 horas.

CD3 - Diseño del modo Debug

Evaluar las diferentes opciones para crear el modo Debug y cómo incorporarlo en la nueva aplicación. Se estiman unas 16 horas.

CD4 - Diseño de la interfaz Gráfica

Definir qué aspecto tendrá la aplicación, los componentes que se mostrarán y la funcionalidad que desempeñarán. Para esta tarea se estiman unas 16 horas.

CI – Implementación

CI1 - Crear el editor

Crear el componente editor que alterna entre dos tipos. Uno es una pantalla entera de editor con diferentes funciones, cómo la de autocompletar cuando se escribe, la

de deshacer y rehacer, etc. El otro tipo de editor consiste en dos pestañas de editor normales una junto a la otra y marca las diferencias entre lo que se escriba en un editor con respecto al otro. Se estiman unas 50 horas.

CI2 - Crear línea temporal

Crear el componente de gestión de versiones utilizando el api de GitLab para obtener el código de versiones anteriores y mostrarlo en el editor. Se estiman unas 25 horas.

CI3 - Implementar interacciones con GitLab Api

Crear la barra de herramientas que permita abrir, guardar, eliminar y cerrar los ficheros del editor. Para esto se necesita conectar con el api de GitLab para poder hacer todas estas funciones. Se estiman unas 50 horas.

CI4 - Implementar el modo Debug

Ejecutar el código que el usuario introduce en el editor, detectar errores y mostrar la visualización gráfica por pantalla. Se estiman unas 75 horas.

CT – Pruebas

CT1 - Pruebas unitarias

Probar cada componente por separado para ver que la funcionalidad es correcta. El número de horas depende de la cantidad de componentes y de la profundidad con que se quieran probar. Se estiman 25 horas.

CT2 - Pruebas de principio a fin

Consiste en comprobar el flujo de datos y la ejecución que sigue la página web con todos los componentes incorporados y una secuencia de acciones predeterminadas. Se estiman 25 horas.

CT3 - Pruebas de usabilidad

Pasar la aplicación a personas de la empresa para que la utilicen y reporten los errores encontrados y posibles mejoras. Se estiman 10 horas de pruebas y 15 horas de corrección de errores.

4.2. Recursos

4.2.1. Recursos humanos

Dentro de un proyecto no todos los trabajadores tienen el mismo rango o responsabilidades. La diferencia principal que podemos encontrar es que hay un "Jefe de Proyecto" (JP) y luego está el "Equipo".

El "Jefe de Proyecto" es quien lidera el equipo, se encarga de la planificación, dirige las reuniones de seguimiento y toma las decisiones importantes.

Por otra parte, está el "Equipo", dentro de este equipo podemos encontrar diferentes roles, están los "Investigadores" (I) que son los encargados de buscar las diferentes soluciones y nuevas tecnologías que se pueden utilizar para cumplir los objetivos.

También encontramos los "programadores". En este rol hay que diferenciar entre los "programadores de funcionalidades" (PF) que su principal tarea es programar, crear código que cumpla con las funcionalidades pedidas, y el otro rol es el "programador /diseñador gráfico" (PD) que es quien se encarga de la visualización de la aplicación y de que los componentes de la página web se vean cómo el cliente quiere. Por último, aunque el programador ha creado pruebas unitarias y de principio a final, una parte importante de los errores se descubren con el uso de la

aplicación y para ello se necesita un conjunto de personas que prueben la aplicación y detecten errores, estos son los "testers" (T).

Este proyecto se realiza por una persona, eso quiere decir que el autor interpretará todos los roles, pero tendrá el apoyo de los miembros del departamento de GiD para cualquier duda que surja, en la Tabla 1 que está a continuación se puede ver una columna donde se especifica que rol va relacionado con cada tarea.

4.2.2. Recursos materiales

En este apartado se comentará todo el material que se necesita para realizar el proyecto, tanto material físico cómo programas. Esto se verá reflejado por cada tarea en la tabla que aparece al final de este apartado.

La empresa proporciona un espacio dentro del despacho del departamento de GiD con mesa y una torre para poder trabajar, pero se utilizará un portátil personal para escribir la documentación. En la tabla todo aparecerá cómo Ordenador.

También se necesita internet para buscar información y poder subir el trabajo realizado. Cómo este recurso se necesitará siempre no aparecerá en la tabla.

Se han descargado diferentes programas, algunos son de pago y los proporciona la empresa. Otros son gratuitos, concretamente programas de documentación y presentación cómo Word [28] y PowerPoint. También se han utilizado aplicaciones gratuitas para programar cómo el Visual Studio Code [29], Sourcetree [24], Angular [33].

También se utilizan paquetes cómo Monaco Editor, PrimeNG [34] para componentes visuales y GitLab api para conectar la aplicación con GitLab CIMNE.

Se utilizan la mayoría de los recursos de Atlassian [23] y la aplicación Gantter [30] para la organización y las reuniones.

Id.	Tarea	Tiempo (horas)	Dependencias	Recursos	Roles
D	**Documentación**	**160 h**	**-**	**-**	**-**
D1	Entregas de GEP	90 h	CP1, CP2, CP3	- Ordenador	-

				- Microsoft Word - Material de GEP	
D2	Escribir memoria	60 h	D1	- Ordenador - Microsoft Word	JP, I, PF, PD, T
D3	Preparar presentación	10 h	D2, S4	- Ordenador - PowerPoint	JP
S	**Reuniones de Seguimiento**	**22 h**	**-**	**-**	**-**
S1	Reunión Inicial	3 h	-	- Ordenador	JP
S2	Reuniones seguimiento semanales	15 h	S1	- Ordenador - Atlassian	JP, I, PF, PD, T
S3	Reunión hito intermedia	2 h	S2	- Ordenador	JP, I, PF, PD, T
S4	Reunión Final	2 h	S2, S3, CT3	- Ordenador	JP, I, PF, PD, T
CP	**Preparación Previa**	**85 h**	**-**	**-**	**-**
CP1	Definir Alcance	10 h	S1	- Ordenador - Microsoft Word	JP
CP2	Planificación Temporal	10 h	CP1	- Ordenador - Microsoft Word - Gantter	JP
CP3	Presupuesto y análisis de sostenibilidad	15 h	CP2	- Ordenador - Microsoft Word	JP
CP4	Aprendizaje Angular y TypeScript	40 h	CP3	- Ordenador - Udemy	PF, PD
CP5	Preparar el Entorno	10 h	CP4	- Ordenador - Sourcetree - Angular - Visual Studio Code	PF, PD
CD	**Diseño**	**60 h**	**-**	**-**	**-**
CD1	Diseño del Editor	12 h	CP5	- Ordenador	I
CD2	Diseño del Gestor de Versiones	16 h	CP5	- Ordenador	I
CD3	Diseño del Modo Debug	16 h	CP5	- Ordenador	I
CD4	Diseño de la Interfaz Gráfica	16 h	CP5	- Ordenador	PD
CI	**Implementación**	**200 h**	**-**	**-**	**-**
CI1	Crear Editor	50 h	CD1, CD4	- Angula - Visual Studio Code - Sourcetree	PF, PD

				- paquete Monaco editor	
CI2	Crear Línea Temporal	25 h	CD2, CD4	- Angula - Visual Studio Code - Sourcetree - PrimeNG	PF, PD
CI3	Implementar interacciones con GitlLab Api	50 h	CD1, CD2	- Angula - Visual Studio Code - Sourcetree - cuenta GitLab CIMNE	PF
CI4	Implementar el modoDebug		CD3, CD4	- Angular - Visual Studio Code - Sourcetree	PF, PD
CT	**Pruebas**	**75 h**	**-**	**-**	**-**
CT1	Pruebas Unitarias	25 h	CI3, CI4	- Angular - Visual Studio Code - Sourcetree	PF
CT2	Pruebas de Principio a Final	25 h	CT1	- Angular - Visual Studio Code - Sourcetree	PF
CT3	Pruebas Usabilidad	25 h	CT2	-	T
-	**Total**	**602 h**	**-**	**-**	**-**

Tabla 1 : Tabla resumen de tareas con la duración y recursos de cada una. Elaboración propia

En el **Anexo A** están las figuras que representan el diagrama de Gantt, en el que se pueden ver las diferentes tareas, en el que cada ámbito está representado por un color diferente.

En el diagrama de Gantt se pueden ver las diferentes dependencias entre tareas, pero también se pueden apreciar de manera más clara las dependencias entre tareas de final a inicio en la siguiente figura.

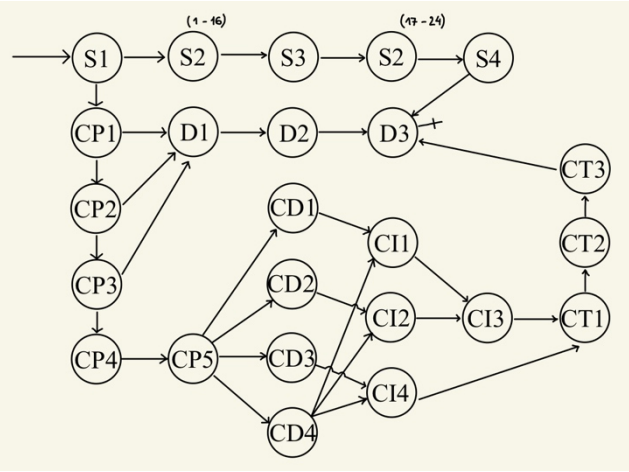


Figura 3 : Diagrama de dependencias de final-inicio. Elaboración propia

4.3. Gestión del riesgo

El primer riesgo está relacionado con la tarea de Aprendizaje del entorno Angular y de TypeScript. Si la curva de aprendizaje no es tan rápida cómo se esperaba en un principio, se tendrán que añadir horas de trabajo según sea necesario, por eso esta tarea se realiza en agosto, si fuera necesario se podría alargar utilizando las dos semanas de vacaciones que el estudiante tiene en la empresa antes de empezar el cuatrimestre. De esta manera no se vería afectada la duración del proyecto.

Los siguientes riesgos son más problemáticos, están relacionados con que una de las decisiones que se han tomado en las fases de Diseño no sea adecuada para este proyecto. Si la solución consiste en crear una tarea de adaptación de otro paquete ya existente, que reemplace el que no se adapta al proyecto, tendrá un coste menor al que si la solución es añadir la tarea de implementar desde cero las funcionalidades que debería tener el paquete.

En el caso de cambiar de paquete se necesitarán los recursos del nuevo paquete. Asumiendo que parte del código que relaciona los componentes ya estará hecho, se estiman 15 horas para la gestión de versiones y 10 horas para el editor. En el caso del módulo de Debug es más complicado, porque se tendría que buscar una solución nueva e implementarla, esto se estima con unas 50 horas de trabajo extra y puede que se necesiten nuevos recursos.

Se intentará no llegar al caso de implementar desde cero, porque si esta casuística se da en el editor y en el componente de gestión de versiones, es muy posible que el incremento de tiempo sea insostenible para el trabajo o se tendrían que reducir pruebas por falta de tiempo.

Todos estos casos se han tenido en cuenta en el diagrama de Gantt que está en el **Anexo A**, se han representado de color amarillo y con el nombre de **tarea extensión de identificador.** Si al final todo va según lo previsto, se terminaría el trabajo un poco antes de la fecha de entrega y se puede aprovechar para hacer más pruebas o añadir alguna funcionalidad extra.

5. Presupuesto

Una parte importante de todo proyecto es hacer una estimación de costes que tenga en cuenta los obstáculos que puedan aparecer y los costes no programados para no sobrepasar el presupuesto acordado para el proyecto.

5.1. Identificación y estimación de costes

Los costes se dividirán en dos tipos, los costes del personal y los costes genéricos que contendrán hardware, software y necesidades varias. También se tendrán en cuenta los imprevistos y se hará un plan de contingencia.

Costes de personal

Cómo ya se ha comentado anteriormente, en este proyecto participan diferentes roles y cada uno tiene diferentes grados de responsabilidad. Por este motivo, dependiendo del rol que tratemos los salarios varían. En la siguiente tabla se puede apreciar el coste por hora dependiendo del rol.

Rol	Coste por hora	Seguridad Social	Total
Jefe de Proyecto	30 €/h	9 €/h	39 €/h
Investigadores	20 €/h	6 €/h	26 €/h
Programador funcionalidades	16 €/h	4,80 €/h	20,80 €/h
Programador gráfico	16 €/h	4,80 €/h	20,80 €/h
Tester	16 €/h	4,80 €/h	20,80 €/h

Tabla 2 : Retribución de cada Rol. Elaboración propia

Una vez fijado el precio por hora de cada rol es el momento de tener en cuenta todas las tareas de la planificación y calcular el coste del personal total. Es lo que se puede ver en la siguiente tabla, donde la columna JP corresponde al Jefe del proyecto, la columna I corresponde al Investigador, la columna PF al Programador de funcionalidades, la columna PD al Programador gráfico y la columna T al Tester.

Id.	Tarea	Tiempo (horas)	JP	I	PF	PD	T	Coste
D	**Documentación**	**160h**	**858€**	**312€**	**249,60€**	**249,60€**	**249,60€**	**1.918,80€**
D1	Entregas de GEP	90h	-	-	-	-	-	-
D2	Escribir memoria	60h[3]	468€	312€	249,60€	249,60€	249,60€	1.528,80€
D3	Preparar presentación	10h	390€	-	-	-	-	390€
S	**Reuniones de Seguimiento**	**22h**	**858€**	**494€**	**395,20€**	**395,20€**	**395,20€**	**2.537,60€**
S1	Reunión Inicial	3h	117€	-	-	-	-	117€
S2	Reuniones seguimiento semanales	15h	585€	390€	312€	312€	312€	1.911€
S3	Reunión hito intermedia	2h	78€	52€	41,60€	41,60€	41,60€	254,80€
S4	Reunión Final	2h	78€	52€	41,60€	41,60€	41,60€	254,80€
CP	**Preparación Previa**	**85h**	**1.365€**	**-**	**520€**	**520€**		**2.405€**
CP1	Definir Alcance	10h	390€	-	-	-	-	390€
CP2	Planificación Temporal	10h	390€	-	-	-	-	390€
CP3	Presupuesto y análisis de sostenibilidad	15h	585€	-	-	-	-	585€
CP4	Aprendizaje Angular y TypeScript	40h[4]	-	-	416€	416€	-	832€
CP5	Preparar el Entorno	10h[5]	-	-	104€	104€	-	208€
CD	**Diseño**	**60h**	**-**	**1.144€**	**-**	**332,80€**	**-**	**1.476,80€**
CD1	Diseño del Editor	12h	-	312€	-	-	-	312€
CD2	Diseño del Gestor de Versiones	16h	-	416€	-	-	-	416€
CD3	Diseño del Modo Debug	16h	-	416€	-	-	-	416€

[3] Estas horas son totales se dividen entre los dos roles, cada rol dedica 12 horas.

[4] Estas horas son totales se dividen entre los dos roles, cada rol dedica 20 horas.

[5] Estas horas son totales se dividen entre los dos roles, cada rol dedica 5 horas.

CD4	Diseño de la Interfaz Gráfica	16h	-	-	-	332,80€	-	332,80€
CI	**Implementación**	**200h**	**-**	**-**	**2.600€**	**1.560€**	**-**	**4.160€**
CI1	Crear Editor	50h[6]	-	-	520€	520€	-	1.040€
CI2	Crear Línea Temporal	25h[7]	-	-	260€	260€	-	520€
CI3	Implementar interacciones con GitlLab Api	50h	-	-	1.040€	-	-	1.040€
CI4	Implementar el modoDebug	75h[8]	-	-	780€	780€	-	1.560€
CT	**Pruebas**	**75h**	**-**	**-**	**1040€**	**-**	**520€**	**1560€**
CT1	Pruebas Unitarias	25h	-	-	520€	-	-	520€
CT2	Pruebas de Principio a Final	25h	-	-	520€	-	-	520€
CT3	Pruebas Usabilidad	25h	-	-	-	-	520€	520€
-	**Total**	**602h**	**3.081€**	**1.946€**	**4.804,80€**	**3.057,6€**	**1.164,8€**	**14.058,20€**

Tabla 3 : Tabla de costes por tarea. Elaboración propia

Después de observar la Tabla 3 podemos deducir que el coste del personal para el proyecto es de 14.058,20€ que se redondeará a **14.059 €.**

Costes genéricos

Para esta parte tendremos en cuenta que la amortización del Hardware es de unos 4 años, la fórmula que se utilizará para calcular las amortizaciones es

$$\frac{\boldsymbol{CosteDispositivo}(€) * \boldsymbol{HorasUso}(\boldsymbol{horas})}{\boldsymbol{VidaÚtil}(\boldsymbol{años}) * \boldsymbol{DíasTrabajados}(\boldsymbol{dias/año}) * \boldsymbol{HorasDiarias}(\boldsymbol{horas/dia})}$$

[6] Estas horas son totales se dividen entre los dos roles, cada rol dedica 25 horas.

[7] Estas horas son totales se dividen entre los dos roles, cada rol dedica 12.5 horas.

[8] Estas horas son totales se dividen entre los dos roles, cada rol dedica 37.5 horas.

También hay que tener en cuenta que un año tiene 365 días de los cuales 104 días forman parte de fines de semana y unos 25 días son festivos, lo que hace que queden 236 días laborables. La jornada laboral que se hará es de 5 horas cada día.

Hardware

En este proyecto se utilizan dos dispositivos: un ordenador de sobremesa, que estará fijo en el despacho, y un portátil personal, que se utilizará para escribir la memoria y para las reuniones de seguimiento.

El precio del Ordenador de sobremesa, que incluye torre [35], pantalla [36], teclado [37], ratón [38] y SAI [39], es de 2.000€.

El precio del portátil, un MacBookPro 2019 [40], es de 2500€.

La siguiente tabla calcula las amortizaciones para los dos dispositivos y se puede ver que el coste total de estos dispositivos es de **275€.**

Hardware	Precio	Unidades	Vida útil	Horas de uso	Amortización
Ordenador sobremesa	2.000€	1	4 años	420h	178€
Portátil	2.500€	1	4 años	182h	97€
Total	**-**	**-**	**-**	**-**	**275€**

Tabla 4 : Costes de los recursos de Hardware. Elaboración propia

Software

Dentro de este apartado se encuentran todos los programas que se han tenido que comprar para la realización del proyecto, los programas que se utilizan y tienen coste cero no aparecen en la Tabla siguiente.

Software	Precio mensual	Unidades	Meses	Coste
Microsoft 365 Empresa Estándar	10,50€	1	6	63€

Gantter	5€	1	6	30€
Total	**-**	**-**	**-**	**93€**

Tabla 5 : Costes de los recursos de Software. Elaboración propia

Otros

Para trabajar también se necesita un espacio de trabajo. En este caso este espacio de trabajo es el despacho del edificio B0 situado en el Campus Nord de la UPC. El precio del alquiler sería de unos 3800€ al año, contando que el tiempo que durará el proyecto es de 6 meses esto sumaría un total de 1900€. Cómo el despacho se comparte con 6 personas más, el precio real del alquiler por persona es de 271,5€. A esta cantidad se le tiene que añadir 300€ de material de oficina cómo una silla y mesa, por lo tanto, el total son **571,5€**.

Otro coste que hay que tener en cuenta es el de Internet. Además de la electricidad, internet cuesta unos 50€ al mes. El proyecto dura unos 6 meses, por lo tanto el consumo total de Internet es de **300€**.

La electricidad media es de 0.27592€/kWh, contamos el consumo del equipo de sobremesa, según el artículo [41] contamos que el consumo del ordenador encendido es de 22W por 5 horas de uso y apagado consume 4W.

Ahora contamos el consumo del portátil [42] que su gasto medio es de 30W cuando se usa.

Otros	Potencia	Unidades	Horas	Consumo	Coste
Monitor	40W	1	420h	16,8kWh	4,64€
Ordenador uso	180W	1	420h	75,6kWh	20,86€
Ordenador apagado	4W	1	257h	1,028kWh	0,29€
Portátil	30W	1	182h	5,46kWh	1,51€
Total	**-**	**-**	**-**	**-**	**27,3€**

Tabla 6 : Costes de energía previstos del proyecto. Elaboración propia

Esto hace que los costes clasificados cómo Otros sumen un total de **898,8€**, si a este coste se le añade el Hardware y el Software sale un total de gastos genéricos del 1.266,8€ que se redondea a **1.267€**

Contingencia

En todo proyecto hay que añadir un pequeño sobrecoste para cubrir posibles imprevistos no anticipados y obstáculos que surjan. En este caso cómo es un proyecto de creación se considera que es más posible que haya problemas en el desarrollo, que en los costes generales por este motivo la contingencia asociada a los gastos de personal es del 20% y la contingencia de los gastos generales es del 10%.

En la siguiente tabla se pueden observa los costes de contingencia calculados con la siguiente fórmula

$$\boldsymbol{CosteFinal = Coste * Contingencia}$$

Descripción	Coste	Contingencia	Coste total
Coste de personal	14.059€	20%	2.811,8€
Costes generales	1.267€	10%	126,7€
Total	**-**	**-**	**2.938,5€**

Tabla 7 : Costes de contingencia. Elaboración propia

La cantidad que se guardará para contingencias es de 2.938,5€ que se redondea a **2.939€**

Imprevistos

En este apartado se tiene en cuenta los riesgos que ya han sido identificados anteriormente con una probabilidad de que sucedan del 30% y riesgos de que falle el Hardware de entre un 5% y un 10%.

Para calcular el tiempo extra de personal se calcula el coste estimado de la siguiente manera

$$CosteEstimado = TiempoEstimado * Salario$$

Una vez ya se saben los costes estimados se calcula el cose total de cada uno de los riesgos

$$CosteTotal = CosteEstimado * Probabilidad$$

En la siguiente tabla se pueden ver el resultado de estos cálculos.

Descripción	Tiempo estimado	Coste estimado	Probabilidad	Coste
Aprendizaje Angular y TypeScript	65h	1352€	30%	405,6€
Crear Editor	10h	208€	30%	62,4€
Crear Línea Temporal	15h	312€	30%	93,6€
Implementar interacciones con GitlLab Api	10h	208€	30%	62,4€
Implementar el modoDebug	50h	1040€	30%	312€
Nuevo ordenador de sobremesa	-	2.000€	10%	200€
Nuevo portátil	-	2.500€	5%	125€
Total	**-**	**-**	**-**	**1.261€**

Tabla 8 : Coste de los imprevistos. Elaboración propia

Coste total del proyecto

En la siguiente tabla se suman todos los costes de cada grupo que se han mencionado anteriormente y se suman para sacar el coste total del proyecto, los

costes totales de cada grupo están redondeados al alza porque cuando se hablan de estas cantidades de dinero no hace falta especificar los céntimos.

Descripción	Coste
Coste del personal	14.059€
Coste genérico	1.267€
Coste de contingencia	2.939€
Coste de imprevisto	1.261€
Presupuesto Final	**19.526€**

Tabla 9 : Presupuesto final del proyecto. Elaboración propia

5.2. Control de gestión

Ahora ya se tienen contados todos los gastos del proyecto y con ello se ha estimado el presupuesto final, pero hay que hacer un seguimiento de los gastos para que no se desvíen demasiado del plan establecido. Durante las reuniones semanales se revisarán los gastos que se hayan podido modificar a lo largo de la semana, con diferentes mecanismos de control e indicadores numéricos que se muestran a continuación.

Desvío en coste por tarifa

- Desviación del coste personal por tarea

$$(\boldsymbol{CosteEstimado} - \boldsymbol{CosteReal}) * \boldsymbol{HorasReales}$$

Desvío en eficiencia

- Desviación de la realización de tareas

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) * \boldsymbol{CosteReal}$$

Desvío en total

- Desviación total en la realización de tareas

$$(\boldsymbol{CosteEstimadoTotal} - \boldsymbol{CosteRealTotal})$$

- Desviación total de recursos (hardware, software, otros)

$$(\boldsymbol{CosteEstimadoTotal} - \boldsymbol{CosteRealTotal})$$

- Desviación total costes de imprevistos

$$(\boldsymbol{CosteEstimadoImprevistos} - \boldsymbol{CosteRealImprevistos})$$

- Desviación total de la electricidad

$$(\boldsymbol{CosteEstimadoElectricidad} - \boldsymbol{CosteRealElectricidad})$$

- Desviación total de la contingencia

$$(\boldsymbol{CosteEstimadoContingencia} - \boldsymbol{CosteRealContingencia})$$

- Desviación total de horas

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales})$$

Esto permite ser consciente de las desviaciones lo antes posibles, clasificar el gasto extra en imprevistos o en contingencias para poder usar parte del dinero que se ha dedicado a esas casuísticas y si no forma parte de ninguna de las dos, buscar una solución antes de que el problema sea mayor.

6. Aplicación GiD de escritorio

Anteriormente se ha mencionado mucho el nombre de la aplicación GiD actual que se puede descargar en [43] y se ha hecho un pequeño resumen del objetivo principal y de lo que quiere aportar a sus usuarios.

En este apartado se quiere profundizar un poco más para ver parte del potencial que tiene y poder entender mejor la explicación del desarrollo. La idea principal de arquitectura de la nueva web reside en crear una web que tenga todas las funcionalidades ya existentes en la aplicación original e incluso añadir algunas nuevas. Pero esta parte solo será el **frontend**, eso quiere decir que convertirá los datos en una interfaz gráfica para que el usuario pueda ver e interactuar con la información de forma digital usando HTML, CSS y JavaScript. Pero si lo que hace es convertir la información, la pregunta que nos interesa es ¿Qué información? o ¿De dónde sale esa información?

La respuesta es muy simple, se conectará a otro componente que nosotros llamamos **ModelManager** que contendrá la información de los usuarios y sus modelos. Una vez el usuario esté preparado para trabajar, solo tendrá que abrir un modelo y empezar a hacer geometrías, cálculos, simulaciones, etc.

Para que esto sea posible, en un servidor estará activa una instancia del programa original de la cual la aplicación web obtendrá los datos de los cálculos, simulaciones y demás para poder mostrarlos al usuario. Pero también se le enviará toda la información de las tareas a realizar con las ordenes precisas para que el programa del servidor las pueda ejecutar.

Las conexiones explicadas anteriormente han sido creadas por los compañeros de la empresa y una vez empezado el proyecto esa parte ya era funcional.

A continuación, se puede ver una imagen de la aplicación de GiD del apartado de Preproceso. Esta es la parte CAD de la aplicación, en la que se pueden hacer operaciones geométricas como generar entidades, manipularlas y destruirlas.

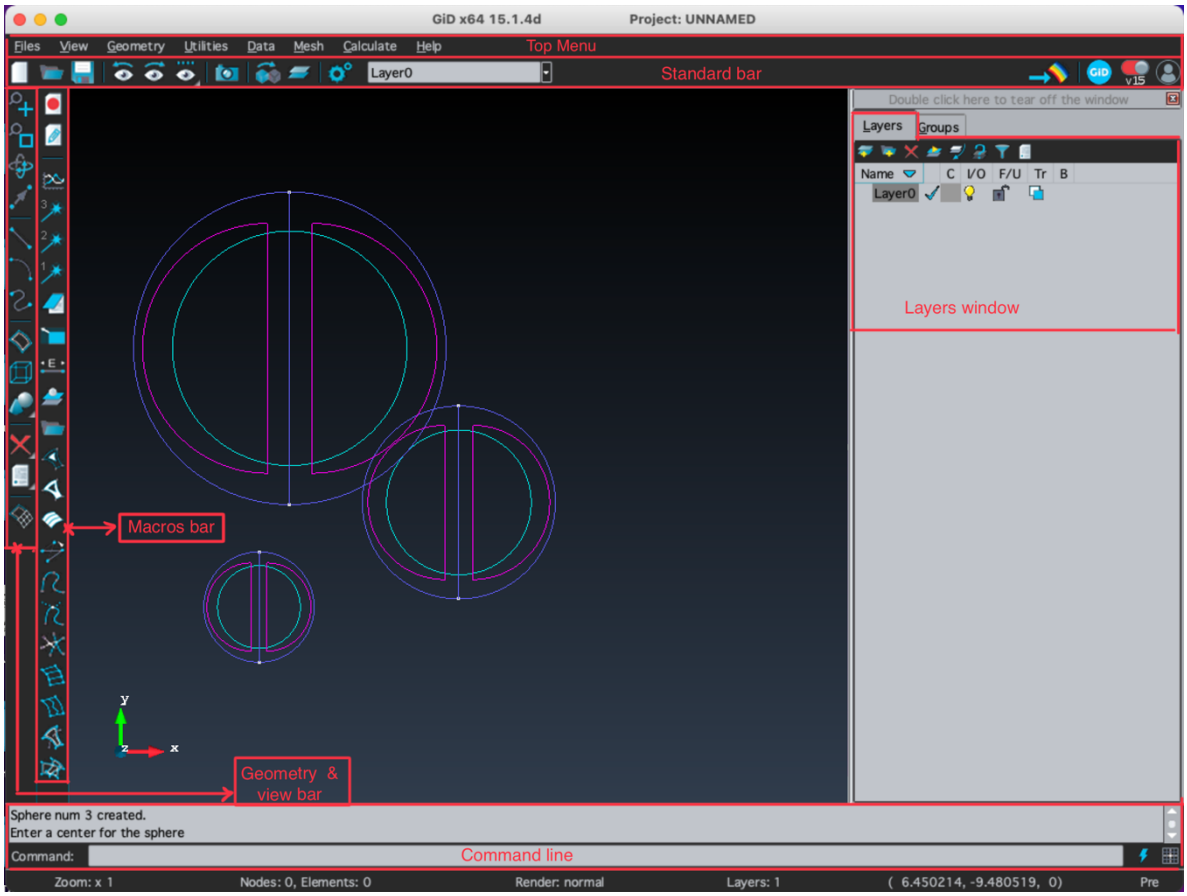


Figura 4 : Aplicación GiD, parte de preporcesado. Elaboración propia

Si se presiona el botón  de la barra estándar, la aplicación te llevará a la parte de postprocesado que se puede ver en la siguiente imagen. Si se presiona  te lleva otra vez a la parte de preprocesado.

El postprocesado permite estudiar los resultados de un solver. La aplicación recibe información de mallas y resultados desde el módulo de resolución, y si el módulo de resolución no crea ninguna malla nueva, se utilizará la malla de preprocesado que genera el programa de GiD.

