Este proyecto de Fin de Grado se inicia con el objetivo de desarrollar una aplicación multiplataforma para gestionar los datos recopilados por el dispositivo IoT Tree Inspection Kit. La aplicación contará con un sistema de gestión de usuarios que permitirá organizar la información recopilada sobre diferentes especies de árboles a través de fichas de datos.

Esta aplicación está planteada en dos partes: frontend y backend. El frontend se encargara de la interfaz de usuario y la interacción con el usuario final. Mientras tanto, el backend implementado en la nube, se encargara de procesar las solicitudes del cliente y gestionar los datos.

Luis Javier Soriano Suárez es un Ingeniero Informático de Granada, España. Es el autor y desarrollador del presente proyecto.

AndrésMaría Roldán Aranda es el responsable académico del presente proyecto y el tutor del estudiante. Es profesor en el Departamento de Tecnologías de Electrónica y Computadoras.

TRABAJO FIN DE GRADO

Aplicación en Flutter para gestión de equipo IOT

Luis Javier Soriano Suárez

GRADO EN INGENIERÍA INFORMATICA

UNIVERSIDAD DE GRANADA

Grado en Ingeniería Informática

Aplicación en Flutter para gestión de equipo IOT

Luis Javier Soriano Suárez

2022/2023

Tutor: Andrés María Roldán Aranda

Impreso en Granada, Junio 2023.

“Aplicación en Flutter para gestión de equipo IOT”

GRADO EN INGENIERÍA INFORMÁTICA

Trabajo fin de grado

"Aplicación en Flutter para gestión de equipo IOT"

AUTOR:

Luis Javier Soriano Suárez

DIRECTOR:

Prof. Andrés María Roldán Aranda

DEPARTAMENTO:

Electrónica y Tecnología de Computadores

“Aplicación en Flutter para gestión de equipo IOT”

Luis Javier Soriano Suárez

PALABRAS CLAVE: *dispositivo IoT, aplicación multiplataforma, gestión de usuarios, proyectos, almacenamiento en la nube, despliegue, captura de imágenes, geoposicionamiento, exportación en Excel*

RESUMEN:

Tras el desarrollo del dispositivo Tree Inspection Kit (TIK), creado por un miembro del grupo GranaSAT, surge la necesidad de contar con una aplicación que permita gestionar los datos recopilados por este dispositivo IoT. Con este propósito, se inicia este proyecto con el objetivo de desarrollar una aplicación multiplataforma para satisfacer esta necesidad. La aplicación contará con un sistema de gestión de usuarios que permitirá organizar la información recopilada sobre diferentes especies de árboles mediante proyectos. Además, se aprovechará el almacenamiento en la nube gracias al despliegue implementado, evitando así el almacenamiento local. Entre las funcionalidades que se incluirán encontramos la captura de imágenes, geoposicionamiento, exportación de proyectos en Excel y gestión de perfil del usuario.

“Aplicación en Flutter para gestión de equipo IOT”

Luis Javier Soriano Suárez

KEYWORDS: *IoT device, multi-platform application, user management, projects, cloud storage, deployment, image capture, geopositioning, Excel export*

ABSTRACT:

Following the development of the TIK device, created by a member of GranaSAT group, there arises a need for an application that allows managing the data collected by this IoT device. As a result, this project is initiated with the goal of developing a versatile application that fulfills this requirement. The application will include a user management system, enabling the organization of collected information on different tree species through project-based organization. Additionally, cloud storage will be utilized to avoid relying on local storage. Exciting features of the application will include image capture, geopositioning, project export to Excel, and user profile management.

Agradecimientos

A mi madre, por todo el sacrificio y dedicación que ha puesto en su familia; por todo el aprecio y amor que ha vertido en sus dos hijos; por los valores invaluables que me ha inculcado a lo largo de mi vida.

A mi padre, por el trabajo sin descanso para que a ninguno de sus hijos les falta nada; por mostrarnos que todo en la vida tiene su recompensa; por cuidarnos y estar cerca de nosotros a pesar de las circunstancias.

A mi hermano, por ser un ejemplo a seguir; por acompañarme, aconsejarme y el apoyo brindado durante todos estos años; por enseñarme y demostrarme que el trabajo duro vence al talento cuando el talento no se esfuerza.

A ellos, les debo todo.

No podría olvidarme de mi gran amigo Adrián, por su ayuda, consejos y modo de ver la vida; por estar presente en los momentos duros; por mostrarme que más allá de la familia, puedes encontrar a personas que se convierten en parte integral de tu vida.

Índice general

Índice de figuras

Índice de Tablas

Glosario

A | B | C | D | E | F | G | I | J | L | M | P | R | S | T | W | X

A

API Siglas de Application Programming Interface (Interfaz de Programación de Aplicaciones). Se trata de un conjunto de reglas y protocolos que permiten la interacción entre diferentes aplicaciones de software.

B

Backend Parte de un sistema o aplicación web que se encarga del procesamiento y la lógica detrás de escena.

C

Cloudinary Plataforma en la nube para la gestión y optimización de imágenes y vídeos. https://cloudinary.com .

D

Dart Lenguaje de programación desarrollado por Google. Es un lenguaje de alto nivel y de tipado estático que se utiliza principalmente para el desarrollo de aplicaciones móviles y web.

E

Endpoint Dirección específica que representa una función o operación en el servicio. Los endpoints definen cómo interactuar con la API para realizar acciones como obtener datos, crear registros, actualizar información o eliminar recursos.

Express Framework de desarrollo web para Node.js. Es una herramienta popular que permite construir aplicaciones web y APIs de manera rápida y sencilla .

F

Fakopp Empresa fundada en 2005 con el fin de desarrollar, producir y analizar para el mercado equipos para la silvicultura y las industrias madereras. https://fakopp.com/es/about/.

Framework Estructura o conjunto de herramientas predefinidas que facilita la creación de aplicaciones.

Frontend Parte visible y interactiva de un sistema o aplicación web que los usuarios ven y con la que interactúan.

G

GitHub Plataforma de alojamiento de repositorios de código fuente basada en la nube. https://github.com/.

GitLab Plataforma de gestión de repositorios de código fuente basada en Git. https://gitlab.com/.

GranaSAT *Grupo de Electrónica Aeroespacial*. Un proyecto académico de la UGR (Universidad de Granada). Esta organización cuenta con un laboratorio de electrónica donde estudiantes de diferentes carreras y niveles educativos desarrollan proyectos multidisciplinarios https://granasat.ugr.es .

I

InnoSetup Herramienta de código abierto utilizada para crear instaladores de aplicaciones para Windows. https://jrsoftware.org/isinfo.php .

IoT Sistema de interconexión digital en el que objetos cotidianos están conectados a Internet y pueden comunicarse y compartir datos .

J

Jira Plataforma de gestión de proyectos y seguimiento de problemas que utiliza la metodología ágil de SCRUM. https://www.atlassian.com/software/jira.

L

LaTex Sistema de composición de documentos utilizado ampliamente en la escritura académica y científica. LaTeX se basa en el lenguaje de marcado TeX y proporciona un conjunto de comandos y macros que permiten crear documentos con un formato profesional y de alta calidad .

M

Microsecond Timer Dispositivo IoT creado por FAKOPP diseñado para medir el tiempo de propagación de la onda de esfuerzo en los árboles.

Microsecond Timer App Aplicación creada por FAKOPP encargada de almacenar y gestionar los datos que se obtienen a través del dispositivo Microsecond Timer.

Middleware Función que se ejecuta en el flujo de solicitudes y respuestas HTTP, permitiendo realizar tareas como validación de datos, autenticación, gestión de sesiones y seguridad.

Mockup Representación visual estática o interactiva de una interfaz de usuario o diseño de un producto, como una aplicación móvil o un sitio web.

MOE Es el módulo de elasticidad de los árboles, una medida de la rigidez de la madera frente a la deformación bajo carga. Cuanto mayor sea su valor, más rígida será la madera. Es una propiedad importante en el diseño de estructuras de madera debido a su influencia en la resistencia y estabilidad de las construcciones. .

MongoDB Sistema de gestión de bases de datos NoSQL que se caracteriza por su enfoque en la escalabilidad, flexibilidad y rendimiento. https://www.mongodb.com/.

MongoDB Atlas Servicio de base de datos en la nube que ofrece MongoDB, un sistema de gestión de bases de datos NoSQL. https://www.mongodb.com/atlas/database.

P

Pre-hook Función o middleware que se ejecuta antes de una operación específica en una base de datos MongoDB.

Provider Widget que permite compartir y acceder a datos entre diferentes widgets de manera eficiente.

R

Rive Plataforma en línea que permite crear y animar gráficos vectoriales 2D de manera interactiva. https://rive.app .

S

SCRUM Marco de trabajo ágil utilizado en el desarrollo de software y gestión de proyectos. Se basa en la colaboración, la transparencia y la adaptabilidad.

T

Tree Inspection Kit App Aplicación desarrollada para este proyecto, la cual se encarga de manejar los datos que han sido obtenido por el dispositivo TIK .

W

Widget Objetos inmutables de Flutter que describen cómo se debe renderizar y comportar un elemento en la pantalla. Pueden representar elementos visuales, como botones o imágenes, o conceptos abstractos.

X

XCode Entorno de desarrollo integrado utilizado para el desarrollo de aplicaciones para dispositivos Apple, como iPhone, iPad, Mac y Apple Watch .

Siglas

B | **C** | **G** | **T**

B

BBDD Base de datos.

C

CRUD Create, Read, Update, Delete. Son las operaciones básicas que se realizan en un sistema o aplicación que interactúa con una base de datos. En español: Crear, Leer, Actualizar, Eliminar.

G

GPS Global Positioning System. Sistema de Posicionamiento Global.

T

TIK Tree Inspection Kit.

Capítulo 1

Introducción

Este Trabajo Final de Grado se presenta como el resultado del conocimiento adquirido en el Grado en Ingeniería Informática y es enfocado al desarrollo de una aplicación móvil para la gestión de los datos obtenidos de un dispositivo IoT. El dispositivo IoT, creado por Juan Del Pino Mena y actualmente mantenido por Irene Gil Martín, será el protagonista del proyecto en general y sobre el cual gira la aplicación desarrollada. Dicho dispositivo, a partir de ahora llamado TIK, es el encargado de encontrar el Módulo de Elasticidad (MOE) de árboles en pie, troncos y tablas de madera.

El MOE, también conocido como Módulo de Elasticidad o Módulo Elástico, que explicado brevemente se refiere, a la capacidad de resistencia de un objeto o sustancia ante la deformación elástica al aplicar una fuerza. Esta deformación elástica es una alteración temporal que aparece al someter el objeto a estrés y desaparece instantáneamente al remover la fuerza (a diferencia de la deformación plástica). El MOE proporciona una indicación de la rigidez de un material: a mayor MOE, mayor será su rigidez. En el caso de la madera, la rigidez es uno de los marcadores de calidad más relevantes en aplicaciones estructurales.

El objetivo principal de este proyecto es proporcionar una aplicación como solución eficiente y fácil de usar para el usuario, que sea capaz de gestionar lo datos ofrecidos por el dispositivo TIK, aprovechando las capacidades y funcionalidades ofrecidas por las tecnologías móviles.

Este Trabajo ha sido realizado en el marco de la Acción C4 (Desarrollo de herramienta Tree Inspection Kit para clasificación no destructiva de la madera) del proyecto europeo LIFE 20 CCM/ES/001656 Recuperación de las alamedas de la Vega de Granada para la mejora de la biodiversidad y el secuestro de carbono a largo plazo en bioproductos estructurales.

(a)Logo del programa LIFE

(b)Logo del proyecto LWFF

Figura 1.1 – *Logos del programa LIFE y proyecto LWFF*

1.1. Motivación

1

Situándonos previamente en el desarrollo del dispositivo TIK, su creación surge en el contexto de los graves problemas de contaminación existentes en el área metropolitana de Granada.

Según informes de calidad del aire en España, Granada se sitúa detrás de Madrid y Barcelona en términos de contaminación. La ubicación geográfica de Granada, rodeada por Sierra Nevada, dificulta la ventilación natural y favorece la inversión térmica. Sin embargo, el problema no se limita únicamente a la geografía, ya que actividades humanas altamente contaminantes, como el alto tráfico de vehículos, el uso de sistemas de calefacción contaminantes en invierno y la quema de residuos de poda y rastrojos, no están siendo abordadas de manera efectiva.

Esto a su vez se suma a que en los últimos años, la silvicultura de álamos en la Vega de Granada ha disminuido considerablemente debido a la falta de competitividad económica en comparación con otros cultivos. Sin embargo, los álamos ofrecen importantes beneficios ambientales, como la capacidad de capturar carbono, mejorar la calidad del aire, conservar la calidad del suelo y regular el ciclo del agua.

Con la situación dada en la capital, surge una oportunidad para desarrollar una industria local de madera estructural en la Vega de Granada, con un enfoque en reducir la huella ecológica y actuar como un filtro de absorción de contaminantes para el área metropolitana. Para lograr este paso, es fundamental contar con madera de calidad y aprovecharla de manera eficiente. La calidad de la madera está directamente relacionada con sus propiedades, como la densidad, la orientación de las fibras, la rigidez y la resistencia mecánica.

Agregamos también que, la legislación actual de la Unión Europea sobre eficiencia energética en la construcción, promueve el uso de la madera como material clave en la construcción sostenible y en edificios de consumo de energía casi nulo. La incorporación de materiales de madera en estructuras mediterráneas ofrece beneficios en términos de rendimiento invernal, costos, comodidad y reducción del impacto ambiental.

Dado este contexto, se satisfizó la creación del dispositivo TIK, una herramienta adecuada para caracterizar la madera y así lidiar con los problemas expuestos que hay en la capital de Granada.

Tras el desarrollo del prototipo, surge una nueva necesidad, la cual se basa en crear un sistema para poder realizar un seguimiento de los datos obtenidos con el dispositivo.

Hasta este momento cualquier dato obtenido, se almacenaba en la aplicación comercial (Microsecond Timer App) del dispositivo precursor al TIK. Dicha aplicación, tiene una interfaz bastante sencilla, atrasada en cuanto a diseño y sin posibilidad de gestión de proyectos por usuarios. Todos los datos que se almacenan, lo hacen de forma local, contando únicamente con la posibilidad de exportar los datos a Excel. Sin embargo cualquier problema derivado, como podría ser la pérdida del dispositivo móvil, rotura del mismo, mal-funcionamiento... haría que todos los datos tomados hasta el momento se perdieran.

Esto, a la hora de una investigación o análisis de una plantación, puede llegar a ser crítico y por tanto un riesgo que no se puede llegar a tomar. Para ello se plantea la creación de Tree Inspection Kit App, una aplicación multiplataforma para dispositivos móviles, gestionada por usuarios y desplegada en la nube.

Figura 1.2 – *Logo TIK App*

1.2. Objetivos y metas del proyecto

El objetivo general es la creación de una aplicación multiplataforma, que gestione los datos en la nube y que cubra las necesidades de un usuario que opera el dispositivo TIK. Para cumplir con dicha finalidad, a continuación, se describen los siguientes objetivos primarios y secundarios del proyecto:

- **Objetivos primarios:**
  1. Crear una aplicación cliente que haga el rol de frontend para el usuario.
  2. Desplegar un servidor gestionado en la nube, que tenga como rol ser el backend, para poder almacenar los datos que son requeridos en la aplicación por los usuarios. Dicho backend, tendrá que gestionar diversas operaciones CRUD para manipular acciones requeridas, ya sean en los propios usuarios o bien en los datos almacenados.
  3. La aplicación deberá ser funcional, tanto en Android como en iOS, para que el uso pueda ser generalizado.
- **Objetivos secundarios:**
  1. Ofrecer una interfaz intuitiva y atractiva al usuario sin que resulte monótona.
  2. Incorporar elementos animados con el fin de dotar a la aplicación de mayor dinamismo y vivacidad.
  3. Realizar la compilación de la aplicación para la plataforma Windows, permitiendo así la consulta de los proyectos desde un equipo de sobremesa.

1.3. Estructura de la memoria

- **Capítulo 1. Introducción.** Breve introducción del proyecto final, en donde se describe la necesidad actual de desarrollar una aplicación móvil para gestionar los datos que captura el dispositivo TIK.
- **Capítulo 2. Estado del arte.** Análisis de la aplicación comercial existente.
- **Capítulo 3. Gestión y planificación.** Explicación de metodología ágil utilizada durante el proyecto, así como la especificación de tareas, requisitos, recursos y estimación de costes del mismo.
- **Capítulo 4. Diseño e implementación frontend.** Explicación de la implementación del frontend de la aplicación, mencionando las tecnologías utilizadas y las funcionalidades que se han desarrollado.
- **Capítulo 5. Diseño e implementación backend.** Explicación de la implementación del backend de la aplicación, mencionando las tecnologías utilizadas y las funcionalidades que se han desarrollado.
- **Capítulo 6. Despliegue.** Explicación del despliegue de la base de datos, backend y otros servicios en la nube para el correcto funcionamiento de la aplicación.
- **Capítulo 7. Conclusiones y trabajos futuros.** Conclusiones tras el desarrollo del proyecto de la aplicación móvil, así como posibles trabajos futuros que se podrían implementar en el software.

