Trabajo Fin de Grado

Desarrollo de una aplicación para el seguimiento de nuevos empleados

Development of an application for tracking new employees

Autor/es

Paula Oliván Usieto

Director

Vanessa Martín Oliete

Project Manager en NTT Data

Ponente

José Merseguer Hernaiz

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2024

Agradecimientos

En primer lugar, me gustaría agradecer a todo el equipo de NTT Data por su confianza al ofrecerme el proyecto de mejora del sistema de *mentoring* actual. En especial a Vanessa y Manu como *Project Managers* del equipo *DEX*. Gracias por aceptar mi solicitud para la realización del TFG en el equipo y por ayudarme en la búsqueda del tema.

También me gustaría agradecer a Ángel por su papel de tutor durante el desarrollo de la plataforma. Todos los consejos ofrecidos y la ayuda en los momentos donde estaba perdida y no sabía cómo seguir avanzando con el proyecto no se pueden pagar. No creo que sin las reuniones semanales donde me ofrecías tu punto de vista la plataforma hubiera tenido todas sus características diferenciadoras y una identidad propia.

Siguiendo con el personal de NTT Data me gustaría agradecer a Enrique. Ambos llegamos a la empresa estando un poco confusos y sin saber qué nos depararía la experiencia de estos meses. Has sido un gran apoyo durante los días en los que los errores no paraban de surgir y en los ratos de cafés desconectábamos de ellos volviendo al trabajo con más energía.

Me gustaría dar las gracias a José Merseguer, por su papel como ponente y la ayuda dada en el desarrollo de esta memoria. Con tus consejos la memoria ha pasado a ser un todo completo y con una estructura mucho más clara y concreta.

Gracias por supuesto a mi familia y amigos que durante estos cuatro años de lucha y esfuerzo han sido un pilar fundamental. Ofreciendo un hombro para los momentos duros y una sonrisa en los momentos de celebración, sin vosotros estos años habrían sido mucho más duros.

Desarrollo de una aplicación para el seguimiento de nuevos empleados

Resumen

Este TFG se ha desarrollado dentro del equipo de *Digital Experience* de la empresa NTT Data. En el mismo se ha creado una herramienta de seguimiento de nuevos empleados para ser utilizada dentro de la empresa. Con el objetivo base de que durante mi estancia como estudiante en la empresa pudiera aprender las herramientas que se usan dentro del equipo. Además, se buscaba un proyecto que en la empresa hiciera más fácil alguna de sus tareas diarias.

Siguiendo esta premisa, con mi directora de TFG y tutor de empresa, decidimos la creación de una aplicación que sirviera para facilitar las labores relacionadas con el *mentoring*. Consiguiendo simplificar el trabajo del departamento *People* a la hora de asignar mentores o conocer la evolución de los nuevos empleados. Además, se ayudaría a los empleados a tener una mejor relación mentor-*mentee* y una evolución continua.

Durante el ciclo de desarrollo de la aplicación se siguió una metodología en cascada mejorada. Esta decisión se tomó ya que tras la investigación inicial se fijaron los requisitos que la plataforma debía seguir. Al tenerlos asegurados se permitió ir completando las fases del proyecto, y una vez aprobadas por los responsables de *DEX*[1] pasar a la siguiente etapa. Con la ventaja de que la metodología en cascada mejorada permite volver a la etapa anterior en caso de detectar errores o fallos.

Tras las fases iniciales se decidió que se usaría el patrón Modelo-Vista-Controlador [2]. Así se separaría la lógica del programa y la interacción con base de datos de la interfaz, que sería accesible por los distintos usuarios. Con ello conseguiríamos una mayor modularidad y poder desarrollar cada parte de manera independiente.

Para la creación de la vista y el controlador se usó React [14] con JavaScript XML, junto a la librería de componentes UI Material UI [15]. Esta decisión se tomó ya que es el lenguaje usado mayormente en el equipo *DEX.* El uso de una librería centrada en UI permite crear una plataforma personalizada de forma rápida y sencilla.

Para el desarrollo del modelo se usó como base de datos FireBase [10] al ofrecer ya un entorno en la nube evitando las configuraciones en local y posibles fallos durante su desarrollo. La interacción se hizo con archivos JavaScript. Finalmente se integró un servidor de *mailing* local programado con la biblioteca NodeMailer [18].

Concluida la etapa de implementación comenzaron las pruebas. Aunque los componentes eran probados conforme se añadían los controladores, en la fase de pruebas se hicieron comprobaciones globales de los componentes en su conjunto. De esta manera se verifica que se cumple el caso de uso y diagrama de secuencia que le corresponde.

Cuando se acabaron las pruebas se realizó una presentación de la plataforma a los responsables del equipo y al tutor de la empresa para conocer su opinión y posibles mejoras. Todo esto fue añadido para la obtención de una web más moderna e intuitiva.

[1] Abreviatura para referirnos al departamento *Digital Experience*

Índice

Índice

1 - Introducción

Este proyecto ha sido desarrollado en la empresa NTT Data en sus oficinas de Zaragoza. Allí se encuentra el departamento *Digital Experience* del cual he formado parte durante todas las etapas de la realización del Trabajo Fin de Grado. El proyecto ha sido dirigido por Vanessa Martín como *Project Manager* del departamento. Como ponente actuó José Merseguer, profesor de la Universidad de Zaragoza.

En NTT Data se valora en especial medida la comodidad y felicidad de los empleados, por ello desde hace años existe el programa de *mentoring* para nuevos empleados. El programa *mentoring* consiste en la creación de parejas de empleados mentor – *mentee* (denominación que reciben los nuevos empleados de la empresa pertenecientes al *mentoring*).

Actualmente el *mentoring* es muy importante para la empresa, ya que ayuda a los nuevos empleados a conocer los valores y cultura de NTT Data de una manera cómoda, al ser enseñada por otros empleados que tienen más experiencia dentro de la compañía. El *mentoring* también consigue que los *mentees* se sientan acompañados durante su proceso de integración en la dinámica de la empresa. Los *mentees* siempre tienen un referente al que le pueden preguntar sin miedo a las posibles repercusiones que podrían existir si esa pregunta se realizase dentro del equipo o con el cliente.

A pesar de esto, el *mentoring* no cuenta con ningún sistema de retroalimentación para que desde el departamento encargado de la gestión se pueda conocer cómo están actuando los integrantes del proceso. Esto también sucedía por la parte de los protagonistas, ni mentores ni *mentees* conocían de manera real su avance dentro de la empresa. La retroalimentación que podían obtener debía ser pedida por alguno de los miembros de la pareja al otro miembro, lo cual podría ser incómodo para los nuevos empleados que podían sentirse abrumados al pedir ese tipo de información de manera directa.

Desde el punto de vista de la compañía, los mentores son considerados empleados capaces de desarrollar este papel, ya que llevan varios años en la empresa. Por eso la conocen y pueden ser los guías de sus *mentees*. Los mentores además deben realizar ciertos cursos para poder desenvolverse correctamente con otros empleados. A pesar de esto, y en especial si es su primera experiencia como mentor, se pueden sentir abrumados, sin saber exactamente qué decisiones tomar o los mejores métodos de ayuda hacia los *mentees*.

Otra situación de riesgo para los *mentees* ocurre cuando su mentor tiene un cargo alto dentro de la empresa. Ello puede ocasionar que no tenga tiempo suficiente para dedicar al *mentee* y este se sienta abandonado por parte del mentor.

Debido a todo esto se pensó en la posibilidad de desarrollar una plataforma de carácter interno. Se conseguiría que la compañía pudiera monitorizar de mejor manera la evolución de *mentees* y mentores. También se ofrecería a mentores y mentees una aplicación donde tener unificadas todas las herramientas necesarias para el proceso de *mentoring*.

La plataforma tendría como objetivo cubrir todas estas necesidades que tanto administradores, mentores y *mentees* podían encontrar al formar parte del sistema de *mentoring.* Aún teniendo en cuenta que la empresa cuenta con distintas herramientas para comunicación y creación de reuniones como Microsoft Teams o Microsoft Outlook no son servicios creados específicamente con el propósito de ayudar al proceso de *mentoring*. Esto obliga a los usuarios a utilizar varias aplicaciones para realizar su papel de manera adecuada.

Por ello los objetivos concretos del proyecto son:

- Estudiar las necesidades reales del equipo que gestiona el *mentoring,* así como las que poseen mentores y *mentees* para poder conocer los requisitos que tendrá la plataforma.
- Implementar una plataforma interna que se ajuste a esos requisitos y pueda ofrecer a todos los usuarios retroalimentación para mejorar el sistema de *mentoring* actual y su experiencia.
- Obtener un sistema que pueda ser presentado al equipo *People* para que la plataforma sea finalmente utilizada dentro de la empresa mejorando y facilitando sus labores de administración del *mentoring*.

En la fase de implementación se necesitó desarrollar distintos elementos de manera independiente. Estos fueron: servidor de *mailing*, base de datos y componentes.

- Servidor de *mailing*: esencial ya que la plataforma cuenta con envío de correos electrónicos para los usuarios. Había varias opciones para la creación del sistema de *mailing,* pero se podría resumir en el uso de una solución en la nube o la creación de un servidor en local que se encargará del envío gracias a una librería como *NodeMailer*. Se decidió implementar el servidor local priorizando el control total sobre la configuración y la lógica, así como la reducción de costos y la dependencia de servicios externos. Aunque opciones en la nube son rápidas de integrar, la solución local ofrece mayor flexibilidad, escalabilidad y privacidad.
- Base de datos: inicialmente se utilizaron archivos Json para poder crear los componentes que únicamente leían datos de la base de datos. Cuando se finalizaron estos componentes iniciales se buscó la mejor solución para la edición de los archivos Json. Se investigaron las distintas opciones existentes: bases de datos en local, librerías que permiten editar los archivos Json y bases de datos en la nube. Tras la búsqueda se optó por hacer uso de una base de datos en la nube, concretamente FireBase debido a que ofrece un sistema estructurado de forma similar a como se organizan los archivos Json, solución inicial usada. Además, ofrece herramientas para almacenar imágenes. Esta característica no se permite en los archivos Json en local. Asimismo, al estar en la nube su configuración fue sencilla facilitando hacer la migración de los

Json a FireBase sin muchos problemas en comparación con la creación de una base de datos en local.

- Componentes: el lenguaje en el que crear los componentes estuvo claro desde el principio, se usaría React. Así serviría de toma de contacto con las herramientas que el departamento *DEX* utiliza. Aun así, hubo decisiones que tomar como la creación del proyecto y el uso de una librería enfocada a UI. Comenzando con la creación del proyecto las opciones eran la creación a mano de las distintas carpetas, *create-react-app* y *vite*. La primera opción fue descartada por la gran cantidad de dependencias existentes ya que la ventaja que ofrece es la configuración total del proyecto, pero no era necesario modificar la estructura básica. Se debía entonces elegir si usar *create-react-app* o *vite*. Después de compararlos se acabó eligiendo *vite* al ofrecer herramientas para optimizar los componentes consiguiendo una plataforma más veloz. En cuanto a la librería de UI inicialmente no se valoró la opción para que pudiera ser usada sin problemas de licencias. Pero tras la primera reunión con los responsables del proyecto se optó por usar Material UI. Esta librería ofrece componentes ya testeados extensamente, a pesar de que esto implicara cambiar la librería en caso de que el proyecto sea aprobado por el departamento *People.*

1.1 - Metodología

Como se ha nombrado anteriormente el proyecto se ha realizado siguiendo los principios de una metodología en cascada mejorada. En concreto, se centró en desarrollar las etapas realizando una reunión con el tutor asignado en la empresa cada vez que una de las fases finalizaba, para así tener una opinión externa y poder realizar los cambios correspondientes antes de seguir avanzando. Además de las reuniones existía comunicación asíncrona de carácter semanal para conocer el avance del proyecto y comentar problemas que se encontraban.

Durante las fases iniciales del proyecto, las reuniones fueron más frecuentes y rápidas al contar con una idea clara del proyecto. Al llegar a la fase de implementación la cantidad de reuniones se minimizó debido a que se necesitaban varias semanas para contar con el desarrollo completo de los componentes de cada uno de los usuarios. Debido a esto se optó por crear un sistema de colores en el mapa de navegación y la tabla de requisitos funcionales. Esto nos permitía conocer rápidamente el estado del desarrollo sin necesidad de analizar el código. Se priorizaron a su vez los componentes más relevantes, para crear al menos un producto mínimo funcional en caso de que las horas necesarias excedieran a las disponibles para mi estancia en NTT Data. El sistema de colores funcionaba de la siguiente manera:

- Rojo: todavía no se había creado el componente o funcionalidad correctamente.
- Naranja: en el mapa de navegación significaba que el componente se renderizaba de manera adecuada pero no poseía la funcionalidad debida. Si el color naranja se ubicaba en la tabla de requisitos representaba lo contrario, las funciones estaban ya creadas, pero faltaba terminar la parte visual del componente.
- Verde: se habían implementado tanto los componentes a renderizar como las funciones necesarias para el tratamiento de datos además había pasado las pruebas iniciales de funcionamiento.

En el **Apéndice II** se especifica con más detalle cómo ha sido la gestión del esfuerzo y de los riesgos encontrados en el proyecto. Los riesgos no se pueden evitar y aparecen independientemente del proyecto y de la metodología elegida.

A pesar de que actualmente, la mayoría de los proyectos de NTT Data, hacen uso de metodologías ágiles, ya que se depende de opiniones y validaciones de clientes, en nuestro caso se optó por hacer uso de una metodología en cascada mejorada. La decisión se debió a que se contaba con unos requisitos claros, que no iban a ser modificados. No se eligió un modelo de cascada clásico debido a que al ser totalmente secuencial y no permitir volver al paso anterior iba a ser prácticamente imposible poder cumplirlo.

1.2 - Estructura del documento

En lo que resta del documento la organización es la siguiente. En el **Capítulo 2** se resume la fase de análisis de la metodología de cascada mejorada dando a conocer el funcionamiento actual del sistema de *mentoring* en NTT Data y el estudio de las herramientas existentes actualmente. En el **Capítulo 3** se muestran tanto los requisitos funcionales como los requisitos no funcionales. Estos dos capítulos se corresponden con la fase de análisis de la metodología en cascada.

La fase de la cascada correspondiente al diseño está resumida en el **Capítulo 4**, en donde se muestra el funcionamiento que se quería que tuviera la plataforma junto a los diseños de tanto la interfaz, base de datos y servidor de *mailing*.

En el **Capítulo 5** se describe cómo fue la fase de la metodología de implementación de la cascada mejorada. Además de algunas decisiones tomadas para generar un proyecto con elementos que pudieran ser depurados de mejor manera.

Pasando al **Capítulo 6** se encontrará resumida la metodología que se siguió para probar el sistema y las comprobaciones realizadas.

Finalmente en el **Capítulo 7** se nombra lo que se ha aprendido durante el desarrollo del proyecto y cómo este podría ser mejorado y usado en la empresa.

Para facilitar la lectura de la memoria en los apéndices se encuentra información adicional sobre las secciones nombradas anteriormente, pueden ser obviados por el lector si la información no es precisada.

Adicionalmente también está disponible el documento **Anexos** donde se puede encontrar toda la información no incluida en la memoria al considerarse menos relevante o repetitiva. Por ejemplo, descripciones de casos de uso, prototipos de interfaces o pantallas finales.

2 - Investigación preliminar

2.1 - Sistema de *mentoring* actual

En NTT Data la gestión del sistema de *mentoring* es realizada por el departamento *People*. Se encargan de la asignación de mentores y *mentees*, de la realización de cursos formativos para mentores y de la evaluación de empleados, clasificándolos como aptos para realizar las labores de mentor.

La asignación se realiza a través de un Excel teniendo en cuenta que mentor y *mentee* no pertenezcan al mismo equipo y si es posible que tampoco pertenezcan al mismo departamento. Esta medida busca que el mentor actúe como guía y no como "jefe" del *mentee*. En cuanto a los cursos, se realizan mediante una plataforma interna de la empresa y la evaluación se lleva a cabo de manera similar una vez finalizada la fase formativa.

Por parte de los mentores y los *mentees* no existe ninguna herramienta marcada por la empresa para que puedan conversar, crear reuniones o conocer la opinión de la otra parte. Debido a esto cada pareja decidía el método que les resultaba más cómodo de utilizar, dependiendo también del nivel de amistad que existiera entre ambos. Por lo general las aplicaciones más usadas actualmente son Microsoft Teams, Microsoft Outlook o WhatsApp.

Fuera del sistema utilizado por la empresa no encontramos en el mercado muchas aplicaciones relacionadas con las mentorías. La mayoría de las plataformas existentes ofrecen sistemas de seguimiento de los empleados en un sentido laboral, viendo su avance en un proyecto o dentro de la compañía. Estos servicios no se adecuan al sistema a crear, ya que se busca ayudar a los *mentees* en su integración y desarrollo en los primeros años de la empresa. Se buscaron sistemas que se centraran en el concepto de mentorías o coaching personalizado. En este sentido destacó mentorcliQ, herramienta usada por empresas para la creación de parejas de trabajadores y su evolución conjunta. Otra herramienta de mentoría encontrada fue BetterUp la cual dista más de lo que se busca conseguir con la creación de la plataforma. Esta aplicación ofrece auditorías externas para conocer y evaluar la satisfacción de empleados y ayudarlos para mejorar su rendimiento. Al ser de carácter externo no se ajusta a los requisitos que se pretenden, pero debido al componente de evaluación que se quiere añadir se analizará también.

Debido a que ambas herramientas son de pago y no han podido ser probadas directamente se realizará la comparación junto a la plataforma a desarrollar de las características y servicios que ofrecen desde sus páginas web, si alguna característica no se conoce se marcará con "---".

Características	NTT Data	mentorcliQ	BetterUp
Administrador o Gestor			
Se pueden añadir nuevos empleados a la plataforma	Sí	Sí	Sí
Los administradores de plataforma crean parejas de mentor-*mentee*	Sí	Algoritmos	Mentores externos
Los administradores reciben información relevante de los usuarios y problemas	Sí	Estadísticas	Opinión externa
Los administradores pueden realizar acciones para solucionar incidencias	Sí	Sí	Sí
Se pueden añadir bibliotecas de recursos y guías	Foro de usuarios	Sí	No
Los administradores puedes conocer estadísticas para una pareja concreta	Sí	---	Informes por *mentee*
Usuarios			
Los usuarios reciben retroalimentación sobre su evolución en la empresa	Sí	Sí	---
Los usuarios poseen de forma centralizada las funciones de hablar y quedar con su mentor o *mentee* asignado	Sí	Sí	No, son mentores externos
Los usuarios obtienen retroalimentación desde un punto de vista externo y neutro	No	No	Sí
Los usuarios pueden realizar videollamadas dentro de la plataforma	No	Sí	Sí
Los mentores pueden proporcionar material a los *mentees*	Pizarra o Foro	Sí	Sí

Tabla 1: Comparación plataforma propia vs software externos

Se puede observar como la plataforma mentorcliQ ofrece un sistema bastante similar al que se busca desarrollar dentro de NTT Data, pero con algunas diferencias. MentorcliQ busca la evolución del empleado dentro de la empresa estando más orientada al aprendizaje de un *mentee* que recibe ayuda de un mentor que actúa como "profesor" de este *mentee*.