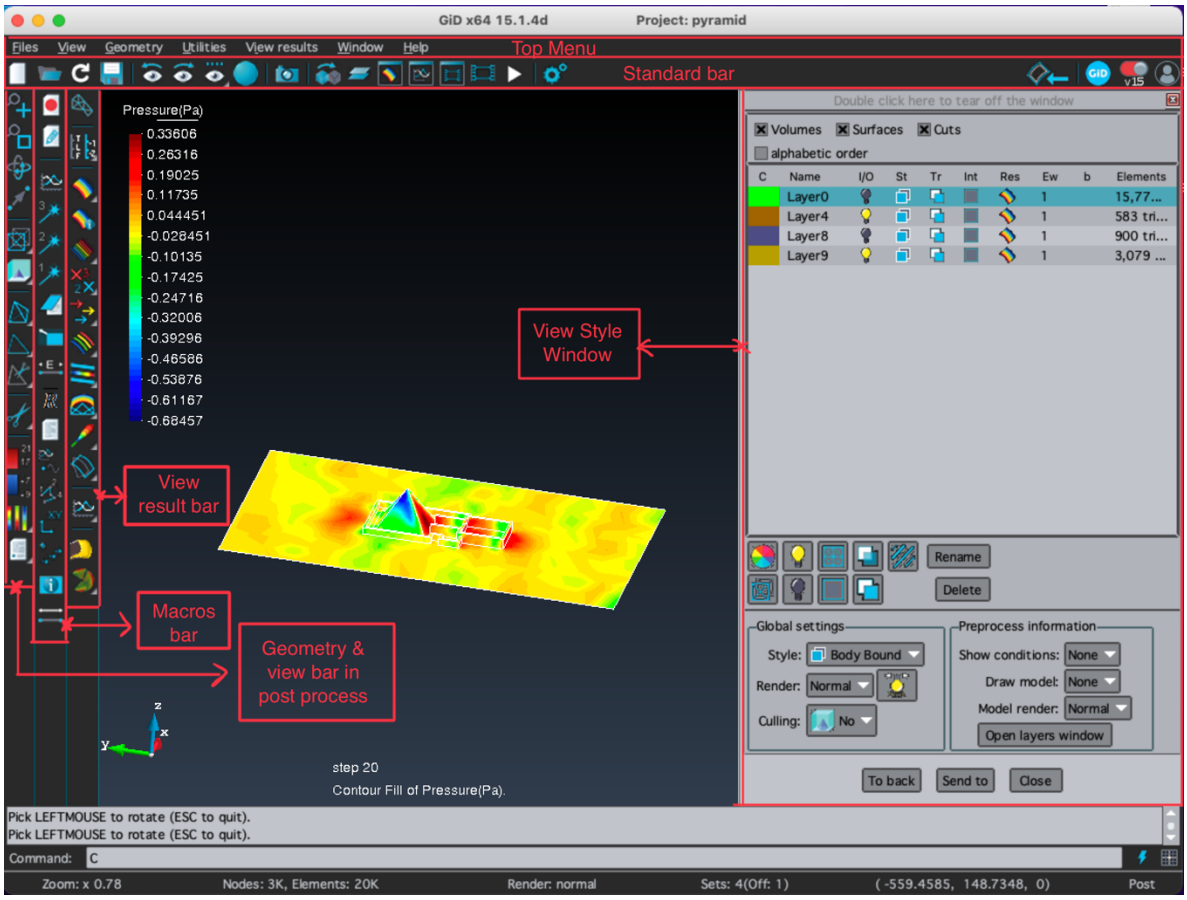


Figura 5 : Aplicación GiD, parte de postporcesado. Elaboración propia

Los ProblemTypes son un conjunto de funcionalidades que permiten al usuario interactuar fácilmente con ellas por medio de una interfaz gráfica de usuario (GUI), y facilita la definición e introducción de todos los datos necesarios para llevar a cabo un determinado cálculo. Para que la aplicación pueda preparar los datos para un programa específico de análisis, es necesario personalizar la aplicación, lo que hacemos con los ProblemTypes que se pueden cargar y descargar fácilmente una vez han sido introducidos en la aplicación.

Este trabajo ha intentado facilitar la creación de funcionalidades con un editor para que esté integrada dentro de la nueva aplicación web, ya que la aplicación de escritorio actualmente se hace creando un directorio al que se le pone el nombre del ProblemType, que ha de contener un conjunto de ficheros.

Los ficheros, definen el tipo de ProblemType y contienen el conjunto de funcionalidades que personalizan el preprocesado. Se añade un fichero que define cómo la aplicación GiD extrae la información del preporcesado para pasarla al solver. Se añade otro fichero que define como sale la información del solver para que el postprocesado la lea de esa manera. La siguiente imagen muestra lo que es la aplicación de GiD y los ficheros externos que consiguen personalizar la aplicación.

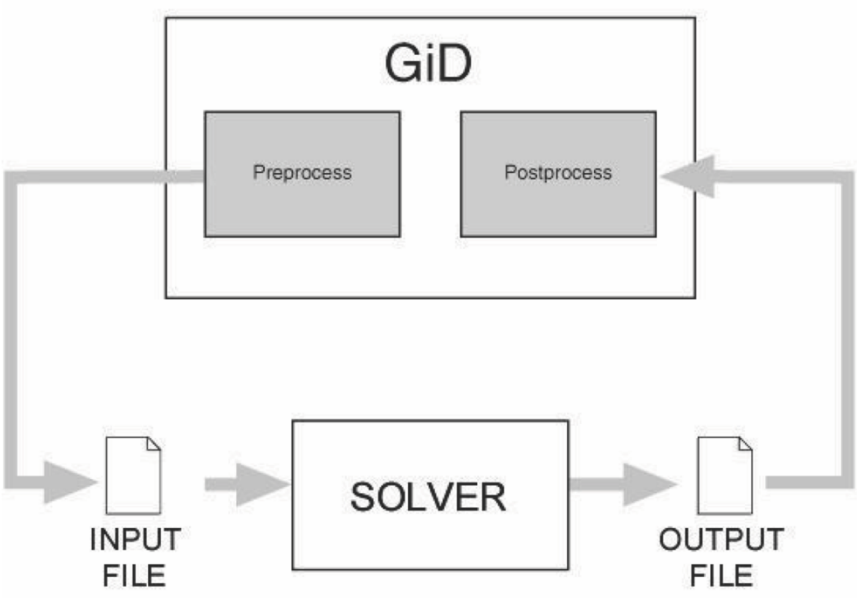


Figura 6 : Ayuda visual para entender que ficheros contiene un ProblemType. Esta imagen se ha extraído de la página 3 de [44]

La aplicación de GiD permite hacer muchas más acciones que no se han explicado en este proyecto, solo nos hemos centrado en explicar una vista general de la aplicación y las partes relacionadas con las funcionalidades y ProblemTypes.

7. Solución propuesta

En este apartado se hará una relación de posibles maneras de solucionar el problema planteado inicialmente. Se evaluarán las diferentes opciones y se tendrán en cuenta las limitaciones de cada una para poder escoger la mejor propuesta como solución final.

El trabajo se abordará por objetivos secundarios, como se ha explicado anteriormente estos son las tareas CD1[9], CD2[10] y CD3[11] del diagrama de Gantt.

7.1. Posibles alternativas del editor

Para resolver el problema del editor se pensó en buscar un componente que proporcionara un editor básico al que se le pudieran añadir funcionalidades y permitiera adaptarse a lo que la empresa necesitara. Otra opción que se barajó fue crearlo desde cero, pero se consideró que era demasiado costoso y que no daría tiempo a terminar el proyecto dentro del tiempo estimado e incluso sería difícil hacer el resto de las funcionalidades si se decidía implementar el editor desde cero.

Una vez tomada la decisión de trabajar con un componente ya existente y adaptar funcionalidades, se pasó a la fase de buscar componentes que se pudieran adaptar, como los editores utilizados en diferentes entornos de desarrollo integrado (IDEs).

De todas las opciones encontradas, las más destacadas fueron "**RunKit**" [16], "**AceEditor**" [17] y "**MonacoEditor**" [18].

A continuación analizaremos con detalle las características de cada una de las opciones para poder determinar qué beneficios y qué inconvenientes proporcionaba cada alternativa.

---

[9] Tarea del diagrama de Gantt que corresponde con el diseño del editor

[10] Tarea del diagrama de Gantt que corresponde con el diseño del gestor de versiones

[11] Tarea del diagrama de Gantt que corresponde con el diseño del modo debug

RunKit

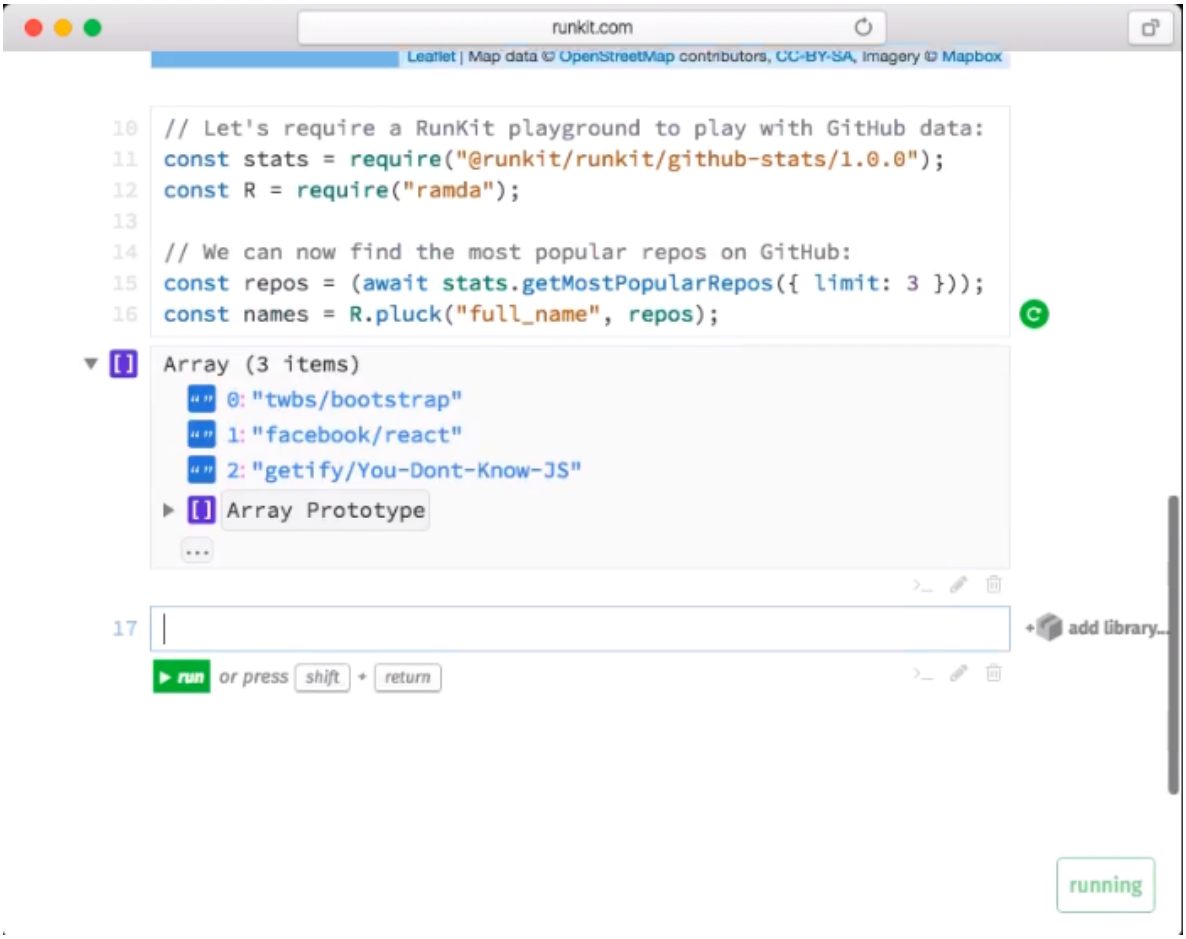


Figura 7 : Un ejemplo del RunKit editor [16]

Como beneficio se puede destacar que es un editor centrado en la parte de ejecución. Principalmente permite ejecutar bloques de código en los llamados notebooks. Estos fragmentos de código se pueden conectar a ficheros de GitHub [19] para extraer los datos que luego se utilizarán en las ejecuciones, e incluso descargar paquetes NPM [45] e incorporarlos a la ejecución.

Los inconvenientes que se han encontrado con esta opción es que no permite mucha personalización o adaptación con respecto a la visualización del editor. La empresa intenta ofrecer al usuario la mayor variedad y flexibilidad con respecto a

la personalización del interfaz y con este editor se tendría que renunciar o invertir el doble de tiempo en sustituir o reemplazar los ficheros de estilos CSS que se utilizan para dar personalidad a las páginas web. Otro de los problemas importantes es que el editor está muy centrado en temas de documentación del código y gráficas para interpretar datos, para ejecutar componentes propios que el usuario pueda programar, se tendría que crear algún paquete NPM para completar el editor y eso complicaría un poco la manera en que el usuario crearía estos componentes y no sería tan intuitivo.

AceEditor

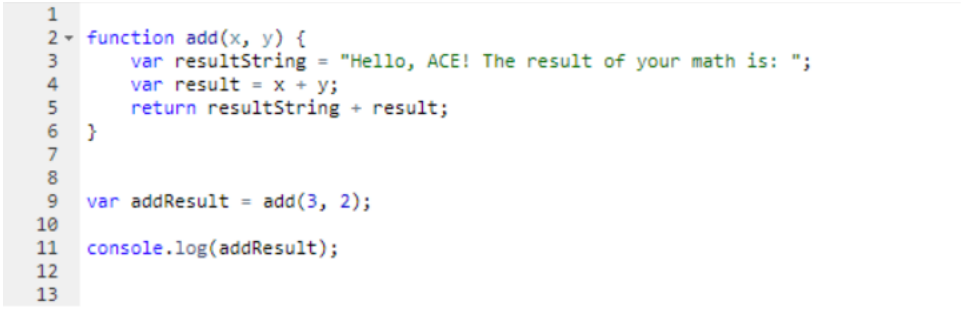


Figura 8 : Un ejemplo del AceEditor [17]

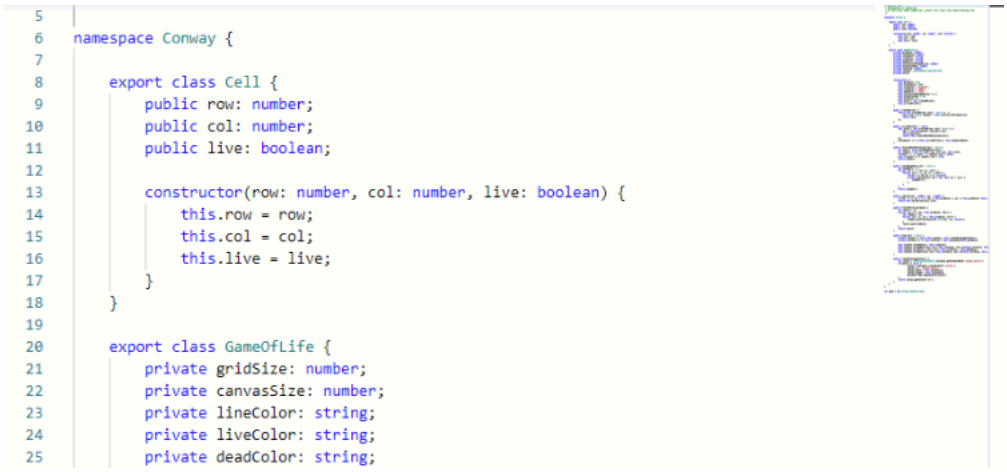
Este componente es algo diferente al anterior, opta más por la parte de edición en vez de dar la opción de ejecutar cómo el anterior. Algunos de los beneficios que aporta este editor es que sí permite cambiar la visualización. También tiene reconocimiento de 110 sintaxis diferentes, con códigos de colores correspondientes a las palabras claves e indentados del lenguaje que se esté utilizando, y acepta hasta 4 millones de líneas. Incorpora opciones básicas de los editores como copiar, pegar, buscar, reemplazar y mucho más.

Pero no es oro todo lo que reluce y a pesar de todos los beneficios que aporta esta decisión, también se ha encontrado un pequeño inconveniente. En la aplicación

que la empresa quiere ofrecer a los usuarios se espera poder gestionar las versiones anteriores, esto quiere decir que la aplicación ha de poder mostrar el código anterior comparándolo con la versión actual. De esta manera el usuario puede ver las modificaciones entre las dos versiones de manera rápida e intuitiva.

Hacer esta funcionalidad con "AceEditor" es algo más complicado de lo que parece a simple vista. El primer paso es obtener el código de las dos versiones, pero ese problema lo trataremos en profundidad un poco más adelante. El siguiente paso es encontrar las diferencias que hay de un código a otro. Esto se puede hacer con un algoritmo que implemente el estudiante. El último paso es poder mostrar las diferencias que detecta el algoritmo de cara al usuario, y aquí reside gran parte del problema. Se tendría que añadir dos componentes del "AceEditor", uno de ellos bloqueado para que no se pueda cambiar el código que corresponde a la versión anterior y añadir ciertos decoradores para marcar las diferencias. Esta opción es viable, pero puede llevar demasiado tiempo de trabajo en la parte de visualización.

MonacoEditor



```
5
6   namespace Conway {
7
8       export class Cell {
9           public row: number;
10          public col: number;
11          public live: boolean;
12
13          constructor(row: number, col: number, live: boolean) {
14              this.row = row;
15              this.col = col;
16              this.live = live;
17          }
18      }
19
20      export class GameOfLife {
21          private gridSize: number;
22          private canvasSize: number;
23          private lineColor: string;
24          private liveColor: string;
25          private deadColor: string;
```

Figura 9 : Un ejemplo del MonacoEditor [18]

Este editor es muy parecido al anterior. Muchos de los beneficios que aportaría utilizar esta alternativa son compartidos con los beneficios de utilizar el editor

anterior. Permite validación de algunos de los lenguajes de programación más utilizados cómo JavaScript, TypeScript, CSS, JSON, HTML, etc. También tiene reconocimiento de diferentes sintaxis, con sus códigos de colores correspondientes a las palabras claves, etc.

Pero uno de los beneficios más llamativos es que tiene incorporado un editor de diferencias, que resalta en verde las líneas nuevas y en rojo las líneas que han sido borradas. Las líneas modificadas tienen un color rojo claro en el código original y un verde claro en el código que ha sido modificado, resaltando el fragmento modificado de un tono más fuerte del color que corresponda.

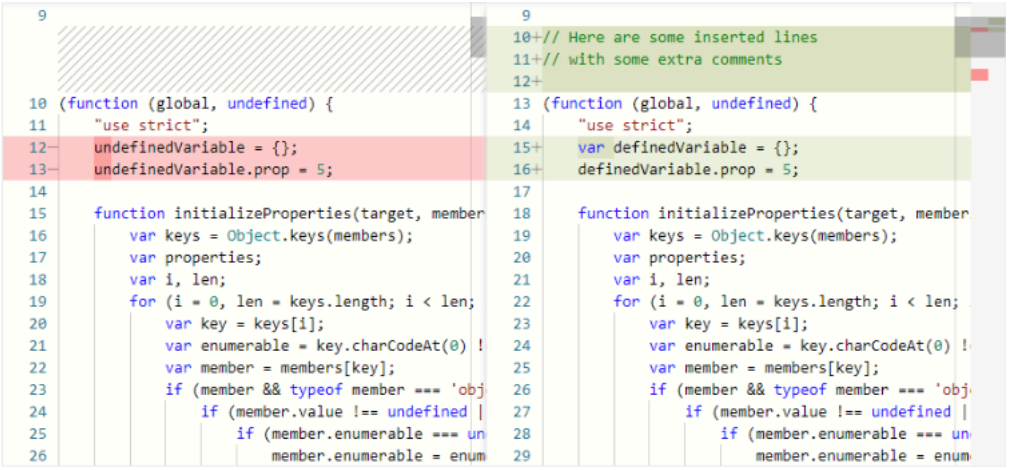


Figura 10 : Un ejemplo del editor de difrencias del MonacoEditor [18]

A simple vista con las investigaciones iniciales no se han detectado inconvenientes que resulten un problema para el trabajo y es el que utilizan de base IDEs cómo Visual Studio Code. [29]

7.2. Posibles alternativas del control de versiones

Otro de los problemas a los que hay que enfrentarse a lo largo de este proyecto es encontrar una manera que permita gestionar las versiones anteriores del código del usuario. Esto ha de permitir poder acceder al código anterior que ha de estar almacenado en alguna parte y poderlo comparar. La parte de comparación ha sido

tratada en el apartado anterior con el problema del editor, en este apartado se evaluarán las diferentes opciones de almacenado.

La primera alternativa en la que se pensó fue en crear una API propia para la empresa que se conectara a una base de datos para poder guardar la información del código. Los beneficios de esta idea son muy evidentes. Al crear algo desde cero no existe el problema de tener que adaptarse o moldearse a lo que ofrece el programa que se está utilizando y además se puede crear a la medida de las necesidades de la empresa. Esta característica es un arma de doble filo, lo que nos lleva a hablar de los problemas que tiene esta solución. Lo primero a destacar es que no sólo se quiere guardar el código, también se quiere gestionar de manera similar a Git. En la primera versión del proyecto solo habrá una única rama con las versiones anteriores en formato commit[12], pero más adelante se querrá dar la opción al usuario de trabajar en diferentes ramas, crear ramas nuevas y poder trabajar simultáneamente en el mismo fichero con más gente. Teniendo en cuenta estos planes de futuro para el proyecto, quizás esta solución conlleva demasiado tiempo y esfuerzo cuando ya hay plataformas que ofrecen APIs de conexión con este tipo de gestores de versiones; así que, se optó por utilizar una plataforma ya existente.

La segunda alternativa fue utilizar APIs que permiten conectar a la aplicación con repositorios de código cómo GitLab [20] o GitHub [19], ambas opciones ofrecen más o menos las mismas funcionalidades. El beneficio de esta solución es que facilitará en un futuro el tema de gestionar las ramas y almacenar la información correctamente. Como inconveniente se puede considerar que permite guardar el código con un mensaje de commit pero no mucha información más, aunque de momento esta alternativa parece suficiente.

[12] Operación que permite Git para guardar.

7.3. Posibles alternativas del apartado de ejecución

El último problema que hay que solucionar es encontrar la manera de ejecutar el código del usuario de forma segura, mostrar los resultados de la ejecución siendo capaces de detectar los errores y proporcionar facilidades para que el usuario los encuentre y los pueda solucionar.

Esta parte no permite demasiadas alternativas, dado que el usuario programará en el lenguaje de programación de JavaScript y para ver el resultado de lo que ha programado se ha de ejecutar. Este lenguaje es interpretado y esto quiere decir que traduce el código mientras se ejecuta, cosa que permite ver los errores en el momento en el que se están produciendo. No compila el código por separado, como harían los lenguajes compilados. Por este motivo no queda más remedio que ejecutar el código para detectar los errores, es la única solución que se nos ha ocurrido, pero esto puede llevar a problemas. La verdadera cuestión requiere encontrar una alternativa para que la ejecución se haga de manera segura y que la aplicación no quede colgada si se cometen errores.

Se ha decidido utilizar el método **eval( )** [46] de JavaScript, que evalúa el código introducido como una cadena de caracteres, y gracias al bloque **try{ } catch(){ }** [47] se consigue capturar los errores que se producen dentro del método eval( ). Han surgido un par de problemas a los que se ha aplicado la siguiente solución. El primero es que al ejecutar el código del usuario, si éste comete errores involuntarios podría llegar a colgar toda la aplicación. El segundo error es que si el usuario escribe código malicioso podría conllevar un problema de seguridad ya que se está ejecutando dentro de la propia aplicación. Para evitar esto se ha decidido utilizar los **WebWorkers** [48], con lo que se consigue ejecutar el código del usuario en un thread [49] secundario. Para que se entienda, sabemos que todo programa es una secuencia encadenada de tareas más pequeñas que ejecuta el proceso principal. El ordenador puede ejecutar además del proceso principal diferentes tareas secuencias de manera simultanea en lo que se llama thread o hilo.

La idea que está detrás del WebWorker, es que el código generado por el usuario se ejecute en un thread separado del proceso principal. Gracias a este método, la ejecución secundaria no puede acceder a toda la información de la aplicación que corresponde al proceso principal y esto proporciona un factor de seguridad. Si por cualquier motivo el código del usuario fallara, solo afectaría a la ejecución secundaria que se pararía, pero el proceso principal seguiría intacto y con un buen funcionamiento.

Una vez solucionado el problema de detección de errores y de ejecución, falta encontrar la manera de visualizar de forma gráfica los resultados. Para eso se ha adoptado la metodología que la empresa tenía pensada. El WebWorker ejecuta código sin respaldo visual, por este motivo el usuario utiliza diferentes tipos de objetos definidos por clases[13] en el WebWorker. Las clases traducen toda la información necesaria a un formato JSON, que es la notación de objetos en JavaScript. La información que se pasa a formato JSON es el tipo de objeto, la acción que hará, el nombre, entre otros. En la siguiente imagen se puede ver un ejemplo de la definición de la clase de tipo botón con su información asociada.

```
definition[id].SphereButton = new Button({
    'name': "Add sphere",
    'icon_type': 'fas',
    'icon': 'circle',
    'onClick': function () {
    context.openWindow(definition[id].CreateSphereWindow, CurrentWorkingMode.mainwindow);
    },
});
```

Figura 11 : Clase de tipo botón y los parámetros necesarios. Elaboración propia

Una vez el objeto JSON está listo se envía al componente de Angular con el método **postMessage(componente_JSON)** y el componente lo recibe gracias al **addEventListener('message', (componente_JSON) =>{})**. A continuación,

[13] Representa un tipo particular de objeto dentro de la programación orientada a objetos. Cada clase tiene un código que la define y determina sus atributos y métodos.

gracias a la implementación que el estudiante ha desarrollado con componentes Angular, según el parámetro tipo del objeto JSON se generará un elemento HTML [50] u otro con toda la información que trae el formato JSON. Los elementos HTML se representan en la aplicación de manera visual. En las siguientes imágenes se puede ver esta transformación que hace el componente Angular. La primera es el objeto JSON que viene del WebWorker, la segunda es el elemento HTML con la información del objeto JSON y la tercera es la visualización del elemento HTML en la página web.

```
{
    "type":"button",
    "icon":"fas fa-circle",
    "action":"Action(){...}",
    "pTooltip":"Add sphere"
}
```

Figura 12 : Objeto JSON. Elaboración propia

```
<button pButton type="button" icon="fas fa-circle"
class="p-button-rounded" (click)='Action()' pTooltip="Add
sphere"></button>
```

Figura 13 : Elemento HTML con la información del JSON. Elaboración propia

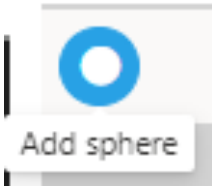


Figura 14 : Representación del elemento HTML en la página web. Elaboración propia

De esta manera se consigue transformar en un objeto visual dentro de la aplicación web, como el WebWorker solo envía objetos de tipo JSON en vez de código entero también es un factor de seguridad.

En la siguiente imagen se puede ver un diagrama de flujo que explica esta idea.

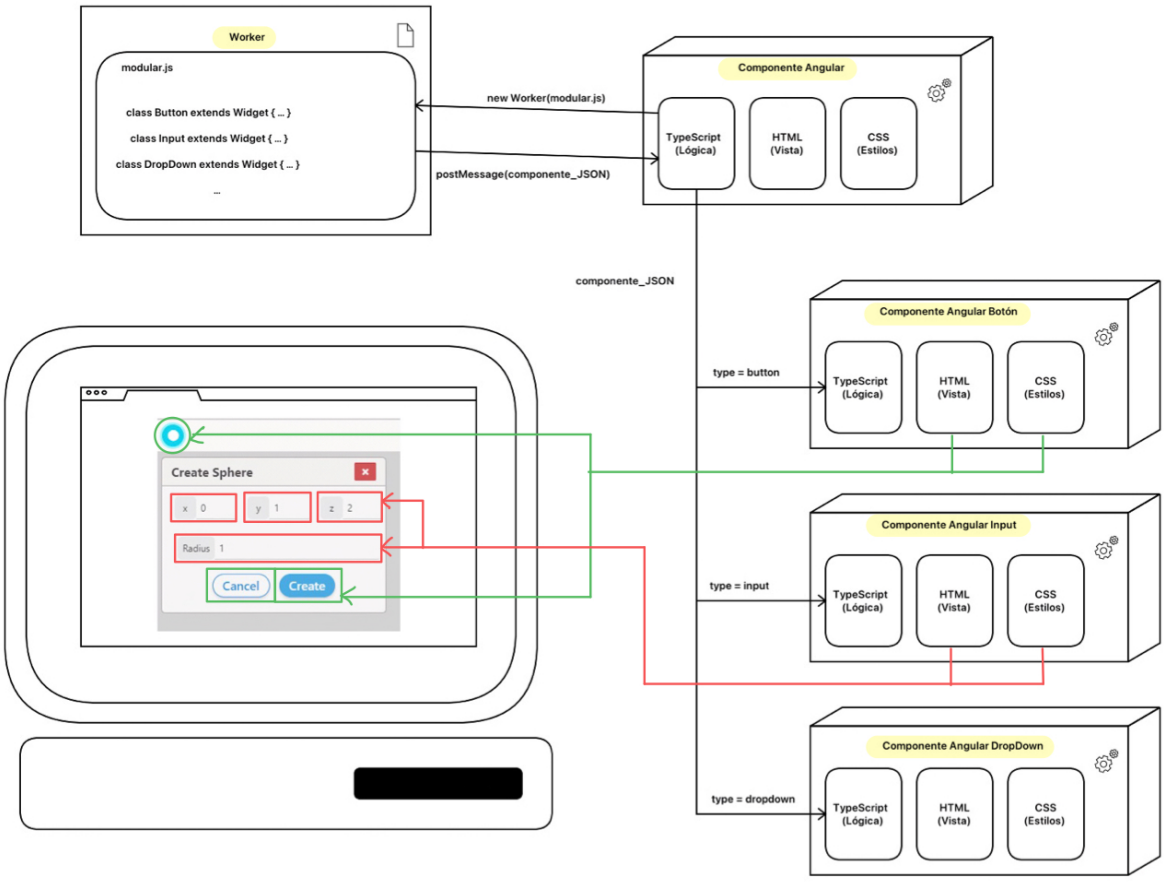


Figura 15 : Diagrama de flujo que representa la transformación de los datos desde el worker a la página web. Elaboración propia

7.4. Solución escogida

Para la solución final se han tomado principalmente tres decisiones. La primera decisión está relacionada con solucionar el problema de crear un editor, en este caso se considera que la mejor alternativa es utilizar el componente de "MonacoEditor" para complementar y facilitar la implementación que realiza el estudiante. Este componente en concreto aporta muchos beneficios y a simple vista no se han detectado inconvenientes a esta solución.

La segunda decisión que se ha tomado corresponde a cómo crear un gestor de versiones. Dentro de la aplicación que se está creando para este proyecto se ha reservado un espacio en el que se pondrán botones para poder acceder a versiones anteriores mediante la conexión con GitLab RESTFUL API [21]. Se ha decidido utilizar GitLab en vez de GitHub porque la empresa ya lo tiene descargado en sus servidores y de esta manera se aprovechan los propios servicios y recursos de la empresa. También se añadirán botones que estarán conectados con GitLab para poder abrir, guardar, crear, cerrar y eliminar ficheros.

La última decisión está relacionada con el apartado de ejecución, pero investigando se ha visto que no habían demasiadas posibles soluciones si se quería incorporar este proyecto dentro de la aplicación que el departamento de GiD está desarrollando. Por este motivo se ha decidido utilizar el método explicado anteriormente con la función eval() que tiene JavaScript para ejecutar el código y hacer esto en un thread secundario con la ayuda de los WebWorkers para conseguir aislar la ejecución del código del usuario por si surgen errores o posibles complicaciones.

8. Desarrollo

8.1. Preparación previa

Antes de empezar cualquier tarea relacionada con el desarrollo el estudiante ha tenido que realizar una preparación previa donde ha aprendido y recopilado toda la información necesaria para poder llevar a cabo el desarrollo.

Ha aprendido a programar con el framework de Angular mediante unos tutoriales de la aplicación Udemy y los manuales de ayudas que proporciona la página [51].

El conocimiento principal requerido para este proyecto es la creación de los componentes, servicios y el flujo de datos entre componentes y servicios Angular.