Capítulo 2

Estado del arte

Una vez que el proyecto ha sido presentado y sus objetivos han sido establecidos, es esencial realizar un análisis del estado del arte. El objetivo de este análisis, es investigar la aplicación de Android ya existente, llamada Microsecond Timer App, desarrollada por Fakopp. Durante el estudio, se examinarán las características y funcionalidades de esta aplicación, así como su rendimiento y usabilidad.

2.1. Caso de estudio: Microsecond Timer App

Fakopp es una empresa dedicada a desarrollar equipos de prueba para la industria forestal y la madera, con más de 25 años de experiencia.

Unos de sus productos desarrollados que podemos encontrar y el que más nos interesa, es el dispositivo Microsecond Timer, precursor del dispositivo TIK. Junto con el desarrollo de este producto, dicha empresa creo una aplicación exclusiva para el sistema operativo de Android, llamada **Microsecond Timer App**.

Figura 2.1 – *Logo de la Empresa Fakopp*

2.1.1. Pantalla principal

Iniciando la aplicación nos encontramos directamente con el menú principal. No se requiere de un inicio de sesión para poder acceder a dicho menú.

Figura 2.2 – *Pantalla principal - Microsecond Timer App*

Podemos observar que tenemos cuatro opciones disponibles:

• **Nuevo proyecto.** Se abre un diálogo que nos pide introducir el nombre del proyecto que deseamos. Por defecto, será la fecha y hora del día actual. Al crear el proyecto nos redirige directamente a la pantalla de fichas del proyecto 2.7 (Pantalla fichas - Microsecond Timer App).

• **Abrir proyecto.** Se abre una diálogo que nos pide elegir el proyecto que deseamos consultar, una vez seleccionado el proyecto, nos redirige a la pantalla de fichas del seleccionado 2.7 (Pantalla fichas - Microsecond Timer App), si tiene fichas disponibles se nos abre la primera de ellas, si no, se abre una ficha vacía para su relleno de datos.

• **Compartir proyecto.** 2.3 (Pantalla exportación - Microsecond Timer App) En dicha opción se nos abre un desplegable para poder seleccionar el proyecto que deseamos exportar a Excel y mandar a quién deseemos en las plataformas que tengamos acceso para compartir ficheros.

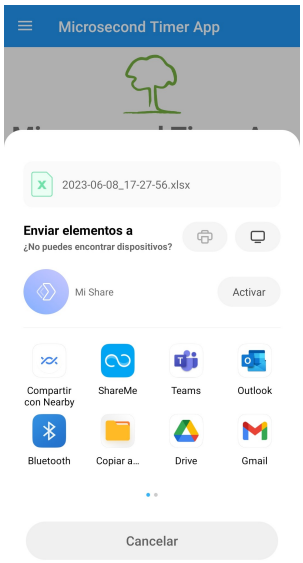


Figura 2.3 – *Pantalla exportación - Microsecond Timer App*

- **Abrir manual.** Esta opción únicamente hace que se abra un PDF en el visor que tengamos por defecto para poder mostrarnos el manual de uso de la aplicación.

2.1.1.1. Pantalla desplegable

En esta pantalla podremos encontrar las opciones de “Ajustes”, “Bluetooth” y “Acerca de”.

Figura 2.4 – *Pantalla desplegable - Microsecond Timer App*

- **Ajustes.** Dicho apartado, permite cambiar el idioma, alternar el sistema de medición (entre el

Sistema Internacional y el **Sistema Imperial**).

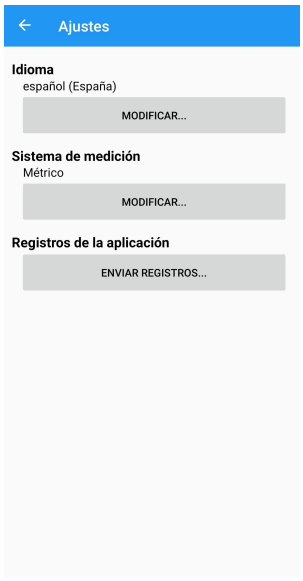


Figura 2.5 – *Pantalla ajustes - Microsecond Timer App*

- **Bluetooth.** Apartado para poder buscar dispositivos bluetooth y poder conectarse con el dispositivo comercial, en este caso, el dispositivo Microsecond Timer.

Figura 2.6 – *Pantalla bluetooth - Microsecond Timer App*

- **Acerca de.** Información genérica de la aplicación, así como información de contacto con la empresa Fakopp.

2.1.2. Pantalla fichas

Podemos acceder a esta pantalla, haciendo click en “Nuevo proyecto” o “Abrir proyecto”, seleccionando el deseado. Si nos fijamos, en la parte superior, podemos ver que tenemos un menú de navegación para poder movernos entre las fichas que forman el proyecto. Tenemos un botón para crear una nueva ficha y un botón para eliminar la ficha actual.

Cada ficha contendrá los siguientes elementos:

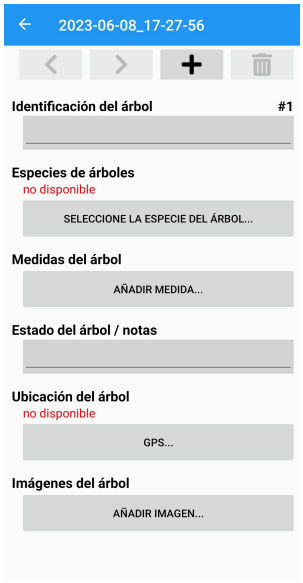


Figura 2.7 – *Pantalla fichas - Microsecond Timer App*

- **Identificador de árbol.** Se necesita especificar para poder identificar cada una de las fichas que componen el proyecto.
- **Especies de árbol.** Se necesita especificar la especie del árbol del cual se está rellenando la ficha de datos.
- **Medidas del árbol.** Apartado para introducir las medidas tomadas que nos devuelve el dispositivo, en este caso, el Microsecond Timer.
- **Estado del árbol.** Sección donde rellenar cualquier anotación derivada del árbol que se está estudiando.
- **Ubicación del árbol.** Botón que nos devuelve los valores numéricos de las coordenadas en donde nos encontramos exactamente, utilizando el GPS del dispositivo móvil.
- **Imágenes del árbol.** Botón el cual abre la cámara para poder tomar una foto del árbol que se está estudiando y poder almacenarla junto a la ficha de datos.

2.2. Planteamientos acerca de Microsecond Timer App

Después de presentar las diferentes pantallas que componen la aplicación comercial y de hacer uso de la misma para poder probar todas sus funcionalidades, se obtienen los siguientes planteamientos:

- No se presenta ningún problema a la hora de hacer uso de ella, funciona con fluidez y responde de manera adecuada.
- Puede generar la sensación de tener un aspecto desactualizado, dando la impresión al usuario de no ser una aplicación mantenida periódicamente por sus desarrolladores. Independientemente de si es mantenida o no, este problema viene dado por el diseño y paleta de colores de la interfaz de usuario.
- A la hora de entrar en un proyecto, nos dirige a la primera ficha de datos de árbol que se agregó. Esto implica que, si queremos movernos a la última ficha del proyecto agregada, tendremos que pasar por todas las fichas que fueron añadidas anteriormente. Esto puede resultar algo tedioso para el usuario si nos encontramos con un proyecto que lo componen muchas fichas de datos.
- Se echa en falta información a la hora gestionar un proyecto. Por ejemplo, poder editar el nombre de este una vez ha sido creado, agregar algún tipo de anotación que haga referencia al proyecto en sí, etc.
- Una vez ingresamos en las fichas de datos, se entra en modo edición, es decir, carece de un modo que sea solamente consulta.
- A la hora de agregar las coordenadas, solo nos indica la precisión de la obtención de estas junto con sus valores correspondientes a la latitud y longitud. Se echa en falta alguna referencia gráfica, como podría ser un mapa.
- Cuando se guarda una imagen en una ficha de datos, para poder consultarla se debe hacer click nuevamente en el botón “Añadir imagen...”. Esto puede resultar poco intuitivo para el usuario, se echa en falta que la imagen se muestre directamente en la ficha de datos.
- Compatible exclusivamente con Android. Esto hace que su uso sea único para usuarios con un teléfono con dicho S.O. Se echa en falta la posibilidad de trabajar en otros sistemas como iOS, Windows, Linux...
- Como adición a los puntos anteriores, para comodidad del usuario, el manual debería poder consultarse desde la propia aplicación, evitando que el usuario necesite tener instalada una aplicación dedicada los formatos de texto como PDFs.

Descritos los puntos anteriores, podemos observar que la aplicación es completamente funcional pero hay varios puntos que se podrían mejorar, haciendo su uso más cómodo.

Capítulo 3

Gestión y planificación

3.1. Metodología de desarrollo

A la hora de planificar un proyecto dentro del ámbito del software, son gran conocidas y aliadas, las metodologías de desarrollo. Dichas metodologías ayudan a desarrolladores a estimar los tiempos y recursos que se va a necesitar para poder llevar a cabo las tareas asignadas dentro de un proyecto.

De todas las metodologías ágiles que hay, nos centraremos en la conocida como **SCRUM**, ya que ha sido la utilizada para este proyecto.

3.1.1. SCRUM

A día de hoy, es altamente utilizada en la industria de proyectos software, teniendo a grandes gigantes tecnológicos que la respaldan, como es el caso de Google, Facebook, AT&T, Nike...

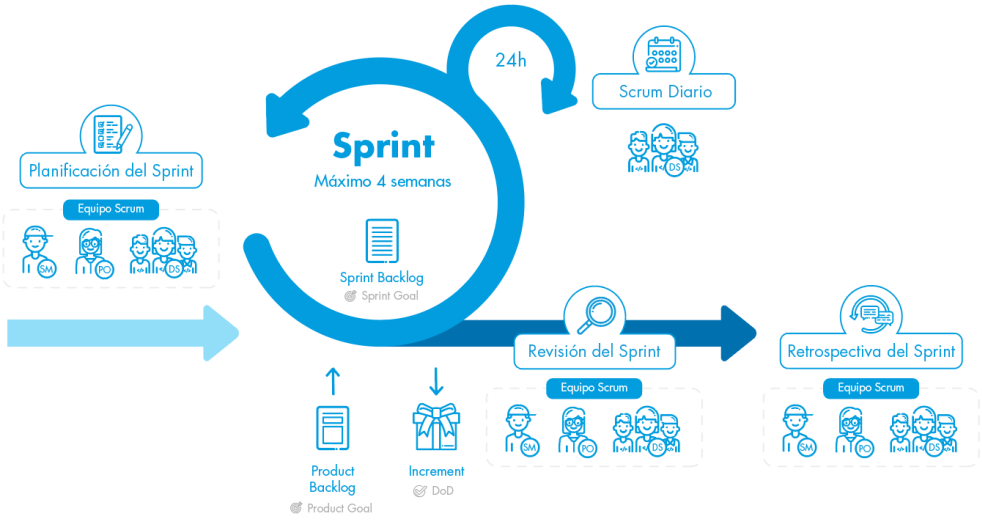


Figura 3.1 – *Esquema metodología ágil - SCRUM*

¿Qué es?[4] Como bien se ha mencionado, es una metodología ágil y flexible que nos permite gestionar el desarrollo de software, cuyo principal objetivo es brindar al proyecto de un plan **adaptativo**, **transparente** y **colaborativo**. Se enfoca en entregar valor al cliente de manera incremental, permitiendo la entrega de funcionalidades en intervalos regulares y obteniendo retroalimentación temprana.

A dichos intervalos de tiempo regulares, se les llaman **Sprints**. Suelen ser tiempos comprendidos de entre una a cuatro semanas en los cuales se tiene un objetivo, más o menos razonable, que cumplir. Este objetivo, lo compondrán nuevas funcionalidades o mejoras en el producto que se incorporan mediante las llamadas tareas.

Teniendo en cuenta el concepto de Sprint, **SCRUM** está compuesto por los siguientes elementos:

- **Planificación de proyecto**
  - **Lista de Producto (Product Backlog)**. Esta es una lista priorizada de todas las características, requisitos y mejoras planificadas para el producto. Priorizada y mantenida por el Product Owner, es utilizada para planificar y ejecutar sprints.
  - **Lista del Sprint (Sprint Backlog)**. Esta es una lista que la componen tareas seleccionadas del Product Backlog, pensadas para ser completadas durante el Sprint. Se utiliza para poder realizar un seguimiento del progreso del Sprint y es creada por el Equipo de Scrum durante la reunión llamada Sprint Planning.
- **Planificación de equipo**
  - **Planificación del Sprint (Sprint Planning)**. Es una reunión en la cual el Scrum Team selecciona funcionalidades y tareas, los cuales los miembros del equipo se comprometen a completar durante el sprint. Se establecen los objetivos del sprint y se realiza una estimación inicial del trabajo.
  - **Reunión diaria de Scrum (Daily Scrum**. Es una breve reunión diaria en la cual el Scrum Team comparte el progreso, discute los obstáculos y se sincroniza.
  - **Revisión del Sprint (Sprint Review)**. Es una reunión que se realiza al final del Sprint, en la que el Scrum Team, lo desarrollado en el producto durante el Sprint a las partes interesadas. Se recibe retroalimentación y se realiza una demostración de las funcionalidades completadas.
  - **Retrospectiva del Sprint (Sprint Retrospective)**. Es una reunión realizada al final del Sprint en la cual sirve como reflexión para el Scrum Team acerca del desempeño durante el Sprint. Se analizan éxitos, desafíos y las áreas de mejora, y se definen cambios para mejorar en el siguiente sprint.
- **Roles**
  - **Propietario del Producto (Product Owner)**. Es el representante del cliente o las partes interesadas, responsable de definir y priorizar los elementos del producto en el Product Backlog. Trabaja en colaboración con el Team Scrum para elaborar los requisitos que han sido pedidos por el cliente y asegurar el valor del producto final.
  - **Scrum Master**. Es el responsable de promover los valores y uso efectivo de SCRUM. Ayuda al equipo a entender y aplicar Scrum, fomenta un entorno de trabajo colaborativo y elimina obstáculos que puedan afectar su rendimiento. El Scrum Master no es el líder del equipo, sino un facilitador.
  - **Equipo de Scrum (Scrum Team)**. Es un conjunto de profesionales, encargados de entregar el incremento del producto al final de cada Sprint. La cantidad de miembros de este equipo oscilan entre las 5-9 personas.

3.1.2. Aplicación de SCRUM al proyecto

Dado a que el desarrollo del proyecto se centra alrededor de un único integrante, se deben aplicar cambios a la metodología SCRUM para poder ser adaptativa al desarrollo mencionado. Las consideraciones que serán tomadas son:

- Las reuniones diarias (Daily Scrum) serán suprimidas al ser una sola persona el equipo de desarrollo.
- El único desarrollador asume todos los roles existentes en la metodología.
- Lo cambios y funcionalidades nuevas implementadas al producto, serán informadas por los canales habilitados al tutor del proyecto.

3.1.3. Especificación de tareas y requisitos

En cualquier proyecto de desarrollo software, es importante establecer los requisitos necesarios para satisfacer por completo las necesidades de los usuarios. Dado que estamos siguiendo la metodología ágil de SCRUM, la definición de requisitos se realizará mediante historias de usuario. Algunas de estas historias de usuarios han sido divididas en sub-historias de usuario.