Al comparar la plataforma a desarrollar con el servicio que ofrece BetterUp se puede observar cómo sigue un enfoque totalmente distinto. A pesar de ello sí poseen características comunes, al tratar ambos de mentorías, pero la mayoría de las cualidades son distintas. Esta aplicación ofrece un mentor externo que realiza mentorías con los empleados. Siendo este mentor el que obtiene y analiza los datos sobre la satisfacción y rendimiento de los trabajadores buscando potenciar estos aspectos.

Se puede concluir entonces que no existe ninguna plataforma externa que ofrezca de forma directa los rasgos que NTT Data quiere tener en sus mentorías. El *mentoring* tiene como objetivo ofrecer apoyo al empleado durante su integración en la empresa desde una posición externa a su departamento. Este factor es importante ya que el mentor es un guía y no un profesor. Pero es necesario que forme parte de la empresa para poder aconsejar y enseñar los valores de esta, otra de las bases fundamentales del *mentoring*.

2.2 - Análisis de debilidades y puntos de mejora

Tras realizar un análisis de los productos existentes en el mercado y entender los motivos por los cuales la empresa no contaba con un software de carácter externo se analizaron las debilidades que poseía el método usado actualmente. Con ellos se obtuvieron los puntos de mejora que la nueva plataforma debía cubrir.

Como se ha nombrado anteriormente, hoy en día la empresa gestiona la asignación de mentores y *mentees* mediante un Excel. Una vez se designa al mentor se desconoce cómo está funcionando el tándem por falta de comunicación con el mismo. Solo se conoce cuando se recibe algún correo electrónico procedente de uno de los empleados. Tampoco se conoce con detalle cómo se está llevando a cabo el proyecto de *mentoring*.

La debilidad más grande que se encuentra por parte de los miembros del proyecto *mentoring* es que no se posee una herramienta real para cubrir las necesidades que pueden tener mentores y *mentees*. Esto causa que pueden existir dudas que no sean compartidas ni resueltas de manera directa.

Otra debilidad que se halló en los aspectos relacionados con los usuarios es que el plan *mentoring* no tiene una finalidad fija para los mentores. O sea, se podría dar el caso en el que los que se sientan perdidos sobre cómo ayudar a sus *mentees* asignados sean los mentores a causa de la duración de las mentorías. O por el contrario, los *mentees* no llegan a tener claro en donde deberían centrar sus esfuerzos y qué aspectos deberían conocer dentro de la empresa.

Al tener claras las debilidades se conocía lo que se debía mejorar dentro del sistema de mentorías. Se creó un documento inicial para ser mostrado a los responsables del equipo, Vanessa y Manuel, de manera que se consiguió una primera validación de la idea preliminar de la aplicación a crear. Este primer archivo se puede consultar en el siguiente enlace (se añadieron posteriormente los requisitos, aunque en esta fase inicial no estaban incluidos) o en el **Anexo A**.

En este documento inicial se definió el nombre de la plataforma, que sería MENTality. También se buscó crear sentimiento de compañerismo entre los integrantes de la pareja. Por ello se decidió asignarle el nombre de tándem a una pareja mentor-*mentee*, ya que ambos son necesarios para que avance dentro del sistema de *mentoring.*

2.3 - Análisis de las tecnologías

En este apartado se van a analizar las principales tecnologías que se tuvieron en cuenta antes de comenzar con el desarrollo de la plataforma considerando aspectos como el coste monetario, el tiempo de implementación y la facilidad de aprendizaje.

Las tecnologías finales usadas durante la creación de la aplicación fueron:

- Navegador para la ejecución: Chrome actualizado a última versión.
- Programación:
    - Entorno de desarrollo: Visual Studio Code con extensiones varias para mejorar la programación en React.
    - Base de datos: FireStore y Storage de FireBase.
    - Servidor local para envío de correos: Node.js con NodeMailer.
    - Lenguajes: React v.18.3.1 con JavaScript XML, JavaScript y CSS.
    - Librería de UI: Material UI.
- Librerías relevantes usadas durante el desarrollo:
    - Enrutamiento y redirecciones: React-Router-Dom [22], elegida por ser una de las más grandes y ofrecer una versión actualizada y estable de sus funciones compatible con la versión de React usada.
    - Notificaciones: React Toastify [25], librería usada para ofrecer avisos de forma personalizable por la plataforma consiguiendo mostrar resultados de acciones a los usuarios de forma agradable.
    - Uso de fechas: dayjs [23], a pesar de que Java posee la clase *Date* se decidió una librería extra para obtener funciones concretas que resultaban útiles para la interacción con los *Calendar* de MUI[2].
    - Elección de color: *rc-component* [17], Material UI posee un componente con un *color-picker,* pero este no es compatible con la versión de React usada por tanto se buscó una librería que ofreciera el componente y fuera estable en la versión 18.3.1.
- Herramientas de producción documental
    - Word 10 Enterprise.
    - Google Draw.
    - Visual Paradigm.
    - Figma.
    - WebSequenceDiagrams.

[2] Abreviatura oficial de Material UI

La primera tecnología que se decidió fue el entorno de desarrollo a usar. El único requisito era que tuviera soporte para React, debido a que era el lenguaje de programación fijado por parte de la empresa, se buscó un entorno que permitiera realizar un análisis de sintaxis y un precompilador.

Teniendo en cuenta todos estos puntos se eligió Visual Studio Code ya que ofrece ambas características. Además, permite la instalación de extensiones para cualquier lenguaje, ya que durante el desarrollo del proyecto se crearon archivos *.css*, *.js*, *.jsx,* por lo cual poseer analizadores de sintaxis propios de cada lenguaje y otras librerías como *Auto Close Tag* (librería que cierra automáticamente las etiquetas HTML) fueron útiles para evitar errores de sintaxis. Aparte de eso, hacer uso de GitHub, como controlador de versiones, es extremadamente sencillo debido a que ambos pertenecen a Microsoft.

2.3.1 - Elección de Base de Datos

Al comienzo del proyecto, para las pruebas, se utilizaron simplemente archivos Json, los cuales son rápidos de crear y más que suficiente para los ensayos iniciales. Los archivos Json locales se usaron para probar funcionalidades básicas del sistema usando Material UI, librería que no se había usado antes. Rápidamente llegaron problemas debidos a que la edición de archivos Json desde el frontend generado en React no es del todo sencilla, al no poder hacerse directamente. Es necesario tener un servidor que actúe como base de datos o como backend para modificar los ficheros Json locales.

Cuando se vio este problema se buscaron soluciones posibles, las cuales se podrían separar en soluciones desarrolladas en local o soluciones en la nube.

En local las opciones eran dos, crear un servidor de backend con Node.js o implementar una base de datos interna de tipo clave-valor para mantener la estructura usada en los Json creados previamente.

- Backend con Node.js: permitiría hacer uso de JavaScript para gestionar los datos que se podrían mantener en el formato Json. Se debía diseñar y programar el backend desde cero, lo cual permite una personalización total, pero requiere un mayor esfuerzo de configuración inicial.
- Base de datos local con Redis: haría uso de un sistema clave-valor que sería similar a como estaban los datos organizados en los archivos Json. Ofrece un sistema de consultas muy veloz pero no es bueno con datos complejos los cuales podían llegar a existir dentro de nuestro sistema (objetos de *arrays* que a su vez poseen otros objetos o *arrays*).

Las opciones que surgieron haciendo uso de la nube y que fueron estudiadas con más detalle fueron CMS Headless como *Contentful* y FireBase.

- CMS Headless: funciona definiendo tipos de contenidos que representan las estructuras de los datos, estos son ofrecidos al frontend mediante peticiones APIs y son devueltos en formato Json.

- FireBase: es una base de datos en la nube de Google de tipo NoSQL. Tiene integradas herramientas para la autenticación de los usuarios que pueden leer y escribir datos, además de permitir guardar datos de tipo binario (imágenes) y/o datos clave-valor.

En la elección se priorizó el tiempo necesario para su configuración. Esto causó el descarte de las soluciones a nivel local, las opciones locales obligaban a hacer un desarrollo completo de una base de datos o sistema de lectura-escritura de archivos. Permitían una población más inmediata de los datos, pero el tiempo de creación de la estructura que los soportaría iba a ser mayor. Teniendo en cuenta que el prototipo debe ser posteriormente aprobado por el departamento *People* para su uso dentro de NTT Data se consideró que el gasto de horas extra no iba a ser tan positivo como la inversión de estas horas en crear una plataforma con más funcionalidades.

Entre las dos opciones en la nube que ofrecen un backend ya establecido se descartó la opción de CMS Headless. Aunque el backend y almacenamiento de los datos es ofrecido por su sistema es necesario crear la API con la que interactuará el frontend. Esto requiere de una configuración de las relaciones y consultas con una curva de aprendizaje pronunciada.

Finalmente, se eligió FireBase. Este servicio permitía mantener la estructura de datos que se creó en los ficheros Json incluyendo las estructuras complejas gracias a los tipos de datos que soporta. Además, la escritura y lectura de sus datos se puede hacer de forma sencilla desde React, instalando su librería y creando el archivo de configuración *firebaseConfig.js*. Este se puede programar fácilmente gracias a la documentación ofrecida por la página web de FireBase o en múltiples tutoriales existentes.

2.3.2 - Elección del sistema de envío de emails

La plataforma creada hace uso de los correos electrónicos para comunicarse con mentores y *mentees* por tanto era una función que se iba a usar en múltiples puntos del sistema. Al igual que con las bases de datos se encontraron soluciones locales y en la nube. Las herramientas halladas fueron principalmente la creación de un servidor local con NodeMailer, uso de servicios externos como EmailJS, uso de APIs específicas como Resend, uso de la extensión de FireBase Email Trigger.

- Servidor local con NodeMailer: ofrece control total sobre los correos a enviar haciendo uso de JavaScript para su configuración y realizando una petición al servidor desde el frontend.

- Servicios externos: ofrecen un servidor en la nube con capacidad de personalización mediante plantillas de los correos electrónicos a enviar desde la aplicación frontend. Son servicios de pago.

- APIs específicas para envío de correos: ofrecen una API que se puede integrar fácilmente en el código sin necesidad de crear un servidor de correo propio, aunque está orientada a ser añadida en el backend. Al igual que la opción anterior son de pago.

- Extensión Email Trigger: es una extensión de FireBase que permite el envío de correos cuando varía alguna colección o documento en la base de datos. Está sujeta a costos extra sobre la base de datos de FireBase.

Debido a que la plataforma inicialmente iba a ser usada para el desarrollo del TFG no se contaba con presupuesto. Se investigaron por tanto las características gratuitas de las soluciones de pago.

En los servicios externos y APIs específicas, el número de correos que se podía enviar era en general de unos cien al mes en la mayoría de las opciones revisadas. El límite de estos servicios no llegaba a ser óptimo ya que, aunque se preveía que no se enviarían tantos correos, en las pruebas realizadas no se podía saber a ciencia cierta el número enviado. Esto podría causar que se implementara un sistema que tendría que ser cambiado durante el desarrollo. Teniendo todo esto en cuenta ambas opciones se descartaron.

La extensión de FireBase era muy adecuada para la mayoría de los correos que se envían en el sistema, ya que están relacionados con nuevas reuniones y usuarios, lo cual implica cambios en la base de datos. Sin embargo, no permite que el administrador envíe correos electrónicos personalizados a mentores y *mentees* cuando se considerara necesario. Estas acciones no provocarían cambios en la base de datos que dispararan el *Trigger*, por este motivo la opción fue descartada.

Se eligió, por tanto, la solución de crear un servidor local. A pesar de que se optó por la opción que más tiempo iba a requerir era la que mejor se adaptaba a cómo iban a ser utilizados los correos electrónicos en la plataforma diseñada. Se encontraron además buenos tutoriales, referenciados en [12], haciendo su programación más liviana lo que fue un punto a favor de la decisión tomada.

2.3.3 - Elección de librería de UI

Debido a que se buscaba generar una imagen de plataforma concreta se debían diseñar componentes comunes entre las distintas pantallas que el usuario visita durante su navegación. De esta manera un empleado podría saber que un componente pertenece a la plataforma. A su vez se buscó utilizar los colores característicos de NTT Data, los cuales se muestran en el logo corporativo, dentro de la aplicación.

Con esta premisa aparecieron tres opciones claras a usar. La primera era crear componentes propios haciendo uso de los componentes básicos de HTML y los estilos de CSS. La segunda opción era el uso de una librería de componentes UI como Material UI o de un *framework* CSS como TailwindCSS.

- Componentes propios: inicialmente es la opción que más tiempo requiere ya que necesita crear componentes complejos haciendo uso únicamente de los componentes básicos de HTML. Posee un mantenimiento más costoso, aunque a largo plazo permite que la empresa haga uso de los componentes diseñados sin necesidad de licencias.

- Material UI: ofrece una librería de componentes amplia y estable al ser probada en profundidad antes de ser lanzados nuevos componentes. Esto se puede ver reflejado en su versión MUI/Lab [16] donde ofrecen componentes que aún no han pasado todas sus pruebas de funcionamiento, aunque ya son bastante estables. Posee una licencia MIT por lo cual debería ser cambiada antes de su integración como herramienta en NTT Data.

- TailwindCSS: es un *framework* CSS que permite crear interfaces con clases de bajo nivel, no ofrece directamente los componentes sino las herramientas para aplicarles estilos. Su licencia es MIT por lo cual no podría ser usado en la compañía sin referenciar su uso.

La decisión fue consultada con Ángel, el tutor dentro de la empresa. Inicialmente se pensó hacer uso de componentes propios, pero por recomendación suya se optó por hacer uso de Material UI o TailwindCSS. Esta sugerencia se debió a que ayudaría a crear una identidad para la plataforma. Posteriormente cuando el proyecto se presentara al departamento *People* se comentaría la necesidad de cambiarlo o incluir el aviso de copyright correspondiente.

Por tanto, entre hacer uso de una librería de componentes (MUI) o un *framework* de CSS (TailwindCSS), se optó por la primera opción. Debido a que los componentes proporcionados son personalizables en su totalidad gracias a la API que poseen, pero incluyen características predefinidas que hacen que su integración sea más rápida. La principal desventaja que tienen es que su tamaño es mayor que un *framework*, pero los componentes personalizados que ofrece evitan tener que crearlos nuevamente (se evita reinventar la rueda) o tener que buscar otras librerías con componentes similares (las cuales no son tan seguras como Material UI y poseen más fallos).

3 - Requisitos

Una vez realizada la investigación preliminar y contando con las necesidades de la empresa y con las características ofrecidas por la competencia se creó la lista de requisitos funcionales (RF) y requisitos no funcionales (RNF). Estos se basaron en el documento preliminar aprobado previamente que se encuentra en el **Anexo A**.

La plataforma MENTality implementará un sistema que optimice y fortalezca la experiencia de mentoría; facilitando la comunicación, la planificación y el seguimiento de hitos clave en la relación mentor y *mentee* (tándem). A través de esta plataforma, se busca fomentar un ambiente de crecimiento y colaboración, donde los tándems trabajen juntos para alcanzar metas y desarrollar una mentalidad de éxito.

Requisitos generales:

RF-1: Existen 3 tipos de usuarios: administrador, mentor y *mentee*.

RF-2: El sistema permite iniciar sesión haciendo uso de las credenciales corporativas*.

RF-3: El sistema permite acceder a OneDesk para tener asistencia técnica**.

RF-4: Se permite a los usuarios cerrar sesión para borrar los datos de sesión.

RF-5: Un usuario que no ha cerrado sesión puede volver a entrar a MENTality sin necesidad de volver a introducir sus credenciales, siempre que los datos de sesión no hayan caducado.

Requisitos para el administrador:

RF-6: El sistema permite al administrador registrar usuarios.

RF-6.1: El sistema no permitirá crear usuarios cuyo correo electrónico, nombre de usuario o número de empleado ya existan en la base de datos.

RF-7: El sistema permite al administrador modificar los datos de un usuario ya registrado.

RF-7.1: El sistema no permitirá al administrador modificar el rol (mentor o *mentee*) de un usuario que ya esté asignado a un tándem.

RF-8: El administrador puede eliminar un usuario

RF-8.1: El administrador no podrá eliminar un usuario que pertenezca a un tándem

RF-9: El sistema permite al administrador crear tándems.

RF-9.1: El sistema permitirá que un mentor pueda estar asignado a varios tándems activos al mismo tiempo.

RF-9.2: El sistema no permitirá que un *mentee* pueda pertenecer a varios tándems.

RF-10: El sistema permite al administrador consultar las estadísticas de los tándems.

RF-11: El sistema permite al administrador ver el número de encuentros generales por día.

RF-12: El sistema permite al administrador conocer las incidencias que tiene un tándem.

RF-12.1: Existirán dos tipos de incidencias: por falta de reuniones y por solicitud de un cambio de mentor.

RF-12.2: El sistema permite al administrador ponerse en contacto con el mentor cuando alguno de sus tándems tenga incidencias de cualquier tipo.

RF-12.3: El sistema permite al administrador separar un tándem que haya solicitado cambio de mentor si se considera necesario.

RF-13: El administrador puede ver los usuarios existentes en la aplicación.

RF-13.1: El sistema permite al administrador filtrar los usuarios por si están asignados, por su rol de MENTality (mentor, *mentee*), nombre, apellidos o nombre completo.

RF-14: El administrador puede ver los tándems creados en la aplicación.

RF-14.1: El sistema permite al administrador filtrar los tándems con incidencias en general, por sus miembros o por un tipo concreto de incidencia.

RF-15: El sistema permite al administrador ver las insignias ganadas por cada tándem.

RF-16: El sistema permite al administrador ver el histórico de un tándem concreto.

RF-16.1: El sistema almacena reuniones de doce meses en el pasado.

RF-16.2: El sistema permite solo ver reuniones creadas como máximo dentro de dos meses.