Estos componentes están formados por cuatro tipos de ficheros. Los primeros son los ficheros de extensión ".ts" que contienen el código de TypeScript que le dice al componente como se tiene que comportar. El segundo tipo de ficheros son de extensión ".html" y contienen el código HTML que define la estructura del componente en la WEB. El tercer tipo de ficheros son de extensión ".css" o ".sass" y contienen los estilos, proporcionan personalidad y le dan la apariencia deseada al componente. El último tipo de ficheros está relacionado con los tests unitarios y es donde se programan todas las funcionalidades que se tienen que comprobar para asegurar que el componente funciona correctamente. En cambio, los servicios Angular solo contienen el fichero de extensión ".ts" y el fichero relacionado con los tests unitarios.

Otra de las tareas importantes antes de empezar a programar consiste en preparar el entorno para ejecutar la parte de la aplicación que la empresa ya tiene desarrollada.

En primer lugar, hay que descargar las aplicaciones y paquetes necesarios para programar, como VisualStudio Code, Sourcetree, Node.js, NPM, CLI Angular entre otros. El siguiente paso es clonar el proyecto de Bitbucket, que descarga una copia del proyecto en local para trabajar y ya está todo listo para ejecutar el proyecto, empezar a mirar el código actual, entenderlo y comenzar a programar.

8.2. Diseño de la interfaz gráfica

Una vez el entorno de trabajo está preparado, se han evaluado las posibles implementaciones con sus beneficios e inconvenientes y se ha decidido cuál es la mejor solución. Es el momento de dar forma e imaginar gráficamente la parte de la aplicación que se implementará en este TFG. Para ello se crea un diseño de la interfaz gráfica que resulte cómodo e intuitivo para el usuario, esta tarea corresponde a la tarea CD4[14] del diagrama de Gantt.

La aplicación en todo momento tendrá la barra superior visible, que llamaremos "cabecera". La cabecera contendrá el logo de la aplicación y diferentes iconos para abrir menús.
La parte central de la pantalla se dividirá en dos, la superior que ocupará mucho menos espacio que la inferior. La parte central-superior será común para los diferentes editores y permitirá ver el nombre del fichero y contendrá botones que permitan gestionar los diferentes ficheros.
La parte central-inferior contendrá un componente de menú en pestañas que irá cambiando según las necesidades del usuario entre el editor, el editor con la parte de ejecución o el editor de diferencias.

A continuación, se muestran una serie de figuras que representan los elementos que tendrá la aplicación y cómo se sitúan en el espacio. En la primera se puede observar la distribución del espacio, en las siguientes observamos una posible representación de cómo quedaría la interfaz en cada pestaña. Al programar la aplicación se utilizarán elementos PrimeNG [34] que son una colección de componentes de interfaz de usuario para Angular. En las figuras podemos observar fotos reales de algunos de estos elementos de la colección PrimeNG como los

[14] Tarea del diagrama de Gantt que corresponde con el diseño de la interfaz gráfica.

botones y la línea temporal, sacados de su pagina web oficial para seguir el estilo que tiene el resto de la aplicación y hacernos una idea del resultado final.

Figura 16 : Organización de la aplicación. Elaboración propia

En la siguiente imagen el editor ocupa todo el espacio central-inferior para que el usuario pueda codificar cómodamente.

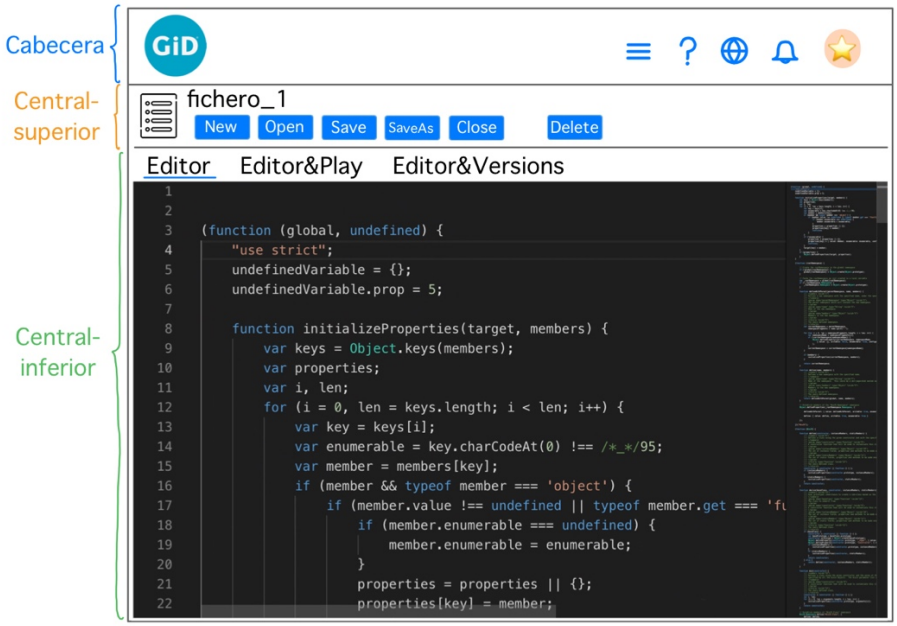


Figura 17 : Diseño de la interfaz en la pestaña Editor. Elaboración propia

En la siguiente imagen el espacio central-inferior está dividido en dos partes. La parte izquierda corresponde al editor, donde el usuario puede seguir modificando el código.

En la parte derecha se puede observar que aparecen dos recuadros. El recuadro superior es una pequeña pantalla en la que se mostrarán las ejecuciones del usuario en caso de que no se hayan cometido errores de código, en este caso se pueden apreciar un conjunto de botones de diferentes colores. El recuadro inferior quiere representar una pequeña consola de ayuda, que mostrará los mensajes de error en caso de que estén y el contenido de las llamadas a la función "conosle.log()".

Además se puede apreciar que en el panel central-superior que contiene todos los botones que permiten gestionar los diferentes ficheros, también ha aparecido un botón nuevo. Este botón es el "Play" que permite ejecutar el código que el usuario tiene escrito en el editor y mostrarlo en la pantalla de visualización.

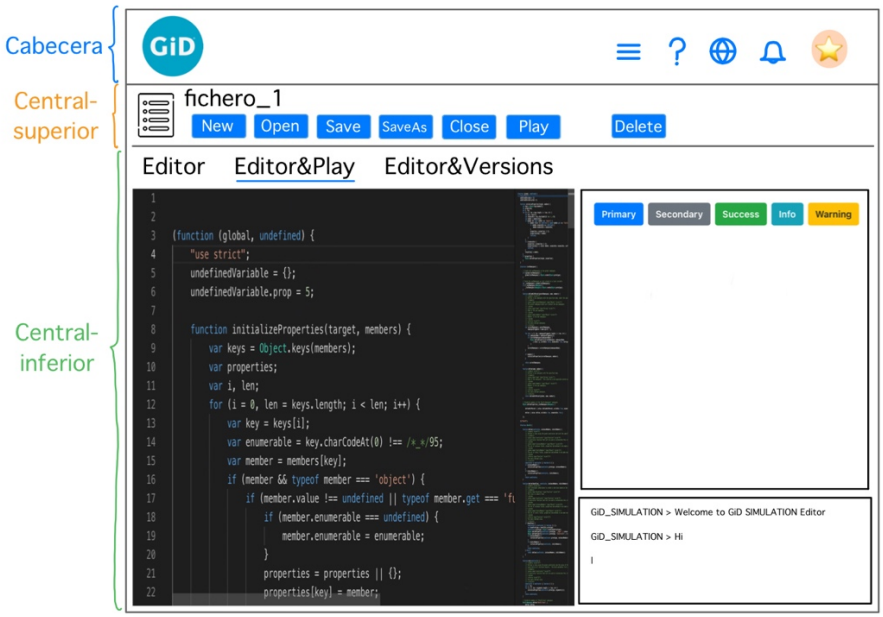


Figura 18 : Diseño de la interfaz en la pestaña Editor&Play. Elaboración propia

Por último, se muestra una figura con la pestaña del Editor&Versions. Se puede ver que el panel central-inferior también se ha dividido en dos partes. La parte de la izquierda contendrá el editor, en este caso se puede observar que el editor es algo diferente a los anteriores. Este componente muestra dos editores en uno, con ayudas visuales para que el usuario pueda ver las diferencias de código que tienen ambos editores. El editor de la izquierda corresponde a las versiones de código anteriores y el de la derecha muestra el código actual, incluso te deja seguir modificándolo.

En el panel-inferior se puede ver que la parte derecha contiene una línea temporal que muestra diferentes puntos con los mensajes relacionados a cada guardado. Los botones inferiores son más antiguos en el tiempo respecto a los que están más arriba en la línea temporal.

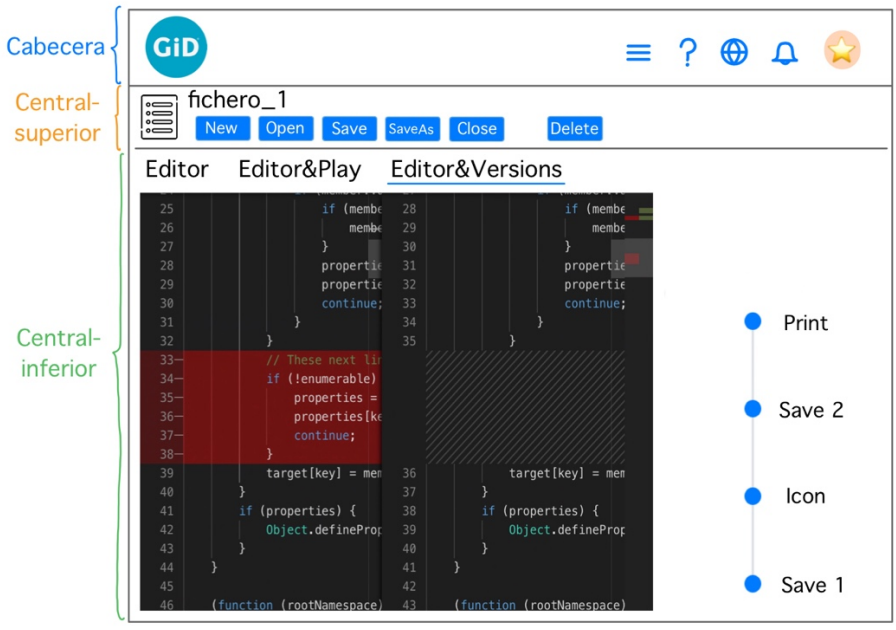


Figura 19 : Diseño de la interfaz en la pestaña Editor&Versions. Elaboración propia

8.3. Creación del editor

Es el momento de empezar a programar. Lo primero que se ha hecho es crear un componente Angular llamado **feature-studio.component** y una ruta web nueva, que llevará a este componente. Esto genera un espacio nuevo dentro de la aplicación web que se utilizará como lienzo. También se crea un servicio llamado **feature-studio.service** que de momento estará vacío, los componentes le pedirán los datos que necesiten de GitLab y el servicio hará las peticiones necesarias y retornará esa información en el formato que el componente necesita. De esta manera el componente **feature-studio.component** funcionará como controlador que se conectará con el servicio **feature-studio.service** y enviará la información a cada uno de sus componentes, que podemos denominar hijos.

Una vez creado el componente **feature-studio.component** ya se puede empezar a dar forma. En primer lugar se añade un menú de pestañas para poder cambiar entre los tres tipos de editores. Se deja un poco de espacio en la parte superior, que es donde se colocarán los botones para gestionar los ficheros.

Ahora nos centraremos en crear el editor, se crea un nuevo componente Angular que se llamará **editor.component** y añadimos uno a cada pestaña. A continuación descargamos el paquete [52] llamado **ngx-monaco-editor** y lo instalamos en el proyecto. Dentro del **editor.component.html** insertamos la etiqueta del editor. Se ha decidido que el componente editor contenga los tres tipos de editores y dependiendo del valor de la variable ***editor_type*** se mostrará uno u otro.
Para el editor normal y el de ejecución se mostrará el **ngx-monaco-editor** y para el de diferencias se mostrará el **ngx-monacodiff-editor**. Cuando se crean los editores se definen unas opciones por defecto, las más importantes son el lenguaje de programación que será JavaScript, el tema que será oscuro, activar la opción de deshacer, activar la opción de rehacer y se asignan unos tamaños para que se vean bien dependiendo del espacio en el que se muestran.

Se crean unas variables llamadas ***code***, ***codePlay***, ***originalModelcode*** y ***modifiedModelcode.*** En estas variables se almacenará el código que el usuario escribe en cada editor. Son variables de "doble binding" como las llama Angula. Esto quiere decir que se pueden actualizar en las dos direcciones. Una es desde la parte html, el usuario escribe en el componente editor y se guarda el código en la variable. El otro es desde el código, se puede asignar un valor a la variable desde el lado del programador y el usuario lo ve reflejado en el componente editor. Esto permite que el editor pueda tener código por defecto y se puedan cargar ficheros que han sido guardados previamente.

Una vez se tienen los tres editores básicos, es el momento de añadir todas las funcionalidades que queremos que tengan.

8.3.1. Actualizar código entre editores

Para que la aplicación no esté continuamente actualizándose, dado que la podría ralentizar demasiado, se ha decidido utilizar el siguiente método que aportará un factor de eficiencia y velocidad a la aplicación.
Cuando el usuario modifica el código del editor, este se guarda en la variable asociada al editor que se está utilizando en el momento. Cuando el usuario decide cambiar de pestaña para visualizar otro editor, el componente controlador **feature-studio.component,** pide el código actual al editor que se ha estado usando y lo actualiza en el nuevo. De esta manera no se actualiza la página para cada click del teclado si no en el momento de cambiar de pestaña y solo se actualizan los editores que se van a utilizar. Para poder hacer esto se han utilizado los decoradores ***@Input()*** y ***@Output()*** que tiene Angular. De esta manera se puede cambiar perfectamente entre pestañas y siempre se tendrá el código actualizado.

8.3.2. Área editable

Se quiere que el usuario sea capaz de crear ProblemTypes. Estos están formados por diferentes funcionalidades que juntas conforman esos cambios que quiere añadir el usuario a la aplicación original. El editor está pensado de la siguiente manera. En cada fichero solo se puede guardar una funcionalidad y esta tiene que venir definida por una cabecera concreta. En la siguiente imagen se puede observar la definición de la cabecera de una funcionalidad.

```
listAllFunctionalities['Untitled'] = function (context, CurrentWorkingMode, definition, id = 'Untitled') {
  // Start of editable area

  // End of editable area
}
```

Figura 20 : Función que define una funcionalidad. Elaboración propia

Para minimizar los errores que pueda cometer el usuario se ha decidido crear un área editable para que la definición de la función que contendrá la programación de esa funcionalidad no la tenga que escribir el usuario.

Para hacer esto se pone el código de la imagen anterior por defecto en el editor. Si el usuario escribe en el área que no es editable, es decir antes de la frase **// Start of editable area** o después de la frase **// End of editable area** se deshace el cambio que haya hecho. De esta manera solo se permite escribir dentro de esas frases.

8.3.3. Cambiar el nombre de la función

Como se ha comentado anteriormente, el usuario solo puede escribir dentro del área editable, así que se ha creado una función que recorre la cabecera y cambia el nombre, dejando el resto de código igual.

Esta función se utilizará cuando se cree un "nuevo" fichero o cuando se haga un "guardar como" que el usuario escribirá el nombre del fichero nuevo y este mismo nombre se utilizará para la función.

8.3.4. Autocompletado

Se quieren proporcionar ayudas al usuario a modo de manual. Solo escribiendo parte de una palabra le aparezcan las posibles opciones que quería escribir y las funciones que puede llamar relacionadas con esa palabra.

Para hacer esto se añade un **triggerCharacters** que se activa cuando se detecta un "**.**" o un "**(**" y sacando la última palabra que se está escribiendo del editor se evalúa para determinar en qué caso está.

Esta evaluación se hace mediante unas expresiones regulares que intentan encontrar unos patrones específicos dentro de la palabra que ha escrito el usuario. En este caso se buscan los patrones relacionados con las palabras **context**, **CurrentWorkingMode** o **definition[id].el_nombre_que_sea.** Una vez se sabe en qué caso se está, se añaden como sugerencias del editor todas las funciones que el usuario puede llamar a partir de esa palabra. Esto se hace mediante la función **registerCompletionItemsProvider()** que tiene el paquete de MonacoEditor.

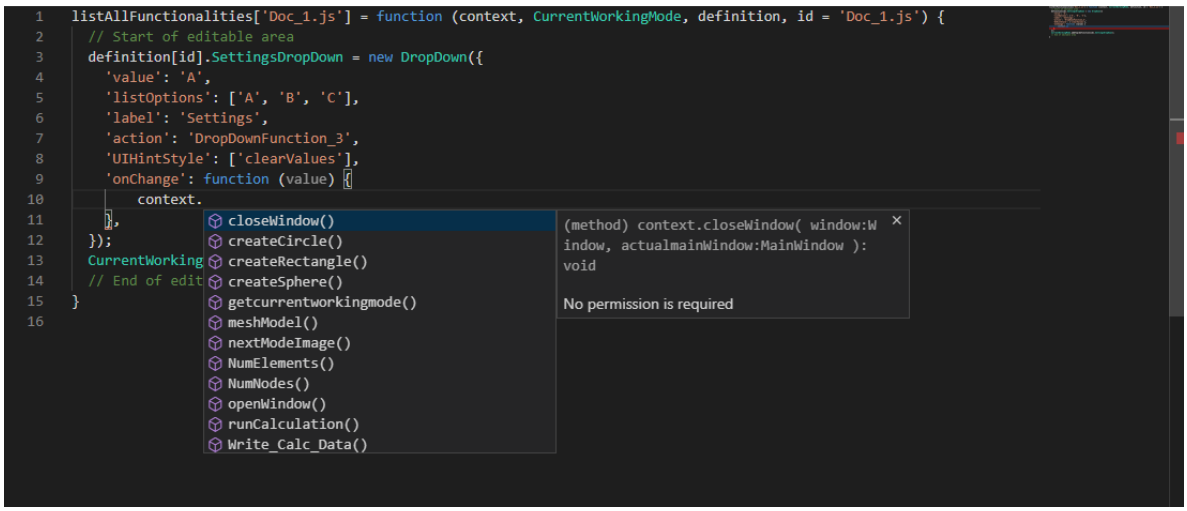


Figura 21 : Autocompletado del Editor. Elaboración propia

Cuando se inicia por primera vez el editor también se añaden unas palabras clave que el usuario puede utilizar como creadoras de los componentes html como botones, ventanas, inputs o elementos más específicos de la aplicación como las coordenadas. Si el usuario presiona el autocompletado de alguna de estas creadoras se añade la creadora con los parámetros obligatorios a modo de ayuda para que el usuario solo tenga que cambiar los valores.

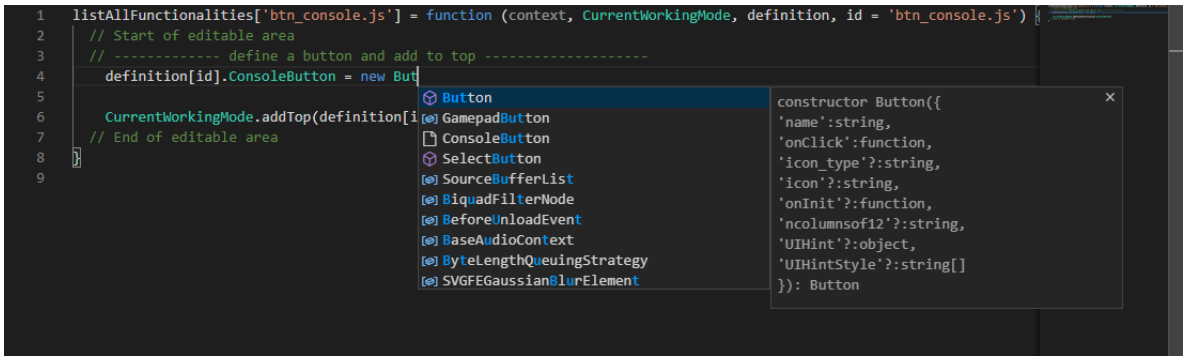


Figura 22 : Autocompletado de una creadora. Elaboración propia

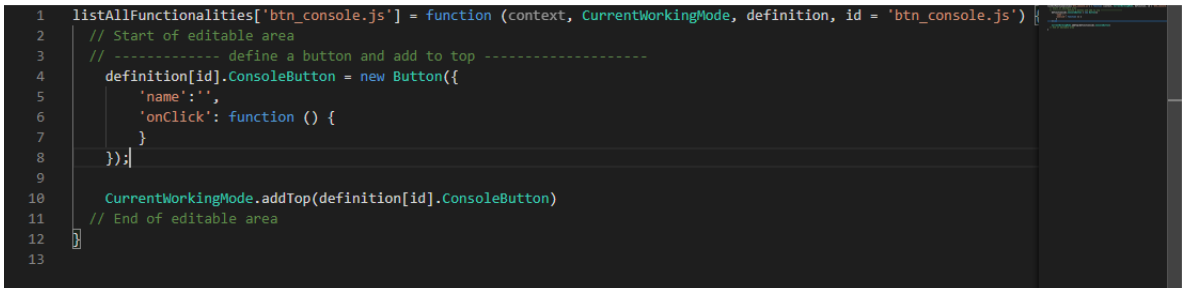


Figura 23 : Código insertado por el autocompletado. Elaboración propia

8.3.5. Validar sintaxis del código

Esta parte está relacionada con la parte de ejecución. El editor lo único que hará es recibir un listado de palabras que sintácticamente son erróneas y las resaltará. El listado lo proporcionará la ejecución, más adelante se explicará cómo se hace esta parte.

La manera en que resalta el editor es con un **marker** que en formato de error deja un subrayado ondulado rojo debajo de la palabra. Hay que indicar en la línea y columna que empieza la palabra y en la línea y columna que termina. Una vez esa palabra ha sido modificada, no se encuentra dentro del código del editor, sale de la lista de errores y se elimina el **marker.** De esta manera se puede comprobar la sintaxis más específica de funciones propias o de las creadoras que se han comentado anteriormente. Palabras clave del lenguaje JavaScript como **function, let, var, private** entre otras, las remarca el propio MonacoEditor.

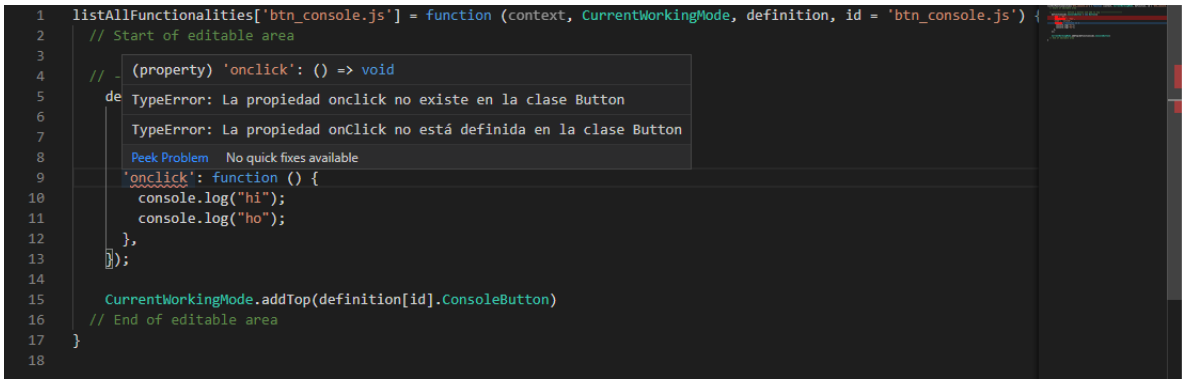


Figura 24 : Resaltado del error onclick con los mensajes de error. Elaboración propia

8.4. Creación de la línea temporal

La línea temporal la programaremos dentro de uno de los componentes que se ha creado anteriormente, el componente **feature-studio.component**. Aparecerá en el tercer panel, que tiene el nombre de **Editor, Versions** y contiene el editor de diferencias.

Se ha de añadir una barra de desplazamiento en el lado derecho. Al principio no será muy necesaria, pero cuando los ficheros acumulen muchos elementos guardados y se vean en la línea temporal, esta barra permitirá desplazarse por toda la línea temporal. Teniendo en cuenta la posibilidad de que el usuario se desplace

con frecuencia en la barra y luego quiera volver a la posición inicial, se ha añadido un botón que tendrá como única función situar la posición de la barra al principio.

Una vez añadida la barra de desplazamiento es el momento de crear la línea temporal. Para ello se usará el componente de PrimeNG con etiqueta **p-timeline**, a este componente se le tiene que añadir un listado de todos los eventos, en este caso concreto serán los mensajes de los commits que se han guardado. Es cuando se ha de utilizar el **feature-studio.service** al cual le pediremos los diferentes commits. Inicialmente esta petición tendrá una lista de retorno por defecto, pero esto no será información real hasta el momento en el que no se realice el siguiente apartado.

El componente de PrimeNG con etiqueta **p-timeline**, tiene una opción a la hora de implementar que se llama plantilla, su función es repetir el código del programador por cada evento.
Esta es la opción que se ha usado, se ha dividido el espacio en dos partes, de manera que la primera parte contiene el botón que el usuario puede presionar para ver el código de esa versión y la segunda parte es el mensaje de guardado que tiene relacionado ese código. Esto se puede ver en la siguiente imagen.

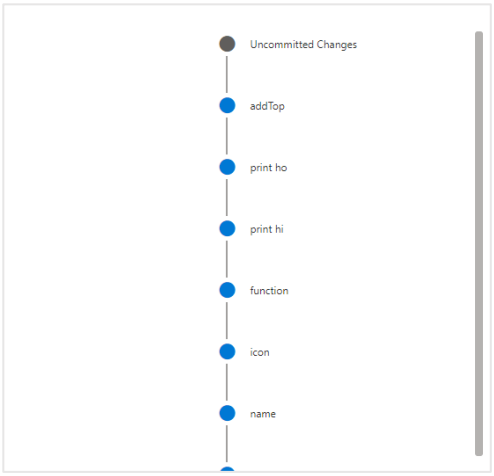


Figura 25 : Línea temporal de versiones. Elaboración propia

Cuando el usuario presiona uno de estos botones el componente **feature-studio.component** hace una petición al servicio **feature-studio.service** que le proporcionará el código de ese commit en concreto. De momento la respuesta a esta petición es la misma para todos los botones, la evolución de este punto se hará en el siguiente apartado.

En la imagen superior también se aprecia una pequeña diferencia entre los botones, esto es porque cuando se detecta que hay algún cambio en el fichero aparece un nuevo punto en la línea superior que avisa de que hay cambios no guardados. Esto se hace mediante la variable ***uncommittedChanges*** que guarda el estado del fichero, al principio el valor de esta variable es "falso" porque no se han hecho modificaciones, es en el momento en el que se cambia algo cuando realmente se asigna a la variable el valor "verdadero". Esta variable ha de estar muy presente en todo el código porque se irá modificando con los botones que gestionan los ficheros. Por ejemplo, si se guarda el código esta variable tiene que ser falsa otra vez. Hay que ser cuidadosos con el tratamiento de esta variable para que el componente controlador no pierda el estado de los cambios.

8.5. Implementación de interacciones con GitLab API

En este apartado se invertirá algo más de tiempo. Se empieza por crear los diferentes botones que gestionan los ficheros. Los botones son **New**, **Open**, **Save**, **Save As**, **Close** y **Delete**. Además, se crean dos botones más, que solo aparecerán cuando el usuario esté en la pestaña de **Editor and Run** que son **Run** y **ExecutionValues**.

Una vez creados los botones es el momento de hacer las conexiones con GitLab. Para ello se crea un repositorio nuevo en GitLab en el que se guardarán los

diferentes ficheros. Lo que se hace a continuación es crear un token de acceso porque este repositorio se ha creado privado y para poder acceder a él se necesitará este token. En el apartado del usuario de GitLab, presionar "editar perfil" lleva a una pestaña que tiene un menú en el lado izquierdo, en el que aparece la opción de **Accesse Tokens** donde se pueden crear.

Ahora en el **feature-studio.service** se añaden las librerías necesarias de Angular para hacer las peticiones http como **HttpCLient** y **HttpHeaders.** También se crean unas variables globales privadas que se utilizarán en las diferentes peticiones, son las siguientes:

- ***Project_id*** : es el identificador del proyecto de GitLab, es necesario el token para acceder al repositorio pero con el id se busca el proyecto que se necesita.
- ***url*** : todas las peticiones http tienen que hacer la conexión con GitLab por lo que necesitan empezar por esta url **http://gitlab.com/api/v4/projects**
- ***token*** : este es el token que se ha generado previamente desde la aplicación de GitLab. Seguramente se pase a una variable de entorno para proporcionar un factor de seguridad.

Una vez está todo listo para hacer las peticiones a GitLab hay que ir botón por botón creando el flujo de datos y haciendo las peticiones. En este momento el estudiante se dio cuenta de que en las peticiones se necesitaba dar el nombre del fichero nuevo, o el mensaje de guardado por ejemplo. Para conseguir esta información del usuario se ha creado un sistema de diálogos dinámicos.

Los diálogos dinámicos dejan la pantalla bloqueada y muestran una ventana con un componente dentro. La ventaja de esta estrategia es que es muy modular, ya que con el mismo diálogo se pueden tener infinitas posibilidades siempre y cuando se hayan creado los componentes que se quieren cargar.

En este caso se han creado 5 componentes diferentes que son los siguientes:

- ***dialog-confirmation*** : muestra el texto y dos botones, uno para aceptar y otro para cancelar.

- ***dialog-input*** : muestra un texto informativo y un elemento html de input donde el usuario puede escribir el nombre del fichero o el mensaje de guardado.
- ***dialog-table*** : muestra una tabla con los nombres de todos los ficheros para que el usuario elija uno. El fichero ecogido se abrirá o borrará dependiendo de qué botón haya activado el diálogo. Para obtener todos los ficheros se hace una petición al **feature-studio.service** que a su vez hace una llamada http de **GET /projects/:id/repository/tree** [53] y que retorna al componente **feature-studio.component** una lista con todos los ficheros.
- ***dialog-text*** : muestra texto informativo y un botón de aceptar.
- ***dialog-execution-values*** : este diálogo se utilizará más adelante, pero está compuesto por inputs en los cuales se tienen que insertar ciertos parámetros para la ejecución.

Estos diálogos se podrán ver con imágenes en el apartado del manual que se encuentra en el **Anexo B Manual**. Y ahora que ya está todo listo es el momento de empezar con los botones

8.5.1. Botón New

Este botón activa el diálogo de ***dialog-input*** en el que se ha de introducir el nombre del fichero que se quiere crear, se añade a la lista de ficheros el nombre y se genera el código por defecto con el nombre del fichero. Con toda esta información se hace la petición de crear el fichero al componente **feature-studio.component** que a su vez hace una petición http de tipo **POST /projects/:id/repository/files/:file_path?branch=main&content=code&commit_message=newfile&access_token=token** [54] para crear el fichero. Una vez creado se realiza la acción de abrirlo, añadiendo el código en el editor para que el usuario pueda empezar a programar.

8.5.2. Botón Open

Este botón activa el diálogo de ***dialog-table*** en el que se tiene que seleccionar el fichero que se quiere abrir. Este apartado tiene dos peticiones al componente **feature-studio.component,** una pide el código del fichero y la otra pide la lista de commits que se añadirán en la línea de versiones.