ID	HU.1 - Inicio de sesión y registro de usuario
Descripción	Como usuario quiero poder registrarme e iniciar sesión en la aplicación para su uso
Criterio aceptación	- Con un registro adecuado el usuario debe registrarse en la BBDD - Con un inicio de sesión correcto, se debe redirigir al usuario a la pantalla principal - Cualquier operación, sea correcta o incorrecta, debe informase al usuario
Tareas	- Crear modelo de datos en frontend para el usuario - Crear modelo de datos en backend para el usuario - Crear formularios para el registro e inicio de sesión - Agregar validaciones para registro y login - Permitir cerrar sesión al usuario - Establecer avisos al completar operaciones - Generar rutas para las operaciones CRUD en el servidor - Implementar las llamadas al servidor en el frontend

Tabla 3.1 – *HU.1 - Inicio de sesión y registro de usuario*

ID	HU.1.1 - Editar perfil de usuario
Descripción	Como usuario quiero poder editar mi perfil dentro de la aplicación.
Criterio aceptación	- El sistema debe permitir al usuario cambiar el nombre utilizado - El sistema debe permitir al usuario cambiar el correo utilizado por uno nuevo - El sistema debe permitir al usuario cambiar su contraseña para el próximo acceso
Tareas	- Crear formulario para edición del perfil - Añadir validaciones al formulario y sus campos - Generar rutas para las operaciones CRUD en el servidor - Implementar las llamadas al servidor en el frontend

Tabla 3.2 – *HU.1.1 - Editar perfil de usuario*

ID	HU.2 - Accesibilidad en la aplicación
Descripción	Como usuario quiero poder tener acceso a la aplicación desde varias plataformas y que este operativa en todo momento.
Criterio aceptación	- La aplicación debe compilar y funcionar correctamente en el S.O Android - La aplicación debe compilar y funcionar correctamente en el S.O iOS - La aplicación debe compilar y funcionar correctamente en el S.O Windows para consultar información - El backend debe estar desplegado en la nube para la disponibilidad de este
Tareas	- Configurar el entorno de desarrollo para admitir la compilación en las plataformas - Desarrollar y probar las funcionalidades en los sistemas mencionados - Realizar pruebas de compatibilidad en diferentes dispositivos y resoluciones de pantalla - Alojar servidor backend en plataforma on-line

Tabla 3.3 – *HU.2 - Accesibilidad en la aplicación*

ID	HU.3 - Ajustes en la aplicación
Descripción	Como usuario quiero poder cambiar entre idiomas en la aplicación.
Criterio aceptación	- La aplicación debe compilar y funcionar correctamente independientemente del idioma seleccionado - Se debe permitir la selección entre varios idiomas - La traducción de idioma se debe hacer de manera local no en línea
Tareas	- Configurar librería de idiomas localmente - Crear plantillas para los idiomas requeridos - Crear opción en frontend para cambiar de idioma

Tabla 3.4 – *HU.3 - Ajustes en la aplicación*

ID	HU.4 - Organización por proyectos
Descripción	Como usuario quiero poder organizar la información obtenida en proyectos.
Criterio aceptación	- La aplicación debe permitir crear proyectos para organizar la información - Se debe permitir al usuario acceder únicamente a los proyectos creados por este mismo - La aplicación debe permitir la edición de los campos que componen los proyectos - Se debe permitir el borrado de proyectos y esto elimine la información asociada
Tareas	- Crear modelo de datos en frontend para el proyecto - Crear modelo de datos en backend para el proyecto - Crear listado para selección de proyecto - Ligar proyecto creado al usuario que lo creó - Generar rutas para las operaciones CRUD en el servidor - Implementar las llamadas al servidor en el frontend

Tabla 3.5 – *HU.4 - Organización por proyectos*

ID	HU.5 - Fichas de datos
Descripción	Como usuario quiero poder almacenar toda la información que reúno de un árbol en fichas de datos y poder exportar dicha información.
Criterio aceptación	- La aplicación debe permitir crear fichas una vez dentro de un proyecto - Se debe permitir al usuario acceder únicamente a las fichas a través de un proyecto previamente seleccionado - La aplicación debe permitir la edición de los campos que componen las fichas de datos - Se debe permitir el borrado de fichas y esto elimine la información asociada
Tareas	- Crear modelo de datos en frontend para la ficha - Crear modelo de datos en backend para la ficha - Crear listado para selección de ficha deseada - Ligar ficha creada al proyecto - Permitir exportar en Excel la información de la fichas - Generar rutas para las operaciones CRUD en el servidor - Implementar las llamadas al servidor en el frontend

Tabla 3.6 – *HU.5 - Fichas de datos*

ID	HU.5.1 - Campos en fichas de datos
Descripción	Como usuario quiero almacenar información específica en la ficha del árbol. Dicha información debe cubrir un identificador, la especie de árbol, anotaciones, la geoposición de este, imágenes y lo más importante, las medidas tomadas.
Criterio aceptación	- La aplicación debe permitir poder agregar todos los campos que el usuario necesita - La aplicación debe permitir la edición de los campos
Tareas	- Crear formulario para contener los campos - Crear modelo de datos en frontend para las especies de arboles - Crear modelo de datos en backend para las especies de arboles - Crear listado de especies de arboles para selección del usuario - Crear funcionalidad para obtener geoposicion y mostrar coordenadas en un mapa - Crear funcionalidad para la toma de imágenes - Crear tabla para agregar los datos tomados del árbol - Generar rutas para las operaciones CRUD en el servidor - Implementar las llamadas al servidor en el frontend

Tabla 3.7 – *HU.5.1 - Campos en fichas de datos*

3.2. Gestión de configuración

Esta sección está destinada a explicar la gestión de configuración de la documentación del proyecto, así como el código de este mismo.

3.2.1. Gestión de la documentación

La principal herramienta utilizada para crear toda la documentación del proyecto a la misma vez que el manual de uso de la aplicación software desarrollada, ha sido **Overleaf**, utilizando su editor de LaTex.

3.2.2. Gestión del código

Para gestionar el código que se ha implementado ha sido necesario y fundamental el uso de herramientas de control de versiones, esto es así, para evitar cualquier riesgo de pérdida de datos y también tener una disponibilidad del código en la nube, sin depender de consultar localmente el código en un único equipo de trabajo. Las herramientas utilizadas han sido: GitLab para mantener una versión en la cuenta de la organización de GranaSAT y GitHub, para mantener una versión personal para el autor. Serán mencionadas de nuevo en la sección 3.4.3 (Software).

Mencionar que el código se encuentra dividido en dos partes: un frontend y un backend. Por un lado tenemos el frontend, el cual ha sido implementado mediante Flutter, un framework de código abierto. Por otro lado, tenemos el backend, el cual ha sido desarrollado con el lenguaje de programación JavaScript utilizando NodeJS para la construcción de la API. Ambas divisiones de códigos, se mantienen en el mismo repositorio.

3.3. Gestión temporal

Para poder cumplir con los objetivos establecidos en el proyecto, se marcan una series de tareas a realizar gracias a la ayuda del software **Jira** 3.4.3 (Software). Con dichas tareas establecidas obtendremos un diagrama de Gantt para recoger la temporización que se ha estimado en el desarrollo del software.

Épica 1. Estudio Microsecond Timer App

- **Tarea 1.** Investigar y ampliar información acerca de Fakopp.
- **Tarea 2.** Descargar Microsecond Timer App.
- **Tarea 3.** Revisar aplicación para estudiar su funcionamiento.
- **Tarea 4.** Recoger toda información para el desarrollo del software.

Épica 2. Elección herramientas en backend

- **Tarea 1.** Investigar opciones de bases de datos a usar en backend.
- **Tarea 2.** Investigar acerca de NodeJS.
- **Tarea 3.** Elección versión de NodeJS.
- **Tarea 4.** Elegir hosting en línea para servidor.
- **Tarea 5.** Crear servidor de pruebas en NodeJS para probar funcionamiento.

Épica 3. Interfaces

- **Tarea 1.** Definir mockups para cada una de las interfaces a crear.
- **Tarea 2.** Crear pantalla para inicio de sesión.
- **Tarea 3.** Crear pantalla para registro.
- **Tarea 4.** Crear pantalla para el menú principal.
- **Tarea 5.** Crear pantalla para proyectos.
- **Tarea 6.** Crear pantalla para fichas de datos.

Épica 4. Sistema de Usuarios

- **Tarea 1.** Crear modelo de datos en frontend para el usuario.
- **Tarea 2.** Crear modelo de datos en backend para el usuario.
- **Tarea 3.** Validar formularios de inicio de sesión y registro el usuario.
- **Tarea 4.** Implementar funcionalidades de inicio de sesión, registro y cerrar sesión.
- **Tarea 5.** Crear formulario para edición del perfil.
- **Tarea 6.** Generar rutas CRUD de usuario en backend.
- **Tarea 7.** Implementar llamadas al servidor desde el frontend.
- **Tarea 8.** Definir avisos para operaciones con usuarios.

Épica 5. Sistema de proyectos

- **Tarea 1.** Crear modelo de datos en frontend para el proyecto.
- **Tarea 2.** Crear modelo de datos en backend para el proyecto.

- **Tarea 3.** Crear modelo de datos en frontend para las fichas de datos.
- **Tarea 4.** Crear modelo de datos en backend para las fichas de datos.
- **Tarea 5.** Crear modelo de datos en frontend para las mediciones que se almacenan en las fichas de datos.
- **Tarea 6.** Agregar tabla de mediciones en las fichas de datos.
- **Tarea 7.** Crear exportación de proyectos en Excel, buscar librerias.
- **Tarea 8.** Crear funcionalidad para obtener geoposición y mostrar coordenadas en un mapa.
- **Tarea 9.** Crear funcionalidad para la toma de imágenes.
- **Tarea 10.** Crear modelo de datos en frontend para las especies de arboles.
- **Tarea 11.** Crear modelo de datos en backend para las especies de arboles.
- **Tarea 12.** Definir avisos para operaciones con proyectos.

Épica 6. Gestión de la aplicación

- **Tarea 1.** Compilar aplicación en iOS.
- **Tarea 2.** Compilar aplicación en Windows.
- **Tarea 3.** Crear instalador de aplicación en Windows.
- **Tarea 4.** Automatizar creación instalador en Windows.
- **Tarea 5.** Brindar de multilenguaje a la aplicación.

3.3.1. Diagrama de Gantt

La siguiente figura representa la estimación temporal del desarrollo del proyecto utilizando el diagrama de Gantt.

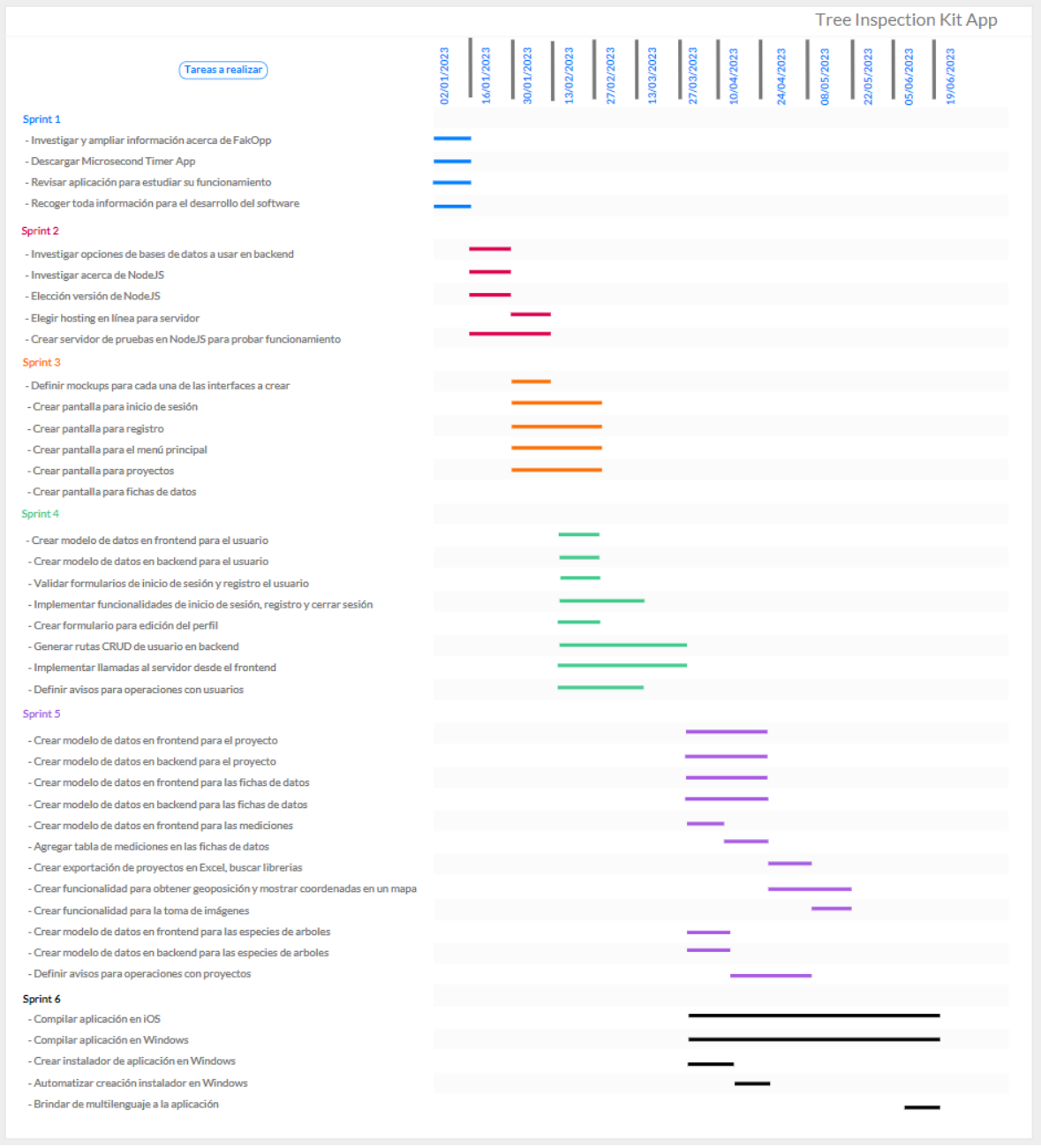


Figura 3.2 – *Estimación temporal - Diagrama de Gantt*

3.4. Gestión de recursos

A continuación se enumeran los recursos utilizados y que han hecho posible el desarrollo del proyecto.

3.4.1. Personal

- **Andrés María Roldán Aranda**, profesor perteneciente al departamento de Electrónica y Tecnología de los Computadores de la Universidad de Granada como tutor del trabajo de fin de grado.
- **Luis Javier Soriano Suárez**, estudiante del grado en Ingeniería Informática de la Universidad de Granada como autor del trabajo de fin de grado.

3.4.2. Materiales

Los principales recursos materiales que han sido requeridos y utilizados para desarrollar este proyecto han sido:

- **Xiaomi Mi Notebook Pro**. Portátil utilizado para trabajar fuera de casa. Dicho equipo de la marca Xiaomi cuenta con un procesador Intel Core i5 8250u con 8GB de memoria RAM, Nvidia MX250 como tarjeta de vídeo y 256GB como capacidad de la unidad de estado sólido.
- **Equipo sobremesa**. Ordenador de sobremesa utilizado para trabajar desde casa. Está compuesto por un procesador Intel Core i5 8600k con 16GB de memoria RAM, Nvidia RTX 3060 Ti como tarjeta de vídeo y 3TB como capacidad de la unidad de estado sólido.

3.4.3. Software

En cuanto a gestión de software se refiere, se ha optado por emplear software de código abierto y hacer uso de las licencias gratuitas ofrecidas a los estudiantes para minimizar en la medida de lo posible los gastos asociados al proyecto. A continuación se detalla el inventario de los recursos de software utilizados:

- **Visual Studio Code.** Es un IDE muy popular entre los desarrolladores que cuenta con una interfaz intuitiva, amplia personalización y un variado soporte de lenguajes de programación. También tenemos una larga colección de extensiones y facilita la escritura de código, la depuración y la colaboración en proyectos.
- **Xcode.** Es un IDE de Apple para desarrollar aplicaciones en plataformas como iOS, macOS y watchOS. Ofrece herramientas de codificación, diseño de interfaces y pruebas. Ha sido utilizado para comprobar el estado de la aplicación software en iOS.
- **ThunderClient.** Es una extensión de Visual Studio Code que permite probar y depurar APIs directamente desde el editor de código. Proporciona una interfaz intuitiva para enviar solicitudes HTTP, ver respuestas, establecer encabezados y parámetros, y realizar pruebas de respuesta.
- **Git.** Es un sistema de control de versiones ampliamente utilizado. Proporciona un registro de cambios en el código fuente, gracias a esto permite rastrear y gestionar eficientemente las modificaciones realizadas en un proyecto. Git es valorado por su capacidad de ramificar y fusionar el código.

- **GitHub.** Es una plataforma de alojamiento de repositorios de código fuente basada en la nube. Permite a los equipos de desarrollo colaborar de manera efectiva en proyectos, proporcionando un lugar centralizado para almacenar, gestionar y compartir el código. GitHub ofrece funciones como control de versiones utilizando Git y otras funcionalidades interesantes como la integración con servicios de implementación continua que ha sido utilizado en este proyecto. En este proyecto se ha obtenido el plan PRO para poder aprovechar el máximo los minutos de CI/CD que nos brinda, esto supone un coste nulo para estudiantes.
- **GitLab.** Es una plataforma de gestión de repositorios de código fuente basada en Git, similar a GitHub. Al igual que GitHub, GitLab ofrece un lugar centralizado en la nube para almacenar y colaborar en proyectos de desarrollo de software. Proporciona características como control de versiones, seguimiento de problemas, solicitudes de extracción y gestión de tareas.
- **NodeJS.** Es un entorno de ejecución de código JavaScript con capacidad para ejecutar código tanto en el lado del cliente como en el lado del servidor. Destaca por su enfoque en la programación asíncrona, su capacidad para manejar múltiples solicitudes sin bloquear el hilo principal y su amplio ecosistema de módulos y paquetes.
- **Express.js.** Se trata de un framework de código abierto de backend para aplicaciones web de Node.js y construir APIs RESTful.
- **Rive** Se trata de una plataforma en línea que permite crear y animar gráficos vectoriales 2D de manera interactiva y fáci para aplicaciones móviles, sitios web, etc.
- **Cloudinary.** Es una plataforma en la nube diseñada para almacenar, gestionar y entregar de manera eficiente imágenes y vídeos. Se ha hecho uso de la API que dispone para su manejo en aplicaciones.
- **MongoDB Atlas.** Es un servicio sobre gestión de base de datos MongoDB basado en la nube. En este proyecto se ha hecho uso de su licencia gratuita para poder desplegar la base de datos de la aplicación software.
- **Render.** Es una plataforma de alojamiento en la nube que simplifica el despliegue de aplicaciones web y APIs. Para este proyecto se ha optado por el servicio gratuito con limitaciones para poder implementar un servidor NodeJS en la nube.
- **Codemagic.** Es una plataforma de CI/CD específicamente diseñada para el desarrollo y la implementación de aplicaciones móviles. En nuestro proyecto se ha utilizado para poder compilar la aplicación en iOS y poder acceder a la máquina virtual y probar el funcionamiento de esta.
- **InnoSetup.** Es una herramienta de código abierto que se utiliza para crear instaladores de aplicaciones en el sistema operativo Windows. Permite empaquetar y distribuir programas de manera sencilla, facilitando la instalación de aplicaciones en los dispositivos de los usuarios.