RF-17: El sistema permite al administrador ver los hitos a cumplir por los tándems.

RF-17.1: El sistema muestra también las insignias ligadas a los hitos de la plataforma.

RF-18: El administrador puede crear nuevos hitos en la plataforma.

RF-18.1: El sistema no permitirá crear hitos que tengan el mismo nombre, descripción o color que otro hito previamente creado.

RF-18.2: El sistema no permite crear hitos cuya imagen de insignia asociada exista previamente en la base de datos.

RF-18.3: El sistema permite al administrador crear las encuestas de finalización de hitos.

RF-19: El sistema permite modificar un hito creado en la plataforma.

RF-19.1: El sistema no permite modificar el nombre o la descripción de un hito que ha sido completado por algún tándem de la plataforma.

RF-19.2: El sistema no permite modificar la encuesta de finalización de hito si algún tándem ya ha contestado a la encuesta.

RF-20: El administrador puede acceder al foro general de la aplicación.

RF-20.1: El sistema permite al administrador publicar mensajes en el foro general.

Requisitos para los mentores y los *mentees*:

RF-21: El sistema permite al usuario desplegar el modal de notificaciones desde cualquier pantalla.

RF-22: El sistema permite al usuario ver su(s) compañero(s) de tándem(s).

RF-23: El usuario puede ver los hitos completados junto a su(s) tándem(s).

RF-24: El sistema permite al usuario acceder a su perfil.

RF-24.1: El usuario puede consultar sus datos personales y laborales desde su perfil.

RF-24.2: El usuario puede ver las insignias ganadas durante su pertenencia a MENTality.

RF-25: El sistema permite al usuario acceder al perfil de su otro miembro del tándem.

RF-25.1: El usuario puede conocer los datos personales y laborales de su compañero de tándem.

RF-25.2: El usuario puede ver las insignias que ha ganado su compañero de tándem.

RF-26: El sistema permite al usuario ver el calendario de reuniones para un mes.

RF-26.1: El usuario puede acceder a su histórico de reuniones de doce meses en el pasado.

RF-26.2: El usuario puede consultar las reuniones que tendrá en los próximos dos meses.

RF-27: El sistema permite al usuario crear reuniones junto a su(s) tándem(s).

RF-27.1: El sistema permitirá crear reuniones con un margen mínimo de una hora desde la hora en la cual se intente crear la reunión y un máximo de dos meses en el futuro.

RF-28: El usuario podrá acceder al listado de encuestas de reuniones.

RF-28.1: El sistema solo mostrará las reuniones que ya hayan o deberían haber tenido lugar.

RF-29: El usuario podrá responder las encuestas de reuniones y ver las respuestas dadas por él y su compañero de tándem.

RF-29.1: El sistema no permitirá responder una encuesta que ya haya sido respondida y guardada en la base de datos.

RF-29.2: El sistema permitirá al usuario visualizar las respuestas únicamente cuando hayan sido respondidas por el mismo.

RF-30: El sistema permite al usuario ver la lista de hitos de la plataforma.

RF-30.1: El sistema muestra claramente una diferenciación entre los hitos no logrados, los hitos en proceso y los hitos completados.

RF-31: El usuario puede marcar los hitos que considera completados, teniendo en cuenta el sistema de doble check***.

RF-31.1: El sistema no permitirá marcar como no completado o marcado por el usuario un hito que previamente estaba marcado por el usuario o marcado por ambos usuarios (completado) respectivamente.

RF-32: El usuario podrá acceder a las encuestas de finalización de hitos de los hitos que estén marcados como completados.

RF-32.1: El sistema permite al usuario rellenar y modificar las respuestas de la encuesta de finalización de hito.

RF-32.2: El sistema muestra las respuestas dadas por el otro miembro del tándem.

RF-33: El usuario puede acceder a la pizarra privada de tándem.

RF-33.1: El sistema permite al usuario publicar contenido en la pizarra privada.

RF-34: El sistema permite al usuario acceder al foro exclusivo de su rol (mentor, *mentee*).

RF-34.1: El usuario puede publicar mensajes en el foro exclusivo.

RF-35: El sistema permite al usuario acceder al foro general de MENTality.

RF-35.1: El usuario puede publicar mensajes en el foro general.

RF-36: El sistema permite al usuario acceder al perfil de otros miembros de la plataforma.

* Por privacidad de los datos de los usuarios los usados en la base de datos son inventados.

** OneDesk es la plataforma interna de asistencia técnica de la empresa. Tienen acceso a los datos de los empleados y serán los encargados de gestionar cambio de contraseña y problemas ligados al inicio de sesión.

*** El sistema de doble check busca que ambos miembros del tándem tengan responsabilidad a la hora de marcar su avance y progreso, por tanto, un hito tendrá 4 estados: no completado, marcado por *mentee*, marcado por mentor o completado.

RNF-1: El sistema debe cumplir con la normativa de protección de datos.

RNF-2: El sistema debe funcionar en los navegadores web modernos y actualizados.

RNF-3: La solución debe ser 100% Web y todas las operaciones deben realizarse desde el navegador.

RNF-4: El sistema debe disponer de conexión a internet.

RNF-5: El sistema debe tener capacidad para dar respuesta al acceso simultáneo de los usuarios registrados con un tiempo de respuesta aceptable incluso cuando el nivel de demanda sea alto.

RNF-6: El sistema debe contar con un sistema de identificación seguro.

RNF-7: El sistema debe ser de fácil uso y entendimiento por parte de los usuarios, con una curva de aprendizaje suave.

RNF-8: El sistema debe dar a conocer al usuario su ubicación dentro de la aplicación.

RNF-9: El sistema contará con una interfaz atractiva y fluida.

RNF-10: El sistema debe poseer mensajes de error personalizados para la identificación y localización de errores durante las fases de desarrollo, pruebas y de operación.

RNF-11: Los *backups* de la base de datos son responsabilidad de la administración de la plataforma que deberá crearlos, almacenarlos y recuperar la información si fuera necesario. Se podrán usar los *backups* ofrecidos por FireBase* si se quiere centralizar el control de la base de datos en dicha plataforma.

* La plataforma FireBase ofrece un sistema de recuperación de datos y *backups* con el plan Blaze o planes superiores que la empresa podría contratar si se considerara necesario o se quisiera implementar la versión final con FireBase.

Los requisitos funcionales fueron creados pensando en la continuidad del proyecto tras la finalización de este Trabajo de Fin de Grado. Por ello, algunos de los requisitos se plantearon a futuro ya que su programación podía requerir de más tiempo del disponible durante la estancia en la empresa. Debido a esto, los requisitos funcionales se sometieron a la técnica MoSCoW para priorizar aquellos requisitos indispensables de cara a la creación del producto mínimo viable. La aplicación de la técnica se detalla en este documento, **Apéndice III**.

4 - Análisis y diseño del sistema

Para realizar el análisis se hizo uso de la técnica “Análisis Orientado a Casos de Uso” [1]. Esta técnica se enfoca en identificar los casos de uso del sistema desarrollado, es decir, las interacciones que tendrán los distintos usuarios con el sistema para lograr objetivos específicos.

Utilizando este método de análisis nos centramos en satisfacer las necesidades específicas del usuario al estar orientado a estas. Además, esta técnica de análisis es compatible con el “Análisis Orientado a Objetos” (AOO[3]) lo cual nos permite utilizar el enfoque AOO en la parte ligada a la base de datos.

El “Análisis Orientado a Objetos” sólo se usará en la base de datos. El patrón que se usa en la plataforma es Modelo-Vista-Controlador. En el modelo, conformado por la base de datos, los JavaScript que interactúan con ella y el servidor de *mailing*, solo la base de datos posee objetos, denominados colecciones en FireBase. Por ello es el único elemento que poseerá un diagrama AOO.

La vista solo se encarga de mostrar los atributos que le proporciona el modelo y los controladores hacen uso de las funciones proporcionadas por los archivos JavaScript para interactuar con la base de datos.

4.1 - Análisis del sistema

Cuando se realizó esta fase se buscaba obtener más claridad en las funciones que el usuario desearía realizar y cuál es la forma más adecuada de presentar la solución a esta necesidad. Se examinaron las necesidades desde la perspectiva de los casos de uso que iban a existir en la plataforma a partir de los requisitos fijados anteriormente.

4.1.1 - Casos de uso

Los casos de uso son utilizados para describir cómo el sistema puede ser utilizado teniendo en cuenta el actor (usuario involucrado en el uso del sistema) que quiere realizar la acción. Un caso de uso (CU[4]) muestra la secuencia de eventos que el usuario efectúa para completar una operación concreta.

Los casos de uso son resumidos por los diagramas de casos de uso que muestran todas las funciones que puede realizar un determinado actor, teniendo en cuenta que estos son siempre usuarios externos al sistema. En nuestra plataforma existen los siguientes actores:

- Actor Admin: referido al administrador de la plataforma; es un actor registrado con rol de admin en la base de datos.
- Actor User: referido tanto a mentores como a *mentees* de la plataforma; es un actor registrado con rol de mentor o *mentee* en la base de datos.

[3] Abreviatura de Análisis Orientado a Objetos.
[4] Abreviatura de caso de uso.

Teniendo en cuenta estos actores podemos identificar los casos de uso que nuestra plataforma tendrá, obtenidos a raíz de los requisitos nombrados anteriormente en el **Capítulo 3**. Con ambos elementos podemos crear el diagrama de casos de uso. Por simplicidad en la comprensión se creó un diagrama por actor implicado en el sistema.

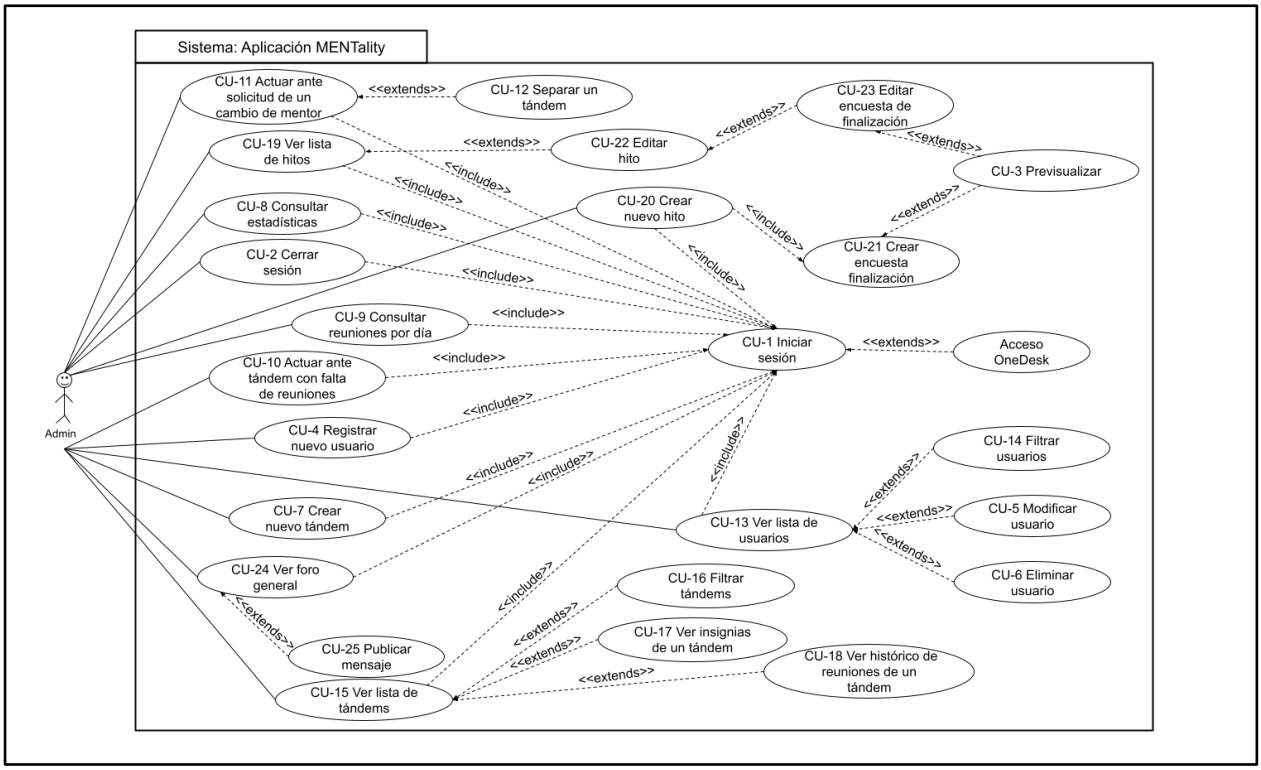

Ilustración 1: Diagrama de Casos de Uso del Administrador

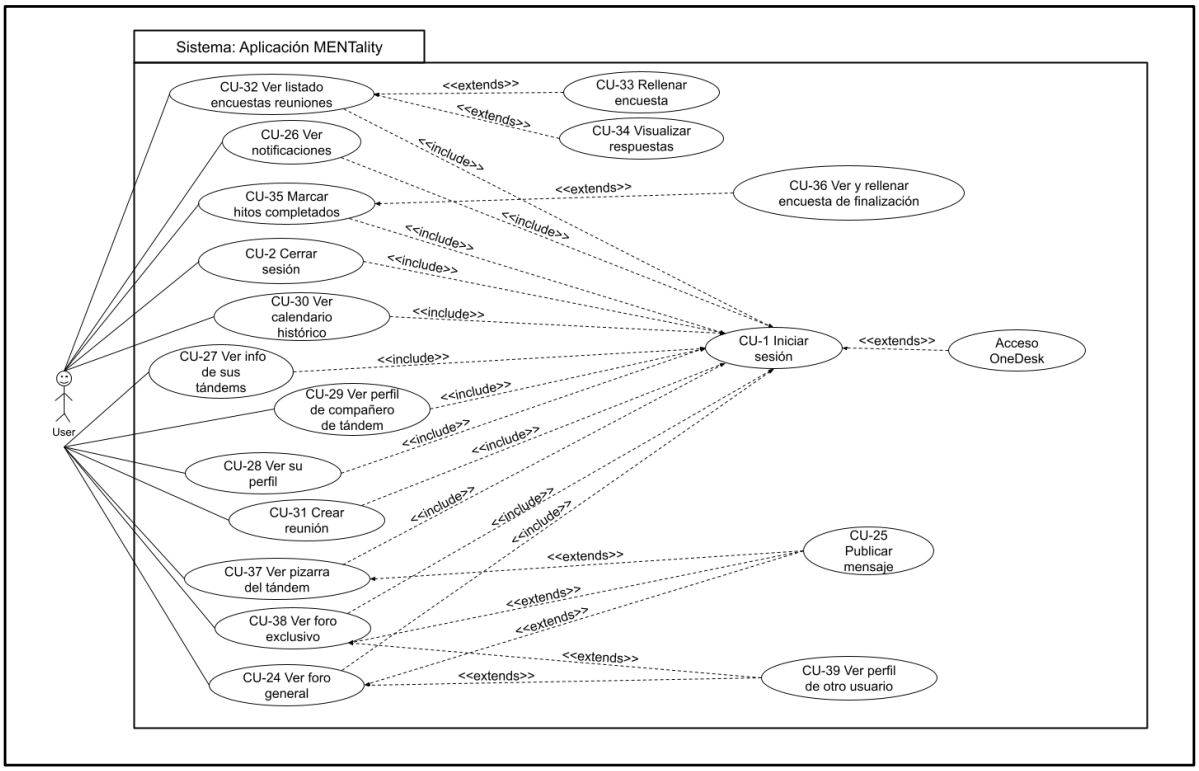

Ilustración 2: Diagrama de Casos de Uso del Mentor o Mentee

Cuando ambos diagramas de casos de uso estuvieron descritos se debía definir a alto nivel cada uno de los casos de uso. En ellos se debía explicar tanto el flujo de eventos principal como los flujos de eventos alternativos, si existen, especificando el actor que interactúa con el sistema en cada caso de uso explicado.

Debido a que existen 39 casos de uso en la aplicación y su descripción tiene la misma estructura en todos ellos, en la memoria se incluirán sólo dos casos de uso detallados. Los casos de uso seleccionados son el CU-7: "Crear nuevo tándem" y el CU-31: "Crear reunión", uno de cada actor del sistema. El resto podrán ser encontrados en el **Anexo B.3 - Casos de Uso**.

Caso de uso 7: Crear nuevo tándem
Actores: Admin
Descripción: el administrador quiere añadir un nuevo tándem a MENTality.
Flujo de eventos principal:

- El caso de uso comienza cuando Admin accede a la plataforma de MENTality.
- Desde la pantalla *Home* (ilustración 3) se pulsa el botón de iniciar sesión.
- Include: "Iniciar sesión" (ilustración 4).
    - Admin introduce sus credenciales en los campos de texto correspondientes.
    - Admin pulsa el botón "Iniciar sesión".
- El sistema muestra a Admin su página principal (ilustración 5).
- Admin accede a la pantalla de "Tándems" que contiene la información relacionada con los usuarios registrados y tándems (ilustración 6).
- Admin pulsa en el botón "Crear tándem".
- El sistema carga el formulario de creación de tándem (ilustración 7).
- Admin rellena los campos de mentor y *mentee* y pulsa en "Crear tándem".
- El sistema elimina al *mentee* de la lista de posibles *mentees* a asignar, guarda en la base de datos el nuevo tándem y finaliza el caso de uso.

Flujo de eventos alternativo:

Si las credenciales de inicio de sesión no son correctas se muestra a Admin un error y se le permite volver a intentarlo. Si no recuerda sus credenciales se realizaría un Extend a "Acceso OneDesk" para crear un ticket informando del problema.

Flujo de eventos alternativo:

Si Admin contaba con su sesión guardada en el navegador se saltaría el Include a "Iniciar sesión" pasando al siguiente paso directamente.

Flujo de eventos alternativo:

Si Admin pulsa cancelar mientras asigna el mentor o el *mentee* se vuelve a la pantalla "Tándems" (ilustración 6) sin realizar cambios en la base de datos.

Si Admin no rellena el mentor o *mentee* a formar parte del tándem se mostraría un mensaje de error indicando que ambos campos son obligatorios.