Ambas peticiones son http de tipo GET, primero nos centraremos en la petición de código que hace un **GET /projects/:id/repository/files/:file_path/ raw?ref=main&access_token=token** [55] que retorna una cadena de caracteres con el código del fichero, solo hace falta pasarlo al controlador y que este lo envíe al componente hijo **editor.component.**

La otra petición http pide el listado de commits de un fichero, esto se hace de la siguiente forma **GET /projects/:id/repository/commits?path=name& access_token=token** que retorna una lista de todos los commits de ese fichero y el servicio lo transforma al formato de la línea temporal antes de enviarlo al componente **feature-studio.component.**

8.5.3. Botón Save

Cuando se presiona el botón save aparece un diálogo ***dialog-input*** que pide al usuario el mensaje con el que se quiere guardar el estado del código. El componente **feature-studio.component** le pide al editor el código actual.

A continuación el componente hace una petición al **feature-studio.service** para que guarde el código. Esta petición está compuesta por tres pasos diferentes.

El primer paso es guardar el código en el fichero actual, para ello se hace una petición **PUT /projects/:id/repository/files/:file_path?branch=main& content=code&commit_message=mensaje&access_token=token** [56] esta petición retorna la dirección en la que se ha guardado el fichero lo que permite realizar el siguiente paso.

El segundo paso es obtener la información del fichero con la dirección mencionada anteriormente se ejecuta la siguiente petición **GET/projects/:id/ repository/files/:file_path?ref=main&access_token= token** [57]. De toda la información que obtenemos después de hacer el paso dos solo nos interesa el identificador del commit, que lo utilizaremos a continuación.

El último paso es obtener la información del commit para poder actualizar la línea temporal. Es diferente a la llamada http del botón de abrir porque en este caso solo se quiere un commit, no toda la lista. Se hace de la siguiente manera **GET /projects/:id/repository/commits/:sha?access_token=token** [58] se transforma el resultado de esta petición en el formato de la línea de versiones y se envía al componente **feature-studio.component.**

8.5.4. Botón SaveAs

Este botón tiene un procedimiento muy parecido al que se utiliza en el botón de un nuevo fichero. Lo primero es activar el dialogo ***dialog-input*** en el que se ha de introducir el nombre del fichero que se quiere crear, se añade a la lista de ficheros y se pide al editor el código actual del que se parte.

La petición que hace es la misma que la del botón New **POST /projects/:id/repository/files/:file_path?branch=main&content=code& commit_message=newfile&access_token=token** [54]

8.5.5. Botón Close

Con Close no se hace ninguna llamada http, su función es poner todas las variables del controlador a null o falso y reestablecer el código por defecto. Antes de hacer eso muestra el dialogo ***dialog-confirmation***. En caso de no haber guardado los cambios avisa para que se realice esa acción.

8.5.6. Botón Delete

Con él se activa el dialogo de ***dialog-table*** en el que se tiene que seleccionar el fichero que se quiere borrar. Una vez seleccionado se elimina de la lista de ficheros que sale en el dialogo y el componente hace la petición al servicio. Esta petición es la siguiente **POST /projects/:id/repository/commits?branch=main& commit_message=delete&actions&[{action:'delete',filePath:filePath}]** con lo que se consigue borrar el fichero.
En caso de que el fichero que se quiere borrar estuviera abierto en ese momento, desaparecería el código del editor y se pondría el código por defecto.

8.5.7. Petición del código de un commit

Cuando se presiona un botón de la línea temporal el componente **feature-studio.component** hace una petición al servicio con el identificador del commit que se ha presionado. La petición http es la siguiente **GET /projects/:id/repository/files/:file_path/raw?ref=commit_id&access_token=token** [55], esto retorna el código de una versión anterior. A continuación, se envía al componente **feature-studio.component** para que este lo envíe al editor y el usuario pueda ver las diferencias entre los dos códigos.

El resto de los botones se tratarán en el siguiente apartado.

8.6. Creación del apartado de ejecución

Lo primero que se ha hecho es crear un componente nuevo llamado **run-feture.component**. Este componente es hijo del controlador **feature-studio.component**.

El componente se divide en dos partes horizontalmente, en la parte superior se ha dejado espacio para mostrar la ejecución dentro de un elemento html de etiqueta **div** y en la parte inferior se ha añadido un recuadro que contiene una barra de

desplazamiento y otro elemento html de etiqueta **div.** En este último elemento es en el que se muestran los diferentes mensajes del componente a modo de ayuda.

Cuando el componente se inicializa se crea un WebWorker indicando la dirección del fichero que se ejecutará en segundo plano, de la siguiente manera **new Worker(dirección_fichero).** Para comunicarse entre la ejecución principal y el WebWorker se utilizan unas funciones que proporcionan las librerías de JavaScript que son las siguientes, **onmessage = function(){}**, **self.postMessage(...)** y **addEventListener((...)=>{...}).**

El fichero con el que se trabaja ya tenía muchas funciones y clases programadas por los compañeros con los que el estudiante está trabajando en la empresa. Pero para que funcione la ejecución se necesita transformar los componentes html que el usuario programa en el editor, a elementos html reales que la aplicación pueda mostrar.

Para hacer esto el estudiante ha creado alguna clase de JavaScript con el nombre de **AcceptCancel**, **Tree**, **ProgressBar**, **DropDown**, **Frame**, **Row**, **Column**, **Button**, **SelectButon**, **Input**, **Listbox**, **Coordinates**, **Window**, **Toolbar**, **Canvas** y **TreeTabel.** Todas ellas son una extensión de la clase **Widget** que ya estaba creada.
Para cada una de las clases mencionadas anteriormente se ha creado un componente Angular. En los ficheros .html se ha mostrado el componente que indica el nombre de la clase y en los ficheros .ts estará todo el código necesario para hacer la conexión con el WebWorker manteniendo los datos actualizados.

Con todas las piezas preparadas solo falta montar la ejecución. Aquí es donde entran en juego los botones de **ExecutionValues** y **Run**. Como se ha comentado anteriormente, la aplicación web se conectará a una instancia del programan GiD original, pero en el caso de la ejecución esto no pasa. Quizás en el futuro se decida

cambiar esta decisión técnica para que la experiencia de ejecución sea más real, pero para este trabajo se ha decidido hacerlo de la siguiente forma. El usuario podrá ver los componentes que ha programado y su funcionamiento, pero cuando este necesite información de la aplicación real, como por ejemplo saber el número de nodos que tiene una figura geométrica, el propio usuario será quien declare el valor de retorno.
Gracias al botón **ExecutionValues** se pueden definir estos valores que la aplicación retornará, así es más fácil comprobar casos muy concretos.

Con esto aclarado es el momento de hacer que el botón **Run** funcione. Cuando el botón se presiona, el componente controlador obtiene el código del editor y lo envía al componente **run-feature.component.** Se transforma el código a un formato ejecutable, esto quiere decir separar los argumentos del cuerpo del código para que gracias a la función de JavaScript **new Function(arguments, code)** el WebWorker pueda utilizar el método **eval()** para ejecutar esta función, acompañado de un **try{}catch(error){}** para detectar errores.

Cuando se utiliza el método **eval()** las constructoras de componentes html que ha utilizado el usuario como la clase **Button** se transforman a objetos JSON. Más tarde el componente Angular **widget.compoente** transforma el objeto JSON a un formato que el componente Angular **button.component** puede leer y mostrar el botón en el área que se había reservado para la ejecución.
Si en este proceso el método **try{}catch(error){}** detecta errores el WebWorker los envía al componente **run-feature.component** y terminará la ejecución del WebWorker. El componente **run-feature.component** los añadirá al apartado del terminal para que el usuario los vea y los pueda solucionar.
En el caso de que detecte la función **console.log()** enviará la información que tiene que aparecer en el terminal pero no terminará la ejecución del WebWorker.

8.7. Implementar tests

En este apartado se explicarán las diferentes validaciones que se han hecho del código. Realmente los tests se han implementado junto con la creación de los componentes, pero se han recopilado en este apartado para mejorar la estructura del trabajo.

Se harán tres tipos de validaciones, las dos primeras son el test **e2e** y el test **unitario.** La última prueba es algo diferente, no requiere programación externa al trabajo, pero si que necesitará la colaboración de compañeros de la empresa. Utilizarán la aplicación para reportar una serie de criticas constructivas.

Los compañeros de la empresa han preparado todo el entorno para que el estudiante pueda programar los tests.

Para el test **e2e** ha sido necesario instalar **protractor** [59] que es un programa de Node.js que utiliza **Jasmine** [60] que es un framework de testeo. También se ha descargado **Cucumber** [61] para definir los tests con frases simples que puede escribir cualquier persona, incluso si no ha participado en la programación del proyecto. Las frases tienen las palabras claves **Given**, **When** y **Then.**

Aquí es donde entra el trabajo del estudiante. Recopila las frases en ficheros dentro de la carpeta **features** y por cada una crea una función en el fichero **common.steps.ts** de los tipos siguientes :

- **Given(`frase_del_test`, (variables) => { código_del_test })**
- **When(`frase_del_test`, (variables) => { código_del_test })**
- **Then(`frase_del_test`, (variables) => { código_del_test })**

El código del test recorre la aplicación y mediante funciones como las que aparecen en la siguiente imagen se seleccionan componentes, se hace click en ellos o incluso se escribe texto si es el caso de un input o el editor para comprobar que la funcionalidad de los componentes es correcta y cohesiona bien con la aplicación.

```
When(`I type {string} into editor {string}`, async(text,editorName)=>{
  if(browser.params.writeSteps=="true") console.log(`When I type "${text}" into editor "${editorName}"`);

  let tabs = await element(by.tagName(`app-feature-studio`)).all(by.tagName(`p-tabpanel`));
  let text_area, lines_area;
  for(let i = 0; i < tabs.length; i++){
    if( await tabs[i].getAttribute("header") === `${editorName}`){
      lines_area = await tabs[i].element(by.tagName(`div ngx-monaco-editor`)).element(by.css(`div[class="view-lines"][role="presentation"]`))
      text_area = await tabs[i].element(by.tagName(`div ngx-monaco-editor`)).element(by.css(`textarea[class="inputarea"]`));
      break
    }
  }
  //la linea 3 está dentro del area editable
  let line_3 = await lines_area.all(by.tagName(`div`)).get(2).all(by.tagName(`span`));
  await line_3[line_3.length-1].click();
  let text_line_3 = await line_3[line_3.length-1].getText();

  if((line_3.length == 2 && text_line_3 === " ") || text_line_3.substr(-1) != " ") await text_area.sendKeys(`${text} `);
  else await text_area.sendKeys(`${text}`);

});
```

Figura 26 : Ejemplo de un test e2e. Elaboración propia

Este tipo de test comprueba ejecuciones completas como abrir un fichero, hacer algún cambio y guardarlo, comprueba que la línea de versiones se haya actualizado y finalmente cierra el fichero. Este caso se puede ver en la siguiente imagen.

```
Scenario: Identified user can open a file, save a change and then close it inside Editor area
  # Abrir fichero
  When I press "OpenFile" button
  Then dialog with title "Choose a File" appears
  When I select "3_e2e_test.js" file inside the dialog
  And I press "Open" button
  Then  dialog with title "Choose a File" closes
  And file name changes to "3_e2e_test.js"
  And function name from "Editor" changes to "3_e2e_test.js"
  # Guardar un cambio
  When I type "let b;" into editor "Editor"
  Then editor "Editor" contains "let b;" text inside function "3_e2e_test.js"
  When I press "SaveFile" button
  Then dialog with title "Save 3_e2e_test.js" appears
  When I type "save_3_e2e_test" into dialog input "infoFile"
  And I press "Save" button
  Then  dialog with title "Save 3_e2e_test.js" closes
  When I press "Editor and Versions" tab
  Then editor "Editor and Versions" contains "editor original" and "editor modified" as visible
  And I can see commit with message "save_3_e2e_test"
  When I press "Editor" tab
  # Cerrar el fichero
  When I press "CloseFile" button
  Then dialog with title "Confirmation" appears
  When I press "Yes" button
  Then  dialog with title "Confirmation" closes
  And file name changes to "Untitled"
  And function name from "Editor" changes to "Untitled"
  And editor "Editor" contains default text
  When I press "Editor and Run" tab
  Then editor "Editor and Run" contains default text
  When I press "Editor and Versions" tab
  Then editor "Editor and Versions" contains default text on the "editor original" part
  And editor "Editor and Versions" contains default text on the "editor modified" part
```

Figura 27 : Ejemplo de una ejecución completa de un test e2e. Elaboración propia

Los tests **unitarios** están incorporados dentro de Angular. Al ejecutar ***ng test*** se abre una instancia de **Karma test runner** [62] y con la información de los ficheros Angular .spec pasa los tests. Los ficheros .spec contienen una instancia del componente que se está testeando con funciones definidas de la siguiente manera:

- **it(`frase_del_test`, () => { código_del_test }**

En este caso las comprobaciones que hacen los test están relacionadas con la funcionalidad del componente que se está testeando, para que las funciones que se testean se ejecuten correctamente. Por ejemplo, si el componente tiene una función que cambia el nombre del fichero del editor, se mira el nombre del fichero antes y después de ejecutar la función que se quiere testear, y finalmente se comprueba que se haya modificado correctamente. La cantidad de tests **unitarios** es algo menor de lo esperado dado que se han desarrollado muchas funcionalidades y no ha dado tiempo a comprobar cada función individualmente, solo las que se han considerado más importantes.

A continuación, hay una imagen con el test explicado anteriormente

```
59    fit('setFunctionName() works', () => {
60      fixture.detectChanges();
61      component.editor_type = 0;
62      component.code = default_code;
63      fixture.detectChanges();
64
65      expect(component.code).toContain("listAllFunctionalities['Untitled']")
66      component.setFunctionName('Doc_1');
67      fixture.detectChanges();
68      expect(component.code).toContain("listAllFunctionalities['Doc_1']")
69    });
```

Figura 28 : Ejemplo de un test unitario. Elaboración propia

Estos tests son totalmente reproducibles. El test **unitario** tiene que estar siempre que se programe en ejecución para asegurar que nada de lo añadido puede hacer que lo existente falle y el test **e2e** se ha de pasar al menos una vez antes de

guardar los cambios en la rama master para comprobar que el componente programado actúa como es debido con el resto de la aplicación.
Para pasar los tests **e2e** primero hay que ejecutar **yarn run start-local,** que prepara la aplicación web, y luego **yarn run e2e-over-running,** que pasa los tests. En el caso de los **unitarios** hay que ejecutar **yarn run test.**

Para el último tipo de prueba el estudiante hizo una pequeña exposición de todo lo implementado al departamento de GiD, esta reunión se hizo por las mismas fechas que la reunión de seguimiento con la ponente del trabajo. Fue una reunión muy constructiva en la que salieron muchas ideas nuevas y posibles mejoras de elementos actuales. Pero este tema se comentará en el apartado de Acciones futuras y se verá más adelante. Una vez expuesto el contenido y una breve explicación de cómo se han implementado las funcionalidades, todos los miembros del departamento han tenido acceso al proyecto para poder descargarlo y ejecutarlo en sus máquinas. De esta manera han podido probarlo y dar su opinión de usabilidad y funcionamiento.

Los tres tipos de pruebas han reportado algún que otro error. En el momento en que esto pasa, se detiene la programación de tests nuevos y se crea una tarea de tipo **bug.** Se soluciona el error lo antes posible para continuar testeando.

9. Estado final y desviaciones

A continuación, se hará un breve recordatorio de los objetivos y los requisitos relacionados con cada objetivo, y se evaluarán para corroborar que se han alcanzado los objetivos propuestos y que el proyecto realmente resuelve el problema planteado. También se explicarán los cálculos de las desviaciones que se han tenido en cuenta durante todo el proyecto para verificar que no se ha sobrepasado el presupuesto inicial.

9.1. Cumplimiento de los objetivos

9.1.1. Resumen de los requisitos del proyecto

Como se ha comentado en el aparatado de objetivos, se considera que el trabajo ha terminado cuando un usuario es capaz de navegar por la aplicación, seleccionar el Editor y escribir el código, ejecutarlo y visualizar las versiones anteriores.

Los requisitos principales que ha de cumplir **el editor** para que se considere funcional y terminado son los siguientes:

1) El usuario puede escribir código en un área editable.
2) Se pueden realizar funciones básicas de los editores como rehacer y deshacer lo que se ha escrito.
3) El editor tiene ayudas de autocompletado.
4) El editor permite crear ficheros nuevos.
5) El editor permite visualizar el código de ficheros que se han guardado anteriormente.
6) El editor permite guardar cambios y guardar como nuevo fichero.
7) El editor permite cerrar un fichero.
8) El editor permite borrar un fichero.

Los requisitos para cumplir el objetivo de **poder controlar las versiones** son los siguientes:

1) El usuario ha de poder ver un listado con las modificaciones anteriores de ese fichero.
2) El usuario puede comparar el código de una versión con el código actual.

El último objetivo es tener un apartado para **ejecutar el código**, y se considera terminado cuando cumpla los siguientes requisitos:

1) El usuario puede ver la ejecución de su código.
2) El usuario puede interactuar con la ejecución del código.
3) Se detectan los errores en el código.
4) Los errores y mensajes de ayuda se muestran en un terminal.

Estos han sido los requisitos funcionales relacionados con un objetivo concreto, pero no hay que olvidar los requisitos no funcionales, que se nombran a continuación:

1) Las implementaciones del trabajo se han de integrar bien con la parte de la aplicación web que está creada.
2) La aplicación ha de ser eficiente.
3) El usuario encuentra sencillo y cómodo el manejo de la aplicación.
4) Los componentes creados son modulares y se pueden reutilizar en otras partes de la aplicación.

9.1.2. Evaluación del cumplimiento de los requisitos

Una vez recordados todos los requisitos es el momento de evaluarlos uno a uno para determinar si el proyecto resuelve el problema planteado inicialmente.

Se empieza por evaluar el **objetivo del editor**, se corrobora que se han desarrollado todas las funcionalidades y que han pasado los tests.

El requisito **1) El usuario puede escribir código en un área editable** está implementado y explicado en el apartado **8.3.2. Área editable,** también se ha

comprobado su funcionamiento dentro de un test **e2e.** Por lo tanto, podemos decir que este requisito se cumple satisfactoriamente.

El siguiente es **2) Se pueden realizar funciones básicas de los editores como rehacer y deshacer.** Esta funcionalidad ya estaba incorporada con el editor y solo se ha tenido que activar. Como también pasa el test, consideramos que el requisito ha sido cumplido.

El requisito **3) El editor tiene ayudas de autocompletado,** se ha explicado en el apartado **8.3.4. Autocompletado,** se comprueba como parte de un test **e2e** y con eso se puede ver que también se cumple.

Los requisitos **4) El editor permite crear ficheros nuevos, 5) El editor permite visualizar el código de ficheros que se han guardado anteriormente, 6) El editor permite guardar cambios y guardar como nuevo, 7) El editor permite cerrar un fichero** y **8) El editor permite borrar un fichero,** han sido explicados en los apartados **8.5.1. Botón New, 8.5.2. Botón Open, 8.5.3. Botón Save y 8.5.4. Botón Save As, 8.5.5. Botón Close y 8.5.6. Botón Delete** respectivamente. Se han ejecutado tests **e2e** para comprobar el funcionamiento. Con todos los requisitos cumplidos podemos decir que el objetivo del que hablamos se cumple satisfactoriamente, por tanto el proyecto ha superado el objetivo del editor.

Para comprobar que se cumple el **objetivo de poder controlar las versiones**, hay que valorar dos requisitos. El primero es **1) El usuario ha de poder ver un listado con las modificaciones anteriores de ese fichero** y la aplicación se lo ha de permitir. El desarrollo ya ha sido explicado en los apartados **8.4. Creación de una línea temporal** y en **8.5.2. Botón Open.** Los tests comprueban que cuando se abre un fichero aparece la línea temporal y que al ser guardado se actualiza, por lo que daremos este requisito por cumplido y finalizado.

El segundo requisito es **2) El usuario puede comparar el código de una versión con el código actual,** gracias al apartado **8.5.7. Petición del código de un commit** vemos que la aplicación permite obtener el código de una versión

anterior y en el apartado **8.3. Creación del editor** nos cercioramos de que el código se mostrará en un editor específico que señala las diferencias con el código actual. El test demuestra que en cada parte del editor se enseñan versiones diferentes del código y al pasar los diferentes test se puede decir que el trabajo cumple con el objetivo.

El último objetivo que es **tener un apartado para ejecutar código,** ha de cumplir con cuatro requisitos diferentes que van asociados. Todos ellos han sido desarrollados y se explican en el apartado **8.6. Creación del apartado de ejecución.** Además, el requisito **3) se detectan los errores en el código,** también ha sido explicado en el apartado **8.3.5. Validar sintaxis del código.** Se han ejecutado algunos tests para comprobar los decoradores de error y los mensajes que aparecen en el componente del terminal. Con esto se puede decir que el proyecto también cumple el último de los objetivos planteados.

No nos tenemos que olvidar de los requisitos no funcionales, que pueden pasar desapercibidos a simple vista, pero también son muy importantes para el buen funcionamiento de la aplicación. El primero, **1) Las implementaciones del trabajo se han integrado bien con la parte de la aplicación web,** se ha cumplido. Todo lo desarrollado está en una pestaña dentro de la aplicación web y sigue el estilo y formato del conjunto existente. También hay que tener en cuenta si se cumple el siguiente requisito **2) La aplicación ha de ser eficiente,** que se ha tenido en cuenta a la hora de pensar las estrategias a usar y la implementación. Es cierto que han surgido ideas nuevas para mejorar la eficiencia, pero se comentarán en el apartado de Acciones futuras. Otro requisito muy importante es **3) El usuario encuentra sencillo y cómodo el manejo de la aplicación.** De momento no se tiene la suficiente información como para decir que este requisito se cumple. Seguramente cuando más gente utilice el editor surgirán dudas del funcionamiento. Como en cada aplicación nueva se necesita un pequeño periodo de adaptación. Para facilitar esto se han puesto todas las acciones visibles y

ejecutables con botones, de modo que es muy intuitivo para el usuario, y para cualquier duda futura que pudiera surgir también se ha hecho un pequeño manual que está en el **Anexo B**.

Ya solo falta hablar del requisito **5) Los componentes creados son modulares y se pueden reutilizar en otras partes de la aplicación**. Es un requisito no funcional importante para la empresa, pero no tanto de cara al usuario. Igualmente se han creado servicios que conectan con componentes y se han separado las funcionalidades en diferentes componentes para que sean reusables.

Con estas últimas verificaciones vemos que el proyecto cumple con todos los objetivos y requisitos planteados inicialmente.

¿Esto quiere decir que el TFG resuelve y proporciona una solución a la necesidad planteada? La respuesta es sí, el usuario es capaz de programar sus funcionalidades en un entorno que permite ejecutarlas y además ver las versiones anteriores. Ahora la aplicación web de GiD será más completa y tendrá funcionalidades que la actual no tiene.

9.2. Desviaciones del proyecto

Como se ha comentado en el apartado anterior el proyecto ha resuelto el problema planteado, pero ahora falta determinar si lo ha hecho con el presupuesto establecido al principio o se ha pasado de costes. Esto se puede saber con las desviaciones. Primero se estudiarán las desviaciones de la planificación temporal que repercutirán en el presupuesto y después nos centraremos en las desviaciones del presupuesto como tal.

9.2.1. Desviaciones en la planificación

Al principio del proyecto se decidieron los objetivos y se dividieron en tareas. Se evaluaron cada una de las tareas minuciosamente para poder determinar el trabajo

que requería cada una y poder hacer una estimación del número de horas que se emplearían.
Con estos datos se preparó una planificación inicial que se puede ver en las imágenes del diagrama de Gantt del **Anexo A** con el nombre de "**Diagrama de Gantt inicial**" y en el apartado de Planificación inicial.
Se puede decir que, a excepción de algún pequeño detalle o modificación en el orden de tareas simultáneas, se ha seguido la planificación inicial en su gran mayoría al pie de la letra.

Para cada tarea se ha evaluado el desvío en coste por tarifa gracias a la desviación del coste personal por tarea, pero como la retribución de cada rol no ha variado, esta desviación siempre será 0:

$$(\boldsymbol{CosteEstimado - CosteReal}) * \boldsymbol{HorasReales}$$

Jefe del Proyecto -> $(\boldsymbol{39€ - 39€}) * \boldsymbol{HorasReales} = \boldsymbol{0}$

Investigador -> $(\boldsymbol{26€ - 26€}) * \boldsymbol{HorasReales} = \boldsymbol{0}$

Los 2 programadores y tester -> $(\boldsymbol{20,80€ - 20.80€}) * \boldsymbol{HorasReales} = \boldsymbol{0}$

Y el desvío en eficiencia que corresponde a la desviación de la realización de tareas con la siguiente fórmula:

$$(\boldsymbol{HorasEstimadas - HorasReales}) * \boldsymbol{CosteReal}$$

Por último, se evaluará el desvío total de horas con la siguiente fórmula:

$$(\boldsymbol{HorasEstimadas - HorasReales})$$

Y la desviación total en la realización de tareas:

$$(\boldsymbol{CosteEstimadoTotal - CosteRealTotal})$$

Las tareas de documentación se han realizado según lo previsto. Concretando un poco más, las entregas de la asignatura de GEP terminaron a finales de octubre y la memoria se ha estado escribiendo a medida que se han terminado las implementaciones. Además, se preparó una pequeña presentación de lo que el estudiante había implementado desde el inicio del trabajo hasta la reunión del hito de seguimiento intermedio, para que tanto el tutor de la FIB como los compañeros de la empresa de CIMNE pudieran ver el trabajo realizado hasta ese momento.
A continuación, se pueden ver los cálculos de la desviación de eficiencia que son 0.

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) * \boldsymbol{CosteReal} =$$

Jefe del Proyecto -> $(\boldsymbol{22H} - \boldsymbol{22H}) * \boldsymbol{39€} = \boldsymbol{0}$

Investigador -> $(\boldsymbol{12H} - \boldsymbol{12H}) * \boldsymbol{26€} = \boldsymbol{0}$

Los 2 programadores y tester -> $(\boldsymbol{12H} - \boldsymbol{12H}) * \boldsymbol{20,80€} = \boldsymbol{0}$

Las tareas relacionadas con la preparación previa también se han hecho en el tiempo estimado de la planificación inicial, no ha sido necesario utilizar el tiempo de extensión reservado para la tarea **CP4 - Aprendizaje de Angular y TypeScript**. Los cálculos de la desviación de eficiencia también son de 0.

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) * \boldsymbol{CosteReal} =$$

Investigador -> $(\boldsymbol{35H} - \boldsymbol{35H}) * \boldsymbol{26€} = \boldsymbol{0}$

Los 2 programadores -> $(\boldsymbol{25H} - \boldsymbol{25H}) * \boldsymbol{20,80€} = \boldsymbol{0}$

Igualmente se han terminado dentro del plazo de la planificación previa, las tareas de diseño. Por otra parte, la tarea **CD4 - Diseño de la interfaz gráfica** se ha hecho en paralelo al resto de tareas de diseño por comodidad del estudiante, lo cual no influye en las horas dedicadas a esa tarea. La desviación también es 0.

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) * \boldsymbol{CosteReal} =$$

Investigador -> $(\mathbf{44H} - \mathbf{44H}) * \mathbf{26€} = \mathbf{0}$

programador gráfico -> $(\mathbf{16H} - \mathbf{16H}) * \mathbf{20,80€} = \mathbf{0}$

Las tareas de implementación también se han realizado según la planificación inicial, aunque han sufrido una ligera modificación, ya que se ha decidido realizar en primer lugar las tareas de implementación y dedicar el tiempo reservado a las extensiones de las tareas al final. Así, se utilizará este tiempo para solucionar imprevistos, errores o incluso hacer mejoras. Como se ha invertido el tiempo reservado a las extensiones de las tareas **CI1 - Crear el editor**, **CI2 - Crear línea temporal**, **CI3 - Implementar interacciones con GitLab Api** y **CI4 - Implementar el modo Debug**, el coste del personal se ha incrementado respecto al coste que se estimó en un principio, esto se ve gracias a las siguientes desviaciones.

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) * \boldsymbol{CosteReal} =$$

Programador -> $(\mathbf{125H} - \mathbf{167.5H}) * \mathbf{20,80€} = \mathbf{-884}$

Programador gráfico -> $(\mathbf{75H} - \mathbf{107.5H}) * \mathbf{20,80€} = \mathbf{-676}$

En total -> $\mathbf{-884 - 676 = -1.560}$

El coste del personal se ha incrementado en 75 horas de implementaciones más. Están distribuidas entre los roles de Programador de funcionalidades y gráficos que cobran unos 20,80 € la hora contando seguridad social. Esto hace que los costes de personal se hayan incrementado en **1.560€**.

Las tareas de pruebas se han realizado según lo previsto, se pueden apreciar los cálculos de las desviaciones que han resultado ser 0.

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) * \boldsymbol{CosteReal} =$$

Programador de funcionalidades -> $(\mathbf{50H} - \mathbf{50H}) * \mathbf{20,80€} = \mathbf{0}$

Tester -> $(\mathbf{25H} - \mathbf{25H}) * \mathbf{20,80€} = \mathbf{0}$

Respecto a las tareas de seguimiento se ha hecho una pequeña modificación con relación a la reunión de hito intermedio, que se llevó a cabo la semana del 22 al 26 de noviembre. Se consideró que de esta manera el estudiante podía enseñar al tutor de la FIB gran parte del trabajo a falta de terminar o corregir algunos errores. Las reuniones semanales se han mantenido de la forma acordada al principio del trabajo y por este motivo se ha decidió aplazar la reunión de hito intermedio. Semana a semana el tutor de la empresa ha visto el avance y evolución del TFG, y los problemas que han ido surgiendo. Lo que ha ayudado al estudiante a cerciorarse de que se han implementado los diferentes puntos tal y como la empresa deseaba. Como no se han modificado las horas dedicadas a esta parte las desviaciones son 0.

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) * \boldsymbol{CosteReal} =$$

$$(\boldsymbol{22H} - \boldsymbol{22H}) * \boldsymbol{2.537,60€} = \boldsymbol{0}$$

Jefe del Proyecto -> $(\boldsymbol{6H} - \boldsymbol{6H}) * \boldsymbol{39€} = \boldsymbol{0}$

Investigador -> $(\boldsymbol{4H} - \boldsymbol{4H}) * \boldsymbol{26€} = \boldsymbol{0}$

Los 2 programadores y tester -> $(\boldsymbol{4H} - \boldsymbol{4H}) * \boldsymbol{20,80€} = \boldsymbol{0}$

Este es el resultado final con respecto a la planificación inicial, lo que muestra que la desviación ha sido mínima ya que se observa en un solo apartado.

Por tanto, dado que los objetivos no han sufrido modificaciones, las tareas a realizar continúan siendo las mismas con alguna alteración en el orden de su ejecución por la comodidad del estudiante, algunas se han hecho en paralelo y otras se han subordinado a unas previas.

Los cambios se pueden ver en las imágenes del diagrama de Gantt del **anexo A** con el nombre de "**Diagrama de Gantt final**".