Para tener un control sobre el desarrollo del proyecto se han utilizado recursos software para la comunicación y documentación del mismo. Las herramientas usadas son:

- **Jira.** Es una plataforma de gestión de proyectos y seguimiento de problemas que utiliza la metodología ágil de SCRUM. Se hace uso de su licencia gratuita para poder llevar un seguimiento de las tareas a realizar.
- **Lucid.app.** Es una herramienta en línea utilizada para crear diagramas, gráficos y visualizaciones. Se ha utilizado para la creación del diagrama de flujo y mockups de la aplicación software.

- **Overleaf.** Es una plataforma en línea que permite la creación, edición y colaboración en documentos LaTex.

- **Telegram.** Es una aplicación de mensajería instantánea. Ha permitido la comunicación entre el tutor y autor del trabajo fin de grado.

- **Google Meet.** Es una plataforma de comunicación y videoconferencia desarrollada por Google. Se ha utilizado para las reuniones telemáticas entre el tutor y el autor del trabajo fin de grado.

3.5. Gestión de costes

En esta sección, nos enfocaremos en calcular y analizar un presupuesto basado en los costes asociados al personal, los recursos software, los recursos materiales y otros posibles gastos derivados que se han dado en el proyecto.

3.5.1. Costes de personal

Para los costes que derivan de recursos humanos se ha tenido en cuenta únicamente al autor del proyecto, que ostentará el rol de ingeniero informático junior. En base al rol, también tendremos que tener en cuenta el periodo que abarca el proyecto para poder obtener una estimación de los costes.

Sabiendo que el desarrollo ha tenido una duración de 6 meses y el autor, debido a su situación de trabajar a jornada completa en paralelo junto al proyecto, ha podido dedicar una media de 3 horas diarias al desarrollo de este, se calculan los siguientes costes:

$$Dias\ trabajados = 6\ \text{meses} \cdot 30\ \text{dias/mes} = 180\ \text{dias}$$

Teniendo los días que se han trabajado calculados, tendremos ahora que calcular la horas trabajadas:

$$Horas\ trabajadas = 180\ \text{dias} \cdot 3\ \text{horas/dia} = 540\ \text{horas}$$

Sabiendo el número total de horas que se han invertido en el desarrollo del proyecto ahora nos queda por calcular el coste teniendo de referencia lo que cobra de media en España un ingeniero informático junior. Según un artículo publicado por la universidad Alfonso X El Sabio[16] y otro artículo publicado por Immune Technology Institute[14] un perfil de ingeniero informático junior de media cobrará aproximadamente 20.000 € brutos anuales.

$$Salario\ mensual = \frac{20000\ €/\text{año}}{12\ \text{meses}} = 1666\ € \text{ brutos}$$

$$Salario\ por\ hora = \frac{1666\ €/mes}{160\ \text{horas trabajadas/mes}} = 10,41\ €/\text{hora}$$

Teniendo el salario por hora y las horas trabajadas en el proyecto el coste total es de:

$$Coste\ total = 540\ \text{horas} \cdot 10,41\ €/\text{hora} = 5621,4\ € \text{ brutos}$$

3.5.2. Costes de materiales

A la hora de calcular los costes relacionados con los dos equipos que han sido utilizados para el proyecto, tenemos que tener en cuenta la depreciación de su valor que sufren con el paso del tiempo.

Para ello, se calculará el valor residual de estos mismos. Este número indica el valor que tendrán los productos una vez hayan alcanzado al final de sus vidas útiles. Con el valor residual, tendremos que calcular su depreciación anual y junto estos datos numéricos y la vida útil de los productos obtenemos el valor actual de los dispositivos y que usaremos para saber el coste.

La fórmulas utilizadas son las siguientes:

$$Valor\ residual = \frac{Coste\ inicial\ del\ producto}{Años\ de\ vida\ util}$$

$$Depreciacion = \frac{Coste\ inicial\ del\ producto - Valor\ residual}{Años\ de\ vida\ util}$$

$$Valor\ actual = Valor\ inicial - (Depreciacion \cdot Años\ antigüedad\ del\ dispositivo)$$

Sabiendo las fórmulas a usar calculamos el coste de los dispositivos:

- **Xiaomi Mi Notebook Pro**

  Partimos de una vida útil de 6 años para este portátil. Sabiendo esto:

$$Valor\ residual = \frac{799\ €}{6\ \text{Años de vida util}} = 133,16\ €$$

$$Depreciacion = \frac{799\ \ € - 133,16\ \ €}{6\ \text{Años de vida util}} = 110,97\ \ €$$

  Valor actual $= 799\ €/\text{hora} - (110,97\ € \cdot 5 mathrm Años\ antigüedad\ del\ dispositivo) =$ **244,15 €**

- **Equipo sobremesa**

  Al equipo de sobremesa le daremos un valor de vida útil de 8 años. Sabiendo esto:

$$Valor\ residual = \frac{1599\ €}{8\ \text{Años de vida util}} = 199,87\ €$$

$$Depreciacion = \frac{1599\ € - 199,87\ €}{8\ \text{Años de vida util}} = 174,89\ €$$

Valor actual $= 1599\ € - (174,89\ € \cdot 6$ Años antigüedad del dispositivo$) =$ **549,66 €**

3.5.3. Costes de herramientas software

El coste relacionado con el uso de herramientas software destinadas al desarrollo del proyecto es de un total de **0.00 €**. Esto es debido a que se ha aprovechado el uso de software gratuito junto a la condición de estudiante del autor del trabajo de fin de grado, permitiendo tener acceso a licencias totalmente gratuitas.

3.5.4. Costes de suministros

Dentro de los costes de suministros que han derivado del desarrollo del proyecto tenemos:

- **Luz.** Aproximadamente 30 €/mes
- **Internet.** 39,99 €/mes

3.5.5. Costes totales

En la presente sección se muestra una tabla resumiendo todos los gastos que están asociados con el desarrollo del proyecto.

Recurso	**Coste**
Personal	
Ingeniero informático junior	5621,4 €
Material	
Xiaomi Mi Notebook Pro	244,15 €
Equipo sobremesa	549,66 €
Software	
No hay gastos asociados	0 €
Suministros	
Luz	30 € x 6 meses = 180€
Internet	39,99 x 6 meses = 239,94 €
COSTES	**6835,15 €**
I.V.A (21 %)	**1435,38 €**
COSTES TOTALES	**8270,53 €**

Tabla 3.8 – *Tabla de costes asociados al proyecto*

Capítulo 4

Diseño e implementación frontend

4.1. Introducción

Para el desarrollo del frontend, se ha utilizado el framework de código abierto de Google llamado **Flutter**.

Figura 4.1 – *Logo del framework Flutter*

Gracias al avance de los últimos años en tecnologías móviles, se han creado potentes herramientas para poder permitir a los desarrolladores tener más facilidades a la hora de crear nuevas aplicaciones. Entre una de estas contamos con Flutter, lanzado por primera vez en Mayo de 2017 y que rápidamente ha ganado mucha popularidad en la comunidad de desarrolladores debido a su rendimiento, facilidad de uso y capacidad para ofrecer interfaces de usuario atractivas y fluidas.

Para poder tener un contexto acerca de lo que es capaz Flutter, se listan las siguientes características[6] que hacen que sea una herramienta muy querida por la comunidad.

- **Rendimiento** - Utiliza el lenguaje de programación Dart, el cual se compila directamente a código nativo, permitiendo que las aplicaciones Flutter alcancen un rendimiento similar al de las aplicaciones nativas. Esto se traduce en una experiencia de usuario más rápida y fluida.
- **Desarrollo rápido** - Proporciona un conjunto completo de widgets personalizables y predefinidos, que facilitan la creación de interfaces de usuario atractivas. Además, la función de **Hot Reload** permite ver los cambios realizados en tiempo real, sin necesidad de tener que compilar desde cero el desarrollo haciendo que todo el proceso de desarrollo sea acelerado y aumente la productividad.
- **Compatibilidad multiplataforma** - Se pueden crear aplicaciones para múltiples plataformas, incluyendo iOS, Android, web y escritorio, utilizando el mismo código para todo. Esto ahorra tiempo y esfuerzo al no tener que desarrollar y mantener diferentes versiones de la aplicación para cada plataforma, aunque se pueden dar casos de incompatibilidades con librerías de terceros.

- **Widgets personalizables** - ofrece una amplia gama de widgets personalizables, que se combinan para construir interfaces de usuario completas. Estos widgets permiten adaptar la apariencia y funcionalidad de la aplicación según las necesidades específicas del proyecto.
- **Acceso a características del dispositivo** - Proporciona acceso nativo a características del dispositivo, como la cámara, geolocalización, sensores y más, a través de su API de plataforma. Esto permite aprovechar todas las capacidades del dispositivo sin restricciones.
- **Comunidad activa** - Cuenta con una comunidad de desarrolladores en constante crecimiento. La comunidad ofrece una amplia variedad de recursos, incluyendo documentación oficial, tutoriales, videos y librerías de terceros, lo que facilita el aprendizaje y el desarrollo en Flutter.

4.2. ¿Cómo funciona Flutter?

Lo primero que se debe tener en claro, es que en Flutter, todo es un widget. Desde los elementos visuales más simples, como botones y cajas de texto, hasta las estructuras más complejas, como columnas y filas, todo se representa como un widget. Los widgets son los bloques de construcción fundamentales para crear la interfaz de usuario en Flutter.

Dentro del corazón de Flutter se encuentran los árboles, que juegan un papel vital en la construcción y renderizado de la interfaz de usuario. Tenemos los siguientes[3]:

- **Árbol de widgets.** Los widgets se organizan en una estructura jerárquica, formando el árbol de widgets. Cada widget tiene propiedades que definen su apariencia y comportamiento, y se anidan dentro de otros widgets para crear una estructura de interfaz de usuario. Esta arquitectura basada en árboles de widgets permite componer interfaces de usuario complejas de manera modular y reutilizable.
- **Árbol de elementos.** Cada widget en el árbol de widgets tiene un elemento correspondiente en el árbol de elementos. Los elementos son objetos de bajo nivel que Flutter utiliza internamente para representar el uso de un widget y configurar una ubicación específica en el árbol. Los elementos indicarán si los widgets son un elementos sin estado (Stateless Elements) o un elemento con estado (Stateful Elements).
- **Árbol de objetos de renderizado.** Es el que se muestra en la pantalla y cada elemento en el árbol de elementos tiene un objeto de renderizado asociado. Este árbol contiene información más detallada sobre cada widget y su ubicación precisa en la pantalla. Controla propiedades como el tamaño, la posición, la rotación, el escalado y otros aspectos visuales de cada widget. Esto permite a Flutter realizar la tarea de renderizado y pintura en la pantalla de manera eficiente y precisa. Flutter utiliza **Skia**, una biblioteca de gráficos de alto rendimiento, para interactuar con el sistema operativo subyacente y lograr un renderizado fluido y de calidad.

Sabiendo la existencia de dichos árboles, destacar que Flutter proporciona una capa de abstracción a través del árbol de widgets que permite a los desarrolladores construir interfaces de usuario sin necesidad de interactuar directamente con los árboles de elementos y de objetos de renderizado. Esto simplifica el proceso de desarrollo y mantenimiento, pero en casos específicos y para el desarrollo más avanzado sería importante tener conocimientos básicos sobre estos árboles.

Como se ha mencionado anteriormente, en el árbol de elementos se definen si los widgets son de tipo **con estado (Stateful)** o **sin estado (Stateless)**.

Un widget con estado es aquel que puede cambiar internamente durante la vida útil de la aplicación. Puede tener propiedades que pueden cambiar y afectar su apariencia o comportamiento

a lo largo del tiempo. Además, puede mantener y actualizar su propio estado interno a medida que cambian las interacciones del usuario o los datos de la aplicación. Están compuestos por dos objetos: el widget en sí mismo y una objeto de estado correspondiente. El objeto de estado es responsable de almacenar y administrar el estado interno del widget. Cuando ocurren cambios que requieren actualizar el widget, Flutter se encarga de reconstruir únicamente el widget y su estado, manteniendo el resto de la interfaz de usuario intacta.

Un widget sin estado, como su nombre indica, es inmutable una vez que se crea. No tiene ningún estado interno que pueda cambiar después de su creación. Esto significa que su apariencia y comportamiento son totalmente determinados por las propiedades que se le pasan en su construcción.

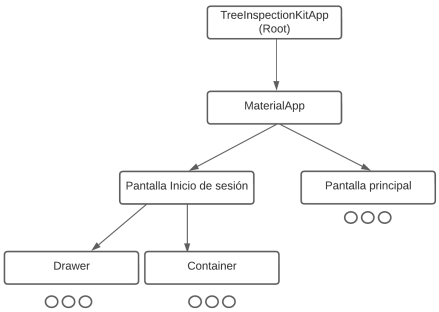


Figura 4.2 – *Ejemplo de Árbol de widget*

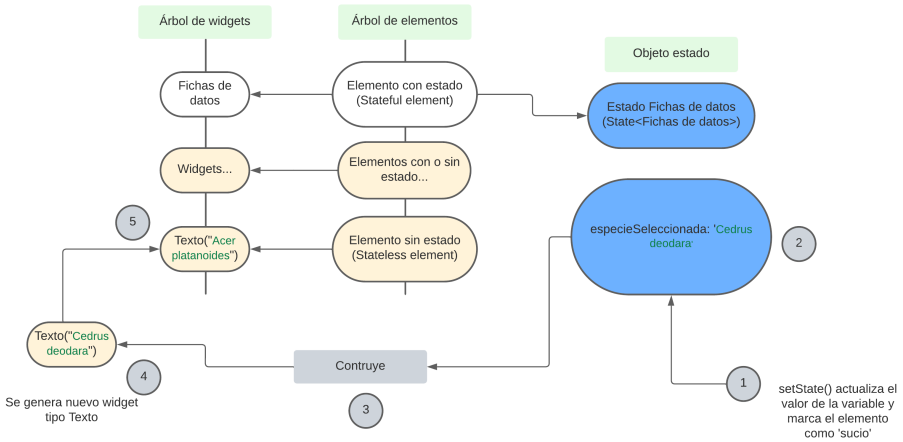


Figura 4.3 – *Explicación de cambio en widget con estado*

En la figura 4.2 (Ejemplo de Árbol de widget) tenemos un ejemplo de como sería un árbol de widgets. Cada widget que se observa en la figura, tiene a su vez más widgets que serán hijos y podrán ser de tipo con estado o sin estado, dependiendo de como haya sido su declaración en el código.

En la figura 4.3 (Explicación de cambio en widget con estado) tenemos un ejemplo de como actuaria un widget con estado dentro de una aplicación, modificando así sus propiedades. El ejemplo esta escenificado con un caso real del proyecto, en el cual un usuario decide cambiar el tipo de especie de

una ficha de datos y se llama al método **setState()** del objeto estado el cuál indica que dicho elemento del árbol de elementos está ”sucio“ y que por lo tanto se debe redibujar y cambiar el valor del widget Text.