Caso de uso 31: Crear reunión
Nota: el *mentee* no está incluido en las ilustraciones porque estas no están en la memoria y solo se han añadido en el **Anexo B.3 – Casos de Uso**.
Actores: User
Descripción: un mentor o *mentee* quiere crear una reunión en su calendario.
Flujo de eventos principal:

- El caso de uso comienza cuando User accede a la plataforma de MENTality
- Desde la pantalla *Home* (ilustración 3) se pulsa el botón de iniciar sesión.
- Include: "Iniciar sesión" (ilustración 4).
    - User introduce sus credenciales en los campos de texto correspondientes.
    - User pulsa el botón "Iniciar sesión".
- El sistema muestra a User su página principal (ilustración 8 para mentor).
- User pulsa en el botón "Calendario" situado en la barra de navegación.
- El sistema mostraría el calendario del usuario junto al formulario de creación de reuniones (ilustración 9 en mentores).
- User rellena los campos del formulario y pulsa "Crear Reunión".
- El sistema muestra un Modal con la información de la reunión a crear y el resto de los eventos existentes en la misma fecha (ilustración 10 en mentores).
- User pulsa "Aceptar".
- El sistema registra la nueva reunión en la base de datos y finaliza el caso de uso.

Flujo de eventos alternativo:

Si las credenciales de inicio de sesión no son correctas se muestra a User un error y se le permite volver a intentarlo. Si no recuerda sus credenciales se realizaría un Extend a "Acceso OneDesk" para crear un ticket del problema.

Flujo de eventos alternativo:

Si User contaba con su sesión guardada en el navegador se saltaría el Include a "Iniciar sesión" pasando al siguiente paso directamente.

Flujo de eventos alternativo:

Si alguno de los campos del formulario no es rellenado por User el sistema muestra un error indicando el campo que falta por completar y no se guardaría la reunión (ilustración 11 si el rol de User es mentor).

Si User pulsa "Cancelar" el sistema no realiza cambios en la base de datos y vuelve a la pantalla de "Calendario" (ilustración 9 si User es un mentor) con los datos anteriores introducidos.

Si la fecha seleccionada es anterior a la fecha actual el sistema mostrará a User un error indicándole que la fecha no es válida (ilustración 12 en mentor).

Si la hora introducida es inexistente o el margen es inferior a una hora, el sistema mostrará un mensaje de error a User indicando que la hora no es válida (ilustración 12 para mentores).

4.1.2 - Diseño de interfaces

El modelo de casos de uso busca principalmente planear cómo será el sistema donde los usuarios van a tener un papel esencial, con ellos podemos crear las interfaces que usará la aplicación. El diseño de interfaces nos ayuda a ofrecer una visión más completa, apoyando las descripciones de los casos de uso, además actúan como base para prototipos iniciales o diagramas de secuencia.

El diseño de interfaces muestra visualmente cómo se presenta la información del sistema a los actores implicados y cómo el sistema recoge los datos que hayan sido introducidos por los actores. Al crear las distintas interfaces fue fundamental reflejar la visión lógica del sistema, consiguiendo solidez entre la imagen que el usuario forma del funcionamiento del sistema y el comportamiento real que este tiene.

Para realizar el diseño se han elaborado las distintas pantallas que debe presentar el sistema según los pasos de cada caso de uso. Se utilizó Visual Paradigm para la creación de todos estos prototipos.

A continuación, se muestran las pantallas relacionadas con el CU-7: "Crear nuevo tándem" y el CU-31: "Crear nueva reunión". Debido a que el CU-31 puede verse desde el punto de vista de mentor o *mentee* en la memoria solo se incluirá la interfaz del mentor. El resto de las interfaces están disponibles en el **Anexo C**.

Como hemos visto en el apartado anterior, el CU-7 parte de la pantalla Home (ilustración 3). Si se cumple el flujo de eventos principal llegamos a la pantalla Login (ilustración 4) para cumplir el *Include* del caso de uso.

Ilustración 3: Interfaz de Home común en todos los usuarios

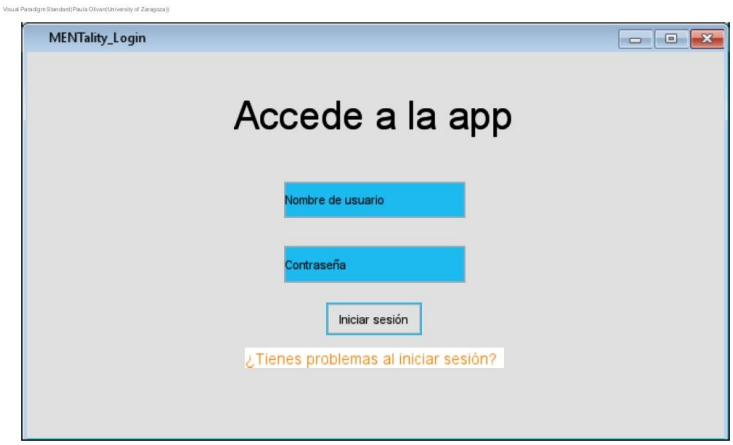

Ilustración 4: Interfaz de Login común en todos los usuarios

Si se inicia sesión de manera correcta, se llega a la pantalla principal del actor Admin (ilustración 5).

Ilustración 5: Interfaz Pantalla Principal de Admin

Admin pulsa el botón de "Tándems" pasando a mostrar la interfaz que lista los usuarios y tándems que forman parte de la aplicación (ilustración 6).

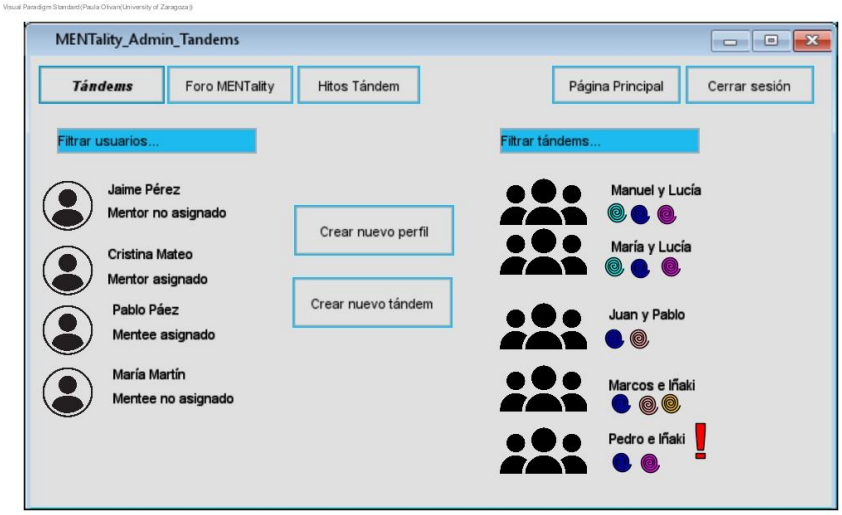

Ilustración 6: Interfaz del componente Tándems

Cuando Admin llegara a la interfaz anterior pulsaría el botón de “Crear nuevo tándem” y se mostrará el formulario de creación de nuevo tándem (ilustración 7).

Ilustración 7: Interfaz final del CU-7 Crear nuevo tándem

Admin realizaría las asignaciones necesarias y pulsaría “Crear tándems marcados” finalizando el flujo de eventos principal del caso de uso.

Vamos a realizar ahora el estudio de interfaces del CU-31: “Crear reunión”. Al igual que para el CU-7: “Crear nuevo tándem” se comienza con la interfaz *Home* (ilustración 3) y se continúa con la interfaz *Login* (ilustración 4). Una vez se ha iniciado sesión, en el escenario del CU-31 se muestra la página principal del mentor o *mentee* (ilustración 8).

Ilustración 8: Interfaz Pantalla Principal de Mentor

User pulsaría el botón “Calendario” y el sistema muestra la pantalla de su calendario con el formulario de creación de nuevas reuniones (ilustración 9).

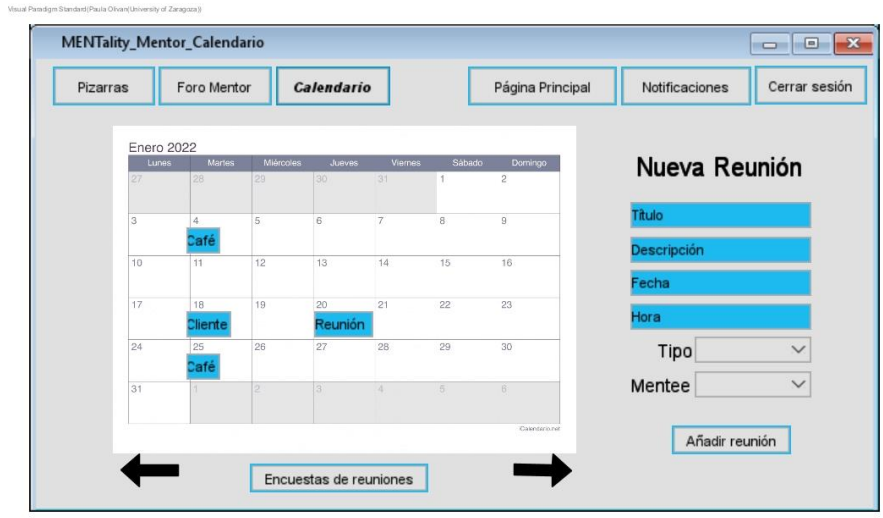

Ilustración 9: Interfaz del CU-31 Crear nueva reunión, formulario

User rellenaría el formulario y pulsaría “Añadir reunión”. En el flujo eventos principales el formulario estaría correctamente rellenado y se mostraría la interfaz de confirmación (ilustración 10).

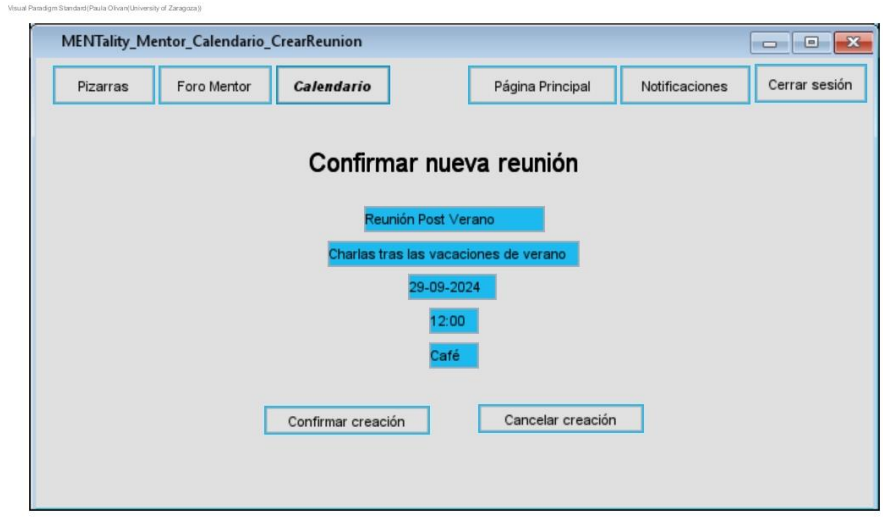

Ilustración 10: Interfaz Confirmación de nueva reunión

User confirmaría la creación de la reunión y esta se añadiría a su calendario personal. La reunión se añadiría también en el calendario del *mentee* implicado.

En caso de error en alguno de los pasos se llegaría a uno de los flujos de eventos alternativos. Si alguno de los campos no ha sido rellenado, se mostrará una interfaz de error por falta de campos (ilustración 11)

Ilustración 11: Interfaz Error por formulario no completo

El otro error que se puede originar es que la fecha marcada para la reunión ya haya pasado, que la hora no sea al menos una hora posterior a la hora de creación de la reunión o una hora inventada (ejemplo: 25:68). En este caso de error la interfaz que se muestra sería de error por fecha/hora incorrecta (ilustración 12).

Ilustración 12: Interfaz Error por fecha y/u hora incorrecta

4.1.3 - Diagramas de secuencia

Para describir cómo se transmite la información de entrada y salida de los casos de uso se han creado diagramas de secuencia. Los diagramas de secuencia muestran la interacción de los componentes del sistema y actores según la secuencia temporal de los eventos.

La plataforma MENTality implementa el patrón Modelo-Vista-Controlador, lo cual divide las responsabilidades en 3 capas.

- Modelo: está conformado por la lógica del sistema, el servidor local de envío de correos electrónicos y la base de datos FireBase. Se encarga de realizar las operaciones para mostrar los componentes y los datos en ellos.
- Vista: se encarga de generar la interfaz que el usuario verá y con la que interactuará.
- Controlador: comunica la vista con el modelo, lee los eventos que el usuario genera en la vista, generando en el modelo los cambios.

Para el diseño de los diagramas se hizo uso de la herramienta online WebSequenceDiagrams. En los diagramas se reflejó esta arquitectura, aunque se mostró el modelo dividido en sus 3 partes. Si en alguno de los casos de uso no se interactuaba con una de las partes estas eran omitidas.

En el CU-7: "Crear nuevo tándem", el diagrama de secuencia que lo describe es el siguiente (ilustración 13).

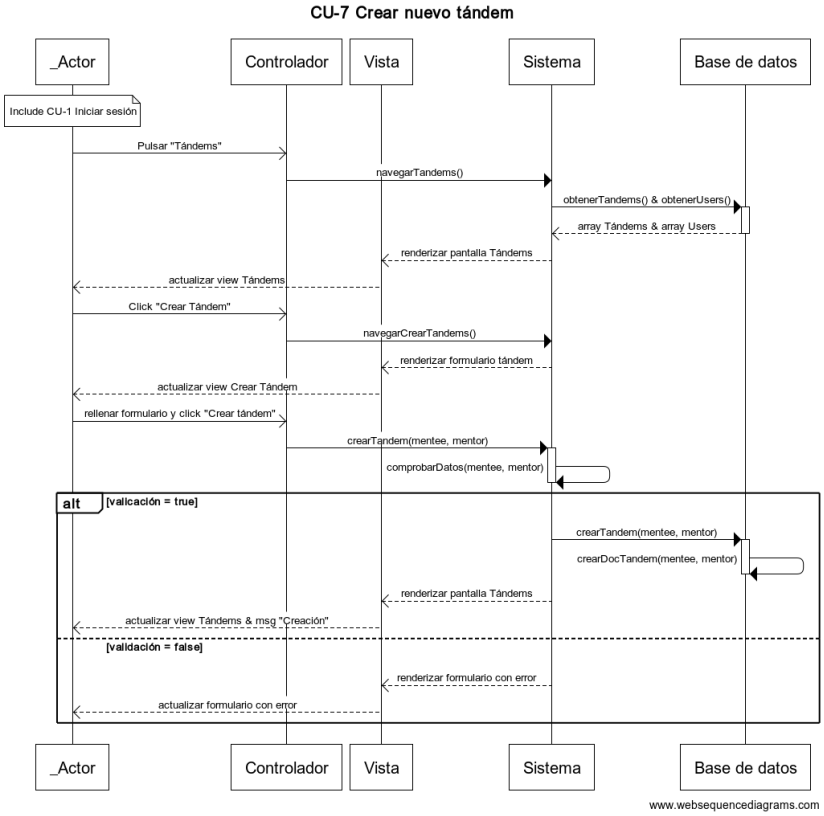

Ilustración 13: Diagrama de secuencia del CU-7 Crear nuevo tándem

El otro caso de uso añadido en la memoria es el CU- 31: “Crear reunión”. Este posee el siguiente diagrama de secuencia (ilustración 14).

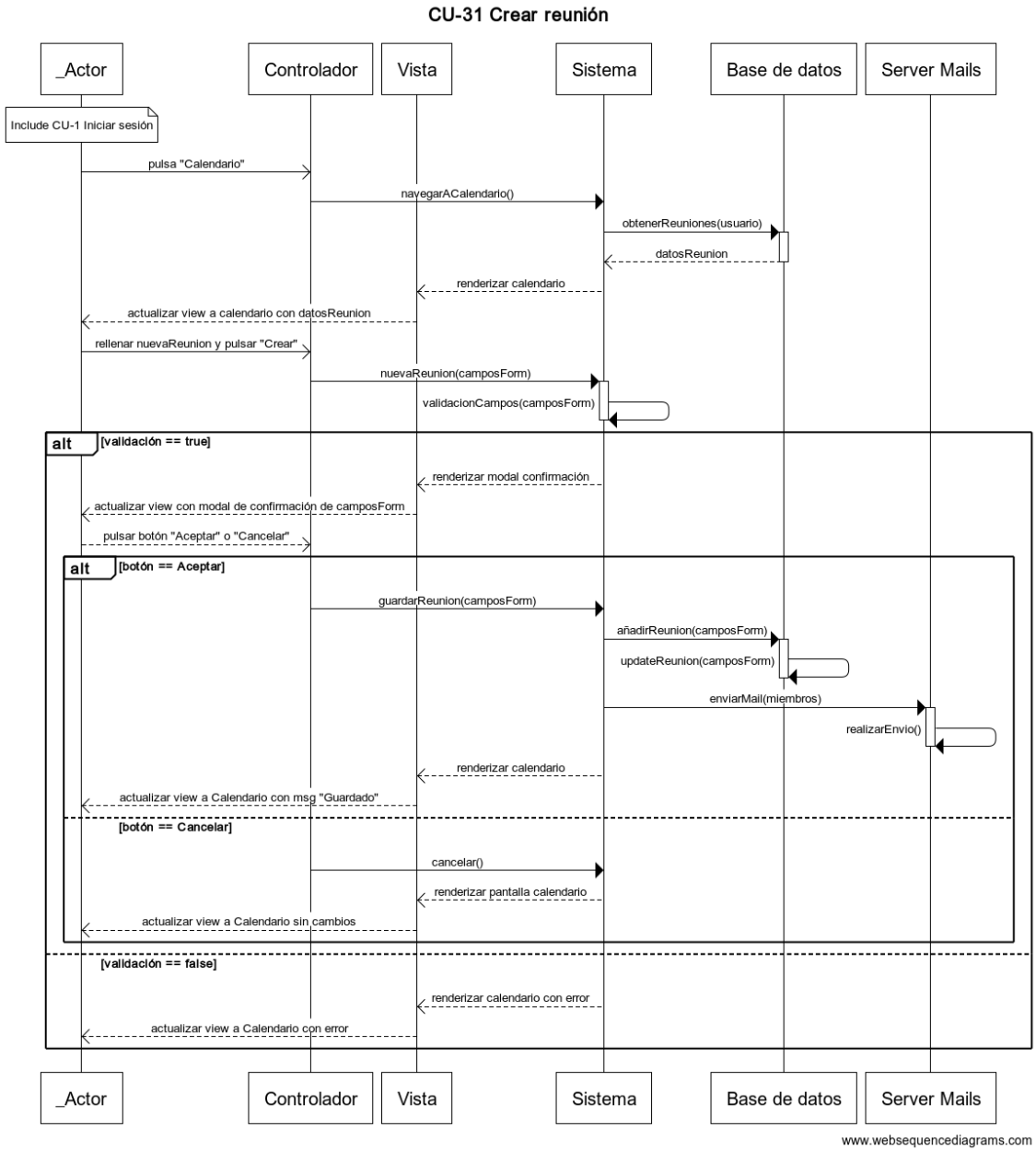

Ilustración 14: Diagrama de secuencia del CU-31 Crear reunión

El resto de los diagramas de secuencia han sido incluidos en el documento de **Anexos**, se pueden encontrar en el apartado **Anexo D**.