Ahora se calculará la desviación total en la relación de tareas:

$$(\boldsymbol{CosteEstimadoTotal} - \boldsymbol{CosteRealTotal}) =$$
$$(\mathbf{14.059€} - \mathbf{15.619€}) = -\mathbf{1.560€}$$

Y el desvío total de horas

$$(\boldsymbol{HorasEstimadas} - \boldsymbol{HorasReales}) = (\mathbf{602H} - \mathbf{677H}) = -\mathbf{75H}$$

9.2.2. Desviaciones del presupuesto

En el apartado anterior se han visto diferentes cambios en las extensiones de las tareas de desarrollo, esto se ha visto reflejado en las diferentes desviaciones temporales que han provocado desviaciones en el coste del proyecto. A continuación se calculará el coste final del trabajo teniendo en cuenta las desviaciones anteriores y las desencadenadas con respecto a los recursos, imprevistos, electricidad y contingencia.

Costes de personal

El coste del personal ha incrementado en 1.560€ al utilizar el tiempo extra de desarrollo dado que los roles de programadores han invertido más horas en esas tareas. El coste de personal asciende a un total de **15.619€**

Costes genéricos

Los costes genéricos también han cambiado dado que dependen de las horas totales de uso. En este apartado se utilizará la desviación total de recursos con la siguiente formula:

$$(\boldsymbol{CosteEstimadoTotal} - \boldsymbol{CosteRealTotal})$$

Al principio del trabajo también se consideró importante tener una desviación propia de la electricidad dado que al empezar el trabajo el precio de la luz aumentaba constantemente, la formula utilizada es la siguiente:

$$(\boldsymbol{CosteEstimadoElectricidad} - \boldsymbol{CosteRealElectricidad})$$

Se empezará por recalcular la amortización del Hardware dado que las horas de uso del ordenador de sobremesa han aumentado, las del portátil siguen siendo las mismas.

$$\frac{\boldsymbol{CosteDispositivo}(€) * \boldsymbol{HorasUso}(\boldsymbol{horas})}{\boldsymbol{Vida}\text{Útil}(\boldsymbol{años}) * \boldsymbol{DíasTrabajados}(\boldsymbol{dias}/\boldsymbol{año}) * \boldsymbol{HorasDiarias}(\boldsymbol{horas}/\boldsymbol{dia})} =$$

$$\frac{\mathbf{2.000}(€) * (\mathbf{420}(\boldsymbol{horas}) + \mathbf{75}(\boldsymbol{horas}))}{\mathbf{4}(\boldsymbol{años}) * \mathbf{236}\left(\frac{\boldsymbol{dias}}{\boldsymbol{año}}\right) * \mathbf{5}\left(\frac{\boldsymbol{horas}}{\boldsymbol{dia}}\right)} = \mathbf{210}€$$

El coste inicial de la amortización del ordenador de sobremesa era de 178€, si le restamos los 210€ de la desviación actual, el resultado es una amortización de **32€** más.

Los costes relacionados con el software seguirán siendo los mismos. Los cálculos se han hecho por precio mensual y como no se ha aumentado el numero de meses del trabajo ese precio no varía.

Los últimos costes que se han de tener en cuenta son los relacionados con el alquiler, internet y electricidad. El alquiler y el coste de internet tampoco han variado dado que el numero de meses del trabajo no ha aumentado.

El coste final de la electricidad ha aumentado por dos razones. Por una parte el número de horas de uso de ordenador ha sido mayor y por otra parte, el coste del

kWh que se utilizó al principio como valor de electricidad media era de 0.27592€/kWh y a lo largo del trabajo se ha detectado un valor medio máximo de 0.49195€/kWh. Para los cálculos actualizados se ha hecho una media de los dos valores y se ha utilizado 0.383935€/kWh.

El coste final no será de 27,3€ sino que se incrementa en **16,91€.** La siguiente tabla muestra los precios actualizados.

Otros	Potencia	Unidades	Horas	Consumo	Coste
Monitor	40W	1	495h	19,8kWh	7,61€
Ordenador uso	180W	1	495h	89,1kWh	34,21€
Ordenador apagado	4W	1	182h	0,73kWh	0,29€
Portátil	30W	1	182h	5,46kWh	2,10€
Total	**-**	**-**	**-**	**-**	**44,21€**

Tabla 10 : Costes de energía actualizados. Elaboración propia

Una vez se tienen todos los costes actualizados, se suman los costes del Hardware, Software y Otros que dan un valor de 1.315,71€ que redondeado son **1.316€**. Esta cantidad es la que se utilizará en el cálculo de las desviaciones.

La deviación total de recursos:

$$(\boldsymbol{CosteEstimadoTotal} - \boldsymbol{CosteRealTotal}) =$$
$$\boldsymbol{1.267€ - 1.316€ = -49}$$

Como el precio de la electricidad ha subido en parte porque se ha hecho un poco más de gasto del esperado y el otro motivo es porque ha subido el precio, a continuación se puede ver el cálculo de la desviación:

$$(\boldsymbol{CosteEstimadoElectricidad} - \boldsymbol{CosteRealElectricidad}) =$$
$$\boldsymbol{27,3€ - 44,21€ = -16,91€}$$

Los costes genéricos han aumentado **49€**

Contingencia

La contingencia tenía en cuenta posibles imprevistos como el aumento de personal o algunos costes generales. El aumento de personal en situaciones reales se puede causar si algún miembro del equipo necesita la baja y se ha de cubrir esa plaza contratando a otra persona. Como este proyecto solo lo realiza el estudiante, este caso no se dará y no será necesario utilizar el dinero.

$$(\boldsymbol{CosteEstimadoContingencia} - \boldsymbol{CosteRealContingencia}) =$$
$$(\boldsymbol{2.939€} - \boldsymbol{0€}) = \boldsymbol{2.939€}$$

Imprevistos

Los imprevistos tienen en cuenta los riesgos que se identificaron al principio del proyecto como el aumento de tiempo en algunas tareas o la necesidad de adquirir un nuevo ordenador para trabajar. Finalmente se ha aumentado el coste del personal resultando un total de 1.560€, y el coste general con un resultado de 49€. Por otra parte, no ha sido necesario adquirir ningún dispositivo nuevo.

$$(\boldsymbol{CosteEstimadoImprevistos} - \boldsymbol{CosteRealImprevistos}) =$$
$$(\boldsymbol{1.261€} - \boldsymbol{1.609€}) = \boldsymbol{-348€}$$

Esto quiere decir que los gastos finales han aumentado, superando la previsión inicial de imprevistos en **348€**, pero como de contingencia no se ha gastado nada, no se supera el cálculo de presupuesto inicial. Esto se puede ver en el siguiente apartado.

Coste total

Para calcular el presupuesto final se tiene en cuenta los costes genéricos y de personal, la contingencia y imprevistos solo forman parte del presupuesto inicial. En la siguiente tabla se puede apreciar el coste final.

Descripción	Coste
Coste del personal	15.619€
Coste genérico	1.316€
Presupuesto Final	**16.935€**

Tabla 11 : Coste final del proyecto. Elaboración propia

En la planificación inicial se estimaron unos 19.526€

$$(\boldsymbol{CosteEstimado} - \boldsymbol{CosteReal}) = (\mathbf{19.526€} - \mathbf{16.935€}) = \mathbf{2.591€}$$

Eso quiere decir que al dar una desviación positiva no nos hemos pasado del presupuesto inicial. Finalmente, el coste total del proyecto es de **16.935€**

10. Informe de sostenibilidad

Una parte que no se puede tomar a la ligera es el análisis de sostenibilidad, este análisis se hará teniendo en cuanta tres dimensiones, la ambiental, la económica y la social. Pero antes de empezar a profundizar en cada una de las dimensiones por separado, se hará una autoevaluación por parte del estudiante.

10.1. Dimensión ambiental

En este apartado se tratará la dimensión Ambiental desde tres puntos de vista: el proyecto en producción, vida útil del proyecto y los riesgos.

Durante la puesta en producción se ha planteado el impacto ambiental que conlleva la realización del proyecto. Se puede decir que no es demasiado grande. Los factores que hacen que el proyecto impacte negativamente en el medio ambiente durante su creación son los ordenadores que se utilizarán y el servidor de la empresa en el que está Instalado GitLab. Al final del trabajo el consumo eléctrico relacionado con los aparatos eléctricos ha sido de 115,09 kW. Si se tiene en cuenta que el consumo de electricidad de una persona en el despacho es de 0.40kWh[15] y que, sin contar las horas de documentación, se pasarán 517 horas en el despacho, el consumo del despacho será 206,8 kWh. Finalmente, el consumo de electricidad de este trabajo ha sido de aproximadamente **322kW.**

A pesar de que el impacto ambiental no sea muy elevado, se busca una manera de reducirlo un poco más y en este proceso se decidió utilizar ordenadores que la empresa y el estudiante ya tenían, es decir no comprar ningún material hardware

[15] El consumo de una persona anual es de 3.487,00kWh/año. Si el año tiene 365 días y los días 24 horas, cada persona gasta un total de 0.40 kWh. [63]

especifico para el proyecto. De esta manera el coste ambiental de fabricación queda amortizado en diferentes proyectos. Algo parecido pasa con la huella ecológica respecto a la electricidad. Dado que en el mismo despacho se realizan diferentes proyectos, este repartirá la huella ecológica en los diferentes proyectos.

Si se pensara en hacer el proyecto de nuevo y se rehiciera un estudio de sostenibilidad para minimizar el impacto medioambiental, resultaría muy similar a este, porque para hacer el proyecto se necesita un ordenador y el consumo eléctrico básico de una persona. Eso quiere decir que si no se pueden recortar recursos por ningún lado no es posible disminuir más el impacto ambiental.

También hay que tener en cuenta como afectará el proyecto durante su vida útil al medioambiente. Los recursos que se utilizarán en la vida útil y el impacto medioambiental del proyecto seguirán siendo los mismos que han resultado en la producción. Se necesitará un ordenador para mantener la aplicación actualizada con las nuevas tecnologías y seguramente también surgirán pequeños errores que detecten los usuarios, los cuales tendrán que solucionarse lo antes posible. Para que la aplicación siga resultando atractiva, a medida que pase el tiempo se tendrán que añadir funcionalidades nuevas, para esto también se necesitará el ordenador con su consumo de electricidad correspondiente.

Además, este proyecto permitirá reducir el uso de otros recursos, actualmente se utiliza el programa de escritorio de GiD en diferentes ordenadores. Al migrar el programa a una aplicación web para que los usuarios sean capaces de utilizarlo desde donde deseen, a la larga se puede considerar como una mejora ambiental, dado que, al no necesitar ningún software específico, ni librerías concretas, hace que sea más fácil poder trabajar en diferentes dispositivos y se elimina la necesidad de comprar un ordenador personal, reduciendo el número total de este recurso. Incluso se podría acceder desde los centros cívicos que proporcionan acceso libre a ordenadores y de esta manera se puede reducir la contaminación que se produce

al fabricar un nuevo ordenador y los residuos que genera una vez termina la vida útil del mismo.

Algunos de los riesgos a los que se enfrenta el trabajo con respecto a la huella ecológica son que se estropee el ordenador y se tenga que comprar otro. Con respecto al coste ambiental de fabricación no quedaría amortizado en diferentes proyectos, sino que se consumiría más en la fabricación de uno nuevo. Otro escenario muy perjudicial sería que la solución escogida al final no resultara la mejor o no funcionara. Esto aumentaría el número de horas de trabajo y con ello el consumo de energía, casi como si se hiciera el trabajo de nuevo.

10.2. Dimensión económica

Antes de empezar el proyecto se ha estimado el coste de su realización tanto en recursos humanos como en materiales. Esto se puede observar en el apartado **5. Presupuesto** donde se detalla con claridad cada uno de los costes e incluso se tienen en cuenta los imprevistos y un plan de contingencia. Se ha intentado seguir al máximo esta planificación para que no aumente el coste del proyecto. Al finalizar el proyecto se han revisado estos costes actualizando las horas por tarea y de uso de los recursos. Gracias a las desviaciones del apartado **9.2. Desviaciones del proyecto** se ha visto que los recursos de personal han aumentado en **1.560€** y los costes genéricos han aumentado en **49€,** como consecuencia de tener que dedicar más tiempo a ciertas tareas de desarrollo**.** Al contar con una partida para contingencias e imprevistos, estos gastos no han supuesto un problema. Se estimó un coste de 19.526€ para realizar el trabajo y finalmente con los incrementos comentados ha resultado un total aproximado de **16.935€,** por lo que el trabajo final ha resultado menor respecto al coste del estimado y el dinero restante se puede ahorrar para otros trabajos.

Se considera que el coste es adecuado con respecto a los beneficios que se pueden obtener con las licencias del producto y sería complicado reducir los costes porque

a simple vista no se aprecia ningún sobrecoste, si bien es cierto que se ha intentado ser precavidos en el apartado de imprevistos y se ha asumido un porcentaje algo mayor para evitar pasarse del presupuesto, cosa que ha resultado muy útil porque al final se ha necesitado parte del dinero de imprevistos y contingencias. El coste final es razonable para un proyecto de esta magnitud.

Durante la vida útil del proyecto se considera que su mantenimiento no será muy costoso dado que se ha creado con lenguajes de programación y herramientas modernas. Si algún día se quieren añadir nuevas funcionalidades tampoco será demasiado costosa la adaptación a las bases del proyecto, dado que se ha intentado modularizar cada componente para que sea simple cambiarlo por uno nuevo o simplemente añadir más. Todas estas tareas las puede realizar un programador, con una retribución de 20,80€/h y un ordenador como el que se ha utilizado, por lo tanto, no se prevé un aumento de los costes. No se puede minimizar más el coste dado que el ordenador y una persona que haga el mantenimiento, actualizaciones y ajustes dentro de la aplicación son esenciales para que los usuarios sientan que hay un servicio técnico detrás que soluciona los posibles problemas que vayan surgiendo.

Al ser un proyecto para una empresa concreta, los riesgos que se pueden correr están relacionados con la posible anulación de los fondos que financien el proyecto, ya que si estos dejasen de existir todo lo realizado, seria difícil o sencillamente no podría implementarse en otro posible proyecto.

10.3. Dimensión social

A nivel social se diferenciará lo que ha aportado la realización del proyecto al estudiante y lo que aportará el proyecto finalizado a la sociedad.

A nivel del estudiante este proyecto ha sido una gran oportunidad para adquirir diferentes tipos de conocimientos. Se clasifican entre conocimientos técnicos y no

técnicos. Dentro del área técnica, el estudiante ha aprendido nuevos lenguajes de programación y el entorno en el que se ha programado, además de poder profundizar en áreas que no se han tocado tanto dentro de la especialidad de Computación. Los conocimientos no técnicos que ha aportado este proyecto son las capacidades de trabajo en equipo y organización, además de ofrecer la oportunidad de una primera experiencia laboral dentro de una empresa y poder ver el funcionamiento y manera de trabajar.

Una vez la aplicación esté finalizada, intentará cubrir la necesidad que ha surgido últimamente de poder trabajar desde cualquier lugar, sin necesidad de tener un software específico, solo con el acceso a internet. Esto no era una necesidad anteriormente, pero con la aparición de la COVID19, lo que antes parecía no ser una necesidad real ahora cada vez resulta más necesario, hasta el punto que se está evolucionando en esa dirección con respecto al trabajo. Esto ha generado que la empresa de CIMNE decidiese migrar una de sus aplicaciones a formato web. Con esta migración se espera que la nueva aplicación tenga todas las funcionalidades que tenía la anterior y se ha intentado replantear alguna de las funcionalidades existentes para que con las nuevas tecnologías resulte más fácil de utilizar por los usuarios. A nivel social se quiere mejorar la comunicación a la hora de trabajar permitiendo compartir los ProblemTypes entre usuarios e incluso a largo plazo permitirá que los usuarios sean capaces de trabajar en el mismo ProblemType a la vez con la gestión de versiones.
Por tanto este proyecto beneficia el área de las simulaciones numéricas y con ello a los estudiantes, ingenieros y científicos que terminen utilizando la aplicación. Si la aplicación tiene mucho éxito podría perjudicar un poco a otras empresas similares a CIMNE. En cualquier caso, el grado en que les puede perjudicar no es muy elevado si desarrollan un solver que se pueda integrar en el programa de GiD, incluso les puede resultar útil proporcionando más herramientas a sus cálculos.

Como se ha comentado en el apartado **9.1. Cumplimiento de los objetivos**, este trabajo ha completado todos los objetivos con sus requisitos y permite que los usuarios programen sus funcionalidades, las prueben y las comparen con otras versiones. Por lo que el programa soluciona perfectamente el problema planteado inicialmente, pero se seguirá avanzando con mejoras y nuevas funcionalidades.

Cualquier herramienta que proporciona ayudas a una tarea puede generar cierta dependencia. Por ejemplo, con el tiempo se ha generado dependencia a los emoticonos, ya que cuando se tiene que expresar los sentimientos solo con palabras puede resultar un poco complicado dado que ya nos hemos acostumbrado a esas caritas amarillas que consiguen captar nuestras emociones. Con este programa puede pasar algo similar: si tienes una manera fácil de visualizar los datos con gráficas y colores te acabas acostumbrando a esta herramienta y ya no mirarás el fichero con todos los números en fila de la solución de los cálculos que resulta mas complicado a la hora de sacar conclusiones. Por eso se intenta tener siempre un buen mantenimiento de las aplicaciones, porque una vez se ha usado y te has acostumbrado a lo que ofrece es algo más complicado volver al punto de partida.

10.4. Conclusión sobre la sostenibilidad

Durante la carrera se nos ha intentado inculcar los valores relacionados con la sostenibilidad y con minimizar el impacto medioambiental mediante conferencias y trabajos. En la encuesta que facilita GEP me he dado cuenta de que he adquirido todos esos conocimientos a nivel teórico, pero no hemos tenido la oportunidad de hacer un trabajo a tan gran escala en la que pudiéramos poner en práctica las soluciones sostenibles que se nos ocurren. El TFG pretende evaluar si se han adquirido todos los conocimientos necesarios para ser un Graduado en Ingeniería Informática y ha sido la ocasión perfecta para ponerlo en práctica.

Todo trabajo intenta cubrir las necesidades que tenga la sociedad para mejorar la calidad de vida, y como este trabajo está dentro del marco de una empresa, a primera vista puede parecer que también se ha centrado en el ámbito económico,

pero quiero remarcar que el ámbito ambiental no ha pasado desapercibido, dado que a lo largo de la carrera he aprendido que la informática tiene un impacto tanto positivo como negativo en las personas, el medio ambiente y en el mundo en general. Por este motivo se ha tenido en cuenta la ética a la hora de tomar decisiones y se ha intentado no malgastar ningún recurso.

Por eso, una vez analizadas las tres dimensiones se ha llegado a la conclusión de que el impacto de este proyecto no puede considerarse negativo. A nivel ambiental se ha intentado minimizar los recursos y tampoco ha supuesto un gran coste dado que los que han generado el impacto medioambiental han sido el ordenador y los trabajadores. Económicamente se ha intentado seguir la planificación inicial al máximo para no generar desviaciones y al final el presupuesto resultante ha sido menor que la estimación inicial. Tampoco se esperan muchos costes de mantenimiento, más o menos en la línea que los del proyecto. Y finalmente a nivel social aporta beneficios y el único escenario perjudicial que puede surgir es con los rivales del sector. Así que el balance del estudio de sostenibilidad lo consideramos positivo.

11. Conclusiones

11.1. Conclusiones del proyecto

Una vez realizado el proyecto es el momento de extraer conclusiones de las diferentes tareas realizadas. El proyecto surge de la necesidad de crear funcionalidades para los WorkingModes en la nueva aplicación formato web de GiD.

Para poder crear funcionalidades dentro de la aplicación web, la empresa quería un apartado de edición con ayudas para el usuario, por lo que el trabajo se ha centrado en hacer un editor con autocompletado que permita ejecutar el código y ver versiones anteriores.

Se estudiaron las diferentes maneras en las que se podía solucionar cada uno de los problemas identificados. Este estudio se ha centrado en las soluciones con más recursos y documentación de las que extraer información. Ha sido un amplio estudio de características con información necesaria que ha permitido tomar decisiones prácticas. Con respecto al desarrollo se ha decidido utilizar MonacoEditor como editor base al que añadir diferentes funcionalidades, GitLab para almacenar los diferentes ficheros generados por los usuarios y utilizar unas clases de JavaScript que se transformarán en componentes Angular para mostrar los componentes HTML que programe el usuario.
Respecto al diseño, se ha tenido en cuenta la usabilidad y nos hemos inspirado en aplicaciones existentes para que sea más intuitiva para los usuarios, consiguiendo una adaptación más rápida. Respecto al editor y a los botones que gestionan los ficheros, nos hemos inspirado en la aplicación de Microsoft Word, por lo que se han mantenido dichos botones en la parte superior, al lado del nombre, y el editor en la parte inferior.
La gestión de versiones ha seguido la idea de una línea temporal parecida a Git que gestiona las diferentes versiones como si fuera un árbol con ramas. Se ha decidido que cada nodo sea un botón que al ser presionado cargue el código de la

versión y lo compare automáticamente. Con esta decisión el usuario determina la versión que quiere ver, simplificando de manera visual y directa la idea principal de Git tal y como hacen muchas interfaces de gestión Git.
Para la parte de la ejecución se decidió poner una pantalla y un pequeño recuadro que mostrara los errores con el formato de un terminal.

Durante la fase de desarrollo no surgieron demasiados problemas, las decisiones tomadas parecían haber sido las correctas. Así que poco a poco y con la ayuda de documentación propia de las aplicaciones utilizadas y foros de internet se ha conseguido implementar todo. La parte de testear detectó errores que no habían surgido a simple vista y se corrigieron.

Con todo lo comentado anteriormente se han conseguido alcanzar los objetivos planteados al principio y añadir una nueva funcionalidad a la aplicación web de GiD. Faltará terminar funcionalidades de la aplicación general para que se utilice y comercialice. Pero actualmente los compañeros del despacho de GiD ya tienen pleno acceso al proyecto y se espera que en las próximas semanas se utilice el editor dentro del departamento como programa beta para facilitar la creación de funcionalidades reales y no solo para evaluar la usabilidad y reportar errores.

11.2. Valoración personal

Este trabajo me ha resultado muy interesante, me ha permitido profundizar en el área de la programación web y en la usabilidad de esta, ya que durante la carrera se ha visto con poca intensidad y solo en alguna de las asignaturas optativas. Me pareció fascinante la variedad con la que se puede trabajar en una programación web, te permite hacer muchas cosas. Seguramente, si no hubiera hecho este trabajo, que ha requerido por mi parte búsqueda de información y el aprendizaje de más conocimientos sobre el tema, habría realizado algún curso para ampliar mis conocimientos.

Algunas de las habilidades que he desarrollado y mejorado durante el transcurso del trabajo han sido mis capacidades de análisis y de toma de decisiones. En los trabajos que había hecho hasta el momento, los puntos a seguir estaban muy claros y las decisiones las había tomado previamente el profesor y los alumnos solo teníamos que seguir sus indicaciones. En este trabajo me he encontrado con la casuística de tener plena libertad respecto a las tomas de decisiones. Obviamente el tutor tenía que estar de acuerdo con las decisiones tomadas. Para mi ha sido totalmente nuevo la toma de decisiones y el análisis de las diferentes opciones que se plantaban con respecto a las tecnologías y estrategias finales que podían marcar la diferencia entre terminar o no el trabajo. En ningún momento me había enfrentado a esta situación de toma de decisiones reales y, observando a mis compañeros en el despacho, he visto cómo ellos en su día a día se enfrentan a estas situaciones constantemente. Creo que este es el aspecto que hace que este proyecto lo tenga que desarrollar un ingeniero informático y no solo alguien que tenga conocimientos de programación.

Con respecto a la implementación he aprendido a utilizar diferentes herramientas y tecnologías que no había utilizado hasta ahora como el framework de Angular y las herramientas para poder crear los tests e2e y unitarios. Durante la carrera no se ha profundizado demasiado en el tema de los tests y en este proyecto me ha parecido muy útil, dado que se han detectado errores de funcionamiento que de otra manera los habría detectado un usuario y así se puede ofrecer una experiencia mejor. También he aprendido a utilizar JavaScript y html, no ha resultado tan difícil como pensaba en un principio gracias a los conocimientos generales de programación que he aprendido durante la carrera, incluso me ha sorprendido la capacidad de adaptación a otros lenguajes que he adquirido sin ser consciente de ello.

Otro aspecto que me ha parecido muy importante está relacionado con las habilidades del trabajo. Es importante tener una organización precisa y bien

estructurada, porque puede marcar la diferencia entre terminar el trabajo en el plazo acordado o no. En este caso considero que la planificación estaba bien definida y estructurada y gracias a eso se ha podido terminar el trabajo a tiempo, porque en un principio he de reconocer que me parecía mucho trabajo y que sería imposible presentarlo en enero. Pero todo tenía su tiempo en la planificación y poco a poco me parecía un objetivo más alcanzable. También quiero remarcar que he visto cómo un buen ambiente de trabajo y una buena comunicación entre los miembros del equipo favorece el avance de todo proyecto. Con una reunión semanal explicando lo que se ha hecho durante la semana y lo que se quiere hacer en la siguiente semana, se detectan posibles bloqueos de dependencias entre tareas y se solucionan problemas en un abrir y cerrar de ojos con la técnica de lluvia de ideas. Esto, acompañado de una metodología ágil, hace que las implementaciones vayan sobre ruedas todo el tiempo.

Otra de las cosas que he aprendido es que una buena documentación del trabajo puede ahorrar tiempo tanto a la persona que realiza la tarea como a los compañeros. Si en algún caso alguien necesita revisar tu código, le resulta más fácil entender la implementación de otra persona en el caso de que haya documentación y, si de repente falla algo que antes funcionaba, se pueden mirar paso a paso las modificaciones del código, de modo que si está bien documentado se pueden encontrar los errores en un momento.

Por todo lo que he comentado anteriormente y todas las cosas que he aprendido estoy muy satisfecha de haber hecho el TFG con la empresa CIMNE, me ha proporcionado una primera experiencia laboral muy enriquecedora. También he podido reorganizar todos los conocimientos adquiridos a lo largo de la carrera y ver en qué ámbitos pueden ser utilizados pudiendo incluso ampliarlos. Por último, quiero decir que este trabajo me ha servido para superar muchos retos y enfrentarme a problemas desde cero sin tener una pauta previa.

11.3. Integración de conocimientos

A lo largo de la carrera se imparten conocimientos de diferentes áreas, en concreto este trabajo se centra en la especialidad de computación por lo que las asignaturas de esta modalidad han sido especialmente útiles. Pero se ha intentado poner en práctica el máximo de aprendizajes adquiridos. Gracias a esto el estudiante ha profundizado en usos de estos conocimientos más específicos en el mundo real.

Dado que este proyecto es de creación, hay que tener en cuenta que todos los conocimientos adquiridos con relación a la programación han sido esenciales para poder implementar la aplicación. Las asignaturas de **PRO1 y PRO2**[16], **EDA**[17], **LI**[18] y **A**[19] han influido en la manera en la que se afrontan los problemas dando paso a una mente más lógica y analítica. También han proporcionado la base respecto a la programación dirigida a objetos, las dependencias entre ellos, los tipados, los cálculos de costes, la eficiencia de algoritmos, etc. Una asignatura que entraría dentro del grupo anterior, pero de la que quiero hacer mención especial, es la de **LP**[20]. Gracias a esta asignatura y a la competencia de **aprendizaje autónomo** he sido capaz de adaptarme a un framework y a lenguajes totalmente nuevos para mi y estos conocimientos me dan la seguridad necesaria para poder enfrentarme a cualquier reto en el futuro.

Otras asignaturas que han resultado útiles a lo largo del proyecto han sido **BD**[21] que me han permitido entender las estructuras de almacenamiento a la hora de

[16] Asignatura de programación uno y dos

[17] Asignatura de datos y algoritmos

[18] Asignatura de lógica en la informática

[19] Asignatura de algorítmica

[20] Asignatura de lenguaje de programación

[21] Asignatura de bases de datos

conectar la aplicación con GitLab, los conocimientos aprendidos en **TC**[22] que han facilitado el uso de expresiones regulares para evaluar contenido del editor y la asignatura de **PAR**[23] que ha proporcionado las bases para entender y utilizar herramientas que trabajaran en segundo plano y threads secundarios.

Toda aplicación ha de ser llamativa para los usuarios, pero a la vez fácil de utilizar e intuitiva. Para poder darle al usuario una mejor experiencia a la hora de utilizar la aplicación, se han puesto en práctica todos los conocimientos de usabilidad que se han impartido en las asignaturas de **G**[24] y de **IDI**[25].

No solo hay que centrarse en los conocimientos técnicos. Durante la carrera también se han enseñado aptitudes de comunicación eficaz y escrita que junto con asignaturas como **IES**[26] me han proporcionado herramientas para entender, dejar documentado y compartir con mis compañeros del trabajo el flujo de datos y las implementaciones hechas dentro de la aplicación. Y asignaturas como **GEOC**[27], **DCS**[28], **G** y **VC**[29] me han ayudado a acercarme más al mundo que trata el departamento de GiD con geometrías, mallados y simulaciones.

A lo largo de este proyecto el estudiante se ha enfrentado a problemas del día a día que tienen las grandes empresas, pero nunca lo había vivido en su propia piel. Un claro ejemplo es la parte económica, hacer un presupuesto. Para este apartado han sido de gran ayuda los conocimientos proporcionados por las asignaturas de

[22] Asignatura de teoría de la computación

[23] Asignatura de paralelismo

[24] Asignatura de gráficos

[25] Asignatura de interacción y diseño de interfaces

[26] Asignatura de introducción a la ingeniería del software

[27] Asignatura de geometría computacional

[28] Asignatura de diseño de curvas y superficies

[29] Asignatura de visión por computador

PE[30] y también **EEE**[31]. Otro ejemplo es la gestión de un proyecto de esta magnitud, que gracias a la asignatura de **PROP**[32], en la que se hace una pequeña práctica a menor escala, pero que ha resultado muy útil para enfrentarse a un proyecto real.

Por último, todo y que no menos importante, se tiene que destacar especialmente la asignatura de **AC** [33] y las asignaturas obligatorias del departamento de computadores que han aportado la concienciación necesaria con respecto al medio ambiente y sostenibilidad que toda persona tendría que tener para su día a día.
No solo se ha enseñado ese valor que considero muy importante, sino que también he aprendido a trabajar en equipo y tener una buena actitud frente al trabajo. Esto me ha sido muy útil en el poco tiempo que llevo en el mundo laboral y estoy segura de que en futuras ocasiones me resultará más provechoso.

Todo lo explicado anteriormente me ha proporcionado el conocimiento necesario para adquirir un criterio propio a la hora de tomar decisiones y conseguir hacerlo de manera efectiva y creativa aportando soluciones con las que ser capaz de resolver perfectamente los problemas planteados.

11.4. Identificación de leyes y regulaciones

En este apartado se hablará de las diferentes leyes a las que el proyecto va ligado. Este proyecto es una parte de una aplicación web, por lo que nos centraremos en las leyes que toda página web debe cumplir.