4.3. Diseño del frontend

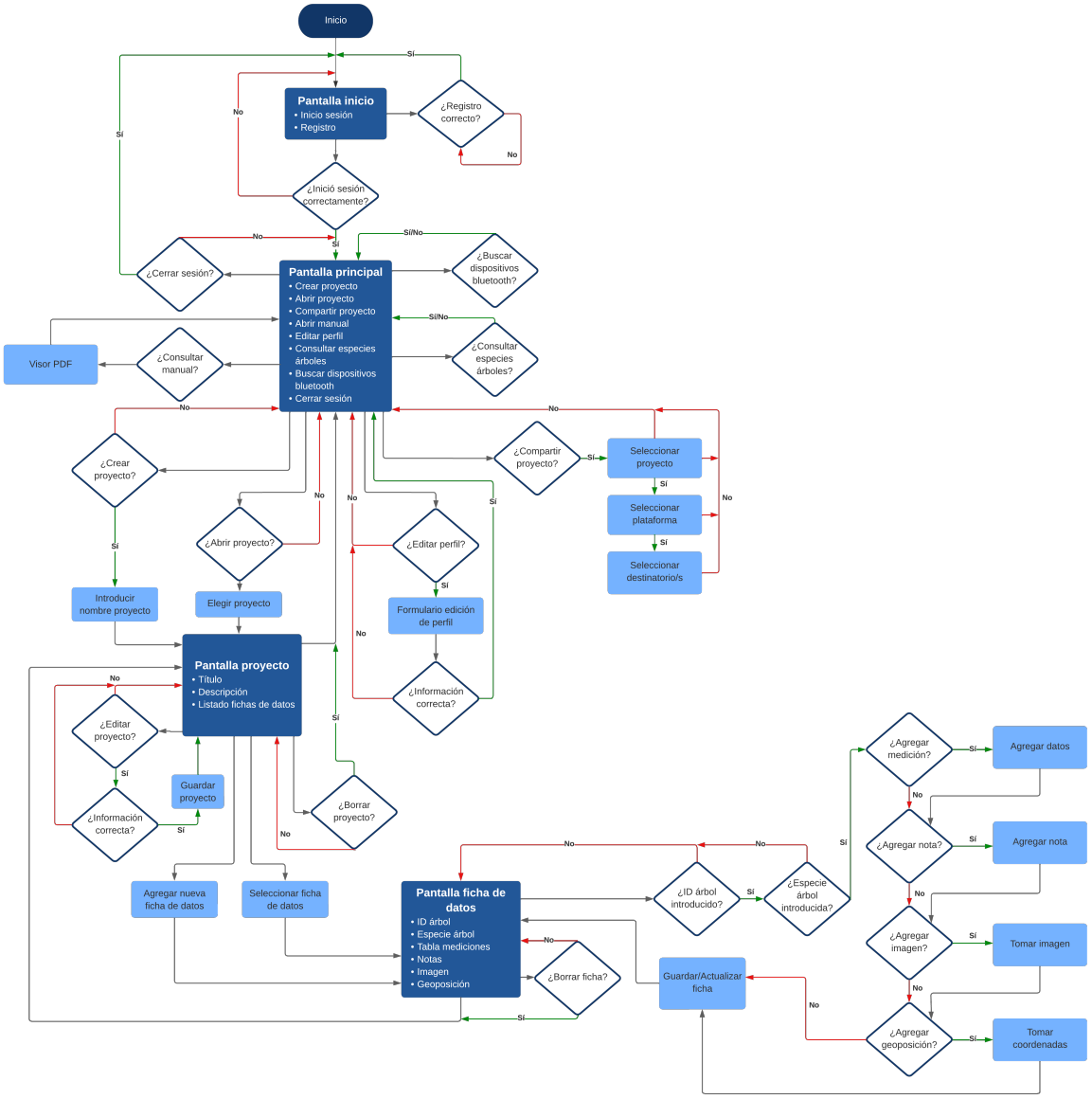


Figura 4.4 – *Diagrama de flujo - TIK App*

Para mostrar el diseño del frontend, se ha escenificado un diagrama de flujo de la aplicación para poder mostrarlo de una manera más adecuada.

Como podemos observar en dicho diagrama de flujo, se pueden apreciar las cuatro pantallas

principales que tiene la aplicación junto con las posibilidades que se pueden explorar una vez nos encontramos dentro de ellas.

Antes del proceso de implementación de código, sobre estas pantallas, se realizaron unos mockups de como podrían ser los diseños de estas mismas, a continuación se muestran los mockups mencionados:

4.3.1. Mockups inicio sesión y registro

Se pensó en una pantalla individualizada tanto para que el usuario realice el acceso a la aplicación, como para poder darse de alta en el sistema y posteriormente iniciar sesión con la cuenta de usuario creada.

(a)Mockup login *(b)Mockup registro*

Figura 4.5 – *mockups de login y registro de usuarios*

4.3.2. Mockup pantalla principal

A la hora de crear la pantalla principal se pensó en el diseño de la aplicación comercial original (2 Estado del arte), ya que la distribución de elementos es adecuada y acorde con lo deseado.

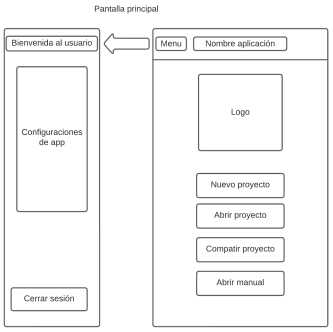


Figura 4.6 – *Mockup pantalla principal*

En dicho mockup disponemos de las opciones principales junto a un menú lateral donde se da la bienvenida al usuario y tendremos las opciones de la aplicación que se vayan a desarrollar.

4.3.3. Mockup pantalla de proyecto

Al principio del desarrollo se vió necesario crear una pantalla intermedia entre la página principal y las fichas de un proyecto seleccionado. Esta pantalla servirá de manera que se muestre en un listado todas las fichas que tenemos asociadas a dicho proyecto, así como una descripción del proyecto para poder recalcar alguna información que sea relevante.

Tendremos también controles para poder editar y borrar un proyecto una vez estemos dentro de él.

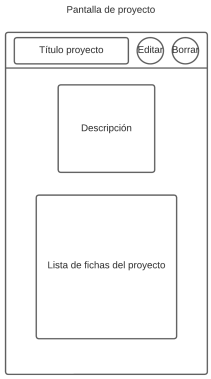


Figura 4.7 – *Mockup pantalla de proyecto*

4.3.4. Mockup pantalla de ficha

Se abrirá una pantalla por cada ficha seleccionada, incluyendo al principio de esta los campos obligatorios para poder crear la ficha deseada. Seguidamente, se establecen los campos que serán opcionales y tendremos controles para poder editar y borrar dicha ficha una vez estemos dentro de ella.

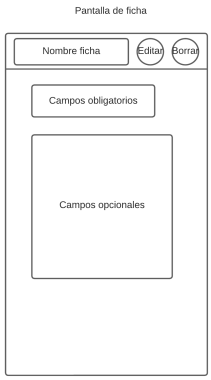


Figura 4.8 – *Mockup pantalla de ficha*

4.4. Estructura

Una vez presentados los mockups iniciales, creados para tener una visión general de como sería la interfaz de usuario, se crea la estructura del proyecto Flutter para comenzar con el desarrollo.

A la hora de crear un proyecto en flutter, se crean una serie de directorios por defecto, los cuales tienen una funcionalidad concreta, así mismo, en este proyecto se ha seguido una estructura organizada en subdirectorios según la funcionalidad de **cada archivo Dart** utilizado. Nos centraremos en la estructura y en explicar los directorios y archivos más importantes para entender un proyecto flutter:

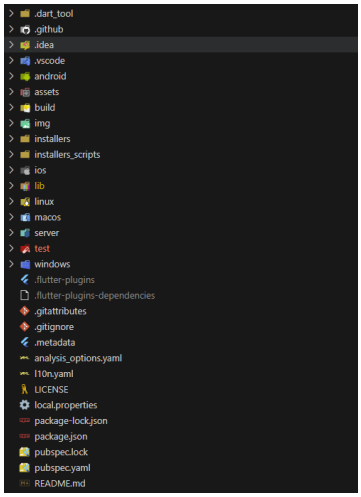


Figura 4.9 – *Estructura de proyecto en flutter*

- **.dart_tool -** Almacena archivos relacionados con el entorno de desarrollo y las herramientas de Dart. Contiene información de compilación, configuración de paquetes y versiones de dependencias.
- **.github -** Directorio utilizado por la plataforma GitHub para almacenar archivos y configuraciones relacionadas con la gestión de repositorios. **Se hará referencia del contenido de este más adelante en el capítulo 6 (Despliegue de aplicación).**
- **.idea -** Directorio utilizado por el entorno de desarrollo IntelliJ IDEA para almacenar configuraciones específicas del proyecto. Contiene archivos relacionados con el estilo de código, preferencias del IDE y configuraciones de ejecución. **En nuestro caso no es relevante para el desarrollo del proyecto.**
- **android -** Crucial para compilar y construir la aplicación en dispositivos Android.
- **assets -** Se utiliza para almacenar archivos estáticos como imágenes, archivos de audio, fuentes y otros recursos que se utilizarán en la aplicación.
- **build -** Contiene los archivos de salida generados durante el proceso de compilación, como el código compilado, los recursos optimizados y otros archivos necesarios para ejecutar la aplicación
- **installers -** Directorio para almacenar el instalador de la aplicación para el sistema operativo Windows. **Se hará referencia del contenido de este más adelante en el capítulo 6 (Despliegue de aplicación).**

- **installers_scripts -** Directorio para almacenar el script que generará el instalador de la aplicación para el sistema operativo Windows. **Se hará referencia del contenido de este más adelante en el capítulo 6 (Despliegue de aplicación).**
- **iOS -** Contiene archivos relacionados con la configuración y compilación específica de iOS, incluyendo el archivo Runner.xcodeproj que representa el proyecto en XCode, archivos de configuración y recursos específicos de iOS.
- **lib - Directorio principal** donde se encuentra el código fuente de la aplicación. Aquí es donde se escriben y organizan los archivos de código Dart que definen la lógica de la aplicación, la interfaz de usuario y otras funcionalidades.
- **server -** Directorio principal para almacenar archivos y subdirectorios que contienen el código fuente, configuraciones y recursos necesarios para implementar y administrar la lógica del backend de la aplicación.
- **windows -** Contiene archivos y configuraciones relacionadas con la plataforma Windows, como recursos gráficos, manifiestos y scripts de compilación.
- **pubspec.yaml (Archivo) -** Archivo de configuración utilizado en proyectos de Flutter para especificar las dependencias del proyecto, incluyendo paquetes externos, así como para definir metadatos y configuraciones adicionales. Permite gestionar las dependencias del proyecto, incluyendo versiones específicas de paquetes, así como configurar otros aspectos como el nombre de la aplicación, la versión, los assets, las dependencias de desarrollo, entre otros. Es esencial para la gestión y construcción del proyecto en Flutter.

4.4.1. Directorio principal

Como se ha mencionado previamente en el listado superior, el directorio "**lib**" alberga los archivos Dart que son fundamentales para la construcción de la aplicación. Es recomendable organizar este directorio en subdirectorios con el fin de mantener una estructura y organización coherente del código. Esta práctica facilita el mantenimiento y comprensión del código a lo largo del desarrollo del proyecto.

El directorio principal lo componen lo siguientes subdirectorios:

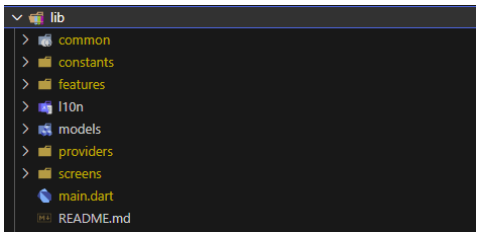


Figura 4.10 – *Directorio principal - lib*

Common contendrá todos los widgets que han sido creados a partir de widgets del estándar, para poder adaptarlos a las necesidades de nuestra aplicación. Dentro podemos encontrar algunos ejemplos como son: **widget del mapa, widget de uso de cámara, widgets campo de texto para formularios...**

Constants como su nombre indica contiene archivos para manejar constantes en la aplicación. Aquí dentro nos encontramos: **URL de conexión a base de datos, funciones y estructuras globales**.

Features alberga la mayoría de código en la que intervienen las funcionalidades principales de la aplicación, como bien pueden ser, **manejo de operaciones del modelo de usuario, manejo de operaciones con proyectos, manejo de operaciones con fichas...**

L10n es un directorio específico para albergar las plantillas de idiomas de la aplicación. Esto nos brinda la posibilidad de agregar tantos idiomas como queramos. **Se tratará este punto en el capítulo 4.5.4 (Funcionalidades genéricas de la aplicación).**

Models contiene los ficheros donde se definen los modelos de datos utilizados en la aplicación. Estos archivos se centran en definir la estructura y el comportamiento de los datos en la aplicación. Dentro de los modelos se definen las propiedades y métodos relacionados con la manipulación y representación de esos datos, como puede ser serialización y deserialización de estos mismos.

Providers es un directorio para albergar lo que se conoce en flutter como providers. Estas clases tienen la capacidad de escuchar el estado de la aplicación y reaccionar en consecuencia cuando se detecta algún cambio que active al provider.

Screens contendrá las pantallas principales que contiene la aplicación. Estás pantallas a su vez están construidas reutilizando los widgets que nos encontramos en la carpeta **Common**.

El fichero main es el punto de partida y la configuración inicial para construir y ejecutar la aplicación flutter.

4.5. Implementación de funcionalidades

4

En este apartado, nos adentraremos en el proceso de desarrollo y programación para llevar a cabo la implementación de las distintas funcionalidades que ha sido diseñadas. Aquí exploraremos en detalle cómo hemos traducido las especificaciones y requerimientos en código funcional. En todo momento se ha hecho uso de la documentación oficial de flutter[8] a la hora de crear los widgets que componen la aplicación.

4.5.1. Sistema de usuarios

Para poder añadir a la aplicación la funcionalidad del sistema que gestiona todo lo relativo a usuarios, se hacen uso de varias clases que se han configurado dentro de los directorios descritos anteriormente.

Se comienza con la definición del modelo de usuario. Este modelo se encuentra declarado en el directorio **Models**, donde se implementan todos los métodos necesarios para poder gestionar un objeto de tipo Usuario. Con lo siguiente en la clase:

- **Variables del modelo**: aquí se definen las variables que forman parte del modelo de datos de un usuario. Estas variables incluyen el ID, nombre, correo electrónico, contraseña, confirmación de contraseña y token.
- **Función toMap**: esta función se utiliza para convertir un objeto de la clase Usuario en un mapa de cadenas clave-valor. El mapa resultante se utiliza para la serialización y almacenamiento de datos.
- **Función fromJson**: esta función es una función de fábrica que se utiliza para convertir un mapa de cadenas clave-valor en un objeto de la clase User. Se utiliza para la deserialización de datos.
- **Función toJson**: esta función se utiliza para convertir un objeto de la clase User en una cadena

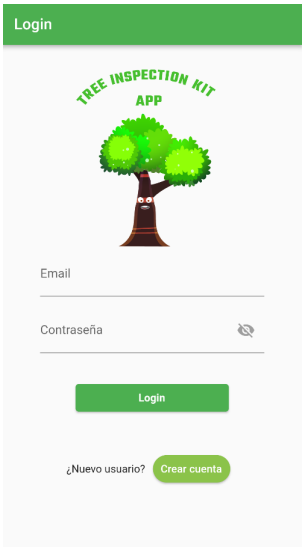
JSON. Se utiliza para la serialización de datos.

```dart
import 'dart:convert';

// Creating user model
class User{
  final String id;
  final String name;
  final String email;
  final String password;
  final String confirmpassword;
  final String token;

  User({
    required this.id,
    required this.name,
    required this.email,
    required this.password,
    required this.confirmpassword,
    required this.token
  });

  // Create a object map
  Map<String, dynamic> toMap(){
    return {
      "id": id,
      "name": name,
      "email": email,
      "password": password,
      "confirmpassword": confirmpassword,
      "token": token
    };
  }

  // Factory fucntion to parse from JSON to object
  factory User.fromJson(Map<String, dynamic> parsedJson){
    return User(
      id: parsedJson["_id"],
      name: parsedJson["name"],
      email: parsedJson["email"],
      password: parsedJson["password"],
      confirmpassword: parsedJson["confirmpassword"] != null && parsedJson["confirmpassword"].isNotEmpty
      ? parsedJson["confirmpassword"]
      : '',
      token: parsedJson["token"] ?? '',
    );
  }

  // Create toJson method
  String toJson() => json.encode(toMap());

}
```

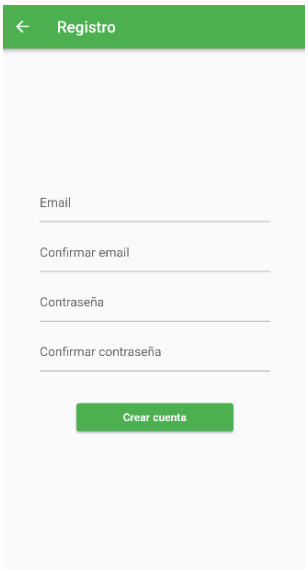
Figura 4.11 – *Modelo de Usuario en frontend*

4

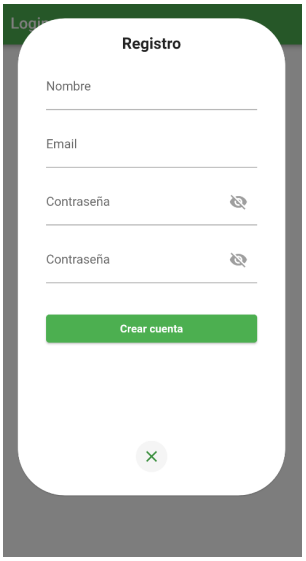
Una vez definido el modelo de usuario en la parte frontend, se diseñan y se crean las pantallas, para que el usuario pueda interactuar con ellas y así poder registrarse e iniciar sesión una vez se haya completado los procesos. Estas pantallas, las cuales fueron basadas en los mockups que se plantearon al principio del desarrollo, finalmente sufren varios cambios, resultando en las siguientes:



(a) Versión final de pantalla de inicio de sesión



(b) Primera versión pantalla de registro



(c) Versión final pantalla de registro

Figura 4.12 – *Versiones de pantallas registro de usuarios*

Pantalla de inicio de sesión. 4.12

Para la creación de esta pantalla, destacamos tres grupos de elementos.