4.2 - Diseño del sistema

Tal y como se ha especificado en los requisitos (**Capítulo 3**), la aplicación MENTality permitirá mejorar el sistema de mentorías de la empresa, con seguimientos y centralización de herramientas. La aplicación funcionará en los navegadores web modernos.

Dado que MENTality cuenta con tres roles claramente diferenciados, se ha decidido implementar la aplicación con tres interfaces distintas. Aunque las interfaces estén marcadas por el rol del usuario que inicie sesión existirán elementos comunes que conformarán la imagen de MENTality.

En los siguientes apartados se explorarán las decisiones que se tomaron a la hora de diseñar el sistema para cada uno de los usuarios. También se verá cómo se diseñó la base de datos para guardar toda la información necesaria para el funcionamiento adecuado. La comunicación entre la aplicación y el servidor de *mailing* también será abordada.

Por último, se verá la arquitectura global del sistema que se ha desarrollado.

4.2.1 - Mapa de navegación

El mapa de navegación da una visión global de cómo será la navegación del usuario en la aplicación desarrollada. En la aplicación de MENTality se usó para facilitar la creación de la navegación entre pantallas, las redirecciones y el enrutador generado con la librería React-Router-Dom.

Para realizar el mapa de navegación se utilizaron los prototipos de interfaces, incluidos en el **Anexo C**. Con estas pantallas y utilizando la herramienta *Figma* se creó el mapa de navegación. Se unieron los “botones” de la pantalla con la interfaz que generará la interacción con el “botón” del prototipo de interfaz, de manera que quedó un esquema de cómo será la navegación.

Por simplicidad se creó un mapa de navegación por cada usuario, los cuales se agregan a continuación. Los mapas de navegación también están en este enlace para poder ser vistos con más detalle.

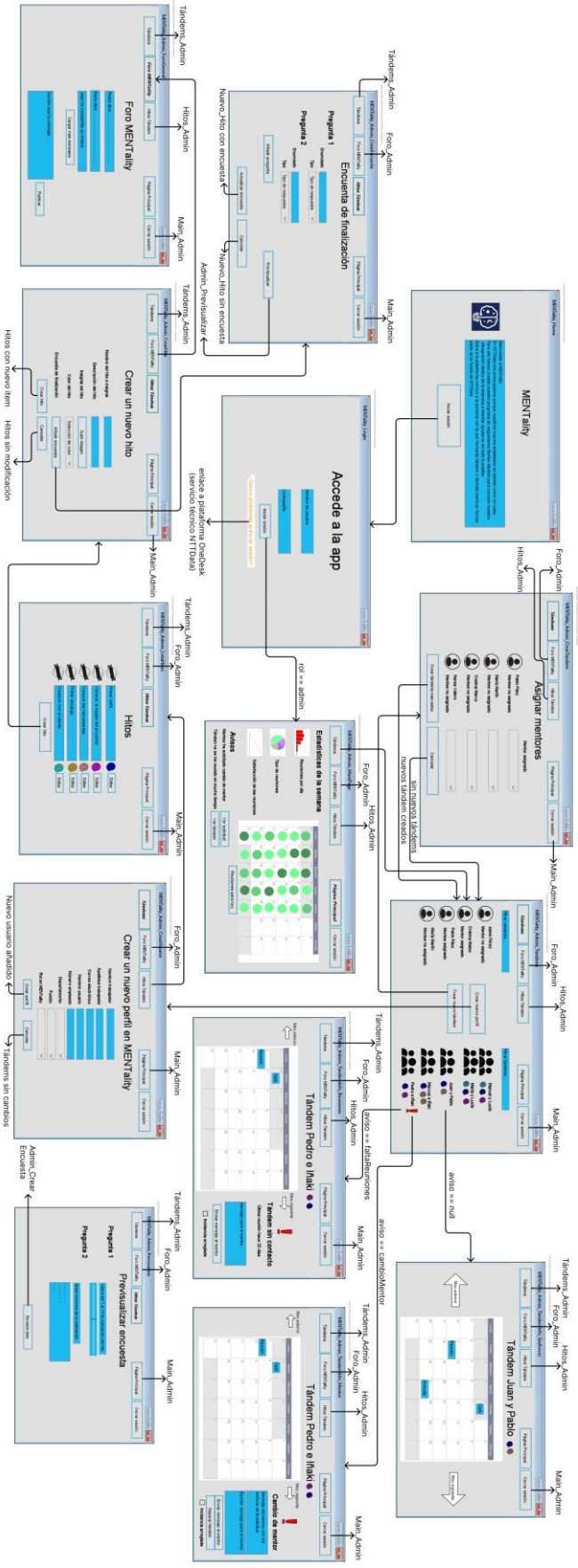

Ilustración 15: Mapa de navegación de un administrador

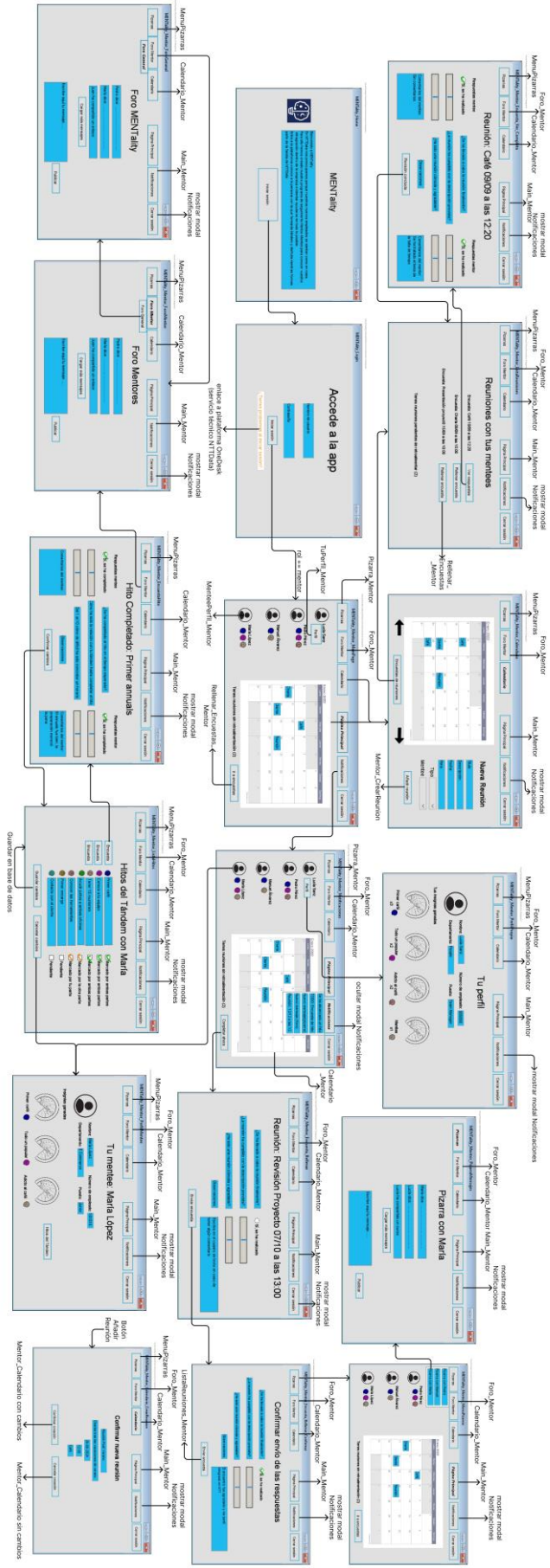

Ilustración 16: Mapa de navegación de un mentor

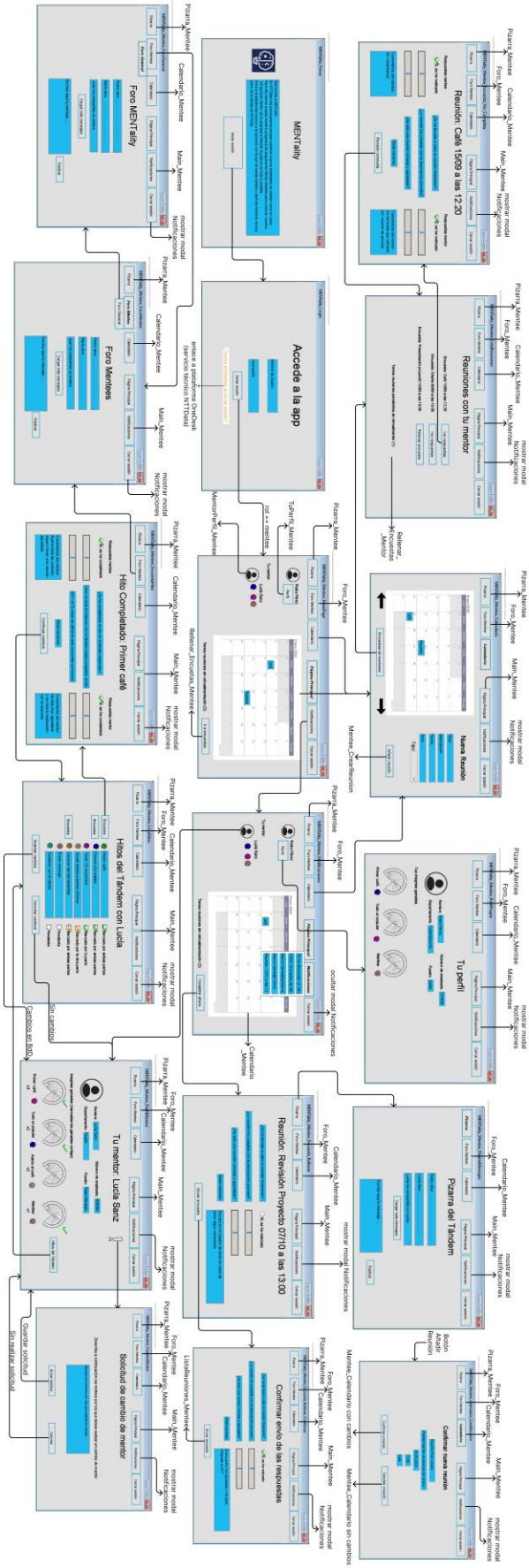

Ilustración 17: Mapa de navegación de un mentee

4.2.2 - Diseño de la base de datos

Como se ha nombrado al comienzo del **Capítulo 4**, en React no existen los objetos. Esto hace que no tenga sentido hacer un Análisis Orientado a Objetos de toda la plataforma. Por ello solo la base de datos se va a representar con esta técnica, aunque FireBase no tiene objetos, sino que son colecciones de documentos.

A continuación, se muestra la estructura que tiene la base de datos creada, como se puede ver las colecciones son independientes entre ellas. La única unión que tienen es con la estructura de los mapas que poseen sus documentos, buscando aclarar los campos de dichos objetos. La unión entre colecciones la hace el sistema con lógica interna mediante campos únicos que contienen los documentos, por ejemplo, el campo miembros de un documento de Tándems está formado los ids únicos de los usuarios que forman el tándem.

El diseño también está disponible en versión PDF en este enlace. Las tablas ampliadas de manera individual pueden ser analizadas en el **Anexo E: Diseño de la base de datos.**

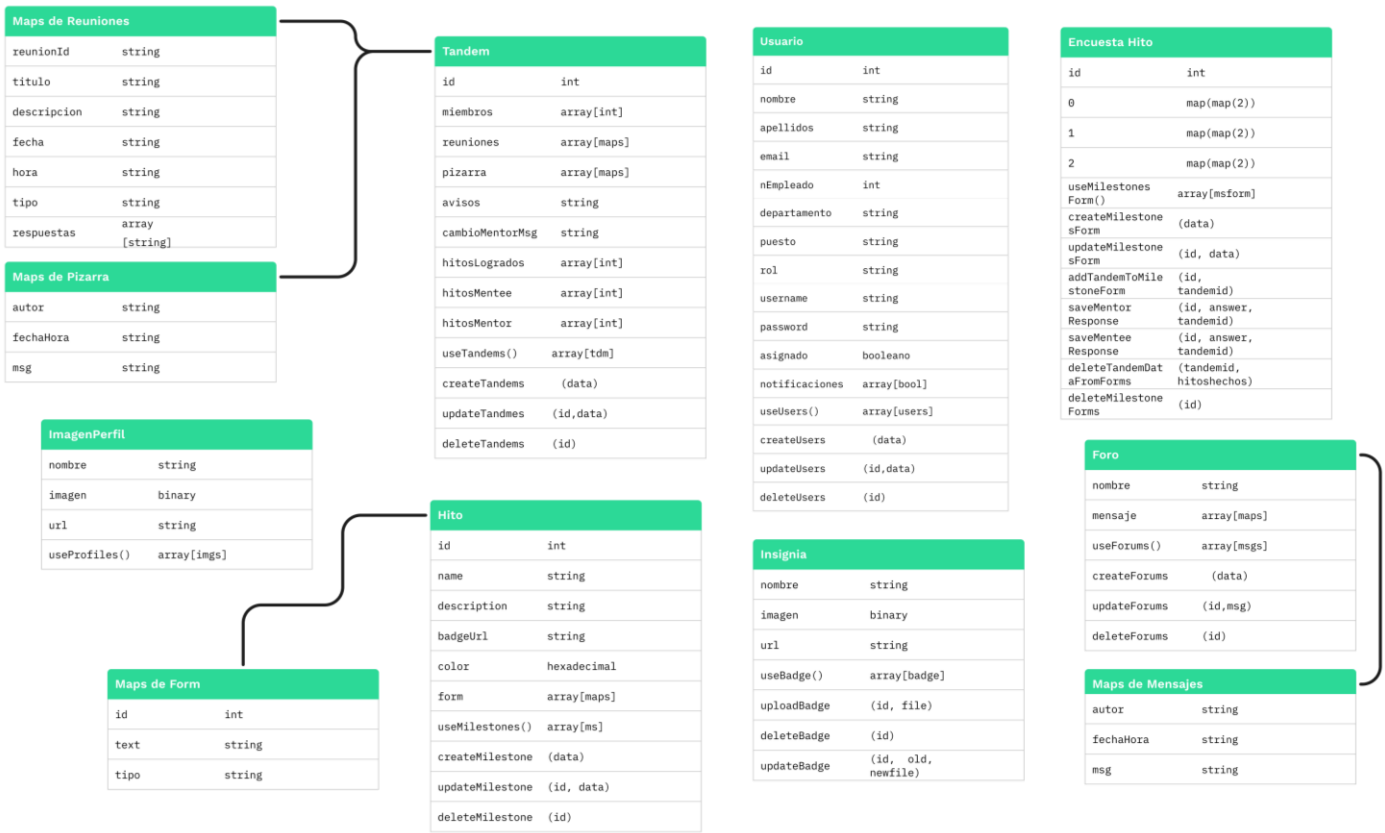

Ilustración 18: Diseño de las colecciones de la base de datos

4.2.3 – Diseño del servidor de mailing

Como se ha visto en los casos de uso (**Capítulo 4.1.1**), una de las características que ofrece la aplicación de MENTality es la posibilidad de facilitar la comunicación entre los distintos usuarios de la plataforma. Haciendo uso de un servidor para el envío de correos electrónicos, alojado en local.

El servidor será sencillo, funcionando como un servidor multicliente que escucha las peticiones de los usuarios de la plataforma. Los clientes le realizan una solicitud de tipo POST junto con la función que se busca ejecutar y sus parámetros. El servidor recibirá la petición y según la función introducida por el usuario debe crear y enviar un correo distinto. La estructura de los *emails* no será igual para todos los casos, lo cual implica distinto número y clase de atributos para generar los correos de manera adecuada.

La estructura del servidor de *mailing* se puede observar en el siguiente diagrama, también se puede ver con más detalle en este enlace.

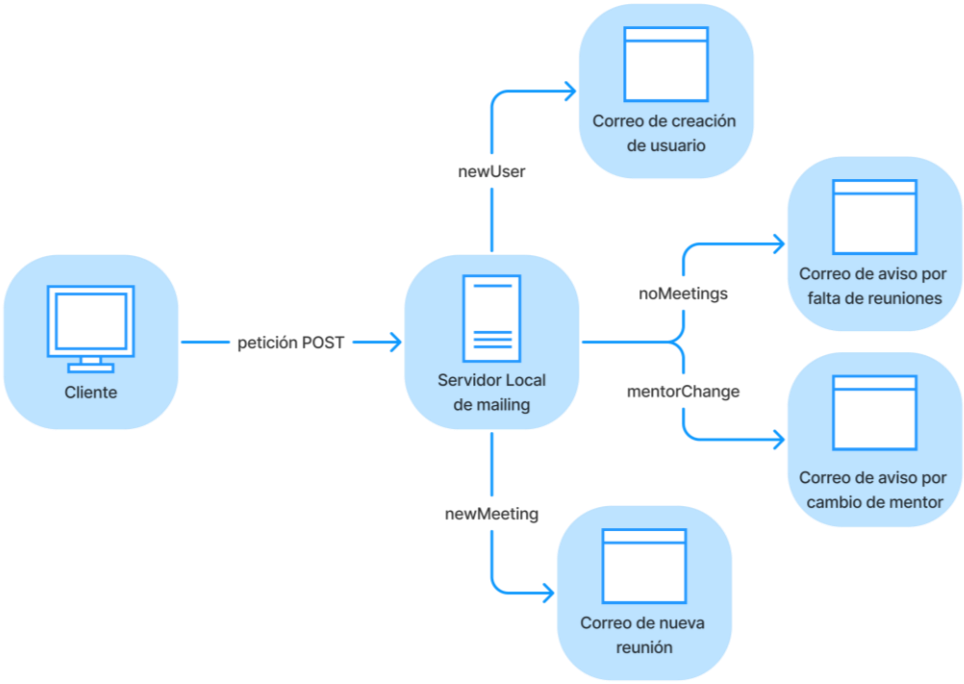

Ilustración 19: Diseño del funcionamiento del servidor de mailing

En el diagrama se puede ver cómo un cliente desde la interfaz web se conectaría al servidor mediante las funciones que tiene disponible desde su rol. Estas funciones realizarían la petición de envío de un *email* de alguno de los tipos ofrecidos por el servidor. El servidor las leería y ejecutaría la función adecuada según el caso, llevando la petición al servicio correspondiente que realizaría el envío del correo a la dirección especificada.