Antes de utilizar cualquier aplicación web el usuario debe registrarse introduciendo unos datos personales y de contacto. En este momento la ley establece que la

[30] Asignatura de probabilidad y estadística

[31] Asignatura de empresa y entorno económico

[32] Asignatura de proyecto de programación

[33] Asignatura de arquitectura de computadores

aplicación ha de informar al usuario de los **Términos y Condiciones de uso**, que los tiene que aceptar para poder finalizar el registro como un nuevo usuario.
El documento de términos y condiciones de uso regula la relación con el usuario respecto al acceso de los contenidos y de los servicios que se ponen a su disposición mediante la aplicación web. Esto sirve para establecer cuáles son las responsabilidades legales, tanto de la empresa que proporciona la aplicación, cómo del usuario que la está utilizando.
La aplicación web a la que pertenece el proyecto que estamos desarrollando está en proceso, pero los usuarios tendrán que registrarse para utilizar los diferentes servicios que proporciona el departamento de GiD. Como ya hay servicios que hoy en día están activos, se están utilizando y contienen datos de usuarios ya existentes, el apartado de registros contiene un documento de Términos y Condiciones de uso que se muestra al usuario para que lo lea y lo acepte antes de registrarse y dar información personal. Dado que el registro es el mismo para todos los servicios, se puede decir que el proyecto cumple esta normativa, quizás antes de comercializarlo o ponerlo en producción se tendría que repasar este documento y añadir alguna cláusula un poco más específica.

El documento mencionado anteriormente contiene información de la empresa, condiciones particulares de uso, las obligaciones que tiene la empresa y el cliente, responsabilidades y limitaciones, los derechos de propiedad intelectual e industrial, tratamiento de datos de carácter personal y política de confidencialidad, derechos de acceso, rectificación, cancelación, oposición, suspensión, limitación del tratamiento y portabilidad de los datos, duración y modificación del servicio, legislaciones aplicables y tribunales competentes, garantía, modificaciones del servicio, variaciones de las condiciones, resolución alternativa de conflictos y uso de cookies.

A este proyecto le afecta especialmente la Política de Privacidad, dado que en la aplicación web se almacenarán datos de los usuarios. Estos datos se tratarán cómo

indica el Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo, de 27 de abril de 2016 [64], relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos para abreviar RGPD, la Ley Orgánica 3/2018, de 5 de diciembre [65], de Protección de Datos Personales y garantía de los derechos digitales.

En un futuro también le afectará la Política de Cookies, ya que se utilizarán las cookies propias teniendo como finalidad el acceso y entrada del usuario, la autenticación o identificación del usuario a la hora de iniciar sesión, así como la seguridad del usuario a la hora de utilizar la aplicación.

11.5. Competencias técnicas

En este apartado se hablará de las competencias técnicas asociadas a él y en que grado se han tratado.

CCO1.1: Evaluar la complejidad computacional de un problema, conocer estrategias algorítmicas que puedan llevar a su resolución, y recomendar, desarrollar e implementar la que garantice el mejor rendimiento de acuerdo con los requisitos establecidos. (Un poco)

Para resolver todo problema, antes de empezar a programar hay que hacer un estudio en el que se evalúen las posibles implementaciones y los costes de cada una.

Para este trabajo, las posibles soluciones tenían mas o menos el mismo coste, excepto las soluciones que exigían implementar el componente desde cero, que se descartaron rápidamente por falta de tiempo.

El resto de las soluciones eran muy parecidas con respecto a los costes, por este motivo la competencia se trata un poco. Dentro de las implementaciones que se han tenido que desarrollar, se ha intentado programar la menos costosa si había más de una manera. Un buen ejemplo de eso puede ser la actualización del

componente editor. Dado que hay tres componentes que tienen que estar siempre actualizados, se ha decidido que se actualice el editor al que se quiere acceder cuando se cambia de pestaña y no actualizar los tres cada vez que se cambia la información del actual. Con respecto al código, se ha intentado parar los ciclos de búsqueda una vez se encuentra el componente, en lugar de recorrer todos los elementos. Son pequeñas estrategias que a la larga favorecen el rendimiento de la aplicación.

CCO1.2: Demostrar conocimiento de los fundamentos teóricos de los lenguajes de programación y las técnicas de procesamiento léxico, sintáctico y semántico asociadas, y saber aplicarlas para la creación, diseño y procesamiento de lenguajes. (Un poco)

El editor creado ejecuta el código del usuario que está escrito en un sublenguaje formado por JavaScript y unas clases específicas, implementadas por el estudiante para que el usuario pueda crear componentes HTML. Mientras se ejecuta el código se detectan errores del propio lenguaje JavaScript, pero también de estas clases específicas que proporcionamos, para ello se ha utilizado parte del conocimiento relacionado con los fundamentos teóricos de los lenguajes. No se ha hecho un parser de un compilador, ni mucho menos, pero en cierta manera la idea que está detrás es similar. El usuario utiliza estas clases, que equivaldrían a la entrada de un parser para transformarlas en objetos del estilo JSON, y finalmente el componente de Angular los transforma en otros objetos Angular, los cuales harán posible la visualización de los componentes HTML por la pantalla.

También se han utilizado estos conocimientos teóricos de los lenguajes para autocompletar expresiones y funciones más específicas de nuestro sublenguaje que no detectaría JavaScript, para ello se han utilizado expresiones regulares. Por todo lo expuesto anteriormente se considera que se ha trabajado un poco esta competencia.

CCO1.3: Definir, evaluar y seleccionar plataformas de desarrollo y producción de hardware y software para el desarrollo de aplicaciones y servicios informáticos de diversa complejidad. (En profundidad)

Algunas de estas plataformas de desarrollo y producción de hardware y software ya han venido impuestas por la empresa. Por ejemplo, el uso de Git, Jira, Confluence, Jenkins, Slack, el framework de Angular y los lenguajes de programación de JavaScript y TypeScript.

Pero no hay que olvidar que el trabajo realzado en si, es una plataforma de desarrollo y producción de software permitiendo crear funcionalidades para los ProblemTypes de GiD. Para poder realizar el proyecto se han tenido que evaluar diferentes plataformas de desarrollo y producción que irán integradas dentro de la aplicación. Es un claro ejemplo de los diferentes componentes de editores y que finalmente se ha decidido utilizar MonacoEditor. También se evaluaron diferentes plataformas para gestionar las versiones y los ficheros y, como explica el apartado de **Solución propuesta,** se decidió utilizar GitLab. Se considera que esta competencia se ha tratado en profundidad porque evaluar y seleccionar las plataformas de desarrollo y producción que más se adaptaban al tipo de proyecto ha sido una decisión esencial para la elaboración del mismo.

CCO2.3: Desarrollar y evaluar sistemas interactivos y de presentación de información compleja, y su aplicación en la resolución de problemas de diseño de interacción persona computador. (En profundidad)

Esta competencia se trata en profundidad dado que la mayoría de los objetivos principales la tratan. A lo largo del trabajo se ha desarrollado un apartado de la página web que consideramos un sistema interactivo para el usuario y que ha de encontrar familiar y fácil de usar. También hay un apartado que muestra la ejecución del código del fichero escogido. Esto se hace por medio de mostrar una pantalla con los componentes html de manera interactuables con todas las funcionalidades que el usuario ha programado. Además, también aparecen pequeños mensajes por un componente terminal que ayudan al usuario a detectar

errores en su código. Se considera que toda esta información compleja se consigue mostrar de una manera intuitiva y solucionando el problema de interacción persona computador.

CCO2.6: Diseñar e implementar aplicaciones gráficas, realidad virtual, realidad aumentada y videojuegos. (Bastante)

En este proyecto se ha colaborado creando funcionalidades dentro de una aplicación web. Para realizar esto se ha tenido que diseñar el apartado del editor teniendo en cuenta la usabilidad y la comodidad de los usuarios y más adelante implementar las funcionalidades necesarias para cumplir los requisitos que la empresa ha exigido. Como al final del trabajo se ha diseñado e implementado una parte de una aplicación gráfica, se considera que se ha tratado bastante el objetivo.

CCO3.1: Implementar código crítico siguiendo criterios de tiempo de ejecución, eficiencia y seguridad. (Bastante)

Toda aplicación tendría que funcionar de manera rápida, eficiente y tratar los datos del usuario de manera segura. Esta aplicación no es una excepción, se ha trabajado generando un identificador único llamado token para que los ficheros del usuario solo los pueda ver él mismo. Respecto a la eficiencia y tiempo de ejecución se han intentado minimizar las peticiones http a GitLab. En los casos que las peticiones tienen que cargar muchos datos o buscar la información entre muchos ficheros, haciendo que la petición se resuelva de manera más lenta, se ha añadido un diálogo de texto informando al usuario. Se considera que durante el desarrollo se ha tenido en cuenta la ejecución, eficiencia y seguridad pero diremos que esta competencia se ha tratado bastante en vez de en profundidad, dado que algunas de las acciones futuras son para mejorar la eficiencia de la aplicación.

11.6. Acciones futuras

Una vez se ha comprobado que el proyecto resuelve el problema planteado se puede decir que ya se ha terminado el trabajo. Pero realmente este proyecto todavía tiene mucho recorrido. A nivel de lo que está implementado, se tendría que hacer un pequeño mantenimiento de la aplicación y, en el momento en que salga al mercado, estar preparados para solucionar pequeños bugs o errores que encuentren los usuarios, o simplemente hacer de soporte técnico para resolver dudas que les puedan surgir.

Otra tarea importante es seguir investigando las tecnologías que puedan ir surgiendo, como se comentó al principio del proyecto. Con el paso del tiempo algunas de estas tecnologías pueden quedar desactualizadas y se necesitan remplazar o llevar a la versión más nueva. Para que esto pueda pasar hay que estar continuamente informado de las novedades del mercado, por si se encuentra un componente que se adapte mejor o simplemente actualizar las versiones de los ya existentes.

De cara a seguir implementando funcionalidades y mejorar las existentes, se ha pensado en incorporar un buen sistema de paginación para ganar eficiencia a la hora de cargar los commits anteriores de un fichero para que el usuario no note que el proceso de abrir un fichero puede ser un poco lento. La idea principal es cargar los primeros 8 commits como máximo y a medida que el usuario bajara la barra de scroll hasta el final se cargarían los 8 commits siguientes.

Otra de las mejoras propuestas es añadir más información al autocompletado, para que el usuario no necesite mirar documentaciones externas, simplemente escribiendo en el editor ya consigue todo lo que necesita saber.

Además, se quiere que la aplicación se vea bien en todo tipo de monitores, tengan más o menos pulgadas y en un futuro también se quiere pasar a aplicación móvil.

Se ha pensado en añadir dentro del editor un "eslint" o "linting", que es una herramienta de análisis de código y se considera que de esta manera se conseguirán encontrar mejor los errores antes de que el usuario presione el botón de ejecutar. En este mismo ámbito se quiere añadir la opción de que al pasar el ratón por encima de un error la aplicación te muestre también las posibles soluciones al error. Pero para este último caso se tendrán que estudiar las posibles maneras de implementarlo.

La última mejora que se quiere hacer es añadir ayudas para la depuración de código. Actualmente permite ejecutarlo y ver los errores, pero en un futuro se quieren añadir puntos de parada y poder visualizar el valor de las variables en cada momento. Esto también requiere un pequeño estudio de las diferentes maneras de implementación.

Estas han sido algunas de las mejoras que han surgido a lo largo del trabajo, por lo que se puede decir que todavía hay mucho margen de ampliación para el proyecto.

12. Bibliografía

[1] «Cimne». https://www.cimne.com/2409/2692/about/mission-and-history (accedido sep. 27, 2021).

[2] «What's GiD | GiD - The personal pre and post processor». //www.gidhome.com/whats-gid/ (accedido sep. 27, 2021).

[3] «Simulación numérica | Itmati». http://www.itmati.com/Simulacion_numerica (accedido sep. 25, 2021).

[4] «01-Introducción.pdf». Accedido: sep. 25, 2021. [En línea]. Disponible en: https://www.gidhome.com/archive/GiD_Convention/2004/courses/gid_basic_and_advanced/01-Introducci%C3%B3n.pdf

[5] «Defining a problem type». https://www.gidhome.com/documents/usermanual/en/DEFINING%20A%20PROBLEM%20TYPE (accedido sep. 27, 2021).

[6] Q. Devs, «¿Qué es Angular y para qué sirve?», *Quality Devs*, sep. 16, 2019. https://www.qualitydevs.com/2019/09/16/que-es-angular-y-para-que-sirve/ (accedido sep. 27, 2021).

[7] D. E. Investigaciones, «Datos e Investigaciones: ¿Qué es depurar?», *Datos e Investigaciones*, nov. 21, 2018. http://datoseinvestigaciones.blogspot.com/2018/11/que-es-depurar.html (accedido sep. 27, 2021).

[8] «Entorno de desarrollo integrado», *Wikipedia, la enciclopedia libre*. ago. 31, 2021. Accedido: nov. 19, 2021. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Entorno_de_desarrollo_integrado&oldid=138036723

[9] «Git». https://git-scm.com/ (accedido sep. 26, 2021).

[10] «Home New», *Tecplot*. https://www.tecplot.com/ (accedido ene. 07, 2022).

[11] «ParaView». https://www.paraview.org/ (accedido ene. 07, 2022).

[12] «Femap | Siemens Software», *Siemens Digital Industries Software*. https://www.plm.automation.siemens.com/global/es/products/simcenter/femap.html (accedido ene. 07, 2022).

[13] «FEMGV - Overview | DIANA FEA». https://dianafea.com/femgv-overview (accedido ene. 07, 2022).

[14] O. Business a PTC, «Onshape | Product Development Platform». http://www.onshape.com/en/ (accedido ene. 07, 2022).

[15] «Simulation Software | Engineering in the Cloud», *SimScale*. https://www.simscale.com/ (accedido ene. 07, 2022).

[16] «RunKit is Node prototyping». https://runkit.com/home (accedido sep. 27, 2021).

[17] «Ace - The High Performance Code Editor for the Web». https://ace.c9.io/ (accedido sep. 27, 2021).

[18] «Monaco Editor». https://microsoft.github.io/monaco-editor/ (accedido sep. 27, 2021).

[19] «GitHub: Where the world builds software», *GitHub*. https://github.com/ (accedido nov. 19, 2021).

[20] «Iterate faster, innovate together», *GitLab*. https://about.gitlab.com/ (accedido nov. 19, 2021).

[21] «REST API resources | GitLab». https://docs.gitlab.com/ee/api/api_resources.html (accedido sep. 27, 2021).

[22] Atlassian, «Kanban: una breve introducción», *Atlassian*. https://www.atlassian.com/es/agile/kanban (accedido dic. 19, 2021).

[23] Atlassian, «Atlassian | Software Development and Collaboration Tools», *Atlassian*. https://www.atlassian.com (accedido sep. 24, 2021).

[24] Atlassian, «Sourcetree | Free Git GUI for Mac and Windows», *SourceTree*. https://www.sourcetreeapp.com (accedido sep. 27, 2021).

[25] Slack, «Slack se encuentra donde trabaja el futuro», *Slack*. https://slack.com/intl/es-es/ (accedido oct. 17, 2021).

[26] «Google Meet (antes, Hangouts Meet) - Videollamadas gratuitas». https://apps.google.com/meet/ (accedido oct. 17, 2021).

[27] «La primera aplicación rápida de escritorio remoto», *AnyDesk*. https://anydesk.com/es (accedido oct. 17, 2021).

[28] «Microsoft Word: software de procesamiento de texto | Microsoft 365». https://www.microsoft.com/es-es/microsoft-365/word (accedido oct. 04, 2021).

[29] «Visual Studio Code - Code Editing. Redefined». https://code.visualstudio.com/ (accedido oct. 04, 2021).

[30] «Gantter | Google-Integrated Project Management & Scheduling», *Gantter*. https://www.gantter.com/google/ (accedido oct. 04, 2021).

[31] B. Web, «Cómo crear tests End To End (E2E Testing) en Angular con Protractor», *Cuaderno de bitácora de un desarrollo web*, may 19, 2020. https://bitacoraweb.info/como-crear-tests-end-to-end-e2e-en-angular-con-protractor/ (accedido sep. 27, 2021).

[32] «Cursos en línea: aprende de todo y a tu propio ritmo», *Udemy*. https://www.udemy.com/ (accedido oct. 03, 2021).

[33] «Angular - Setting up the local environment and workspace». https://angular.io/guide/setup-local (accedido oct. 04, 2021).

[34] «PrimeNG | Angular UI Component Library». https://www.primefaces.org/primeng/ (accedido oct. 04, 2021).

[35] «Mini PC Negro DELL OptiPlex 7090, 2,9 GHz, Intel® Core™ i7 de 10ma Generación, 16 GB, 512 GB, DVD±RW, Windows 10 Pro en bruneau.es». https://www.bruneau.es/product/mini-pc-negro-dell-optiplex-7090-2-9-ghz-intel-core-i7-10ma-generacion-16-gb-512-gb-dvd-rw-windows-10-pro/8149726 (accedido oct. 09, 2021).

[36] «Monitor LED Dell SE2717H 27" Full HD IPS Mate Negro - Monitor LED - Los mejores precios | Fnac». https://www.fnac.es/mp5201289/Monitor-LED-Dell-SE2717H-27-Full-HD-IPS-Mate-Negro/w-4?oref=5d768367-3642-6d93-1ab1-face80c94685&Origin=GOO_PLAS_MKT_Micro&gclid=Cj0KCQjw-4SLBhCVARIsACrhWLXVM9kFPXloT_pT4JWPeVQijf3RDYZ1npMUBU84Tf3Hk5BWD7Lls0gaAnFbEALw_wcB&gclsrc=aw.ds (accedido oct. 09, 2021).

[37] «Teclado DELL KB813 (Con Cable - Layout Alemán)», *Worten ES*. https://www.worten.es/productos/informatica/accesorios-y-perifericos/teclados/teclado-dell-kb813-con-cable-layout-aleman-MRKEAN-5397063785056 (accedido oct. 09, 2021).

[38] «DELL - 570-11147 - Dell Maus - rechts- und linkshändig - optisch», *tonitrus.com*. https://www.tonitrus.com/es/redes/accessories/other2/10219946-003-dell-570-11147-dell-maus-rechts-und-linkshaendig-optisch/ (accedido oct. 09, 2021).

[39] «APC Back UPS Pro BR900G-GR SAI 900VA | PcComponentes.com».

https://www.pccomponentes.com/apc-back-ups-pro-br900g-gr-sai-900va?gclsrc=aw.ds&gclid=Cj0KCQjw-4SLBhCVARIsACrhWLUpmdk5OFnyOcsDVB1d_MNYnfbo4iHb__gF9_nGqf-jk15CrJsgSeIaAiWaEALw_wcB (accedido oct. 09, 2021).

[40] «Apple MacBook Pro 16" Touch Bar Core i7 2,6Ghz | 512GB SSD | 16GB RAM | Gris Espacial». https://www.macnificos.com/apple-macbook-pro-16 (accedido oct. 09, 2021).

[41] «Cuál es el consumo de un ordenador de sobremesa», *tarify.es*. https://tarify.es/noticias/cual-consumo-ordenador-sobremesa (accedido oct. 09, 2021).

[42] «Consumo de energía de una computadora portátil contra una de escritorio», *Techlandia*. https://techlandia.com/consumo-energia-computadora-portatil-contra-escritorio-sobre_481530/ (accedido oct. 10, 2021).

[43] «Download | GiD - The personal pre and post processor». //www.gidhome.com/download/ (accedido dic. 27, 2021).

[44] «GiD_15_Customization_Manual.pdf». Accedido: ene. 03, 2022. [En línea]. Disponible en: https://www.gidhome.com/archive/GiD_Documentation/Docs/GiD_15_Customization_Manual.pdf

[45] «¿Qué es npm? Un Tutorial para Principiantes sobre el Gestor de Paquetes de Node», *freeCodeCamp.org*, ene. 04, 2021. https://www.freecodecamp.org/espanol/news/node-js-npm-tutorial/ (accedido nov. 19, 2021).

[46] «eval - JavaScript | MDN». https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/eval (accedido nov. 19, 2021).

[47] «try...catch - JavaScript | MDN». https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Statements/try...catch (accedido nov. 19, 2021).

[48] «Usando Web Workers - Referencia de la API Web | MDN». https://developer.mozilla.org/es/docs/Web/API/Web_Workers_API/Using_web_workers (accedido nov. 19, 2021).

[49] «Threads en Java».

http://profesores.elo.utfsm.cl/~agv/elo330/2s10/lectures/Java/threads/JavaThreads.html (accedido nov. 19, 2021).

[50] «HTML: Lenguaje de etiquetas de hipertexto | MDN». https://developer.mozilla.org/es/docs/Web/HTML (accedido nov. 19, 2021).

[51] «Angular - Setting up the local environment and workspace». https://angular.io/guide/setup-local (accedido dic. 27, 2021).

[52] «ngx-monaco-editor», *npm*. https://www.npmjs.com/package/ngx-monaco-editor (accedido dic. 28, 2021).

[53] «Repositories API List repository tree | GitLab». https://docs.gitlab.com/ee/api/repositories.html#list-repository-tree (accedido dic. 29, 2021).

[54] «Repository API files Create new file in repository | GitLab». https://docs.gitlab.com/ee/api/repository_files.html#create-new-file-in-repository (accedido dic. 29, 2021).

[55] «Repository API files Get raw file from repository | GitLab». https://docs.gitlab.com/ee/api/repository_files.html#get-raw-file-drom-repository (accedido dic. 29, 2021).

[56] «Repository API files Update existing file in repository | GitLab». https://docs.gitlab.com/ee/api/repository_files.html#update-existing-file-in-repository (accedido dic. 29, 2021).

[57] «Repository API files Get file from repository | GitLab». https://docs.gitlab.com/ee/api/repository_files.html#get-file-from-repository (accedido dic. 29, 2021).

[58] «Commits API Get a single commit | GitLab». https://docs.gitlab.com/ee/api/commits.html#get-a-single-commit (accedido dic. 29, 2021).

[59] «Protractor - end-to-end testing for AngularJS». https://www.protractortest.org/#/ (accedido dic. 30, 2021).

[60] «Jasmine Documentation». https://jasmine.github.io/ (accedido dic. 30, 2021).

[61] «BDD Testing & Collaboration Tools for Teams | Cucumber». https://cucumber.io/ (accedido dic. 30, 2021).

[62] «Karma - Spectacular Test Runner for Javascript». https://karma-

runner.github.io/latest/index.html (accedido dic. 30, 2021).

[63] «¿Cuánto cuesta la luz al mes? Consumo medio de luz en España», *tarifasgasluz.com*. https://tarifasgasluz.com/faq/cuanto-cuesta-luz-mes (accedido ene. 02, 2022).

[64] «REGLAMENTO (UE) 2016/ 679 DEL PARLAMENTO EUROPEO Y DEL CONSEJO - de 27 de abril de 2016 - relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/ 46/ CE (Reglamento general de protección de datos)», p. 88.

[65] M. B. Samper, *Protección de datos personales: Esquemas*, 1.ª ed. Dykinson, 2020. doi: 10.2307/j.ctv17hm980.

13. Anexo A

Diagrama de Gantt inicial

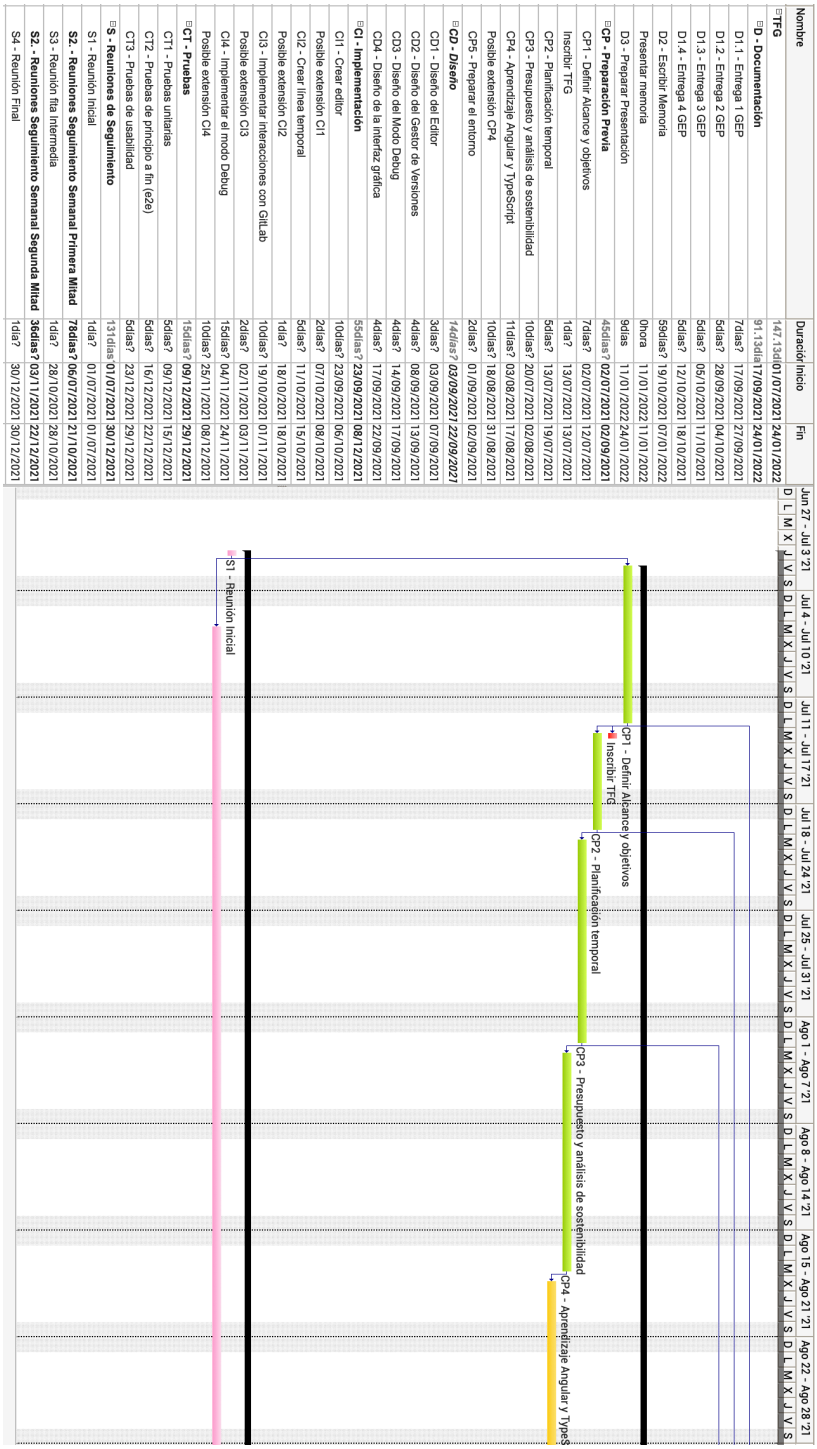


Figura 29 : Primera parte del Diagrama de Gantt inicial. Elaboración propia

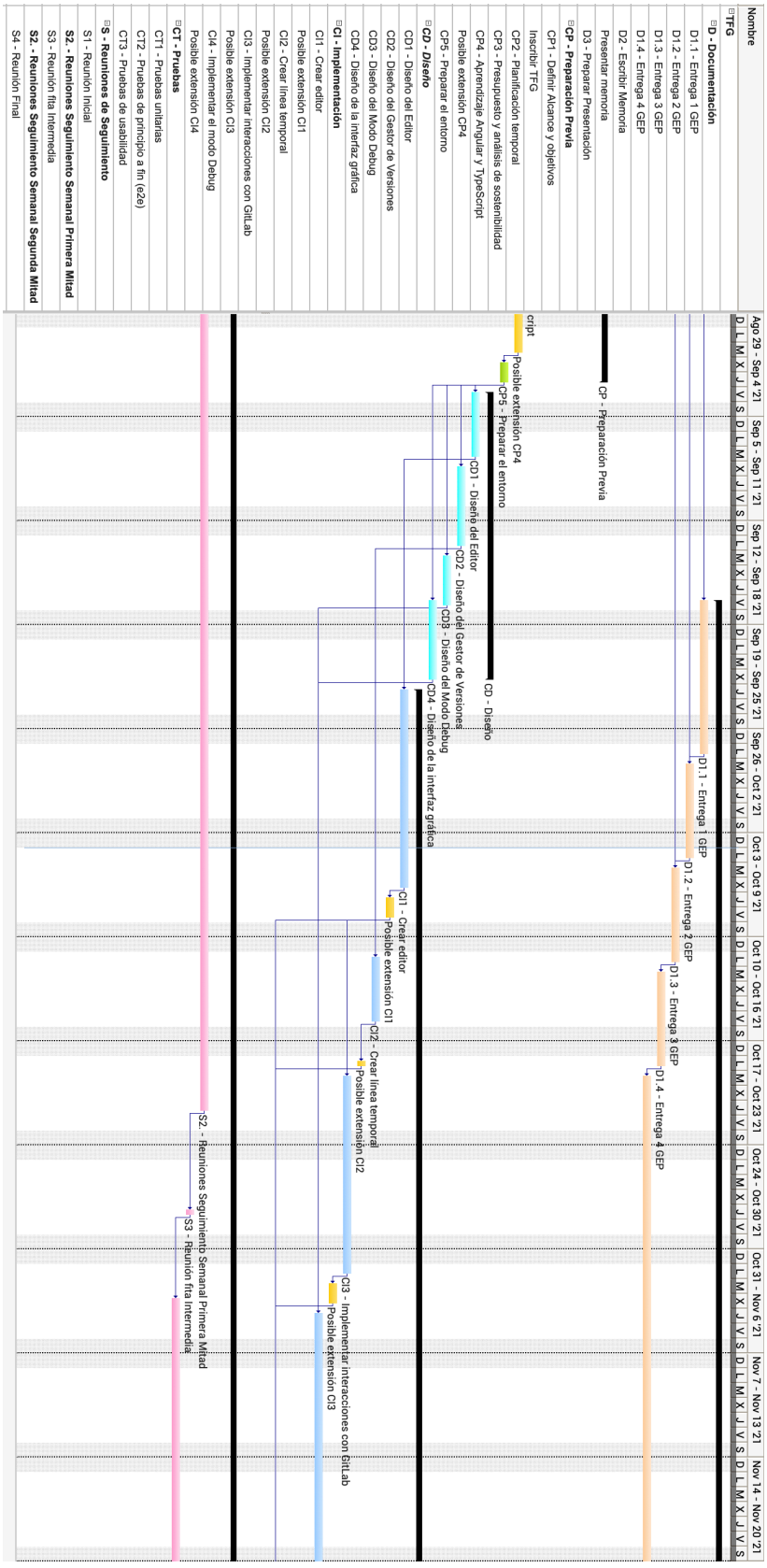


Figura 30 : Segunda parte del Diagrama de Gantt inicial. Elaboración propia

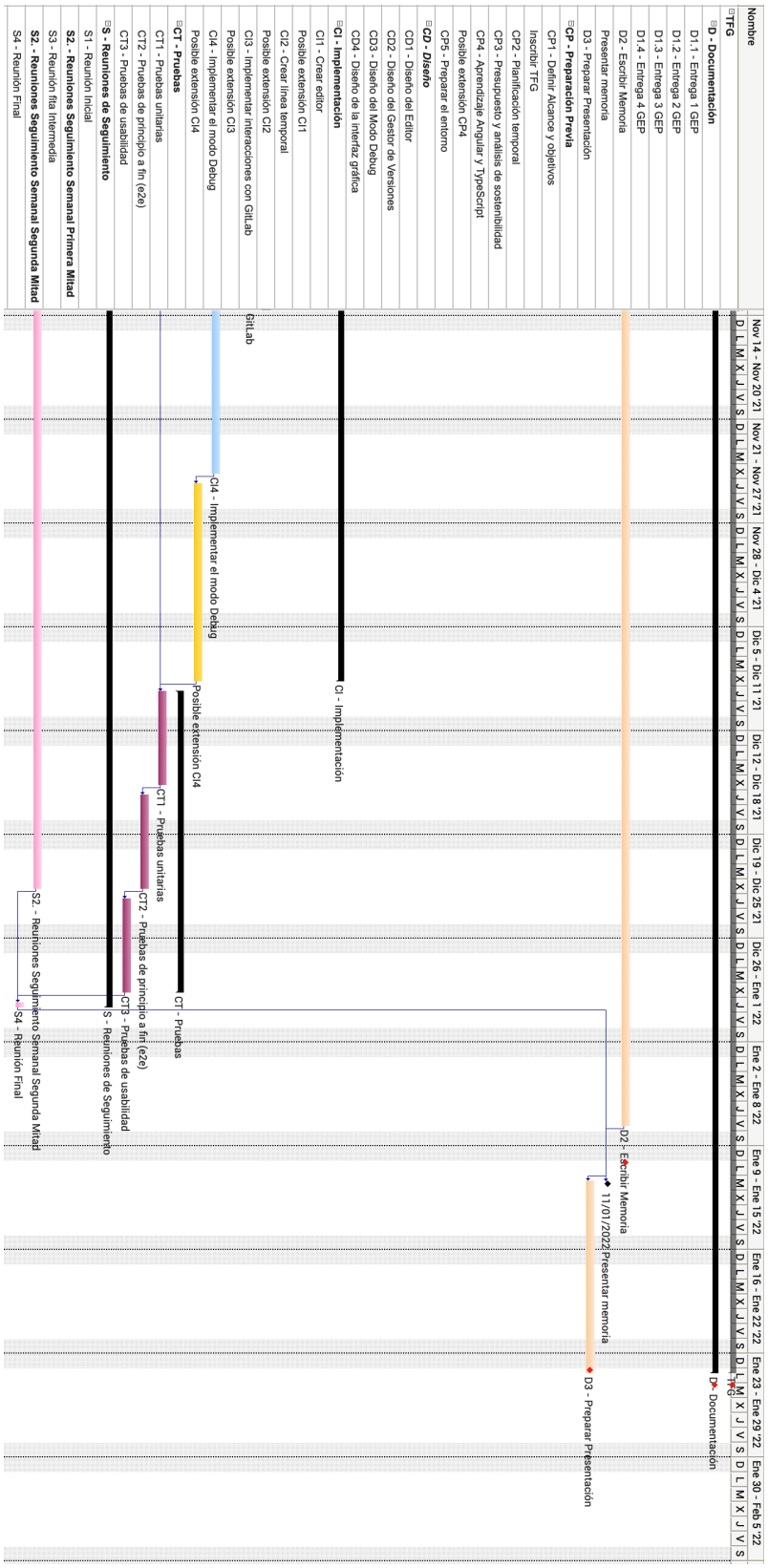


Figura 31 : Tercera parte del Diagrama de Gantt inicial. Elaboración propia.