- **Formulario**: está compuesto por dos campos de texto para poder introducir los valores de **Email** y **Contraseña**. Así mismo, el campo de contraseña, brinda la posibilidad de poder ver el contenido de esta misma para comprobar errores de escritura. Esto hará que facilite la experiencia de usuario.

```
children: [
  CustomTextField(controller: _emailController, labelText: "Email"),
  SizedBox(height: 15),
  CustomPasswordFormField(
    controller: _passwordController,
    onVisibilityPressed: (isPasswordVisible) {
      widget.onVisibilityPressed!(isPasswordVisible);
    },
    editing: false,
  ), // CustomPasswordFormField
```

Figura 4.13 – *Campos del formulario en pantalla de inicio de sesión*

Para estos dos campos, se han utilizado widgets personalizados, **CustomTextField** y **CustomPasswordFormField**, dentro de estas clases que se encuentran dentro del directorio **Common**, se aplica la lógica y funcionalidad de los campos. Para poder controlar la validación de cada uno de ellos, se hacen uso de los llamados **controladores**, en los cuales se determinan las directivas de dichas comprobaciones.

- **Botones**: son dos, los cuales tienen como funcionalidad el poder iniciar sesión una vez introducidos los datos correctamente y poder abrir la pantalla/diálogo del formulario de registro de usuarios. El botón de inicio de sesión, tiene como función llamar al servicio que se implementa en el archivo **auth_service** del directorio **features**.

```
// Login user
Future loginUser({
  required BuildContext context,
  required String email,
  required String password
})
async{
  final client = IOClient(HttpClient()..connectionTimeout = Duration(seconds: timeoutDurationSeconds));
  try{

    String encryptedPassword = getPasswordHash(password);

    User user = User(
      id: '',
      name: '',
      email: email,
      password: encryptedPassword,
      confirmpassword: '',
      token: ''
    );


    Response res = await client.post(
      Uri.parse('$url/accounts/login'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: user.toJson(),
    );

    return ValidResponse(res);

  }on SocketException catch (_) {
    showFlutterToast(msg: 'Se ha excedido el tiempo límite de la solicitud', isSuccess: false);
  }catch(err){
    showFlutterToast(msg: err.toString(), isSuccess: false);
  }finally {
    client.close();
  }
}
```

Figura 4.14 – *Servicio para inicio de sesión*

Una vez se devuelve la respuesta del servidor, se procede a comprobar dicha respuesta para mostrar una animación en pantalla de inicio de sesión correcto o por el contrario una animación de inicio de sesión incorrecto.

El botón de registro, tendrá como objetivo el permitir abrir una nueva pantalla para mostrar al usuario un nuevo formulario donde poder darse de alta.

- **Animación de logo**: creada y realizada con la ayuda de la herramienta en línea **Rive**.

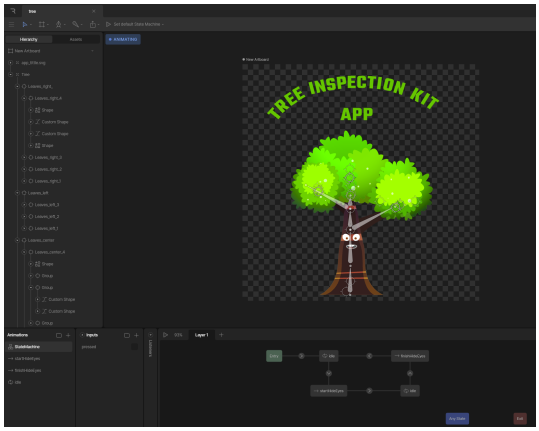


Figura 4.15 – *Editor de Rive*

```
// Create animations variable
late RiveAnimationController _controller;
SMIInput<bool>? _hidePassword;


void _onRiveInit(Artboard artboard) {
// Create a StateMachineController from the provided artboard and state machine name
final controller = StateMachineController.fromArtboard(artboard, 'StateMachine');

// Add the controller to the artboard so it can start managing animations
artboard.addController(controller!);

// Find the SMIBool input with the name 'pressed' in the state machine and store it
_hidePassword = controller.findInput<bool>('pressed') as SMIBool;


void _hitHidePassword(bool hideText) {
  if(hideText == false)
  {
    _hidePassword?.change(false);
  }
  else{
    _hidePassword?.change(true);
  }
}
```

Figura 4.16 – *Parte del código implementado para la animación*

Dentro del editor de Rive, tendremos la posibilidad de crear cualquier forma vectorial que queramos y darle vida gracias al conjunto de herramientas que esta nos brinda. Podemos también declarar varios estados en la animación según algunas condiciones que se den. En nuestro caso, se crea un estado estático para que el árbol se mueva de un lado al otro una vez se haya iniciado la aplicación. También se declara otro estado que hará que el árbol se tape los ojos una vez el usuario haga click en el campo de texto de la contraseña, simulando así tener vida propia y ser consciente de los sucesos que ocurren dentro de la aplicación.

Para poder lograr esto, Rive nos proporciona una librería para Flutter y poder hacer uso de las animaciones creadas. Al instalar la librería, se debe declarar en el fichero **pubspec.yaml** la ruta de donde se encuentra la animación, en nuestro caso la almacenamos en el directorio **Assets**. Una vez configurada la librería y el archivo, se debe proceder a implementar el código para mostrarla en la aplicación. Para poder hacer uso de los estados, se declaran diversas variables que controlan los mismos y pueden ser activados gracias a los métodos que nos proporciona la librería. Así mismo, la animación de inicio de sesión correcto o incorrecto se obtuvo haciendo uso de Rive.

Pantalla de registro. 4.12

Para la creación de esta pantalla, destacamos dos grupos de elementos.

- **Formulario**: está compuesto por cuatro campos de texto, dos de ellos siendo **CustomTextField** y otros dos **CustomPasswordFormField**. Los dos primeros campos de texto serán para que el usuario introduzca su nombre y el correo que desea tener asociado a su cuenta. Los dos siguientes campos servirán para definir la contraseña que utilizará para poder hacer el ingreso con su email establecido. Se hace uso de dos campos para que se confirmen las contraseña y añadir una validación extra el formulario. Así mismo, como ocurre en el formulario de inicio de sesión, en los campos de confirmación de contraseñas, se brinda la posibilidad de poder ver el contenido de esta misma para comprobar errores de escritura.
- **Botones**: son dos, el primero tiene como funcionalidad el poder registrar al usuario una vez introducidos los datos de este y poder procesar la inserción en la base de datos del usuario en la

parte backend de la aplicación. Una vez se haya validado la respuesta, se devuelve al usuario a la pantalla de inicio de sesión para que pueda hacer login con el usuario que recién ha creado.

```dart
// Register user
future registerUser({
  required BuildContext context,
  required String name,
  required String email,
  required String password,
  required String confirmpassword
})
async{
  final client = IOClient(HttpClient()..connectionTimeout = Duration(seconds: timeoutDurationSeconds));
  try{

    String encryptedPassword = getPasswordHash(password);

    User user = User(
      id: '',
      name: name,
      email: email,
      password: encryptedPassword,
      confirmpassword: encryptedPassword,
      token: ''
    );

    Response res = await client.post(
      Uri.parse('$url/accounts/register'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: user.toJson(),
    );

    return res;

  }on SocketException catch (_) {
    showFlutterToast(msg: 'Se ha excedido el tiempo límite de la solicitud', isSuccess: false);
  }catch(err){
    showFlutterToast(msg: err.toString(), isSuccess: false);
  }finally {
    client.close();
  }
}
```

Figura 4.17 – *Servicio para registro de usuario*

El segundo botón, tendrá como función cerrar la pantalla de registro para poder volver a la pantalla de inicio de sesión.

Pantalla de edición de perfil de usuario

Esta pantalla se encuentra una vez estamos dentro de la pantalla principal. En dicha pantalla, la única diferencia respecto del formulario de registro en cuanto a código se refiere, es la llamada al web service para editar el perfil, que por supuesto es otra llamada diferente a la de registro. Para evitar crear un nuevo formulario, se reutiliza el mismo del registro pero usando algunas variables booleanas para indicar que uso le estamos dando, bien si es para registrar usuarios nuevos o bien para editar ya los existentes.

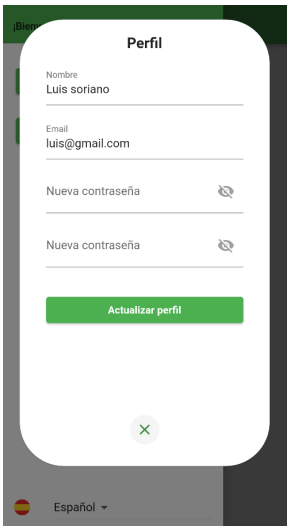


Figura 4.18 – *Pantalla de edición de perfil*

```dart
// Edit user profile
Future editUserProfile({
  required BuildContext context,
  required User user,
})
async{
  final client = IOClient(HttpClient()..connectionTimeout = Duration(seconds: timeoutDurationSeconds));
  try{

    Response res = await client.put(
      Uri.parse('$url/accounts/editprofile'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: user.toJson(),
    );

    if(res.statusCode == 200)
    {
      print(jsonDecode(res.body)['user']);
      Provider.of<UserProvider>(context, listen: false).setUser(jsonDecode(res.body)['user']);
    }
    return res;
  }on SocketException catch (_) {
    showFlutterToast(msg: 'Se ha excedido el tiempo límite de la solicitud', isSuccess: false);
  }catch(err){
    showFlutterToast(msg: err.toString(), isSuccess: false);
  }finally {
    client.close();
  }
}
```

Figura 4.19 – *Servicio para edición de perfil*

4

Para poder recoger la información del usuario dentro de la aplicación y dar la bienvenida sabiendo el nombre del usuario o bien, al abrir el formulario de edición de perfil y que el nombre y el email del usuario estén rellenos en sus campos correspondientes, se hace uso de lo que se llaman en flutter las clases **providers**. Estas clases tienen la capacidad de escuchar el estado de la aplicación y reaccionar en consecuencia cuando se detecta algún cambio que active al provider. Debajo encontramos el código de la clase provider de usuario.

```dart
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:tree_inspection_kit_app/models/user.dart';

// Class use to get and set user data
class UserProvider extends ChangeNotifier {
  // Private user variable
  User _user = new User(id: '', name: '', email: '', password: '', confirmpassword: '', token: '');

  User get user => _user;

  // It's a string because where are going to pass value
  // through response body
  void setUser(Map<String, dynamic> user){
    _user = User.fromJson(user);
    notifyListeners();
  }
}
```

Figura 4.20 – *Clase provider de usuario*

4.5.2. Sistema de proyectos

Una vez terminada la gestión de usuarios y que estos puedan acceder a la aplicación con su cuenta, se comienza con el desarrollo funcional de la gestión de proyectos. Para ello, siguiendo los mismos pasos que en el desarrollo de gestión de usuarios, se toman como base el mockup inicial del menú principal 4.6 (Mockup pantalla principal) para poder crear las pantallas que el usuario necesitará para esta funcionalidad.

En este caso, el mockup diseñado cumple la función de ser una interfaz fácil, intuitiva y atractiva para el usuario. Como pantalla final se establece la siguiente:

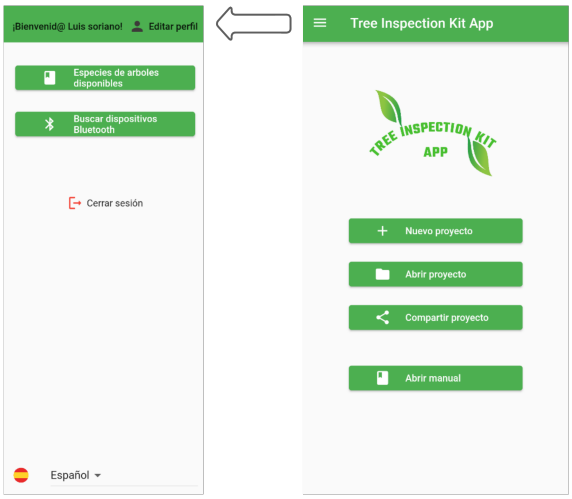


Figura 4.21 – *Versión final pantalla principal*

Como se puede observar se mantiene el logo de la aplicación en la parte superior de la pantalla, seguido de las opciones disponibles que tiene el usuario para hacer manejo de los proyectos que se han ido creando. Antes de continuar con el funcionamiento de cada botón, **debemos explicar el modelo de proyecto** que se ha definido en el frontend para poder llevar a cabo todas las operaciones disponibles.

Como todos los modelos que han sido declarados, se encuentra en el directorio **Models**. La clase contiene lo siguiente:

- **Variables del modelo**: aquí se definen las variables que forman parte del modelo de datos de un proyecto. Estas variables incluyen el ID, nombre, descripción y el id del usuario que ha creado el proyecto.

- **Función toMap**: esta función se utiliza para convertir un objeto de la clase Project en un mapa de cadenas clave-valor. El mapa resultante se utiliza para la serialización y almacenamiento de datos.

- **Función fromJson**: esta función es una función de fábrica que se utiliza para convertir un mapa de cadenas clave-valor en un objeto de la clase Project. Se utiliza para la deserialización de datos.

- **Función toJson**: esta función se utiliza para convertir un objeto de la clase Project en una cadena JSON. Se utiliza para la serialización de datos.

```dart
import 'dart:convert';

// Creating user model
class Project{
  final String id;
  String name;
  String description;
  final String user_id;

  Project(
    {required this.name, required this.id, required this.description, required this.user_id}
  );

  // Create a object map
  Map<String, dynamic> toMap(){
    return {
      "_id": id,
      "name": name,
      "description": description,
      "user_id": user_id,
    };
  }

  // Factory fucntion to parse from JSON to object
  factory Project.fromJson(Map<String, dynamic> parsedJson){
    return Project(
      id: parsedJson["_id"] ?? '',
      name: parsedJson["name"],
      description: parsedJson["description"] ?? '',
      user_id: parsedJson["user_id"],
    );
  }

  factory Project.parseProjects(String responseBody) {
    final parsed = json.decode(responseBody).cast<Map<String, dynamic>>();
    return parsed.map<Project>((json) => Project.fromJson(json)).toList();
  }

  // Create toJson method
  String toJson() => json.encode(toMap());

}
```

Figura 4.22 – *Modelo de Proyecto en frontend*

4

Una vez definido el modelo de proyecto en la parte frontend, podemos entrar en detalle de las funciones que tienen los botones en el menú principal 4.21 (Versión final pantalla principal).

4.5.2.1. Nuevo proyecto

Cuando un usuario desea crear un nuevo proyecto, hará click en dicho botón, esto una vez haya ocurrido este evento, esto hará que se abra un dialogo en la pantalla que indique al usuario que nombre desea ponerle a dicho proyecto. Si el usuario crea el proyecto sin definir un nombre, este proyecto tendrá tendrá como nombre la fecha actual con horas, minutos y segundos.

Figura 4.23 – *Diálogo para crear proyecto*

```dart
Future newProject({
  required BuildContext context,
  required String name,
  required String user_id,
})
async{
  final client = IOClient(HttpClient()..connectionTimeout = Duration(seconds: 10));
  try{
    Project project = Project(
      id: '',
      name: name,
      description: '',
      user_id: user_id
    );


    Response res = await client.post(
      Uri.parse('$url/projects/new'),
      headers: <String, String>{
        'Content-Type': 'application/json; charset=UTF-8',
      },
      body: project.toJson(),
    );

    return res;

  }on SocketException catch (_) {
    showFlutterToast(msg: 'Se ha excedido el tiempo límite de la solicitud', isSuccess: false);
  }catch(err){
    showFlutterToast(msg: err.toString(), isSuccess: false);
  }finally {
    client.close();
  }
}
```

Figura 4.24 – *Servicio para crear proyecto*

Una vez hecho click en el botón crear que aparece en el diálogo, primeramente, se llama al servicio implementado en el archivo **project_service** del directorio **features** para poder hacer una llamada al servidor y crear un nuevo proyecto en la base de datos. Una vez terminado el proceso de creación del proyecto en el servidor, el usuario será redirigido a la pantalla del proyecto recién creado 4.26 (Pantalla de proyecto con nombre Demo).

4.5.2.2. Abrir proyecto

En esta opción al usuario se le muestra un diálogo 4.25 (Listado de proyectos de un usuario) donde aparecerán los proyectos que este mismo ha creado. Por defecto, se le muestran por orden de creación, es decir, el primer proyecto que se ve en el listado es último que ha sido creado. Una vez seleccionado el proyecto, se le abrirá una nueva pantalla donde se expone el nombre del proyecto, la descripción y un listado de fichas de datos asociadas 4.26 (Pantalla de proyecto con nombre Demo).