4.2.4 – Arquitectura del sistema

En los apartados anteriores del **Capítulo 4.2** hemos explicado con detalle las interfaces del usuario reflejadas en los mapas de navegación, la base de datos y el servidor de mailing. El siguiente paso en el diseño del sistema es representar su arquitectura.

La arquitectura del sistema que conformará la aplicación de MENTality se basa en los tres nodos que hemos descrito previamente. El usuario interactúa con la plataforma mediante las interfaces explicadas detalladamente en el **Capítulo 4.2.1**. Las interfaces conforman la vista, las acciones que realice en las distintas pantallas afectarán a los controladores. Los controladores invocarán a las distintas funciones ofrecidas por el modelo, el cual recordemos está dividido en sistema, base de datos y servidor de *emails*.

Teniendo todo ello en cuenta la arquitectura del sistema podría mostrarse resumida en la **ilustración 20**, se puede consultar también en el siguiente enlace.

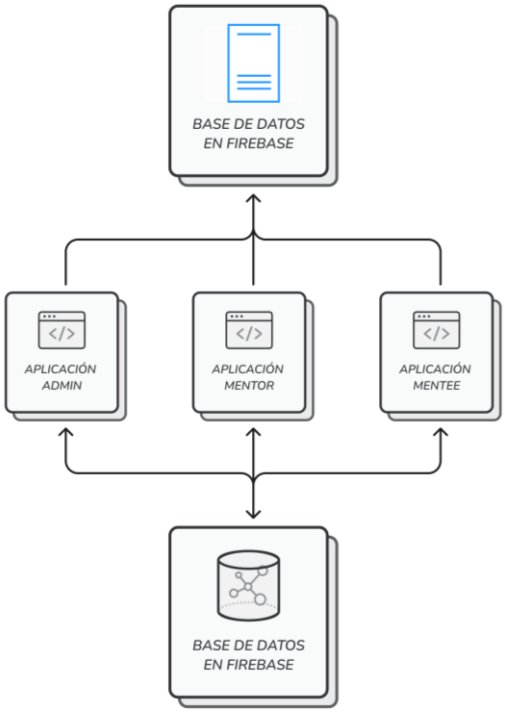

Ilustración 20: Diseño de la arquitectura del sistema

Vamos a detallar a continuación cada elemento y cómo interactúan entre ellos. Las aplicaciones son tanto la vista como el controlador del patrón MVC aplicado en el sistema. La vista se podría resumir en el código HTML de las pantallas; los controladores serían aquellas funciones que se disparan al hacer ciertas acciones en la vista y que generan llamadas al modelo.

El modelo en la aplicación está conformado por 3 partes independientes. El sistema, que está integrado junto a las aplicaciones de los usuarios, es el código JavaScript que los controladores disparan. Los otros elementos del modelo son la base de datos y el servidor de *mailing*, ambos son externos a la plataforma y son accedidos a través del sistema con las funciones públicas que proporcionan. Como ambos elementos han sido explicados en profundidad durante este capítulo no incidiremos nuevamente en ellos.

5 - Implementación

Una vez se finalizó y revisó la fase de diseño se dio inicio a la fase de implementación, en la que se desarrolló todo lo necesario para obtener una aplicación funcional. Este desarrollo se llevó a cabo utilizando JavaScript XML (JSX), y estuvo guiado por los recursos creados en la fase previa. Se usaron sobre todo el mapa de navegación, que facilitó la definición de la comunicación entre las diferentes interfaces. Además, los diagramas de secuencia se emplearon para garantizar la correcta interacción entre los elementos del sistema y el usuario.

En el **Capítulo 2, apartado 3** se explicó y justificó el uso de React como lenguaje de desarrollo. Este enfoque, basado en componentes, permitió dividir el sistema en unidades independientes y reutilizables. Cada componente representa un archivo que puede ser renderizado e incluir código JavaScript, aunque se recomienda mantener la lógica separada para promover la claridad del código y una correcta organización.

En esta sección se destacarán los aspectos más relevantes de la implementación, bien por su importancia en el proceso o por los desafíos que supusieron. Este análisis no tiene como objetivo ser un tutorial paso a paso sobre el desarrollo, sino una exposición de los aprendizajes y experiencias adquiridos. Se describirá la implementación final, dado que algunos casos de uso clasificados como *Could* o *Won't* en el **Apéndice III** no han podido ser implementados durante la duración del proyecto.

Por último, todas las interfaces en su versión final están adjuntadas en el **Anexo F,** donde se puede consultar su diseño.

5.1 - Creación base de datos

Anteriormente se ha explicado cómo en las pruebas iniciales de uso de React con MUI se usaron archivos Json locales para recuperar y mostrar los datos que contuvieran. Para mantener esta información se decidió utilizarla para poblar la base de datos. Siguiendo el diseño mostrado en la **ilustración 18** se crearon scripts en JavaScript. Estos leían los Json ya creados y subían dichos objetos como documentos en la colección correspondiente.

Un ejemplo de estos códigos es:

```
const admin = require('firebase-admin');
const fs = require('fs');
const serviceAccount = require('./serviceAccountKey.json');
admin.initializeApp({credential: admin.credential.cert(serviceAccount)});
const db = admin.firestore();
const usersData = JSON.parse(fs.readFileSync('./data/users.json', 'utf-8'));
async function uploadUsers() {
 const usersRef = db.collection('users');
 for (const user of usersData){
  const docRef = usersRef.doc(user.id.toString());
  try{
   const docSnapshot = await docRef.get();
   if(docSnapshot.exists){
```

```
      console.log(`User ${user.id} ya existe. Actualizando el documento...`);
      await docRef.update(user);
    }
    else{
      console.log(`User ${user.id} no existe. Creando documento en la colección...`);
      await docRef.set(user);
    }}
   catch(error){
console.error(`Error actualizando/creando user ${user.id}: `, error);
 }}
 console.log('Carga de usuarios completada');
}
uploadUsers();
```

Se creó un código similar para cada colección que se posee en la base de datos, excepto para la colección *MilestonesForm*, ya que no poseía archivo Json asociado. La colección *MilestonesForm* almacena las respuestas de cada tándem a las preguntas de la encuesta de finalización de un hito (*form* en FireBase).

La creación de la base de datos en FireBase se hizo siguiendo la documentación que proporciona su propia web, referenciada en [10], la subida de imágenes se realizó adaptando el código de la web [11], con las colecciones concretas mostradas tanto en el **Capítulo 4.2.2** como en el **Anexo E**. Estas colecciones poseen elementos denominados documentos, los cuales alojan cada uno de los objetos a almacenar, este proceso sería la población de la colección. Haciendo uso de los ficheros nombrados al comienzo de esta sección se poblaron las colecciones con datos iniciales de prueba.

5.2 - Componentes de usuarios

Una vez la base de datos estuvo finalizada se comenzó con la creación de las pantallas a las que tendría acceso cada tipo de usuario. Las pantallas previas que se crearon para comprender el funcionamiento de React fueron rehechas. Esta decisión se tomó porque no contaban con componentes de MUI, lo que originaba que no mantuvieran concordancia con el resto de las interfaces.

En este apartado se explicarán los aspectos más importantes en la creación de los componentes de React y en la finalización de la aplicación en sí.

Como se explicó al comienzo del capítulo los componentes en React son los distintos archivos que posee el directorio del sistema, pero solo los que poseen código XML y por tanto, al lanzar la aplicación, serán renderizados como interfaces para el usuario. El resto de los archivos del directorio son los *hooks* y *helpers* creados para la obtención y tratamiento de los datos.

5.2.1 - Imagen de la aplicación

El aspecto que más se destacó en la primera reunión que se tuvo con el tutor de empresa fue la necesidad de tener una imagen de plataforma. Con dicha imagen se conseguiría que los empleados conocieran la aplicación y cómo navegar por ella. Además, facilitaría la posterior presentación al departamento *People* al poder ver todos los componentes como una sola aplicación.

Se dedicaron algunas horas a crear la base de todos los componentes que tendría la plataforma. Se eligió una paleta de colores que se usaría en todos ellos, también un logo y un tipo concreto de lenguaje para usuarios (mentor-*mentee*), para hitos (primer café…).

Fondo	linear-gradient (#f0f6f8, #e0e1e7)	Fondo de cada componente
Pasar ratón	#686868	Componente apuntado por el ratón
Botón	#0586C7	Componente Button de MUI personalizado
	#006DA6	Componente Button:hover de MUI personalizado
Switch	#0586C7	Componente Switch track de MUI personalizado
	#0586C7	Componente Switch track checked de MUI personalizado
	#2AB1F5	Componente Switch thumb de MUI personalizado
	#006DA6	Componente Switch thumb checked de MUI personalizado

Tabla 2: Colores de la imagen de la aplicación

Además de estos colores aplicados a los distintos componentes, se organizaron todas las pantallas de manera similar, para ello se crearon clases personalizadas de CSS. Estas clases eran aplicadas a los componentes *Box* y *Grid2* proporcionados por MUI.

```
.vbox-container {
  display: flex;
  flex-direction: column;
  height: 100vh;
}
/* NavBar */
.top-section {
  flex: 0.1;
  display: flex;
  justify-content: center;
  align-items: center;
}
/* Pantalla principal de los usuarios */
.bottom-section {
  flex: 1;
  display: flex;
  border-top: 2px solid black;
}
```

Al componente inferior le aplicaba el estilo *bottom-section* y se le añadía el componente *Paper* de MUI que renderiza un div con fondo blanco que hace destacar los elementos que contenga respecto al fondo.

El otro elemento principal de cada componente era la barra de navegación. Se creó una barra de navegación por cada rol de usuario, ya que cada uno cuenta con accesos directos diferentes. La estructura de todas ellas, aun así, era la misma, comenzando de izquierda a derecha:

- Logo de MENTality: imagen en formato pequeño del logo que permite la redirección a la pantalla principal.
- Notificaciones: solo mostradas para mentores y *mentees*. Al pulsarlo se muestra un *Dialog* de MUI con la lista de notificaciones del usuario.
- Enlaces a las pantallas principales: se muestran 3 enlaces a las 3 funciones principales que tiene el usuario. Para el administrador son gestión de tándems, gestión de hitos y foro general. Para mentores y *mentees* son la pizarra de los tándems de los que forman parte, el foro general y su calendario.
- Saludo al usuario: se muestra un pequeño texto "Hola, nombreUsuario". El texto tiene como objetivo permitir al usuario saber fácilmente que sesión está iniciada, además cuenta con el ícono asociado al rol para poder distinguir también esta característica. Pulsando en este texto se permite cerrar la sesión y eliminar los datos de sesión guardados.

5.2.2 – Modularidad

Una vez la imagen que se quería dar a los componentes estaba acordada se debía comenzar con la programación de estos. Debido a la gran cantidad de componentes que conforman la aplicación, un total de 86, la organización durante su desarrollo fue fundamental. Con la modularidad se permite evitar repetición de código innecesario y facilita la lectura del código.

Se hizo por tanto uso de esta creando componentes de un número reducido de líneas. Los componentes se dividieron en 3 partes para conseguir modularidad:

- El código HTML que renderizaba los elementos en la pantalla.
- Los *helpers* que son pequeños scripts solo usados por un componente para obtener o tratar algún dato.
- Los *hooks* que son funciones en React que se usan en varias partes del código y suelen hacer uso de otros *hooks* básicos de React.

Haciendo uso de dicha estructura se fueron creando los componentes, los cuales se organizan en carpetas según a qué grupo principal pertenecían (hitos, calendarios, tándems, …).

Un ejemplo de código modular es el siguiente:

```
return (
  <>
   <Box sx={{display: 'flex', justifyContent:'center', alignItems: 'center', gap: 2, mb: 1}}>
      <TextField label="Buscar usuario" variant="outlined" type="search" value={filtro}
onChange={handleCambioFiltro}
           size="small" sx={{textTransform:'none', width:'225px', '& .MuiInputLabel-
root':{textTransform:'none'}}}/>
    <FormControlLabel className="custom-switch" value={filtrarPorAsignacion}
     label={filtrarPorAsignacion ? "Sin tándem" : "Todos"} labelPlacement="end"
     sx={{textTransform:'none', width:'200px', height:'56px', edge:'end',
      '&:hover': { outline: 'none', border: 'none' }}}
     control={
      <Switch checked={filtrarPorAsignacion} onChange={handleFiltrarPorAsignacion}/>
     }
    />
   </Box>
   {/* <Typography variant='h6'>Usuarios</Typography> */}
   {
    loadUsers && <Typography variant='h7'>Cargando...</Typography>
   }
   {
    !loadUsers && (filtradoUsuarios.length === 0) && (
     <Typography variant="h7" color="black" sx={{alignContent:'center'}}>
      <Tooltip title={"sin tándems encontrados"} arrow sx={{alignSelf:'center'}}>
       <SearchOffRoundedIcon color="error" sx={{ fontSize: 32, verticalAlign:'middle' }}/>
      </Tooltip>
      No hay usuarios registrados con las características seleccionadas
     </Typography>
    )
   }
   {
    !loadUsers && (filtradoUsuarios.length > 0) && (
     <Box sx={{height:'70vh', overflowY: 'auto', p: 0.5}}>
      {filtradoUsuarios.map((user) => (
                <UserItem key={user.id} id={user.id} nombre={user.nombre}
apellidos={user.apellidos}
          rol={user.rol} asignado={user.asignado} openCreateUser={openCreateUser}
setOpenCreateUser={setOpenCreateUser}
       />
      ))}
     </Box>
    )
   }
  </>
 )
```

```
 return (
  <>
      <Card key={id} sx={{mt:1, background:"linear-gradient(#f0f6f8, #e0e1e7)",
maxHeight:'80px',
      border: '2.5px solid', borderRadius:'10px', borderColor: asignado ? '#1ba500' :
'#a5001a'}}>
```

```
    <CardContent sx={{justifyContent:'space-between', alignContent:'center'}}>
     <Typography variant="h6" sx={{mb:0, display:'flex'}}>
      <Tooltip title={rol.charAt(0).toUpperCase() + rol.slice(1)} arrow>
       {rol === 'mentee' ? (
                  <SchoolRoundedIcon color="black" sx={{fontSize:20, alignSelf:'center',
verticalAlign:'middle', mr:1}}/>
        ) : (
         <CastForEducationRoundedIcon color="black" sx={{fontSize:20, alignSelf:'center',
verticalAlign:'middle', mr:1}}/>
        )}
       </Tooltip>
       {nombre} {apellidos}
       <Tooltip title={"Eliminar usuario"} arrow onClick={handleOpenDialog}>
        <DeleteRoundedIcon
              sx={{fontSize:20, ml:'auto', verticalAlign:'middle', fill: !asignado ? 'black' :
'#686868', '&:hover':{cursor:'pointer', color:'#686868'}}}/>
       </Tooltip>
       <Tooltip title={"Editar usuario"} arrow onClick={handleEditUser}>
        <EditNoteRoundedIcon color="black"
              sx={{fontSize:20, ml:'5%', verticalAlign:'middle', '&:hover':{cursor:'pointer',
color:'#686868'}}}/>
       </Tooltip>
      </Typography>
             <Typography variant="h7" color="black" sx={{mt:0.1, fontSize:'small'}}>
{tandemMensaje}</Typography>
    </CardContent>
   </Card>
</>
 )
```

Como podemos observar, el primer componente invoca al segundo al hacer un *map* en un array, de esta manera eliminamos líneas de código del primer componente. Cabe destacar que estos ficheros no son reales, ya que han sido simplificados, eliminando *import* y definición de variables.

5.2.3 - Pantallas comunes

Las pantallas comunes en la aplicación son aquellas que no varían su código independientemente del rol que tenga el usuario que haya iniciado sesión. Del mismo modo se incluyen en este apartado aquellas pantallas que se muestran antes de que el usuario inicie sesión.

Estas últimas son sencillas de programar de manera común ya que no tienen cambios, lo cual evita tener que crear componentes generalizados.

Las pantallas comunes que muestran pequeños cambios dependiendo del rol del usuario que quiera renderizar la pantalla fueron un reto mayor. Por ejemplo, se debía tener en cuenta la barra de navegación a mostrar, lo cual obligó a la creación de otra *navbar* encargada de renderizar la adecuada según el usuario.

Estas situaciones se dieron en mayor medida entre las pantallas de mentor y *mentee*, debido a su gran similitud en los componentes. Se buscó reducir al máximo cada componente, siguiendo el principio de modularidad anterior, de manera que fuera más sencillo crear los *helpers* que trataran los datos de mentor y *mentee* de manera diferente.

Se incluye a continuación un código de pantalla común para los tres roles existentes en MENTality (el código está editado eliminando algunas declaraciones no relevantes):

```
export const CommonForum = () => {
 useEffect(() => {
  if (!id){
   if (location.pathname.includes('mentor')){ navigate('/mentor'); }
   else if (location.pathname.includes('mentee')){ navigate('/mentee'); }
   else if (location.pathname.includes('admin')){ navigate('/admin'); }
  }
   const handleSendMessage = async() => {
  if (newMessage.trim() === ''){
   toast.error('El mensaje no puede estar vacío');
   return;
  }
  try{
       const newMsg = {autor: id.toString(), fechaHora: dayjs().format('YYYY-MM-
DDTHH:mm:ss'), msg: newMessage, };
   const updatedForum = [...messages, newMsg];
   await updateForum('general', updatedForum);
   setMessages(updatedForum);
   toast.success('Mensaje publicado con éxito');
   setNewMessage('');
  }
  catch(error){ toast.error('Error enviando mensaje:', error); }
 }

 if (loadInfo || loadForum){ return <LoadingPage/>}

 return (
  <>
   <Box className='vbox-container' sx={{.....}}>
```

```
      <Box className='top-section'>
        {rol && (<RolNavbar rol={rol}/>)}
      </Box>

      <Box sx={{display:'flex', flexGrow:1, gap:2, p:1}}>
       <Paper sx={{flex:1, p:2}}>
        <Typography variant='h4' color='black' sx={{...}}>Foro MENTality</Typography>

        <Box ref={scrollRef} sx={{.....}}>

         {messages.length === 0 && (
          <Typography variant='body1' sx={{.....}}> No se ha publicado ningún mensaje en el
foro general</Typography>
         )}

         {messages.slice().map(message => (
          <Box key={message.fechaHora} sx={{...}}>
           <Typography variant='body1' sx={{....}}>
            {rol ? (
             <>
              <RolIcon rol={...}/> {getNombreById(...)} {getApellidosById(...)} ha dicho:
             </>
            ) : (
             'Cargando...'
            )}
           </Typography>
           <Paper key={message.fechaHora} sx={{.....}}>
            <Typography variant='body1' sx={{...}}>{message.msg}</Typography>
           </Paper>
           <Typography variant='body1' sx={{....}}>
            {dayjs(message.fechaHora).format('HH:mm DD-MM-YY')}
           </Typography>
          </Box>
         ))}
        </Box>

        <Box sx={{display:'flex', gap:1, justifyContent:'center'}}>
         <TextField multiline rows={4} variant='outlined' placeholder='...'
         value={newMessage} onChange={(e) => setNewMessage(e.target.value)} sx={{...}}/>
         <Button variant='contained' color='boton' onClick={handleSendMessage}
          sx={{...}}>Enviar</Button>
        </Box>
       </Paper>
      </Box>
     </Box>
    </>
   )
}
```

5.2.4 - Pantallas propias de cada usuario

Además de las pantallas comunes cada usuario tiene algunas pantallas exclusivas de su rol, en el caso del administrador, por ejemplo, son la mayoría de sus interfaces. De igual forma se crearon pantallas propias para mentor/*mentee* en aquellos casos que entre ambos roles los datos de los que se partían eran distintos, tener el objeto completo o un campo del objeto, por ejemplo. En los escenarios en los que teníamos dicho problema intentar crear componentes generales necesitaba mayor cantidad de líneas de código que la creación de un componente para cada rol. Como la meta principal durante la implementación era la modularidad se optó por crear los dos componentes para que el código fuera más legible.