Diagrama de Gantt final

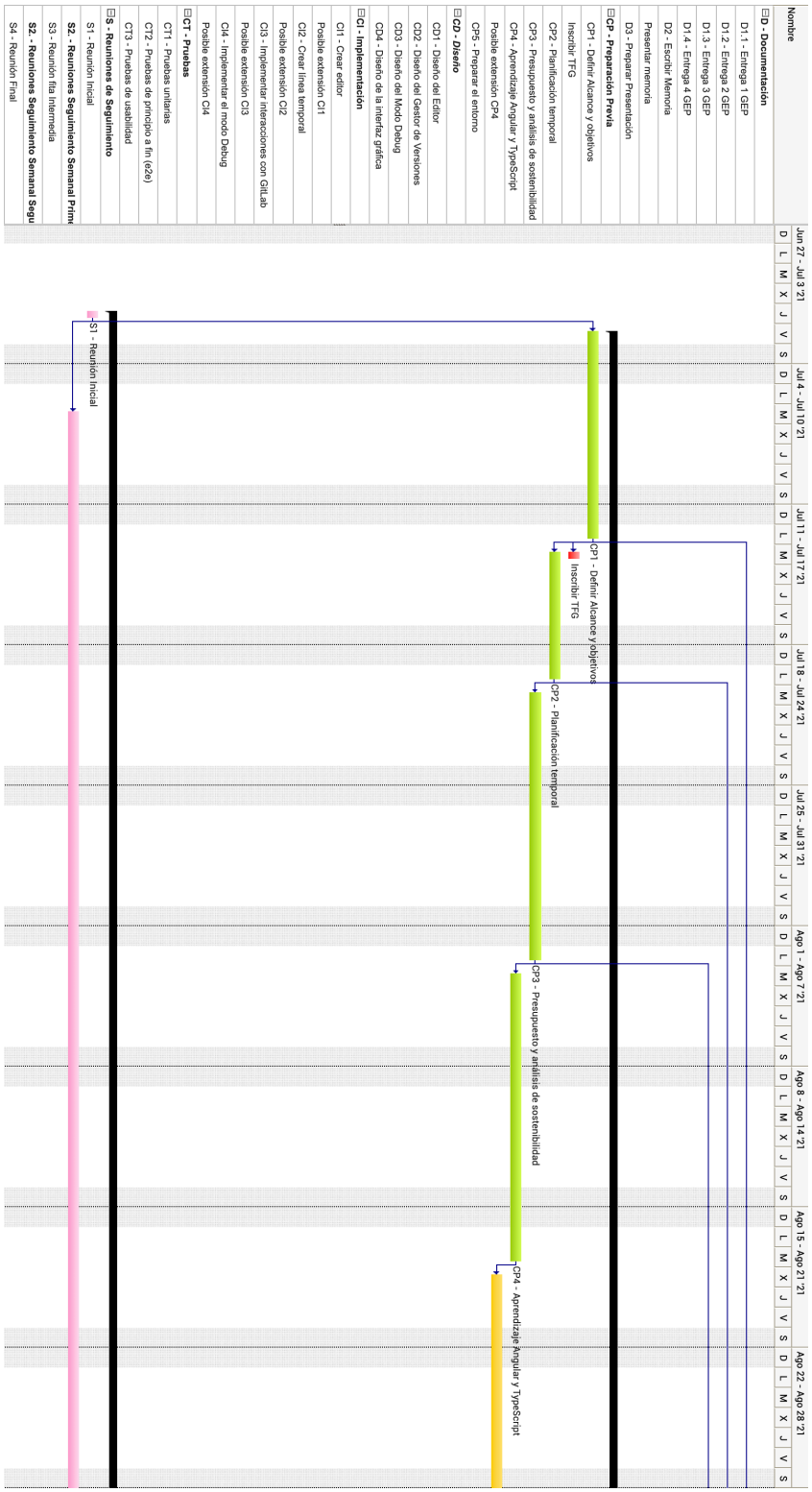


Figura 32 : Primera parte del diagrama de Gantt final. Elaboración propia

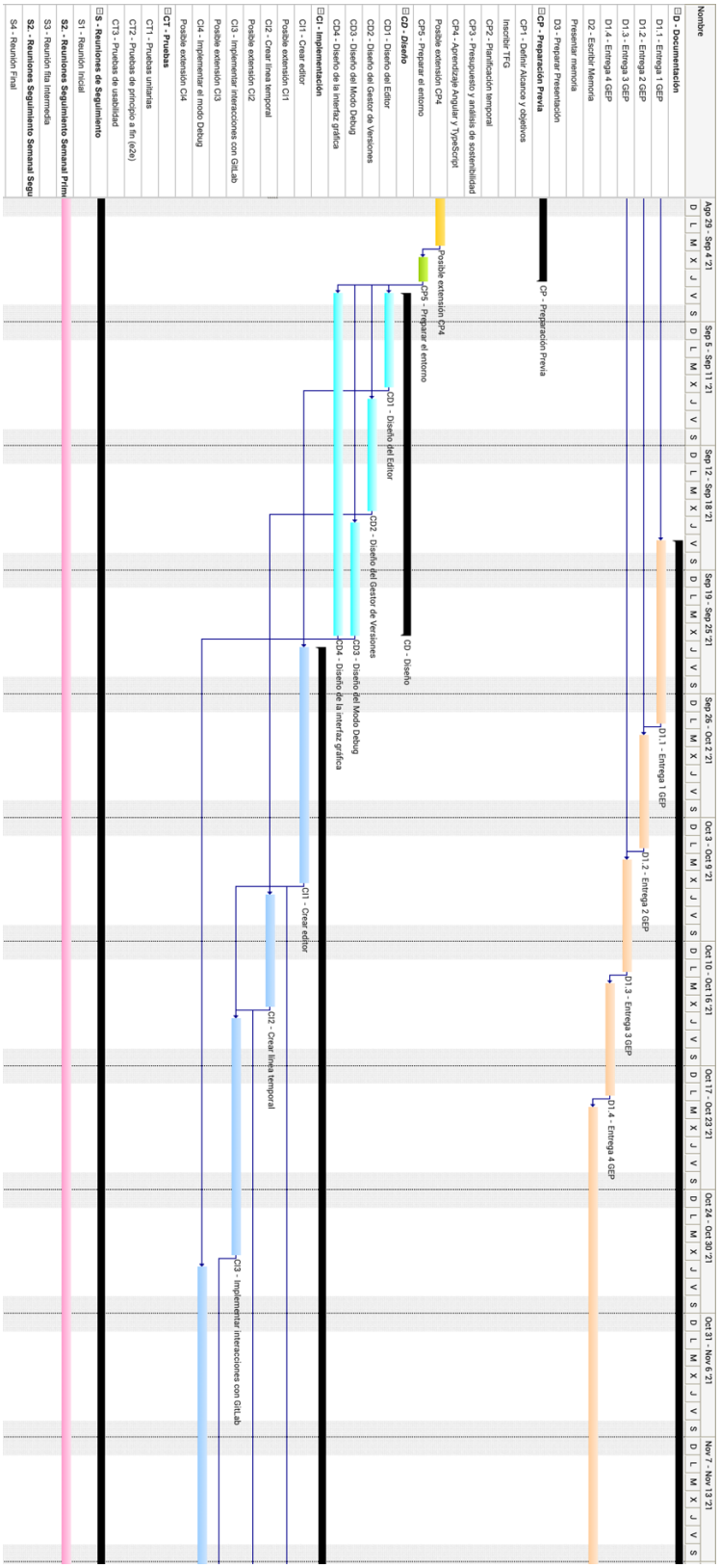


Figura 33 : Segunda parte del diagrama de Gantt final. Elaboración propia

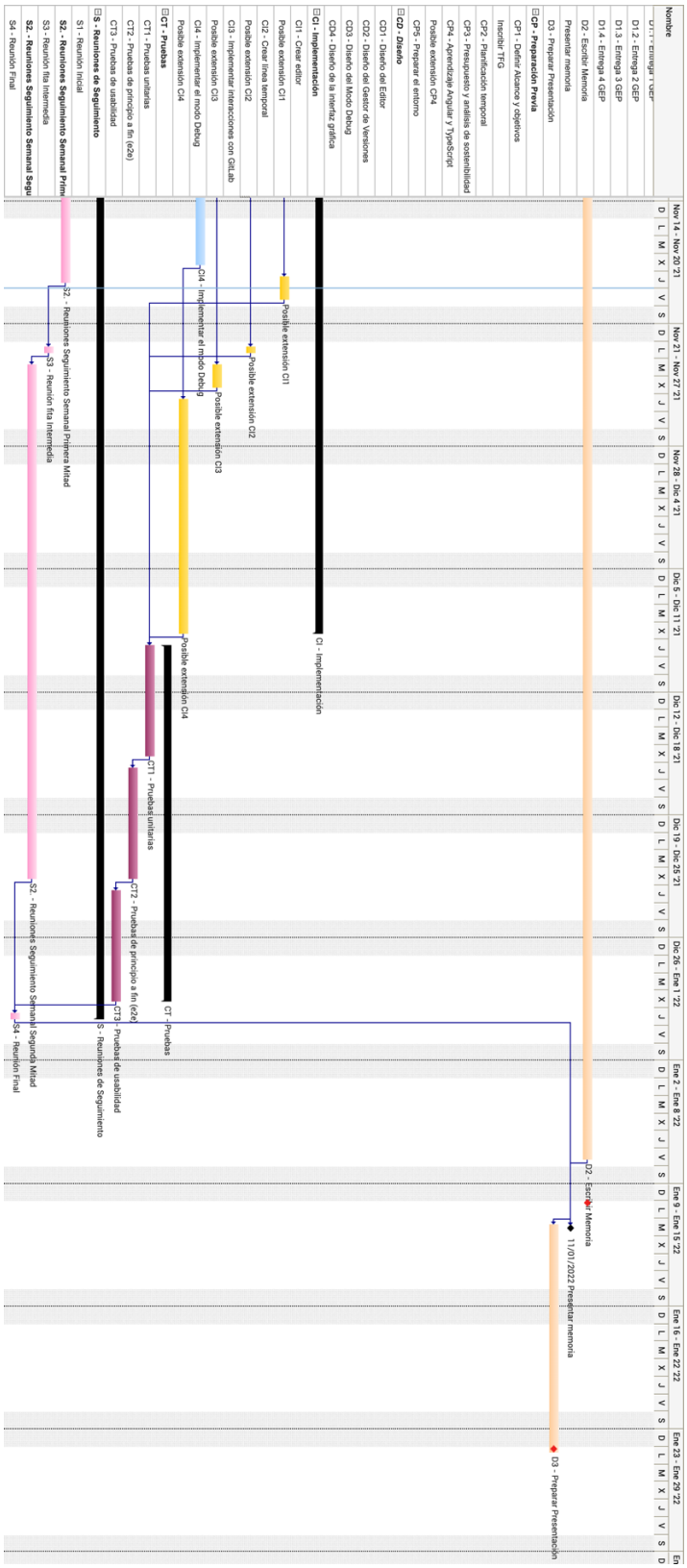


Figura 34 : Tercera parte del diagrama de Gantt final. Elaboración propia

14. Anexo B Manual

Este apartado es una pequeña guía para poder utilizar la parte del editor y programar alguna funcionalidad.

Como en toda aplicación lo primero es introducir el usuario y contraseña, una vez registrado, la aplicación te llevará a la página principal de modelos. Para llegar al editor hay que entrar en el menú que se ve en la imagen inferior y presionar la opción editor.

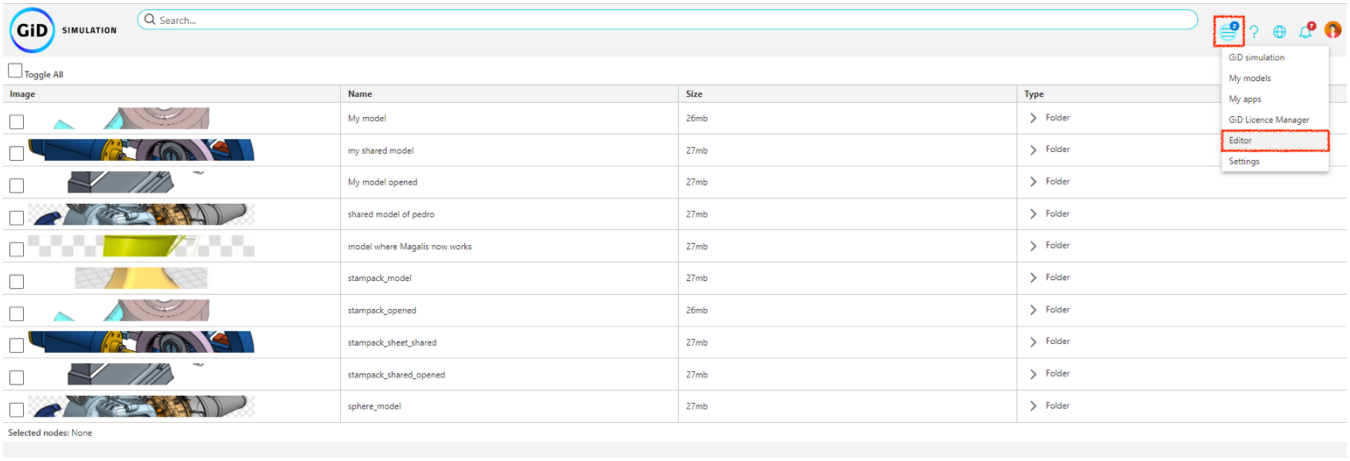


Figura 35 : Manual, pantalla de modelos. Elaboración propia.

Vista rápida

Una vez en la pantalla del editor lo primero que se ve es el editor con los botones de gestión de ficheros.

Funciona como cualquier editor, se presiona en la línea en la que se quiere escribir, en este caso solo se puede escribir dentro del área editable que está marcada por los comentarios de *//Start of editable area* y *//End of editable area* y ya se puede escribir.

También se explicará como acceder a los diferentes editores "Editor and Run" y "Editor and Versions".

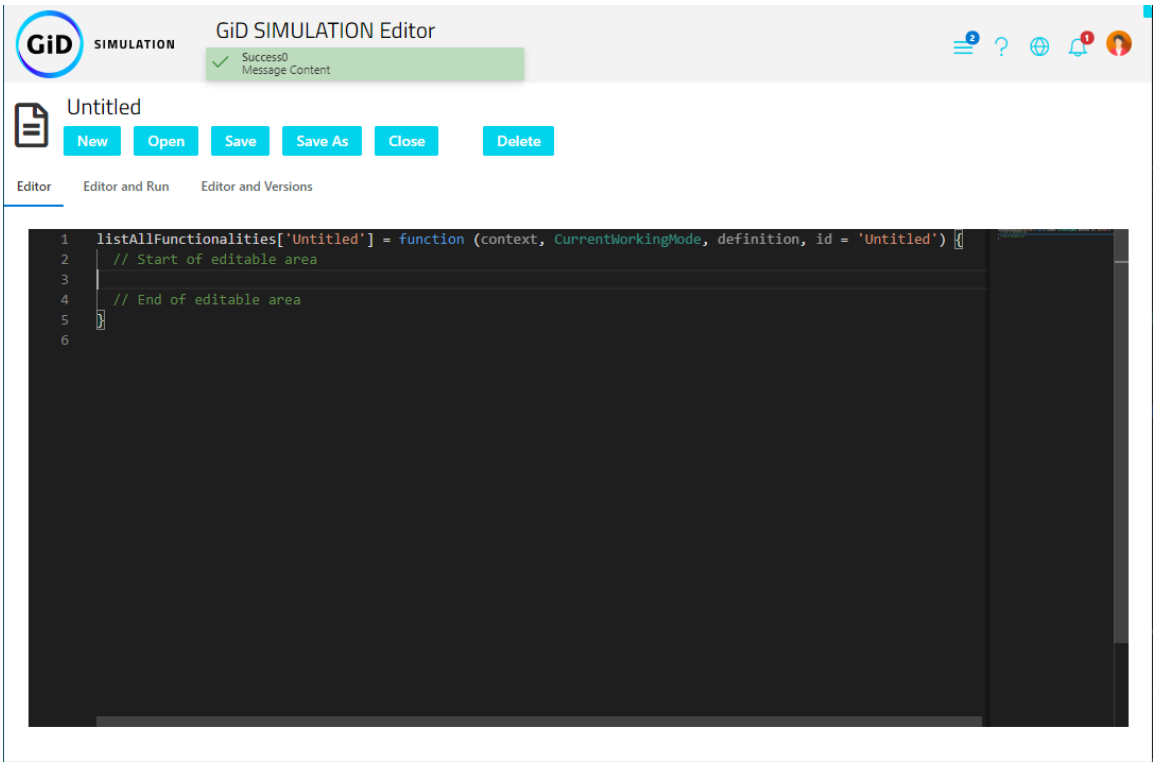


Figura 36 : Manual, pantalla del Editor. Elaboración propia

Para acceder al editor con el apartado de ejecución hay que presionar el botón de la pestaña de "Editor and Run" que se encuentra justo encima del Editor tal y como muestra la siguiente imagen.

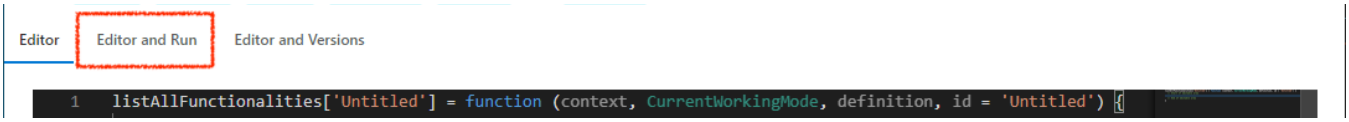


Figura 37 : Manual, botón pestaña Editor and Run. Elaboración propia

A continuación, se puede ver el apartado de "Editor and Run"

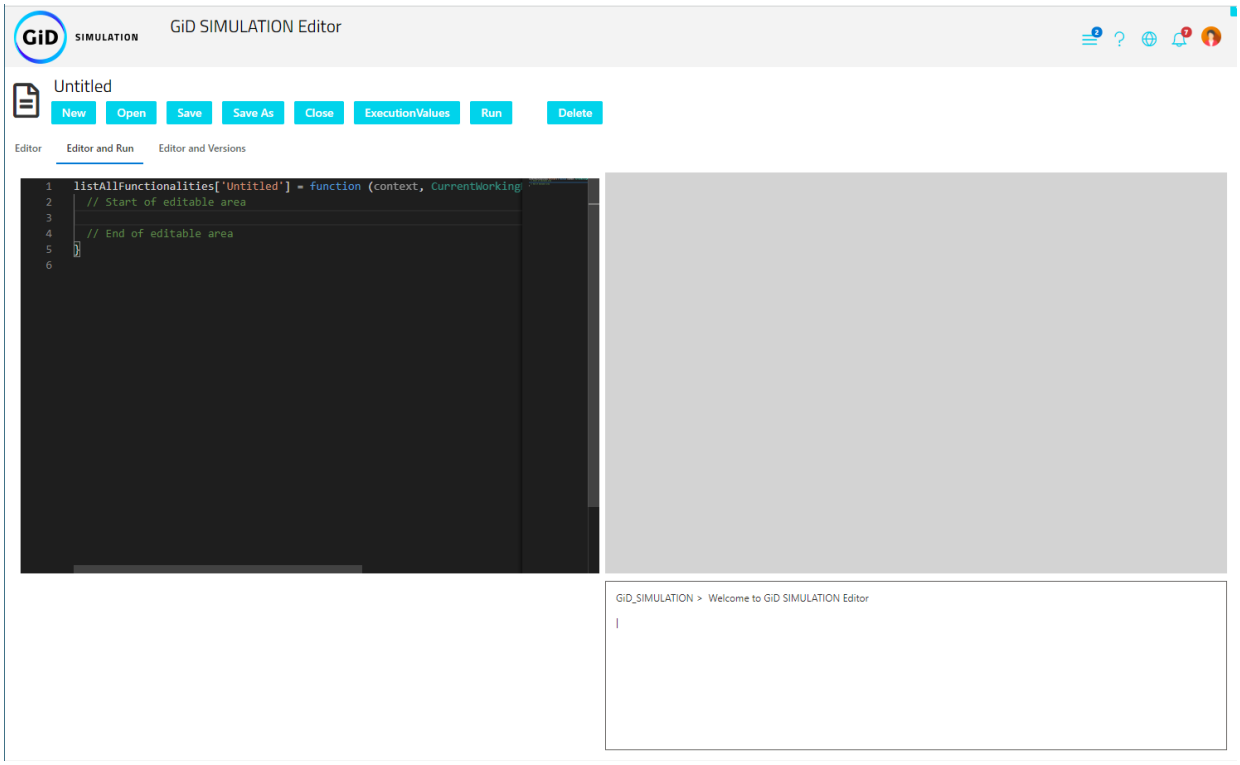


Figura 38 : Manual, pestaña "Editor and Run". Elaboración propia

Para acceder a la pestaña de "Editor and Versions" hay que presionar el botón que de la pestaña "Editor and Versions"

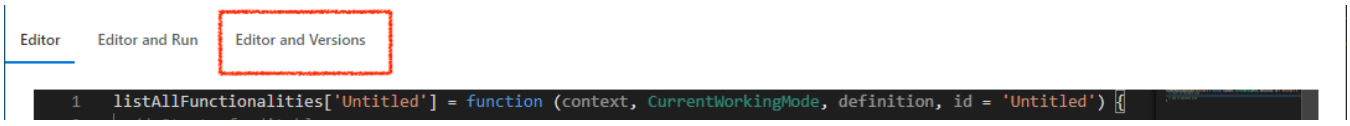


Figura 39 : Manual, botón pestaña "Editor and Versions". Elaboración propia

A continuación, se puede ver el apartado de "Editor and Versions" con un fichero cargado y todas las versiones anteriores del fichero en el lado derecho en la línea temporal. En este caso la línea temporal ha bajado lo suficiente para que aparezca el botón con el icono de una flecha, situado en la parte inferior derecha. Si se presiona este botón la línea temporal se situará otra vez al principio.

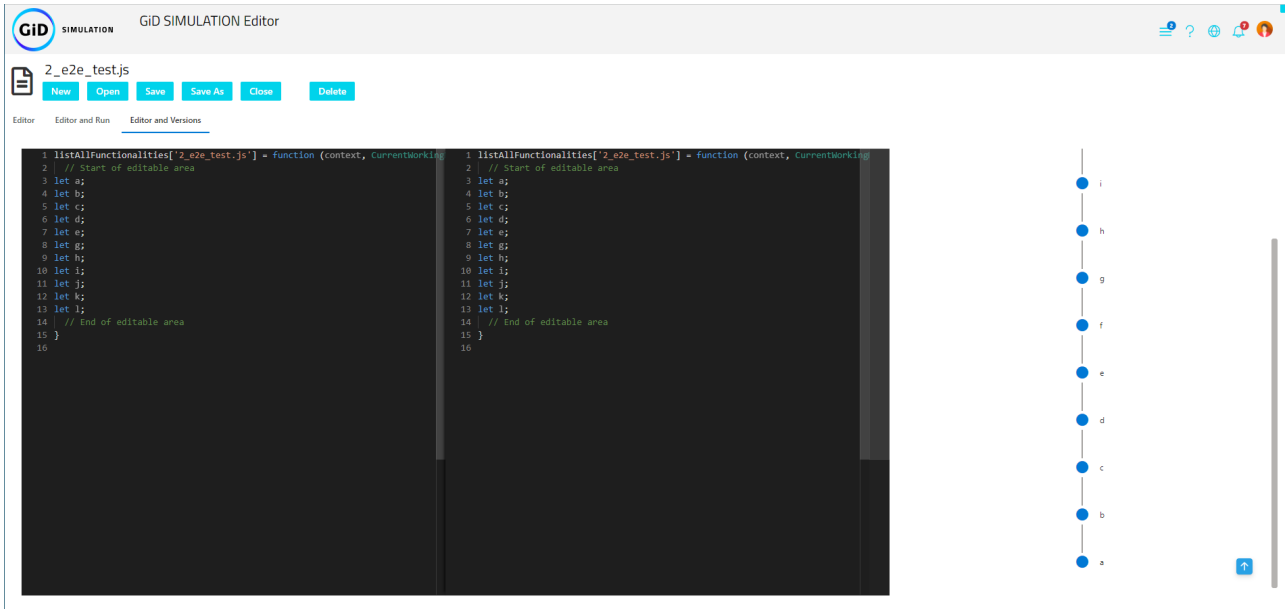


Figura 40 : Manual, pestaña "Editor and Versions". Elaboración propia

Crear un fichero

Para crear ficheros solo hay que presionar el botón 

y aparecerá el mensaje que se puede ver en la siguiente imagen.

En el diálogo se puede ver un campo de entrada donde hay que poner el nombre del nuevo fichero, este nombre será tanto para el fichero como para la funcionalidad que contendrá, en la siguiente imagen se ha escogido el nombre de "Doc_1". Una vez se ha rellenado el campo del nombre hay que presionar el botón 

del mensaje para que el fichero se cree.

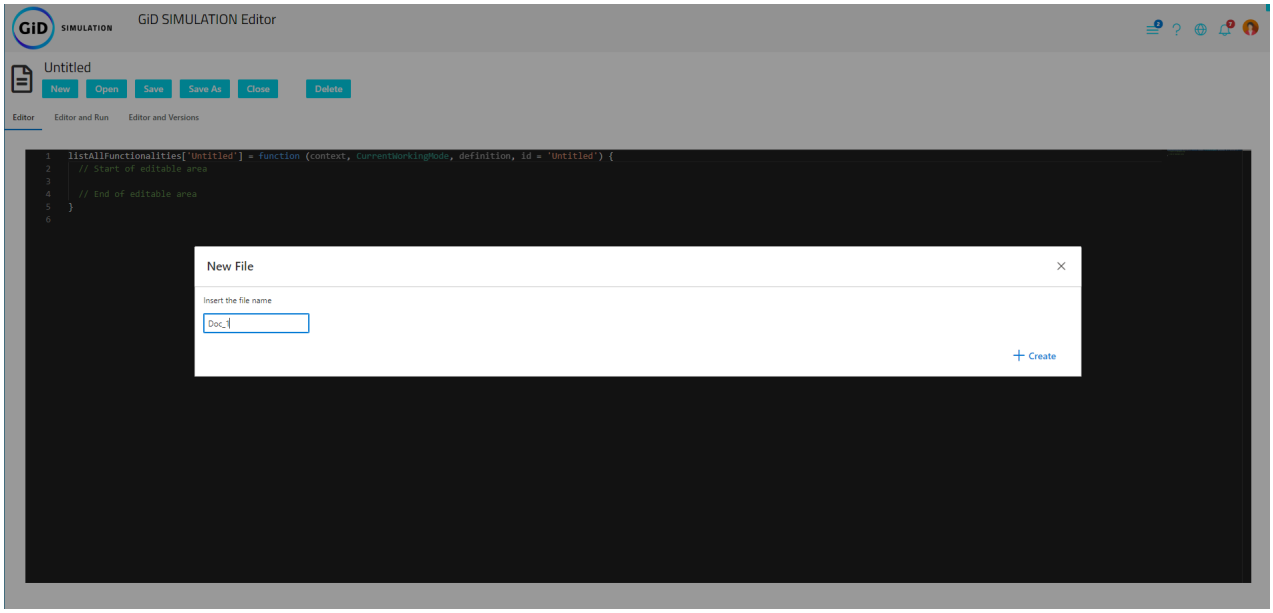


Figura 41 : Manual, crear fichero. Elaboración propia

Una vez creado este fichero se abrirá automáticamente para que el usuario pueda empezar a programar. En la parte superior de la aplicación aparecerá el nombre del fichero y en el editor el texto por defecto, pero con el nombre de la función ya cambiada.

Figura 42 : Manual, fichero abierto después de ser creado. Elaboración propia

Abrir un fichero

Para Abrir un fichero hay que presionar el botón Open, entonces se mostrará un diálogo que se puede ver en la siguiente imagen. Este diálogo contiene una tabla con todos los posibles ficheros que se pueden abrir.

Se selecciona el fichero que se quiere abrir y se presiona el botón Open para abrirlo.