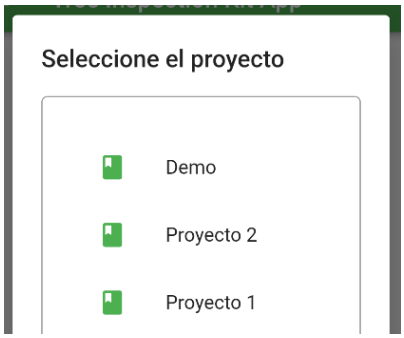
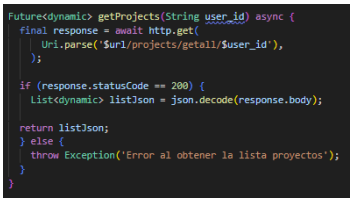


Figura 4.25 – *Listado de proyectos de un usuario*

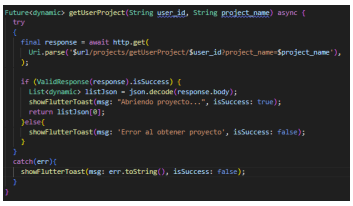
Figura 4.26 – *Pantalla de proyecto con nombre Demo*

Con respecto al mockup que se diseñó al principio 4.7 (Mockup pantalla de proyecto), la pantalla de proyecto ha sufrido el cambio de los botones de borrado y edición, pasando a estar en parte inferior de dicha pantalla. Dichos botones tienen la función que su nombre indica, estos llamarán a dos servicios diferentes que mandarán al servidor la petición de borrado o actualización y una vez la respuesta de vuelta, si ha sido un borrado redirige al usuario a la pantalla principal, si bien ha sido una actualización cambia los valores de los campos del proyecto que el usuario haya modificado.

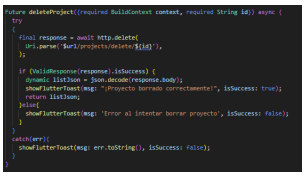
Para poder tener toda esta funcionalidad en el sistema de proyectos, se hacen uso de múltiples servicios. Estos servicios incluyen, conseguir listado de proyectos asociados al usuario, conseguir el proyecto que el usuario ha seleccionado, borrado del proyecto, edición del proyecto y mostrar las fichas de datos asociadas al proyecto. Este último servicio se encuentra en el archivo **tree_data_sheets_service** del directorio **features**. A continuación se muestran los servicios mencionados respectivamente:



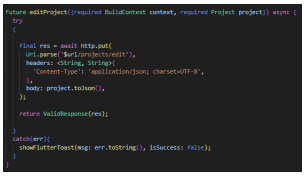
(a) Servicio obtener proyectos de un usuario



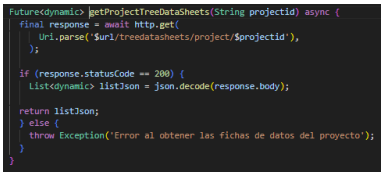
(b) Servicio obtener proyecto seleccionado



(c) Servicio borrado



(d) Servicio edición



(e) Servicio listado fichas proyecto

Figura 4.27 – *Servicios usados en pantalla de proyecto*

4

4.5.2.3. Exportar proyecto

Para terminar con la sección de gestión de proyectos, tenemos la última funcionalidad que engloba tanto proyectos como fichas, ya que nos proporciona una manera de obtener los datos de forma organizada en un fichero Excel y poder realizar una consulta desde el mismo dispositivo si tenemos una aplicación software para abrir fichero **.xls**. Dado que la parte funcional está aplicada en la parte del servidor, ya que es este es el encargado de devolver el fichero con todos los datos, entraremos en profundidad de como se ha implementado en la sección 5.4.3 (Funcionalidad exportación de proyectos). Sin embargo, en la parte frontend entra una componente muy importante que es la posibilidad de poder exportar para compartir dicho fichero obtenido del servidor con las personas que queramos a través de múltiples plataformas.

Esto lo conseguimos gracias a la librería de flutter llamada **Share Plus**. La pantalla de exportación es realmente sencilla, contamos con el mismo diálogo que nos encontramos en la pantalla 4.25 (Listado de proyectos de un usuario) para seleccionar el proyecto a exportar. Una vez el usuario lo selecciona, se utiliza un servicio de proyecto, el cuál mandará una petición al servidor para que devuelva el fichero Excel con todos los datos asociados. Al recibir el fichero, se la abre un nuevo dialogo al usuario para pedir que seleccione en que plataforma desea compartir el fichero. Tenemos muchas opciones como Gmail, Google Drive, Bluetooth y también si disponemos de aplicaciones de mensajería instantáneas como Whatsapp, Telegram, etc...

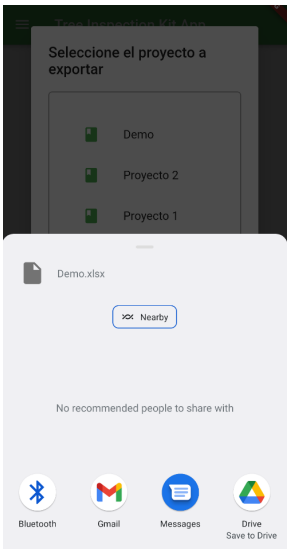


Figura 4.28 – *Diálogo para compartir proyecto en diferentes plataformas*

```
void onTapProjectExport(Project project) async {
  try {
    // Call to projectService to retrieve excel file
    Response res = await projectService.exportProject(context: context, project: project);
    if(res.statusCode == HttpStatus.ok)
    {
      // Get app documents directory
      final documentsDirectory = await getApplicationDocumentsDirectory();
      // Create file with project name
      var file = await File('${documentsDirectory!.path}/${res.headers['filename']}').create();
      // Write data into file
      await file.writeAsBytes(res.bodyBytes);
      // Share file
      Share.shareXFiles([XFile('${file.path}')],
          subject: '${AppLocalizations.of(context)!.emailSubjectExport}: ${project.name}', text: AppLocalizations.of(context)!.successfulExport);
    }
  } catch (e) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(
        content: Text('${AppLocalizations.of(context)!.errorExport}: ${e.toString()}'),
      ), // SnackBar
    );
  }
}
```

Figura 4.29 – *Código llamada de servicio para exportar proyecto y apertura del diálogo para compartir*

4.5.3. Sistema de fichas de datos

Una vez explicada la sección del sistema de proyectos, nos toca toda la implementación de las fichas de datos para poder tener una visión completa de la funcionalidad que gira alrededor del almacenamiento de datos que se obtienen de los árboles estudiados.

Para comenzar, se indican las clases modelo que se necesitan a la hora de crear una ficha de datos. Como se ha descrito anteriormente, estos modelos se encuentran en el directorio **Models**:

Modelo ficha de datos

- **Variables del modelo**: aquí se definen las variables que forman parte del modelo de datos de una ficha de datos de árbol. Estas variables incluyen el ID, el ID del proyecto al que pertenece, el ID del árbol, el **ID de la especie a la que pertenece el árbol**, una descripción, la latitud y longitud para las coordenadas, la url del sitio web donde se aloja la imagen tomada al árbol y una **lista de la clase de medidas**.

- **Función toMap**: esta función se utiliza para convertir un objeto de la clase TreeDataSheet en un mapa de cadenas clave-valor. El mapa resultante se utiliza para la serialización y almacenamiento de datos.

- **Función fromJson**: esta función es una función de fábrica que se utiliza para convertir un mapa de cadenas clave-valor en un objeto de la clase TreeDataSheet. Se utiliza para la deserialización de datos.

- **Función toJson**: esta función se utiliza para convertir un objeto de la clase TreeDataSheet en una cadena JSON. Se utiliza para la serialización de datos.

- **Función parseTreeDataSheets**: función de fábrica que se utiliza para analizar y convertir una respuesta HTTP en forma de cadena de texto en una lista de objetos de tipo TreeDataSheet.

Figura 4.30 – *Modelo de ficha de datos en frontend*

Modelo especie de árbol Este modelo se crea para poder almacenar las diferentes especies de árboles que se necesitan dar de alta en la aplicación, para poder proceder más tarde a la asignación de una especie para cada ficha de datos, ya que es un campo obligatorio.

- **Variables del modelo**: aquí se definen las variables que forman parte del modelo de datos de una especie de árbol. Estas variables incluyen el ID, nombre, y la descripción.
- **Función toMap**: esta función se utiliza para convertir un objeto de la clase TreeSpecie en un mapa de cadenas clave-valor. El mapa resultante se utiliza para la serialización y almacenamiento de datos.
- **Función fromJson**: esta función es una función de fábrica que se utiliza para convertir un mapa de cadenas clave-valor en un objeto de la clase TreeSpecie. Se utiliza para la deserialización de datos.
- **Función toJson**: esta función se utiliza para convertir un objeto de la clase TreeSpecie en una cadena JSON. Se utiliza para la serialización de datos.

```dart
class TreeSpecie{
  final String id;
  final String name;
  final String description;

  TreeSpecie(
    {required this.name, required this.id, required this.description}
  );

  // Create a object map
  Map<String, dynamic> toMap(){
    return {
      "_id": id,
      "name": name,
      "description": description,
    };
  }

  // Factory function to parse from JSON to object
  factory TreeSpecie.fromJson(Map<String, dynamic> parsedJson){
    return TreeSpecie(
      id: parsedJson["_id"] ?? '',
      name: parsedJson["name"],
      description: parsedJson["description"] ?? '',
    );
  }

  factory TreeSpecie.parseTreeSpecies(String responseBody) {
    final parsed = json.decode(responseBody).cast<Map<String, dynamic>>();
    return parsed.map<TreeSpecie>((json) => TreeSpecie.fromJson(json)).toList();
  }

  // Create toJson method
  String toJson() => json.encode(toMap());

}
```

Figura 4.31 – *Modelo de especie de árbol en frontend*

Modelo medidas Este modelo se crea para poder tener una representación de las medidas tomadas del árbol y también poder forma dicho objeto cuando consultamos una medida de una ficha de árbol desde la base de datos. Para esto última necesitaremos la función fromJSon.

- **Variables del modelo**: aquí se definen las variables que forman parte del modelo de datos de una medida de árbol. Estas variables incluyen la distancia, el tiempo de la medida y la velocidad media.
- **Función fromJson**: esta función es una función de fábrica que se utiliza para convertir un mapa de cadenas clave-valor en un objeto de la clase Measurement. Se utiliza para la deserialización de datos.

```dart
class Measurement{
  double? distance;
  double? time;
  double? avgVelocity;

  Measurement({ this.distance, this.time, this.avgVelocity});

  // Create a object map
  Map<String, dynamic> toMap(){
    return {
      "distance": distance,
      "time": time,
      "avgVelocity": avgVelocity?.toDouble(),
    };
  }

  // Factory fucntion to parse from JSON to object
  factory Measurement.fromJson(Map<String, dynamic> parsedJson){
    return Measurement(
      distance: double.parse(parsedJson["distance"].toString()),
      time: double.parse(parsedJson["time"].toString()),
      avgVelocity: double.parse(parsedJson["avgVelocity"].toString()),
    );
  }

  // Create toJson method
  String toJson() => json.encode(toMap());

}
```

Figura 4.32 – *Modelo de medidas en frontend*

Una vez definidos los modelos, mostramos la pantalla de fichas de datos que nos servirá tanto de consulta como para crear una nueva.

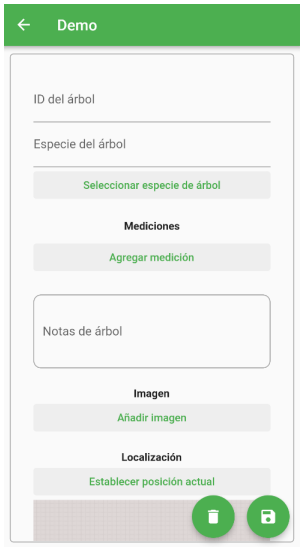


Figura 4.33 – *Pantalla de ficha de datos*

Con respecto al mockup que se diseñó al principio 4.8 (Mockup pantalla de ficha), el único cambio notable que se ha introducido ha sido la relocalización de los botones de borrado y edición, pasando a estar en parte inferior de dicha pantalla. Dichos botones tienen la función que su nombre indica, estos llamarán a dos servicios diferentes que mandarán al servidor la petición de borrado o actualización y una vez la respuesta de vuelta, si ha sido un borrado redirige al usuario a la pantalla del proyecto, si bien ha sido una actualización cambia los valores de los campos de la ficha de datos por lo cuales el usuario haya modificado.

El diseño final recae en la sencillez de una única lista de campos a rellenar, donde encontraremos primeramente los que son obligatorios, seguidos de los opcionales. Tenemos la siguiente lista:

- **ID del árbol.** Compuesto por un widget de tipo **CustomTextFormField** donde el usuario agregará el identificador del árbol. Este campo será también el nombre que poseerá la ficha de datos.
- **Especie del árbol.** En este campo tendremos un widget **CustomTextFormField** protegido para que el usuario no pueda escribir contenido, si no que para poder establecer la especie deba hacer click en el botón justamente debajo y seleccionar la especie deseada. Aquí entra en juego el modelo de especies de árboles, ya que una vez el usuario haga click, se hará uso de un servicio que realice una llamada al servidor para traer al frontend las especies disponibles que hay en la base de datos. Esto se muestra al usuario en forma de listado con un buscador que permitirá la búsqueda por nombre, también se agrega la facilidad de permitir al usuario moverse al primer o último registro del listado haciendo click en las flechas que aparecen. Una vez se selecciona la especie, se llama a otro servicio para poder traer toda la información de dicha especie.

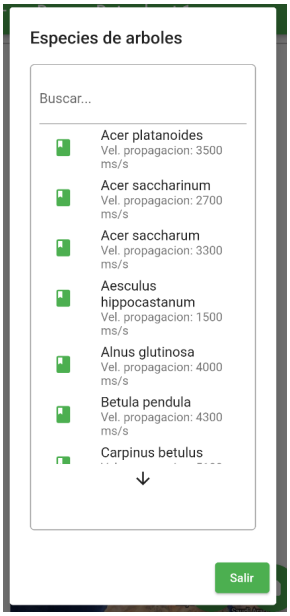


Figura 4.34 – *Diálogo mostrar especies de árboles*

```dart
Future<dynamic> getSpecies() async {
  final response = await http.get(
      Uri.parse('$url/treespecies/getall'),
    );

  if (response.statusCode == 200) {
    List<dynamic> listJson = json.decode(response.body);

    return listJson;
  } else {
    throw Exception('Error al obtener las especies de árboles');
  }
}

Future<dynamic> findSpecie(String _id) async {
  final response = await http.get(
      Uri.parse('$url/treespecies/${_id}'),
    );

  if (response.statusCode == 200) {
    Map<String, dynamic> listJson = json.decode(response.body);

    return listJson;
  } else {
    throw Exception('Error al obtener la especie de árbol seleccionada');
  }
}
```

Figura 4.35 – *Código llamadas a servicios de especies*

- **Mediciones.** En este apartado, nos encontramos únicamente con un botón si no hay mediciones guardadas en la ficha de datos. Una vez que se haga click en el botón, aparece un diálogo que pedirá al usuario introducir las medidas de distancia y tiempo, una vez agregadas, aparecerá en la ficha de datos una tabla con las mediciones ya en pantalla. Aquí entra en juego el modelo de mediciones mencionado más arriba.

Figura 4.36 – *Tabla de mediciones en ficha de datos*

Podemos observar que hay un botón al lado de cada registro que se introduce en la tabla para borrarlo si es necesario.

- **Notas de árbol.** Un campo dedicado a exponer cualquier información de cierta relevancia en la ficha de datos. Se aplica un widget **CustomFormField** con múltiples líneas.
- **Imagen.** Para este apartado se crea un botón especifico para poder acceder a la cámara del dispositivo del usuario y poder tomar una foto en tiempo real del árbol que se está analizando. Esto lo conseguimos gracias a la librería de flutter **Camera**, todo el código referente a su implementación, lo encontramos en el archivo de **custom_camera** del directorio **common**.

Figura 4.37 – *Ejemplo de imagen tomada y mostrada en la aplicación*

Una vez el usuario haya tomado la foto, se muestra en la ficha de datos, almacenándose temporalmente en el dispositivo móvil antes de ser guardada en el servidor cuando el usuario presione el botón de guardado de ficha de datos.**Como se pudo observar en el modelo de ficha de datos, la imagen se almacena como una URL, para ello profundizaremos más en esto en la sección 5.4.4 (Funcionalidad almacenamiento de imágenes).**

- **Localización.** En esta parte, definimos un botón y un mapa que mostrará las coordenadas del usuario en tiempo real, haciendo uso de la función GPS del dispositivo móvil. Para este apartado se necesita la librería de flutter **Geolocator** que permite acceder a dicha funcionalidad del dispositivo. Así mismo para el mapa, se hace uso de la librería **Google Maps Flutter**[7] que nos permite a través de una API key (se debe configurar en los archivos de configuración del proyecto de flutter) del servicio de Google Maps, establecer un mapa en nuestra aplicación.El mapa nos permite tener una integración con la app Google maps y así trasladar las coordenadas para mostrar una ruta hacia la localización del árbol. También nos encontramos con otro botón definido debajo del mapa, que permitirá al usuario variar entre la topología de este mismo, cambiando de tipo satélite a tipo relieve. Todo el código de implementación del mapa se encuentra en el archivo **custom_map** del directorio **common**.