En las pantallas de administrador antes nombradas no existía la posibilidad de generalizar. Como resulta obvio, el administrador de la aplicación tiene funciones exclusivas que deberán ser mostradas en componentes únicos que no sean usados por mentores y *mentees*. A pesar de eso se buscaron los elementos que se iban a repetir en otros componentes, como podrían ser las insignias de los hitos logrados, representadas por el color único del hito. Este componente se diseñó para poder ser usado por varios usuarios y componentes mediante el paso de parámetros en su invocación.

```
export const MilestoneMiniItem = ({milestone, size}) => {
 return (
  <Tooltip title={milestone.name} arrow sx={{'&:hover': {cursor:'normal'}}}>
    <EmojiEventsRoundedIcon sx={{color: milestone.color, fontSize: size, verticalAlign:
'text-top'}}/>
  </Tooltip>
 )
}
```

6 - Pruebas del sistema

En este apartado se muestran los resultados principales que se obtuvieron al iniciar la fase de pruebas del sistema. Esta fase se llevó a cabo una vez la implementación de las pantallas de los 3 roles de MENTality habían sido revisados y aprobados por los responsables del departamento.

Aunque durante la implementación de los distintos componentes se hacían pequeñas pruebas para revisar que las funciones creadas realizaran las tareas para las que habían sido diseñadas, es en esta fase en la que se realizan pruebas globales del sistema.

El sistema se ha sometido a evaluaciones sin usuarios reales, ya que no se ha mostrado la aplicación a sus posibles usuarios futuros. Esto es algo que queda pendiente y que se debería hacer si el proyecto fuera aprobado por NTT Data para su uso interno. Una vez todas las pruebas obtuvieron un resultado positivo, se puede decir que el objetivo de esta fase se ha cumplido y que el sistema funciona perfectamente.

Como se ha nombrado en las fases anteriores, no se considera que pueda haber fallo al guardar en la base de datos la información, ya que estas no son acciones que dependan del usuario. Al hacer uso de FireBase en el proyecto y estar en la nube siempre debería ser accesible.

Siguiendo con los ejemplos usados anteriormente, se van a incluir a continuación algunas de las pruebas realizadas sobre el CU- 7 “Crear nuevo tándem” y el CU- 31 “Crear reunión”. Las pruebas incluidas son: que un usuario realice el flujo de eventos principal y uno de los flujos de eventos alternativos. Sobre el CU-31 solo se añaden las pruebas sobre un usuario con rol de mentor, aunque se hayan hecho en ambos roles, debido a su similitud.

El resto de los casos se pueden encontrar en **Anexo G: Casos de prueba**.

Caso de prueba 1: CU-7 Crear nuevo tándem correcto

Descripción: se realiza la asignación de un mentor a un *mentee* correctamente.

Precondiciones: un usuario con rol admin ha iniciado sesión correctamente en un sistema ya en ejecución.

Entradas: se asigna al *mentee* "Jaime Manuel Fernández" el mentor llamado "Juan Marquina de Miguel".

Procedimiento esperado: el administrador selecciona en el formulario de creación de tándems tanto el *mentee* como el mentor y pulsa el botón "Crear tándem marcado". El sistema comprueba que el *mentee* no estaba asignado, y lo crea en la base de datos, además cambia el campo asignado de ambos usuarios para que sea *true.* Se notifica al usuario con un aviso de tipo *toast.success*.

Resultado: Correcto, en la base de datos se ha creado el tándem con todos sus campos inicializados de manera correcta. Ambos usuarios ahora aparecen como asignados en la lista de usuarios, su tándem también se muestra en la lista de tándems. Se ha mostrado el *toast* informando de la creación correcta del tándem. Al acceder nuevamente a crear un tándem no se muestra el *mentee* "Jaime Manuel Fernández".

Caso de prueba 2: CU-7 Creación de un nuevo tándem sin usuarios

Descripción: se comienza la creación de la asignación de miembros a un tándem, pero no se rellenan los campos del formulario.

Precondiciones: un usuario con rol admin ha iniciado sesión correctamente en un sistema ya en ejecución.

Entradas: no se realizan asignaciones en ninguno de los desplegables.

Procedimiento esperado: el administrador no selecciona miembro en el formulario y pulsa "Crear tándem marcado". El sistema muestra un *toast* de error sobre los campos que se deben rellenar, se añade también un *helpertext* para indicar el campo que ha producido un error. El sistema se mantendrá en la interfaz de creación de nuevo tándem.

Resultado: Correcto, en la base de datos no se ha creado el tándem. Ambos usuarios mantienen su campo asignado con el valor previo en la lista de usuarios. Se mantiene la pantalla con los avisos y *helpertext* marcados para ayuda del usuario.

Todos los casos de prueba del CU-7: "Crear nuevo tándem", tanto los que se refieren al flujo de eventos principal como los alternativos, han finalizado con un resultado positivo. Gracias a esto, podemos concluir que el caso de uso ha sido implementado según el objetivo previsto.

Se muestran algunas de las pruebas a las que se sometió el CU-31: “Crear reunión”.

Caso de prueba 3: CU-31 Creación de reunión correctamente en mentor

Descripción: el usuario crea una nueva reunión para uno de sus tándems

Precondiciones: el usuario de tipo mentor “Paula Oliván” ha iniciado sesión correctamente en un sistema ya en ejecución.

Entradas: título: “Primer café”, descripción: “Conocer el proyecto de mentorías y las oficinas”, fecha: “13-11-2024”, hora: “12:00”, tipo de reunión: “Café”, *mentee*: “Juan”.

Procedimiento esperado: el mentor introduce cada entrada en su campo correspondiente y pulsa “Crear reunión”. El sistema muestra un *Dialog* con la información de la reunión que se creará y el resto de las reuniones del día. El mentor pulsa “Aceptar”. El sistema registra en la base de datos del tándem la reunión y la añade al calendario del mentor y del *mentee*. Ambos usuarios son notificados mediante un correo electrónico. Se muestran en la pantalla del usuario los *toast.success* ligados a la creación de la reunión y el envío correcto de los mails.

Resultado: Correcto, se muestra el *toast* de reunión creada y esta se muestra en el calendario del mentor que tiene la sesión iniciada. Al correo electrónico de mentor y *mentee* ha llegado un mensaje informando de la nueva reunión, en la interfaz del mentor ha aparecido el mensaje de envío correcto.

Caso de prueba 4: CU-31 Creación de reunión con fecha no válida en mentor

Descripción: el usuario crea una nueva reunión para uno de sus tándems, pero la fecha es inválida al ser pasada o fin de semana.

Precondiciones: el usuario de tipo mentor “Paula Oliván” ha iniciado sesión correctamente en un sistema ya en ejecución.

Entradas: título: “Primer café”, descripción: “Conocer las mentorías y las oficinas”, fecha: “05-11-2024”, hora: “12:00”, tipo de reunión: “Café”, *mentee*: “Juan”.

Procedimiento esperado: el mentor introduce cada entrada en su campo correspondiente y pulsa “Crear reunión”. El sistema muestra un *toast.error* informando sobre la invalidez de la fecha, se muestra un *helpertext* en los campos concretos que no han sido completados. No se realizan cambios en la base de datos y se permanece en la misma interfaz.

Resultado: Correcto, se muestra el *toast* de error por falta de campos junto al *helpertext* de fecha pasada. El sistema mantiene los datos anteriores en la interfaz.

Las pruebas que se realizaron para la comprobación del CU- 31: “Crear reunión” han funcionado como se esperaba tanto para mentores como para *mentees*. Por ello se considera que el caso de uso ha sido implementado adecuadamente.

7 - Conclusiones y líneas futuras

Este proyecto ha sido llevado a cabo con éxito, puesto que se han cumplido todos los objetivos fijados en la propuesta inicial del Trabajo Fin de Grado. Durante este capítulo se van a exponer las conclusiones obtenidas con la realización de este proyecto, además se incluyen las líneas futuras que podría tener la plataforma.

7.1 - Conclusiones

La realización de la aplicación MENTality descrita en este documento duró aproximadamente 3 meses. Este fue el primer reto al que me enfrenté, ya que en un tiempo reducido debía realizar un prototipo funcional para ser usado tanto en NTT Data como para realizar el depósito del TFG. Debido a esto fue fundamental la adopción de la metodología de cascada mejorada que permitió tener una visión global del proyecto y poder organizarlo. Otra herramienta fundamental fue el uso de la técnica MoSCoW para enfocar los esfuerzos en aquellos requisitos que permitirían el desarrollo del producto mínimo viable.

Con la realización del trabajo se ha aprendido a desarrollar un proyecto del ámbito de la ingeniería del software en un entorno real para un cliente con sus propias exigencias. Esto ha sido muy útil para entender cómo funcionan este tipo de encargos en el mundo laboral, en el cual no tenía experiencia previamente.

Esto también causó problemas de carácter técnico, la empresa pedía desarrollar la aplicación en un lenguaje concreto, React. Antes de comenzar el proyecto no había desarrollado ninguna aplicación en este lenguaje, lo que me obligó a aprender a utilizarlo. Por este motivo, la organización del directorio fue cambiando a lo largo de la fase de implementación por no contar con el volumen real de la aplicación. Esta fue la dificultad principal del proyecto, tener que realizar de manera ordenada una aplicación en un lenguaje desconocido y con una gran variedad de requisitos predeterminados.

El desconocimiento en React y la falta de conocimiento en el desarrollo de software en el mundo laboral causó algunos problemas que tuvieron que ser solucionados durante la implementación. Los principales se basaban en la versión de React utilizada, como se explicó en el **Capítulo 2.3**, se optó por usar la última versión estable de React. Esta decisión se tomó porque se consideró que sería la más segura, pero provocó problemas ya que algunas librerías o funciones propias de React ya no existían o habían pasado a estar obsoletas. Hubo que programar componentes propios en vez de usar los proporcionados por librerías y fue necesario leer distintas APIs por no contar con códigos de ejemplo que pudieran ser usados como base.

A pesar de todo esto, todas las partes del proyecto han sido finalizadas correctamente y aunque no están perfectamente optimizadas y/o depuradas, cada una de ellas realiza su función de manera eficiente y adecuada para que una versión beta pueda ser usada dentro de la empresa.

Personalmente, mi objetivo al comienzo era conocer cómo funcionaba una empresa importante en el ámbito de la informática en el país. Este objetivo fue evolucionando cuando se fijó el tema que tendría el Trabajo Fin de Grado. Conocer que la aplicación podría tener importancia real en la empresa y podía ser utilizada por gran cantidad de usuarios actuó como un aliciente a la hora de crear una aplicación lo más completa posible y con una usabilidad real en NTT Data.

Vanessa Martín, directora del TFG, deseaba crear una aplicación útil en la empresa, la cual pudiera ser usada como base para añadirle diversas mejoras para la oficina de Zaragoza. Con un plan de futuro amplio, para que la aplicación se convirtiera en base del sistema de *mentoring* en la división de NTT Data España.

Podemos decir por tanto que el proyecto ha cumplido todos los objetivos y expectativas que se tenían, dejando una sensación de trabajo bien.

7.2 - Líneas futuras

Desde un comienzo la aplicación MENTality fue pensada para tener continuidad dentro de NTT Data. MENTality busca ser la aplicación de mentoría de referencia dentro de la empresa.

Para ello, en primer lugar se deberían completar aquellos requisitos y casos de uso que no se han podido desarrollar durante la creación de esta primera aplicación, como son los foros exclusivos por rol y ver los perfiles de usuarios ajenos al tándem del usuario que haya iniciado sesión.

Debido a que este periodo de finalización de casos de uso obligaría a volver a la fase de implementación, una vez evaluada, se debería volver a realizar la fase de pruebas. Además de repetir las pruebas anteriores, descritas en esta memoria, la creación de pruebas para los nuevos casos de uso sería esencial. A su vez, podría ser interesante que la empresa comenzara con una fase de prueba en usuarios potenciales. Esto ayudaría a ver puntos débiles a mejorar, funciones que no son utilizadas como se pensaron inicialmente o dificultades para entender algunos de los casos de uso.

Sería sumamente importante realizar una pequeña investigación de las herramientas usadas para ver si podrían ser utilizadas en la empresa o su coste sería demasiado. Por este motivo, quizás, se tendría que usar algunaa de las soluciones descartadas para este proyecto, las cuales pueden ser encontradas en el **Capítulo 4, apartado 3.**

Finalmente se podría dar por concluida la fase de pruebas comenzando con la fase de despliegue y mantenimiento de la aplicación. En caso de funcionar en las oficinas de Zaragoza podría seguir ampliando su uso a otras oficinas de NTT Data, tanto dentro como fuera de España. Al hacer esto se deberían añadir otras funcionalidades como cambio de idioma, alto contraste, accesibilidad AAA… de manera que cualquier empleado de la empresa pudiera hacer uso de la aplicación sin dificultades.

Apéndice I: Glosario

Debido a que la plataforma se ha creado para ser utilizada dentro de la empresa NTT Data se ha aplicado su terminología durante el desarrollo de la aplicación y durante esta memoria para mantener la coherencia entre ambas partes.

- MENTality: nombre de la plataforma creada para conseguir un mejor sistema de mentorías en la empresa NTT Data.
- Mentor: es un empleado de la empresa que tiene experiencia y conoce tanto la cultura como los valores de la empresa y busca ayudar al *mentee* durante su integración en NTT Data.
- *Mentee*: es un nuevo empleado el cual va a recibir el apoyo del mentor que se le asigne con el objetivo de que su incorporación sea más fácil y no se sienta perdido durante la misma teniendo siempre un trabajador veterano con el que puede contactar.
- Tándem: nombre que reciben las parejas de mentor y *mentee* en la plataforma, de manera que ambos miembros sientan que ambos son necesarios para su evolución.

Apéndice II: Gestión del Proyecto

En este apéndice se realiza un estudio de dos aspectos relacionados con la gestión de proyecto.

En la sección de gestión de esfuerzos se detallan las horas estimadas y las horas empleadas, desglosándolas por categorías, con la distribución del esfuerzo desde el inicio hasta el final del proyecto.

Se analiza también la gestión de riesgos en donde se describe el análisis de los riesgos identificados, la evaluación de los mismos y la estrategia de mitigación de los riesgos de mayor exposición.

Gestión de esfuerzos:

Gestionar el tiempo empleado en cualquier tema es realmente importante, pero en el proyecto desarrollado fue una parte fundamental por la limitación de tiempo que existía para finalizarlo.

En este trabajo se ha hecho uso de la herramienta Track Toggl que permite contabilizar las horas dedicadas a las distintas tareas que forman un proyecto. En su plataforma se pueden crear las tareas se quieren contabilizadas y especificando en cada tarea cuál es la función que se ha realizado.

El uso de la herramienta ha permitido que al terminar el proyecto se puedan comparar las horas que se estimaron inicialmente en cada fase y el esfuerzo real que se ha debido dedicar a cada etapa.

En total se han creado 7 fases principales con un total de 414 horas estimadas y 487’25 horas invertidas realmente, como se muestra en la **tabla 3**. Las estimaciones, por tanto, han sido inferiores a la dedicación real que ha necesitado el proyecto, con 87,25 horas de diferencia (subestimación). Esta diferenciación como se puede observar en la **tabla 3** se debe principalmente a las horas dedicadas al diseño ya que las horas invertidas en realizar los casos de uso, diagramas de secuencia, y los mapas de navegación fueron muy superiores a las estimadas.

Track Toggl permite crear tareas dentro de las distintas fases que se realizan. Cada tarea se corresponde a una de las necesidades que se debían realizar para completar la fase. Se puede observar en la **ilustración 22**, en total se han creado 87 tareas, siendo 63 de la fase de desarrollo.

En la **ilustración 23** se refleja como la implementación abarca un 64% del tiempo invertido (311:22 horas), seguida del diseño con un 14’70% (71:36 horas). Las tareas relativas a la memoria han costado 49:14 horas, mientras que a la verificación y validación realizada en la fase de pruebas se les han dedicado 7:33 horas. Las tres tareas restantes, reuniones, formación e investigación previa, suman un total del 9,75% del tiempo invertido.

Podemos ver en la **ilustración 24** como se han invertido las hora en las 13 semanas que ha durado la estancia en NTT Data. La primera semana tiene una cantidad menor de horas (21 horas) porque el proyecto comenzó el 4 de septiembre. Además, las dos primeras semanas cuentan con una cantidad inferior de horas debido al horario de verano que duró hasta el 15 de septiembre. El resto de semanas las horas dedicadas se han mantenido estables con una media de 40 horas invertidas semanalmente al proyecto.

En el siguiente enlace se puede analizar todas las gráficas correspondientes a la gestión de esfuerzos, aunque las más relevantes son las que se han incluido en este documento.