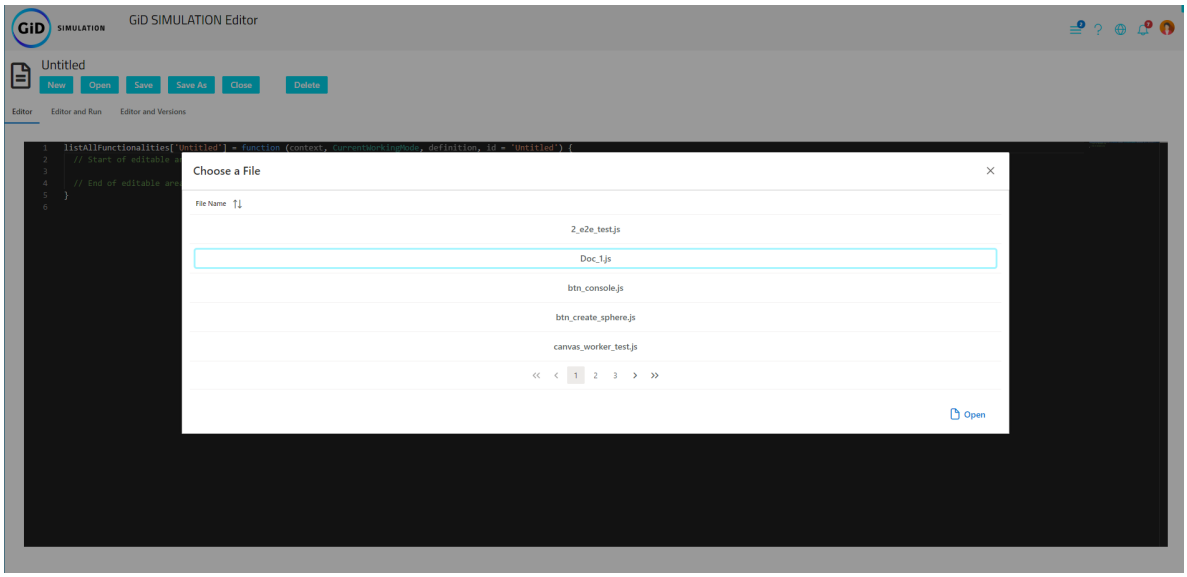


Figura 43 : Manual, abrir fichero. Elaboración propia

Una vez abierto en la parte superior de la aplicación aparecerá el nombre del fichero y en el editor se cargará el código del fichero.

Guardar cambios

Una vez se han hecho modificaciones en el fichero, la línea temporal los marca con un nodo gris y el nombre de "Uncommitted Changes". Lo podemos ver en la imagen que aparece a continuación.

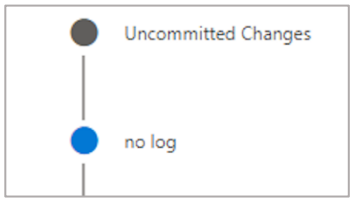


Figura 44 : Manual, línea temporal con cambios no guardados. Elaboración propia

Cunado el fichero ha sido modificado y lo queremos guardar para que no se pierdan los cambios, se presionará el botón Save. Aparecerá un dialogo en el que insertar el mensaje de guardado, este campo es obligatorio.

Cuando se ha escrito el mensaje se presiona el botón Save para guardar definitivamente los cambios.

En la siguiente imagen se puede ver el diálogo que aparece al presionar el botón de guardar y cómo el mensaje que se ha escrito es **let m**.

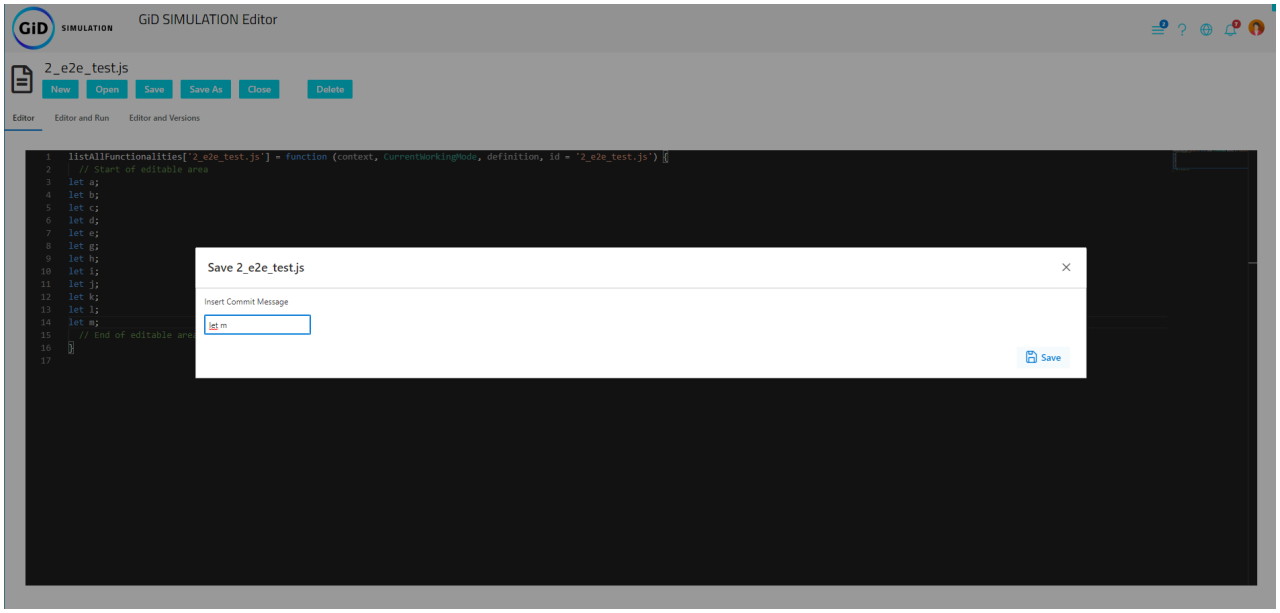


Figura 45 : Manual, guardar cambios. Elaboración propia

Una vez guardado, la línea temporal se actualizará y aparecerá el nuevo mensaje en ella, en este caso **let m**.

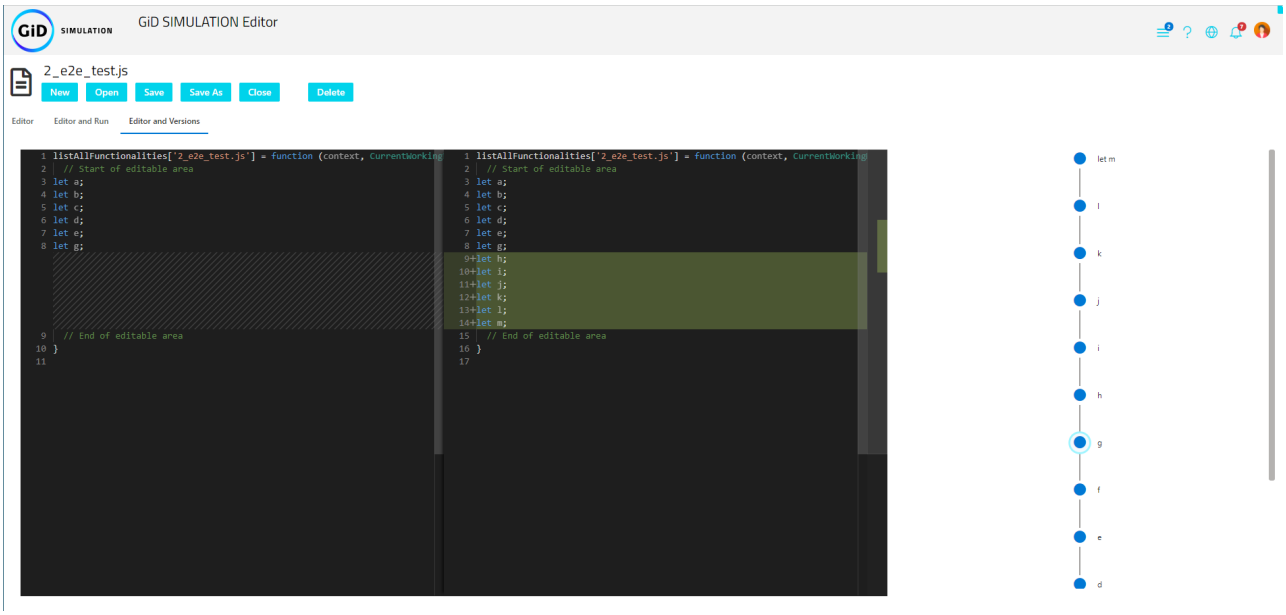


Figura 46 : Manual, línea temporal actualizada. Elaboración propia

Guardar Como

Si se quiere guardar el fichero actual como uno nuevo, hay que presionar el botón Save As. Se abrirá un diálogo que se puede ver en la siguiente imagen, hay que rellenar el campo con el nombre del nuevo fichero y presionar el botón SaveAs

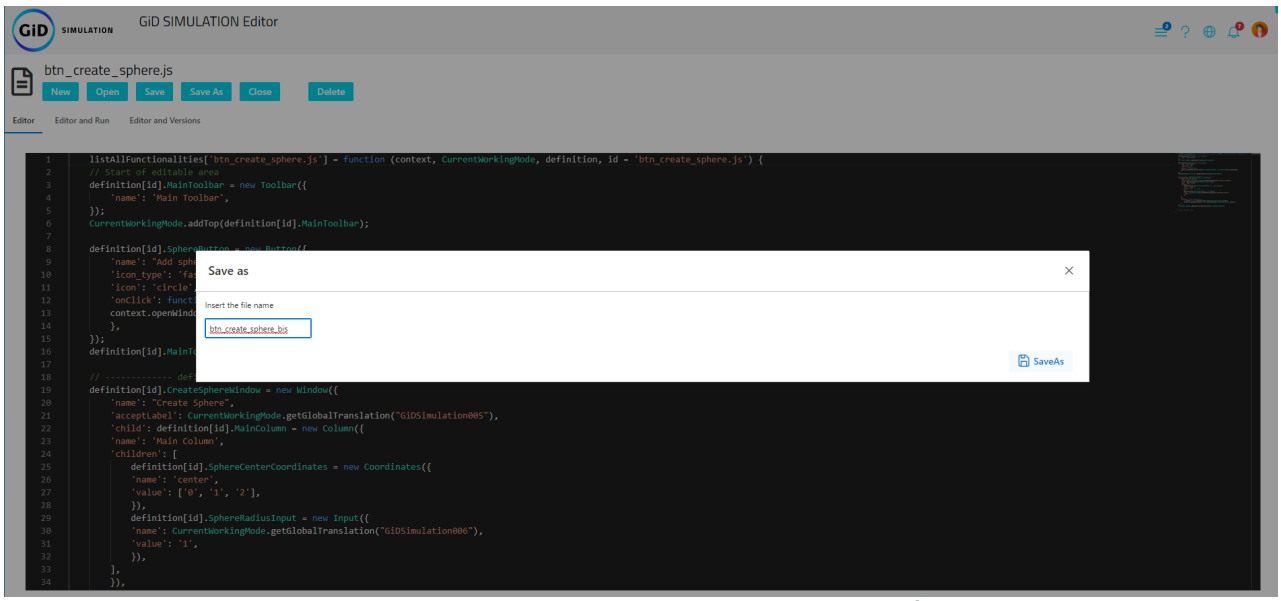


Figura 47 : Manual, guardar como. Elaboración propia

Cerrar fichero

Para cerrar un fichero que está abierto hay que presionar el botón 

.

Si se presiona este botón sin que haya ningún fichero abierto aparecerá un mensaje diciendo que no hay ningún fichero que cerrar.

En caso de que se hayan hecho cambios en el fichero, antes de cerrarlo te preguntará si quieres guardarlo. Este mensaje se puede ver a continuación. Si se presiona el botón de guardar aparecerá el diálogo que pide el mensaje de guardado y si se presiona que no, el fichero se cerrará directamente dejando visible el texto por defecto del editor.

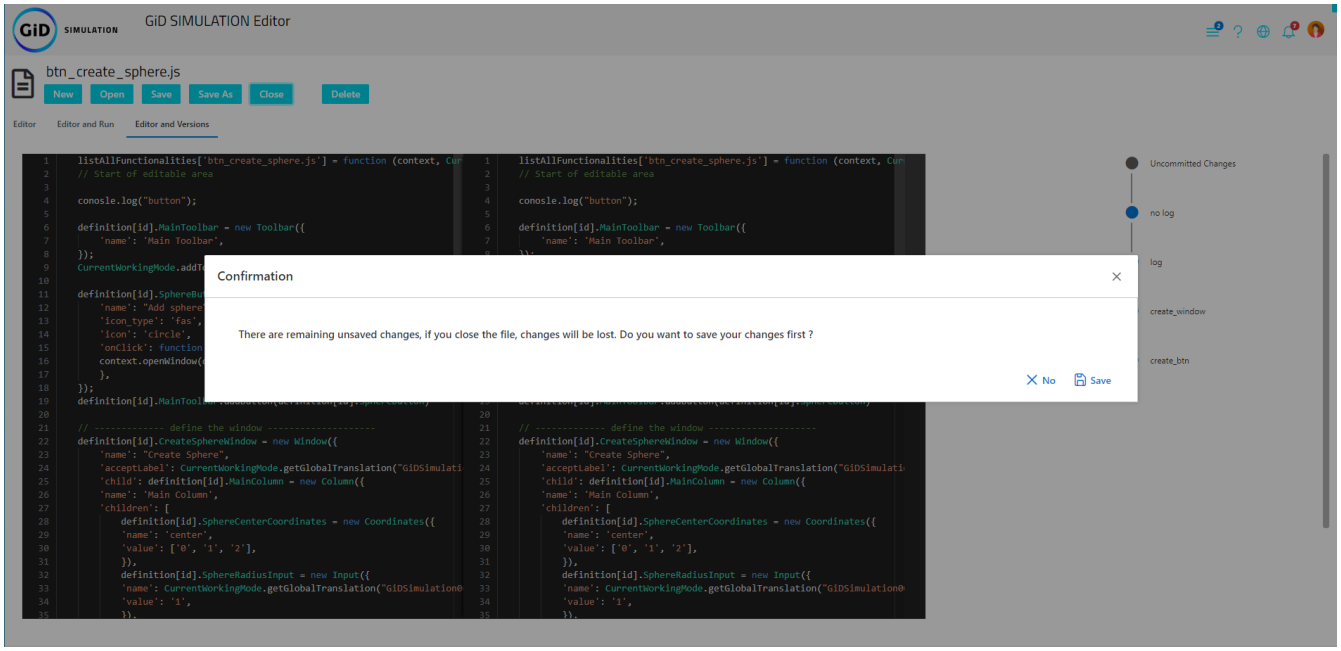


Figura 48 : Manual, cerrar un fichero con cambios. Elaboración propia

Si no hay cambios en el fichero aparecerá un mensaje de confirmación para cerrar el fichero.

Borrar fichero

Para borrar un fichero se tiene que presionar el botón Delete. Esta acción abrirá un diálogo con la tabla de ficheros, que se puede ver en la siguiente imagen.

Se selecciona el fichero a borrar y se presiona el botón Delete.

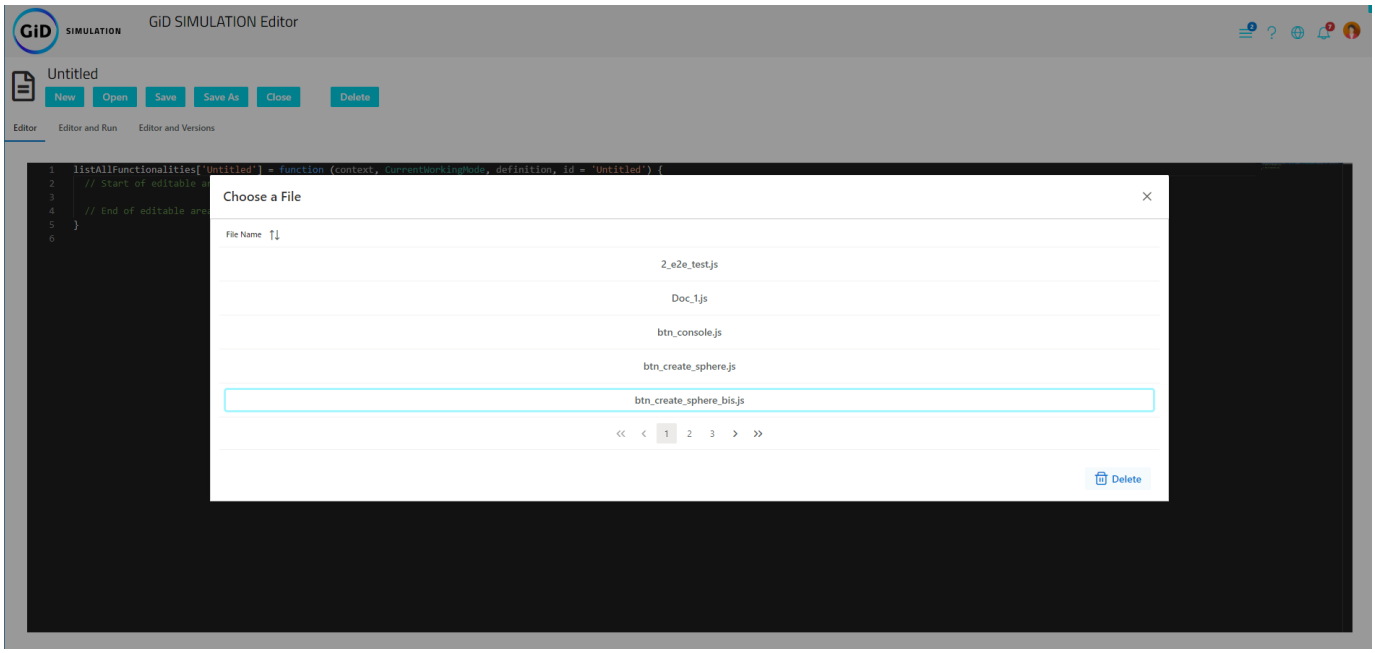


Figura 49 : Manual, borrar fichero. Elaboración propia

En caso de que se quiera borrar el fichero que está abierto, la aplicación lo cerrará primero, dejando el texto por defecto y luego se borrará. Si se borra un fichero que no está abierto la acción de borrar se hace directamente sin interactuar con el editor que en estos momentos puede tener otro fichero abierto.

Ejecutar Código

Para ejecutar el código lo primero es ir al apartado de "Editor and Run". Una vez en este apartado encontramos el botón ExecutionValues Con él podemos cambiar ciertos valores que supuestamente la aplicación retorna para comprobar casos muy

concretos. En la siguiente imagen se puede ver el diálogo que aparece, de momento permite controlar el número de nodos y el número de elementos.

Una vez cambiado se puede regresar al valor anterior con el botón  .

Cuando ya se tienen todos los datos modificados se presiona el botón  para guardar los valores actualizados.

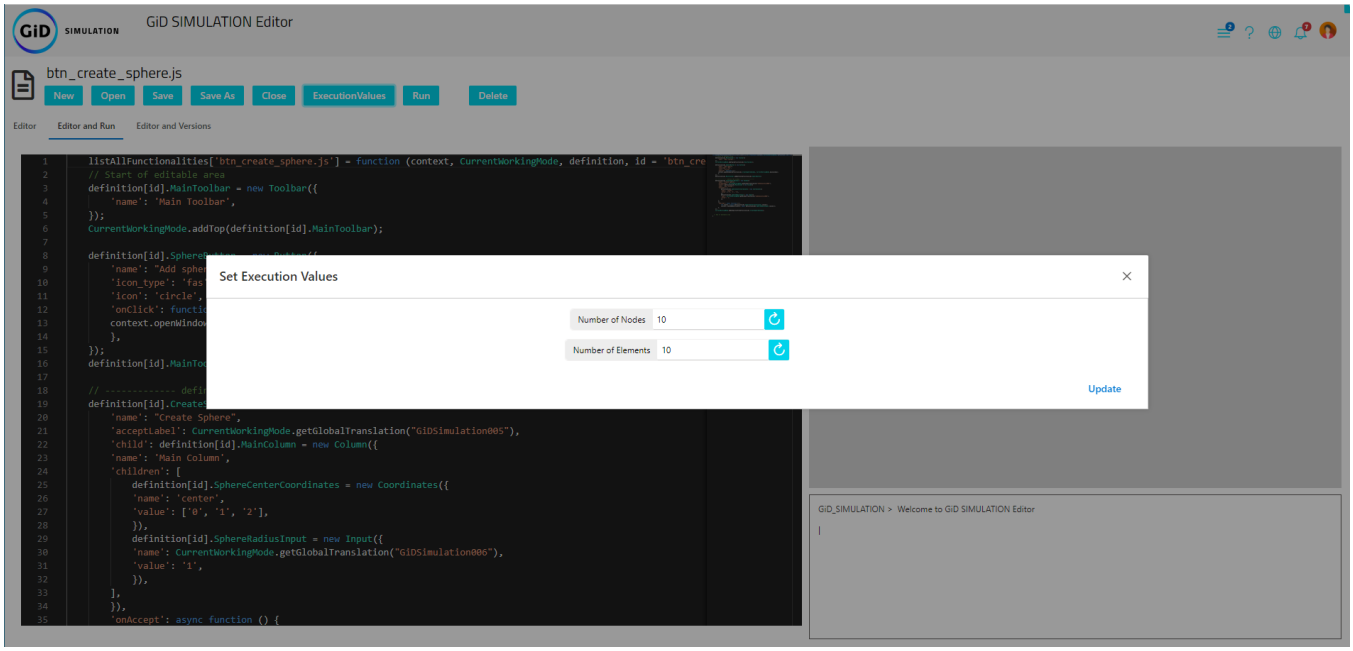


Figura 50 : Manual, cambiar valores por defecto de la ejecución. Elaboración propia

Cuando está todo listo para ejecutar el código que está ahora mismo en el editor, se presiona el botón  para que empiece la ejecución.

En la siguiente imagen se puede ver una ejecución finalizada, en este caso el código hace aparecer un botón de forma circular para crear una esfera.

Al presionar el botón se abre una ventana en la que te pide el radio y la posición del centro de la esfera. Cuando se presiona el botón de crear en el apartado del terminal, aparece toda la información de la esfera, pero no aparecerá en la pantalla de visualización. Esta pantalla es para comprobar el funcionamiento de la interfaz, las geometrías no aparecen, solo te informa si la creación ha sido exitosa o no.

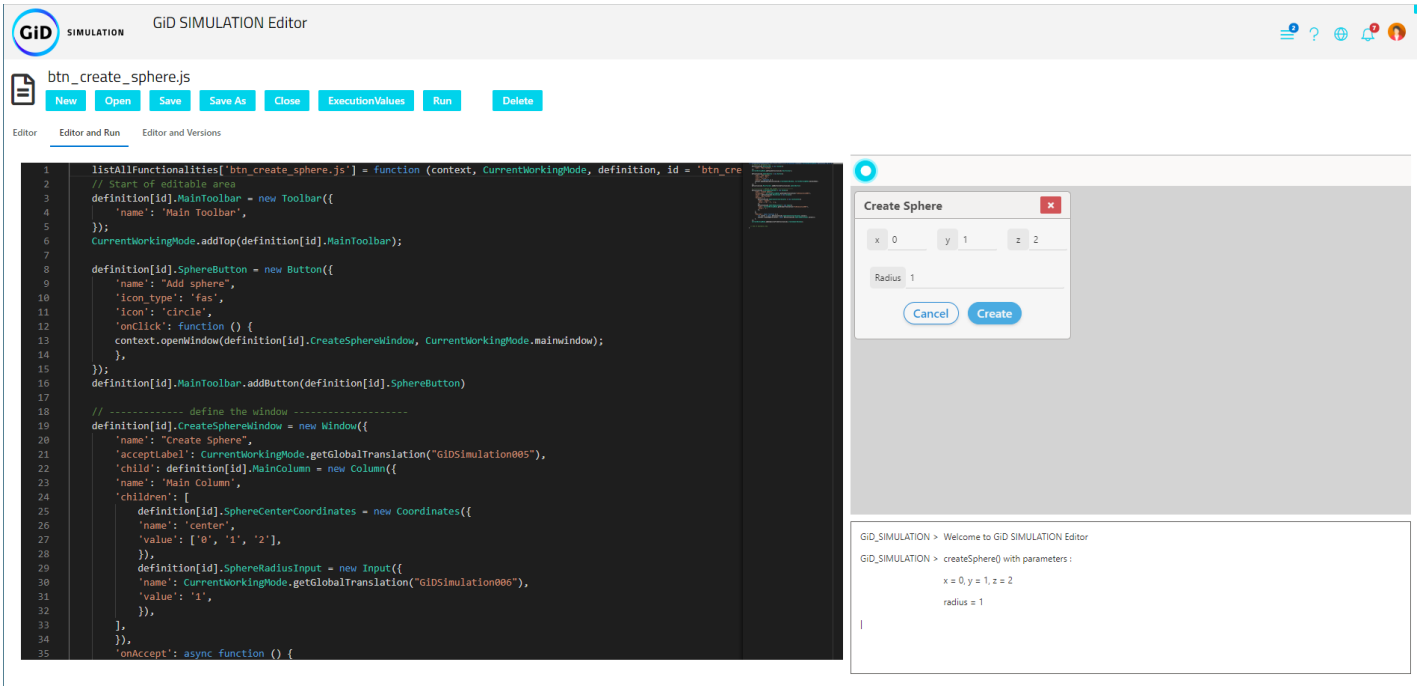


Figura 51 : Manual, ejecución de la creación de una esfera. Elaboración propia

En este caso la ejecución ha sido exitosa, pero en la siguiente imagen podemos ver otra ejecución que no lo ha sido, ha tenido errores en el código.

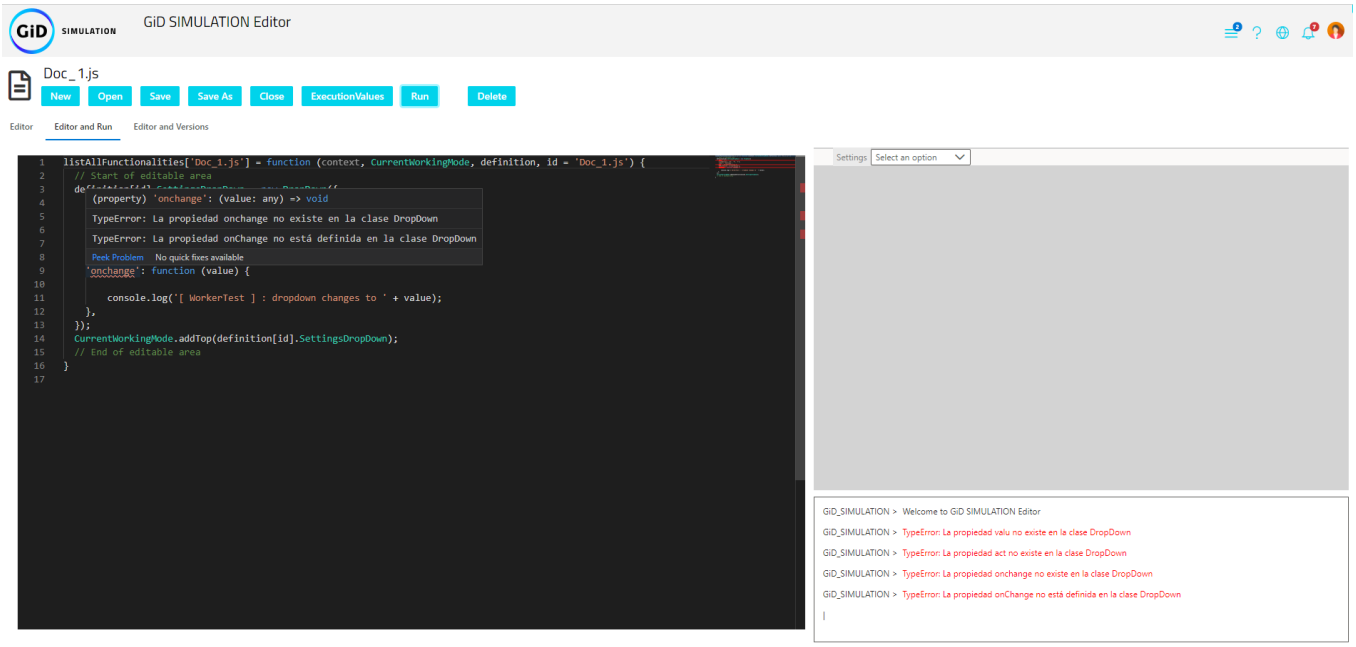


Figura 52 : Manual, errores en la ejecución. Elaboración propia

Comparación de versiones

Para acceder a esta funcionalidad hay que ir al apartado de "Editor and Versions". Una vez en esta ventana es necesario tener un fichero abierto para poder ver la línea temporal. Si se presiona en cualquiera de los nodos 

se cargará el código de esa versión en el editor izquierdo, mientras que en el editor derecho seguirá el código actual. A continuación, se puede ver una imagen que compara dos versiones de un fichero.

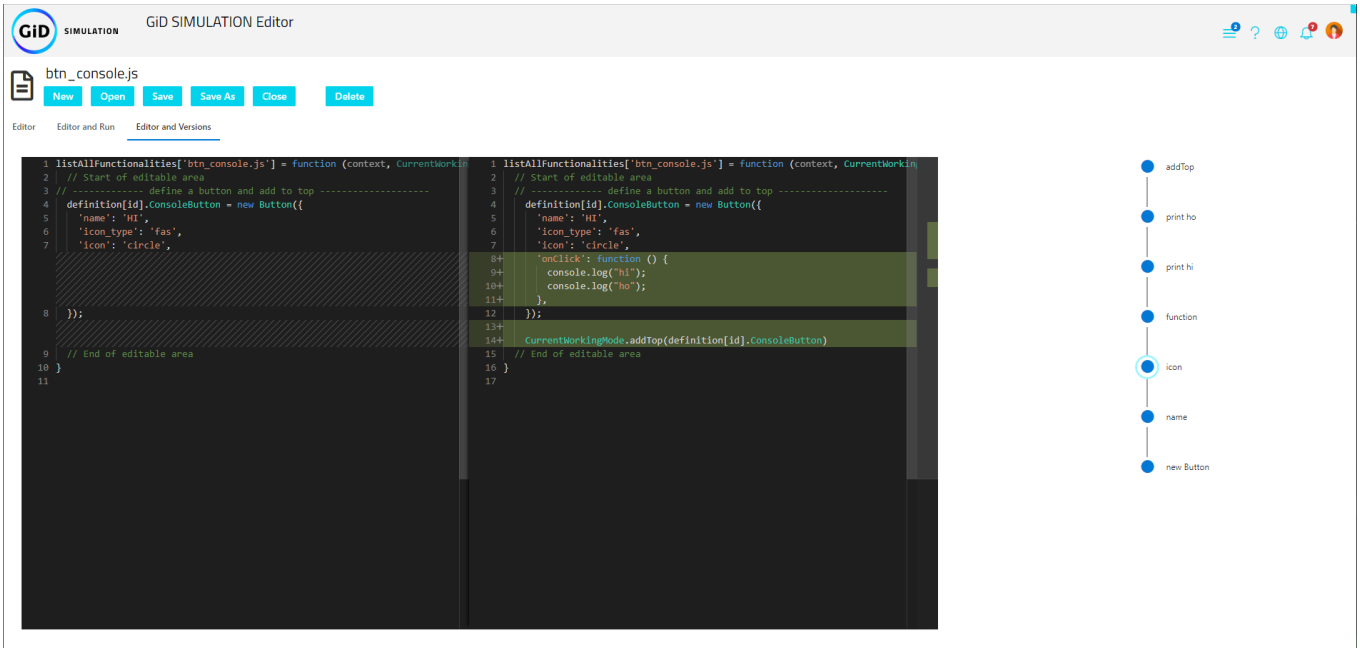


Figura 53 : Manual, comparación de dos versiones. Elaboración propia

Todo lo que aparezca de color verde son modificaciones que se han hecho o líneas de código que se han añadido y todo lo que se marca en rojo son líneas de código que se han borrado.