Fases	Duración real	Duración estimada
Formación	39:00 h	40 horas
Investigación previa	4:00 h	4 horas
Diseño del sistema	71:36 h	20 horas
Desarrollo	311:22 h	300 horas
Pruebas	7:33 h	9 horas
Memoria	49:14 h	35 horas
Reuniones	4:30 h	6 horas
	487:17 h	414 horas

Tabla 3: Comparación de las horas reales y la horas estimadas

	TITLE	DURATION	PERCENTAGE
63	Desarrollo	311:22:08	63.90%
8	Diseño del sistema	71:36:29	14.70%
3	Formación	39:00:00	8.00%
3	Investigación previa	4:00:00	0.82%
1	Memoria	49:14:41	10.11%
1	Pruebas	7:33:28	1.55%
8	Reuniones	4:30:34	0.93%

Ilustración 22: Numero de tareas por fase registrado en Track Toggl

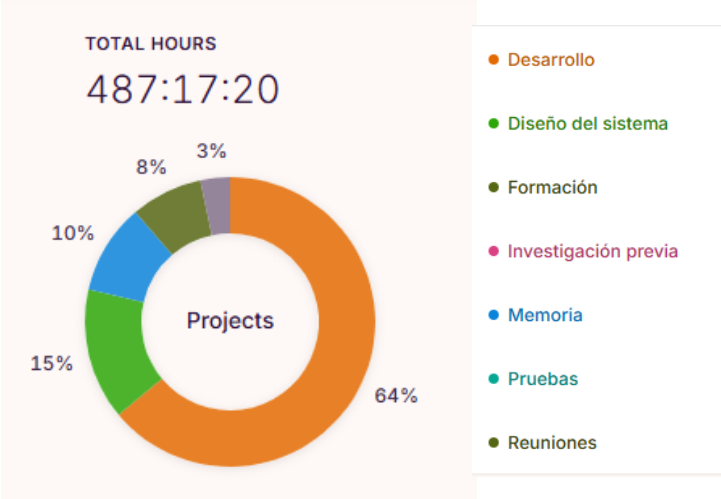

Ilustración 23: Porcentaje de horas por proyecto

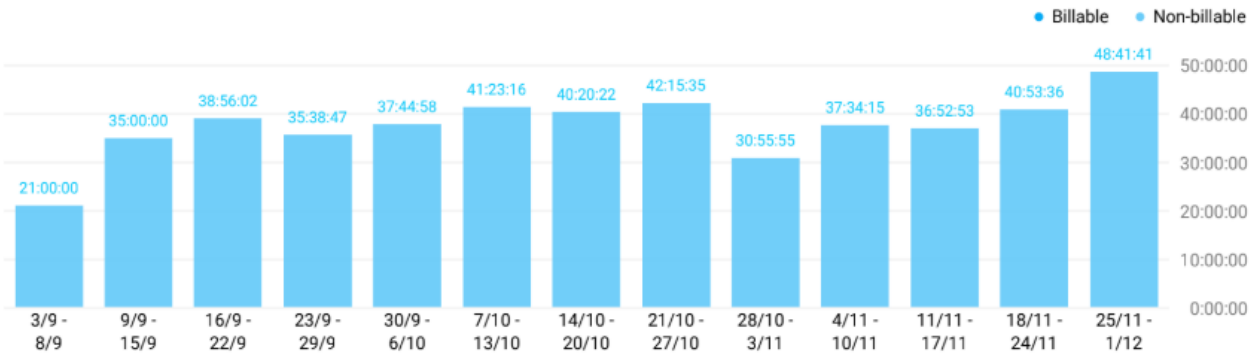

Ilustración 24: Dedicación semanal al proyecto

Gestión de riesgos:

En todo proyecto hay un conjunto de riesgos que pueden materializarse. Su importancia está marcada por la probabilidad que tienen de aparecer y el impacto que provocan al hacerlo. Es necesario priorizarlos en base a dicha importancia, ya que si algún riesgo llega a materializarse podría complicar seguir adelante con el proyecto.

Partiendo de la base de que nunca un riesgo puede reducirse hasta ser nulo, se debe buscar reducir el posible impacto de los más preocupantes hasta un nivel aceptable. Para controlar los riesgos se creó una tabla de los posibles riesgos. Se evaluaron del 1 al 5 en la probabilidad de aparecer y el impacto para el proyecto, multiplicando ambos valores obtenemos la exposición a dichos riesgos. Los riesgos y su evaluación se encuentra en la **tabla 4.**

Posteriormente, de los riesgos más importantes, se creó la estrategia de mitigación de los mismos, se puede analizar en la **tabla 5**.

ID	Descripción	Probabilidad	Impacto	Exposición
1	El proyecto no está bien definido	1	5	5
2	El proyecto no está soportado por las tecnologías actuales	1	3	3
3	**El proyecto no puede ser finalizado en las horas disponibles**	**3**	**5**	**15**
4	Las fases del proyecto no se revisan	2	3	6
5	Se necesitan usar librerías no compatibles entre sí	2	3	6
6	No se prueban los componentes incluyendo todos los casos corner	2	2	4
7	La base de datos elegida no está configurada correctamente desde React y se pierde información	2	4	8
8	El servidor de envío de correos electrónicos no realiza el envío solicitados	3	3	9
9	El código no está correctamente documentado	4	2	8
10	La arquitectura que se quiere desarrollar no está correctamente documentada	4	2	8
11	El directorio del proyecto no está organizado de forma intuitiva	3	2	6
12	No se utilizan herramientas y librerías asentadas y bien documentadas	2	2	4
13	Las tecnologías a usar no son conocidas por la desarrolladora	1	4	4
14	**La aplicación se debe poder ejecutar en navegadores heterogeneos**	**5**	**3**	**15**
15	**La aplicación depende de servicios externos y de sus fallos internos**	**5**	**4**	**20**
16	La retroalimentación de la directora del proyecyo no es suficiente	1	3	3
17	Las habilidades que se poseen no son las correctas para la realización del proyecto	1	5	5
18	La desarrolladora no está comprometida	1	5	5
19	La desarrolladora no tiene tiempo suficiente para el proyecto	1	5	5
20	**La desarrolladora no está capacitado**	**3**	**5**	**15**

Tabla 4: Detección y evaluación de riesgos

ID	Estrategia de mitigación
3	Comprobar periódicamente el avance que tiene el proyecto con el sistema de colores usado. Seguir la priorización de los requisitos para invertir las horas disponibles en los requisitos necesarios para crear el producto mínimo viable e ir añadiendo componentes extra cuando lo básico este realizado y probado.
14	Eliminar la configuración por defecto que los navegadores añaden a las etiquetas HTML que muestran en el navegador. Probar los componentes en varios navegadores principales para comprobar como se visualizan los componentes creados en ellos.
15	Hacer el menor uso posible de sistemas de terceros. En los casos que la opción más viable sea hacer uso de servicios externos usar aquellos pertenecientes a empresas fiables de manera que la probabilidad de fallos sea baja o la reparación de los mismos sea veloz.
20	Dedicar a la formación en las herramientas el tiempo suficiente para que el desarrollo posteriormente sea fluido y buscando reducir la preocupación en el proyecto haciendo uso de herramientas bien documentadas y con una curva de aprendizaje suave. Compartir las dudas con la directora de TFG para ver soluciones más rápido.

Tabla 5: Estrategia de mitigación de los riesgos principales

Apéndice III: Priorización de los Requisitos

Como se ha expuesto durante la memoria la idea básica de este proyecto era crear un prototipo totalmente funcional que pudiera ser presentado al departamento *People* de NTT Data con el objetivo de que se integre en la empresa. Debido a esto, el proyecto se desarrolló para que si era aprobado por dicho departamento, se terminara la implementación en caso de que algunas funcionalidades no pudieran ser integradas durante los meses que duró la estancia en la empresa. Se decidió dar prioridad a ciertos requisitos indispensables para mostrar la idea de la nueva plataforma de *mentoring*.

Para conseguir esta priorización se hizo uso de la técnica MoSCoW que divide los requisitos en 4 categorías: Mo (imprescindible de implementar), S (importante de añadir), Co (estaría bien programarlos si hubiera tiempo) y W (descartados para la entrega a *People*). La técnica también ayuda a saber enfocar los esfuerzos en los puntos que se consideran más importantes evitando invertir horas de desarrollo en funcionalidades más superficiales o secundarias.

Requisitos funcionales:

RF-1	Existen tres tipos de usuarios: administrador, mentor, *mentee*.	Must
RF-2	El sistema permite iniciar sesión haciendo uso de las credenciales corporativas.	Must
RF-3	El sistema permite acceder a OneDesk para tener asistencia técnica.	Must
RF-4	Se permite a los usuarios cerrar sesión para borrar los datos de sesión.	Must
RF-5	Un usuario que no ha cerrado sesión puede volver a entrar a la plataforma MENTality sin necesidad de volver a introducir sesión, siempre que los datos de sesión no hayan caducado.	Must
RF-6	El sistema permite al administrador registrar usuarios	Must
RF-6.1	El sistema no permitirá crear usuarios cuyo correo electrónico, nombre de usuario o número de empleado ya existan en la base de datos.	Must
RF-7	El sistema permite al administrador modificar los datos de un usuario ya registrado.	Must
RF-7.1	El sistema no permitirá al administrador modificar el rol (mentor o *mentee*) de un usuario que ya esté asignado a un tándem.	Must
RF-8	El administrador puede eliminar un usuario.	Must
RF-8.1	El administrador no podrá eliminar un usuario que pertenezca a un tándem.	Must
RF-9	El sistema permite al administrador crear tándems.	Must
RF-9.1	El sistema permitirá que un mentor pueda estar asignado a varios tándems activos al mismo tiempo.	Must
RF-9.2	EL sistema no permitirá que un *mentee* pueda pertenecer a varios tándems.	Must
RF-10	El sistema permite al administrador consultar las estadísticas de los tándems.	Should
RF-11	El sistema permite al administrador ver el número de encuentros generales por día.	Must

RF-12	El sistema permite al administrador conocer las incidencias que tiene un tándem.	Must
RF-12.1	Existirán dos tipos de incidencias: por falta de reuniones y por solicitud de un cambio de mentor	Must
RF-12.2	El sistema permite al administrador ponerse en contacto con el mentor cuando alguno de sus tándems tenga incidencias de cualquier tipo.	Must
RF-12.3	El sistema permite al administrador separar un tándem que haya solicitado un cambio de mentor si se considera necesario.	Must
RF-13	El administrador puede ver los usuarios existentes en la aplicación.	Must
RF-13.1	El sistema permite al administrador filtrar los usuarios por si están asignados, por su rol de MENTality (mentor, *mentee*), nombre, apellidos o nombre completo.	Should
RF-14	El administrador puede ver los tándems creados en la aplicación.	Must
RF-14.1	El sistema permite al administrador filtrar los tándems con incidencias en general, por sus miembros o por un tipo concreto de incidencias.	Should
RF-15	El sistema permite al administrador ver las insignias ganadas por cada tándem.	Must
RF-16	El sistema permite al administrador ver el histórico de un tándem concreto.	Must
RF-16.1	El sistema almacena reuniones de doce meses en el pasado	Must
RF-16.2	El sistema permite solo ver reuniones creadas como máximo dentro de dos meses.	Must
RF-17	El sistema permite al administrador ver los hitos a cumplir por los tándems.	Must
RF-17.1	El sistema muestra también las insignias ligadas a los hitos de la plataforma.	Should
RF-18	El administrador puede crear nuevos hitos en la plataforma.	Must
RF-18.1	El sistema no permitirá crear hitos que tengan el mismo nombre, descripción o color que otro hito previamente creado.	Must
RF-18.2	El sistema no permite crear hitos cuya imagen de insignia asociada exista previamente en la base de datos.	Would
RF-18.3	El sistema permite al administrador crear las encuestas de finalización de hitos.	Must
RF-19	El sistema permite modificar un hito creado en la plataforma	Must
RF-19.1	El sistema no permite modificar el nombre o la descripción de un hito que ha sido completado por algún tándem de la plataforma	Must
RF-19.2	El sistema no permite modificar la encuesta de finalización de hito si algún tándem ya ha contestado la encuesta.	Must
RF-20	El administrador puede acceder al foro general de la aplicación	Should
RF-20.1	El sistema permite al administrador publicar mensajes en el foro general.	Should
RF-21	El sistema permite al usuario desplegar el modal de notificaciones desde cualquier pantalla.	Must
RF-22	El sistema permite al usuario ver su(s) compañero(s) de tándem(s).	Must
RF-23	El usuario puede ver los hitos completados junto a su(s) tándem(s).	Must

RF-24	El sistema permite al usuario acceder a su perfil.	Must
RF-24.1	El usuario puede consultar sus datos personales y laborales desde su perfil.	Must
RF-24.2	El usuario puede ver las insignias ganadas durante su pertenencia a MENTality	Should
RF-25	El sistema permite al usuario acceder al perfil de su otro miembro del tándem.	Must
RF-25.1	El usuario puede conocer los datos personales y laborales de su compañero de tándem.	Must
RF-25.2	El usuario puede ver las insignias que ha ganado su compañero de tándem.	Should
RF-26	El sistema permite al usuario ver el calendario de reuniones para un mes.	Must
RF-26.1	El usuario puede acceder a su histórico de reuniones de doce meses en el pasado.	Must
RF-26.2	El usuario puede consultar las reuniones que tendrá en los próximos dos meses.	Must
RF-27	El sistema permite al usuario crear reuniones junto a su(s) tándem(s).	Must
RF-27.1	El sistema permitirá crear reuniones con un margen mínimo de una hora desde la hora en la que se intente crear la reunión y un máximo de dos meses en el futuro.	Must
RF-28	El usuario podrá acceder al listado de encuestas de reuniones.	Must
RF-28.1	El sistema solo mostrará las reuniones que ya hayan o deberían haber tenido lugar.	Must
RF-29	El usuario podrá responder las encuestas de reuniones y ver las respuestas dadas por él y su compañero de tándem.	Must
RF-29.1	El sistema no permitirá responder una encuesta que ya haya sido respondida y guardada en la base de datos.	Must
RF-29.2	El sistema permitirá al usuario visualizar las respuestas únicamente cuando hayan sido respondidas por el mismo.	Must
RF-30	El sistema permite al usuario ver la lista de hitos de la plataforma	Must
RF-30.1	El sistema muestra claramente una diferenciación entre los hitos no logrados, los hitos en proceso y los hitos completados	Must
RF-31	El usuario puede marcar los hitos que considera completados, teniendo en cuenta el sistema de doble check	Must
RF-31.1	El sistema no permitirá marcar como no completado o marcado por el usuario un hito que previamente estaba marcado por el usuario o marcado por ambos usuarios (completado) respectivamente.	Must
RF-32	El usuario podrá acceder a las encuestas de finalización de hitos de los hitos que estén marcados como completados.	Must
RF-32.1	El sistema permite al usuario rellenar y modificar las respuestas de la encuesta de finalización de hito.	Must
RF-32.2	El sistema muestra las respuestas dadas por el miembro del tándem.	Must
RF-33	El usuario puede acceder a la pizarra privada de tándem	Must

RF-33.1	El sistema permite al usuario publicar contenido en la pizarra privada.	Must
RF-34	El sistema permite al usuario acceder al foro exclusivo de su rol (mentor, *mentee*).	Could
RF-34.1	El usuario puede publicar mensajes en el foro exclusivo.	Could
RF-35	El sistema permite al usuario acceder al foro general de MENTality.	Should
RF-35.1	El usuario puede publicar mensajes en el foro general.	Should
RF-36	El sistema permite al usuario acceder al perfil de otros miembros de la plataforma.	Would

Los requisitos no funcionales se consideran imprescindibles en el sistema por ello, todos deberían contar con clasificación Must, pero debido a que no puede haber un requisito no funcional con otra clasificación en la columna de prioridad se marca con '-'.

RFN-1	El sistema debe cumplir con la normativa de protección de datos.	-
RFN-2	El sistema debe funcionar en los navegadores web modernos y actualizados.	-
RFN-3	La solución debe ser 100% Web y todas las operaciones deben realizarse desde el navegador.	-
RFN-4	El sistema debe disponer de conexión a internet.	-
RFN-5	El sistema debe tener capacidad para dar respuesta al acceso simultáneo de los usuarios registrados con un tiempo de respuesta aceptable incluso cuando el nivel de demanda sea alto.	-
RFN-6	El sistema debe contar con un sistema de identificación seguro.	-
RFN-7	El sistema debe ser de fácil uso y entendimiento por parte de los usuarios, con una curva de aprendizaje suave.	-
RFN-8	El sistema debe dar a conocer al usuario su ubicación dentro de la aplicación.	-
RFN-9	El sistema contará con una interfaz atractiva y fluida.	-
RFN-10	El sistema debe poseer mensajes de error personalizados para la identificación y localización de errores durante las fases de desarrollo, pruebas y operación.	-
RFN-11	Los backups de la base de datos son responsabilidad de la administración de la plataforma que deberá crearlos, almacenarlos y recuperar la información si fuera necesario. Se podrán usar los backups ofrecidos por FireBase si se quiere centralizar el control de la base de datos en dicha plataforma.	-

8 - Bibliografía

Referencias

[1] Grady Booch, Robert Maksimchuk, Michael Engle, Bobbi Young, Jim Conallen y Kelli Houston. *Object-Oriented Analysis and Desing with Applications, 3rd Edition.* Publicado en 2007 por Addison-Wesley Proffesional

[2] Erich Gamma, Richard Helm, John Vlissides y Grady Booch. *Design Patterns: Elements of Reusable Object-Oriented Software*. Publicado en 1994 por Addison-Wesley Professional

Formación en React y FireBase

[3] React y testing

[4] JavaScript Moderno

[5] React y FireBase

Páginas usadas para desarrollo de la plataforma

[6] Picker de Colores en hexadecimal

[7] Tutorial React-Router-Dom

[8] Hook de autenticación

[9] Personalizar Calendario de MUI

[10] Crear base de datos en FireBase

[11] Subir imagen a FireBase Storage

[12] Enviar correos desde React

[13] Creación selector de color – anteriormente poseía un ejemplo de usos del componente, pero actualmente está caído el servidor que lo alojaba.

Librerías utilizadas en el proyecto

[14] React v18.3.1: API de React

[15] Librería de componentes UI: Material UI - Web oficial

[16] Librería de componentes UI: Material UI/Lab

[17] Componente selector de color: rc-components/color-picker

[18] Envío de correos con servidor local: nodemailer

[19] Gestión de rutas para solicitudes HTTP: express

[20] Permitir intercambio de recursos entre dominios: cors

[21] Comunicación base de datos FireBase: FireBase (funciones usadas: app, firestore, storage)

[22] Enrutado de componentes: React Router Dom

[23] Tratamiento de fechas: dayjs

[24] Iconos extras: React Icons (se usó solo Ionicons 5 al tener licencia MIT)

[25] Notificaciones y avisos: React Toastify

[26] Creación optimizada de formularios: React Hook Form