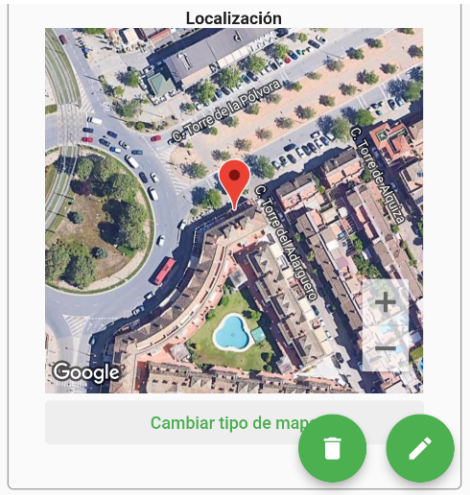


Figura 4.38 – *Ejemplo de localización obtenida*

4.5.4. Funcionalidades genéricas de la aplicación

Una vez se han explicado los sistema de usuarios y proyectos que componen el groso de la aplicación, dedicaremos ahora esta sección a comentar diversas funcionalidades que tiene la aplicación haciendo que esta sea complementada para poder brindar una mejor experiencia al usuario. Dichas funcionalidades se dividen en tres bloques:

- **Visor PDF de manual de uso**. Para poder ofrecer al usuario una manera sencilla de consultar el como manejar la aplicación sin tener que ingeniárselas por si solo, se desarrolla un visor de archivos PDF. Esto podrá lograrlo el usuario haciendo click en el botón **"Abrir manual"** de 4.21 (Versión final pantalla principal), una vez hecho el click se la abrirá una nueva pantalla donde podrá seleccionar si desea descargarlo o simplemente consultar el manual desde la propia pantalla.

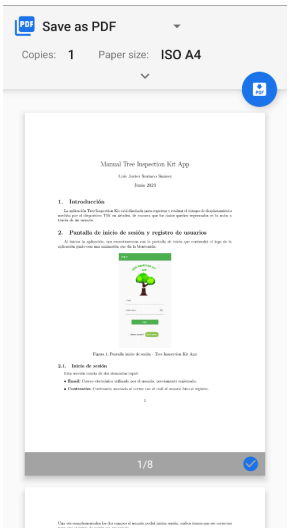


Figura 4.39 – *Pantalla de visor de manual*

```
SizedBox(
  width: 300,
  child: CustomElevatedButton(
    title: AppLocalizations.of(context)!.openManual,
    onPressed: () async
    {
      // Load pdf bytes from asset (version depends on user selected language)
      final pdfData = await loadPdfFromAsset(languageCode: Localizations.localeOf(context).languageCode);
      // Using printing library, open pdf and visualize
      Printing.layoutPdf(
       onLayout: (format) => pdfData,
      );
    },
    icon: const Icon(Icons.book),
  ), // CustomElevatedButton
), // SizedBox
```

Figura 4.40 – *Código para poder construir la pantalla del visor*

Para poder cargar el PDF en la aplicación se debe definir en el archivo **pubspec.yaml** como un **asset** y luego llamar internamente al archivo usando la librería **printing**. Actualmente la aplicación cuenta con dos versiones del manual para el idioma español e inglés, dependiendo del idioma que tenga seleccionado el usuario en la aplicación.

- **Sistema multilenguaje [11]** Este sistema facilita al usuario a lidiar con la aplicación en diversos idiomas. Actualmente los idiomas disponibles son el español e inglés, sin embargo, de la forma en que este sistema se ha implementado, permite en cualquier momento poder agregar tantos idiomas como se desee, únicamente agregando más plantillas. A continuación se procede a explicar como se ha gestionado y configurado el sistema de idiomas en el proyecto.

  Para poder llevar a cabo dicha tarea, es necesario instalar las librerías **intl** y **Flutter Localizations**. Una vez se han instalado, necesitamos configurar varios archivos para definir los idiomas junto con sus traducciones.

  El primero de ellos lo llamaremos **l10n.yaml**, en dicho archivo definimos lo siguiente:

```
l10n.yaml
1  arb-dir: lib/l10n
2  template-arb-file: app_es.arb
3  output-localization-file: app_localizations.dart
```

Figura 4.41 – *Fichero l10n.yaml para configuración de idiomas*

  - **arb-dir.** Especifica la ruta del directorio donde se encuentran los archivos de traducción .arb. En nuestro caso la ruta será **lib/l10n**. que se menciona en la sección 4.4.1 (Directorio principal)
  - **template-arb-file.** Especifica el nombre del archivo .arb que se utilizará como plantilla para generar el código de localización. En nuestro caso será la plantilla en español.
  - **output-localization-file.** Especifica el nombre del archivo de salida que contendrá el código de localización generado. Este archivo es generado al hacer uso del comando **flutter gen-l10n**, lo que permite actualizar las plantillas en la aplicación.

Una vez definido el archivo de configuración, tendremos que crear las plantillas de clave-valor para crear los idiomas deseados y comenzar con las traducciones. Como hemos mencionado esto se encuentra en el directorio **lib/l10n**.

```
l10n.yaml
1  arb-dir: lib/l10n
2  template-arb-file: app_es.arb
3  output-localization-file: app_localizations.dart
```

Figura 4.42 – *Contenido de directorio l10n*

Figura 4.43 – *Plantillas de idiomas español e inglés*

Ahora, necesitaremos en cada apartado que se requiera mostrar una traducción, utilizar la siguiente línea **AppLocalizations.of(context)** seguido de un punto y elegimos la clave que queremos utilizar. Por ejemplo, si necesitamos la traducción de **"Crear cuenta"**, debemos poner lo siguiente, **AppLocalizations.of(context).createAccount**, que se ha definido previamente en las plantillas que deseamos usar.

Por último, necesitamos crear una opción en la aplicación donde el usuario pueda alternar entre los idiomas disponibles. Para ello, en la pantalla 4.21 (Versión final pantalla principal), una vez desplegamos el menú lateral, podemos observar abajo a la izquierda el control para el cambio de idioma.

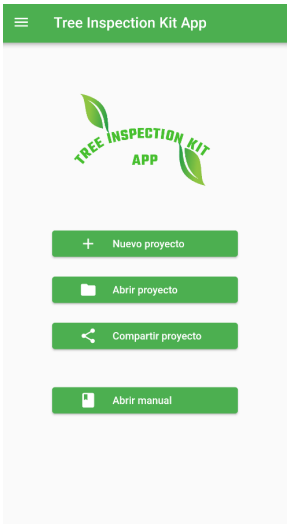


Figura 4.44 – *Pantalla principal en español*

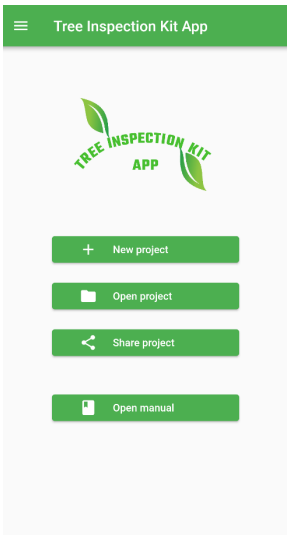


Figura 4.45 – *Pantalla principal en inglés*

- **Funcionalidad Bluetooth**. Esta funcionalidad esta parcialmente completada debido al tiempo de desarrollo que se ha invertido al proyecto. Dicha función de bluetooth, una vez completada, tiene como objetivo poder conectarse al dispositivo TIK para poder obtener directamente los datos calculados de este dispositivo. Comentar por un lado, que el módulo de bluetooth del dispositivo TIK todavía no ha sido implementado por sus creadores, por lo que dicha funcionalidad quedará como trabajo futuro en el desarrollo tanto de la aplicación como del dispositivo.

Para poder implementar el apartado del bluetooth, se hizo uso de la librería **Flutter Reactive BLE** [13] la cual es una de las más populares para lograr este objetivo. Una vez que se instala la librería debemos crear varias clases que gestionarán todo lo relacionado a ello, inclusive el botón que encontramos en el desplegable de la pantalla principal 4.21 (Versión final pantalla principal).

Cuando el usuario realice un click sobre el botón **"Buscar dispositivos bluetooth"** se abrirá un diálogo en el cual aparecerá un listado de los dispositivos bluetooth de bajo consumo que el dispositivo móvil está escaneando en sus alrededores. Si bien no encuentra ninguno, se mostrará un circulo de progreso para indicar que está realizando la búsqueda. Además el usuario dispondrá de un botón en el propio diálogo para poder refrescar el listado en cualquier momento, esto hará un borrado del mismo e iniciará una búsqueda desde cero.

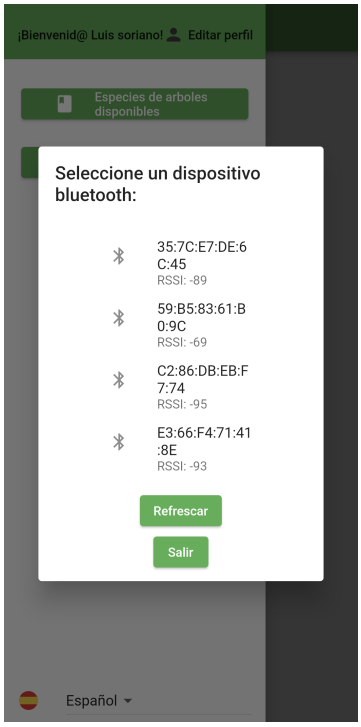


Figura 4.46 – *Diálogo bluetooth con dispositivos encontrados*

Ahora pasaremos a mostrar el contenido de las clases para poder obtener el listado de los dispositivos bluetooth.

Contamos con tres archivos diferentes que harán todo el trabajo, dos de ellos widgets y uno de ellos una clase que hará el papel de escáner bluetooth:

- **BluetoothSimpleDialog**. Está clase será la encargada de mostrar por pantalla el diálogo del listado de dispositivos. Una clase bastante sencilla que no requiere de ningún método a excepción de **build** (obligatorio para todos los widgets) y **didChangeDependencies** usado para poder obtener el contexto del idioma usado en la app y poder usar la traducción adecuada.

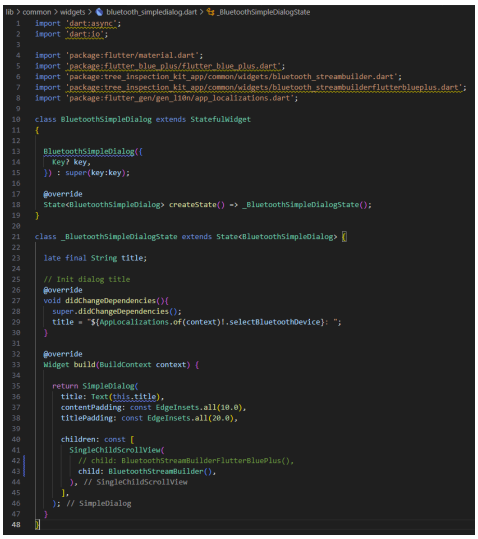


Figura 4.47 – *Diálogo bluetooth con dispositivos encontrados*

- **BluetoothScanner**. Esta clase será la encargada de tener el rol de escáner de bluetooth para poder usar las funcionalidades que nos ofrece la librería **Flutter Reactive BLE**. En dicha clase nos encontramos con una instancia de **flutter reactive ble**, un **controlador de flujo de dispositivos descubiertos**, **una suscripción de flujo de dispositivo descubierto**, **lista de dispositivos descubiertos** y **lista de uuid de servicios** por si queremos buscar dispositivos bluetooth con dichos uuids de servicio. El dispositivo descubierto es un tipo de Objeto de la librería.

Figura 4.48 – *Diálogo bluetooth con dispositivos encontrados*

Necesitamos implementar un método para poder preguntar al usuario si concede acceso a que la aplicación utilice el servicio de localización de su dispositivo móvil, esto se realiza por motivos de privacidad. Este método es llamado **requestLocationPermission**

También necesitamos implementar los métodos de comienzo y finalización del escaneado y un borrado de la lista para cuando el usuario quiera refrescar el listado. **startScan** se encargará de preguntar el permiso al usuario, una vez concedido, empezamos a escanear los dispositivos y vamos añadiendo a la lista de dispositivos descubiertos y al controlador de flujo aquellos que son escuchados a través de la suscripción de flujo de dispositivo descubierto.

- **BluetoothStreamBuilder**. Esta clase se encargará de construir el listado de dispositivos en tiempo real, gracias a la instancia de la clase **BluetoothScanner** que nos permite acceder a los datos que recoge por momentos.

Figura 4.49 – *Widget bluetooth stream builder*

Para poder lograr esto, necesita hacer uso de un widget especial de flutter llamado **StreamBuilder**, el cual tiene la peculiaridad de escuchar un flujo de datos y permitir realizar cualquier acción que sea posible con estos mismos. Este widget necesita un flujo para poder construir, aquí entra en juego el flujo de dispositivos descubiertos por el escáner, por lo tanto, usaremos BluetoothScanner.devicesStream para dicho flujo. Junto a esto, y la lista de dispositivos encontrados en **BluetoothScanner.devicesList** podemos mostrar por pantalla gracias a otros widgets genéricos de flutter, aquellos dispositivos encontrados.

La información que se muestra por pantalla, será el nombre del dispositivo bluetooth (si bien no tiene se muestra la dirección MAC) y el RSSI que es la intensidad de la señal de radio que emite el dispositivo bluetooth expresada en decibelios.

4.5.5. Seguridad en los datos

En un proyecto software debemos de tener especial cuidado a la hora de tratar los datos que se manejan dentro del mismo.

Para ello en la implementación del frontend, se ha optado por hacer uso de la librería **Http dart** para la transmisión y recepción de los datos hacía el servidor.

Dicha librería utiliza una conexión segura a través del protocolo **HTTPS**, lo cual es una buena práctica para proteger la confidencialidad de los datos mientras son transmitidos. Además, se especifica en las cabeceras de ciertas peticiones enviadas al servidor el formato y codificaciones, lo cual es útil para asegurar que los datos sean interpretados correctamente por el servidor y evitar problemas de seguridad como la inyección de código malicioso.

La librería también realiza automáticamente la verificación de certificados al establecer una conexión segura. Esto significa que comprueba que el certificado del servidor es válido y confiable, y que coincide con el nombre de dominio al que se está accediendo. Si el certificado no es válido, la conexión se considera insegura y se rechaza. Esto ayuda a prevenir ataques de suplantación de identidad.

También debemos hacer especial hincapié a la hora de transmitir contraseñas a través de las peticiones que se mandan al servidor. Es impensable enviar dichas contraseñas sin antes realizarles una encriptación y almacenar el hash en la base de datos. Para lograr este punto, se hace uso de la librería **crypto**, la cuál nos facilita todo el trabajo. Antes de enviar la contraseña del usuario al servidor en los servicios de login, registro y edición de perfil se tiene que encriptar. La función que se encarga de realizar el hash se encuentra en el archivo **utils.dart**.

```dart
String getPasswordHash(String password)
{
  final bytes = utf8.encode(password);
  final hash = sha256.convert(bytes);
  final encryptedPassword = hash.toString();

  return encryptedPassword;
}
```

Figura 4.50 – *Función utilizada para obtener hash de contraseña*

Por desgracia, la aplicación no dispone de un sistema de tokens que haría a la aplicación adquirir un plus en seguridad. Este sistema queda registrado como un trabajo a futuro, reflejado en el capítulo 7.2 (Trabajos futuros).

Capítulo 5

Diseño e implementación backend

5.1. Introducción

Para gestionar las peticiones que realiza la aplicación a través de los sistemas de usuarios, de proyectos y funcionalidades varias, es necesario implementar un backend que se encargue de dichas peticiones y devolver las respuestas.

Durante el desarrollo del mismo, se ha optado por hacer uso de **NodeJS**, el cuál es un entorno de ejecución de código abierto y multiplataforma que permite ejecutar JavaScript fuera del navegador web.

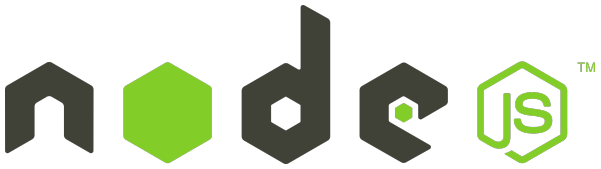


Figura 5.1 – *Logo de NodeJS*

Para poder tener un contexto acerca de lo que es capaz NodeJs, se listan las siguientes características que hacen que sea una opción popular para el desarrollo de aplicaciones del lado del servidor, servicios web y aplicaciones en tiempo real, ofreciendo un rendimiento eficiente y un entorno de desarrollo productivo.

- **Rendimiento y escalabilidad** - Node.js utiliza el motor V8 de Google Chrome, que compila el código JavaScript en código máquina altamente optimizado. Esto permite un rendimiento rápido y eficiente, lo que hace que Node.js sea adecuado para aplicaciones que requieren una alta concurrencia y procesamiento de solicitudes en tiempo real.
- **Modelo de E/S sin bloqueo y orientado a eventos** - Node.js utiliza un enfoque de E/S sin bloqueo y basado en eventos, lo que significa que puede manejar múltiples solicitudes concurrentes sin bloquear el hilo principal.
- **Amplio ecosistema de paquetes** - Node.js cuenta con npm (Node Package Manager), un sistema de gestión de paquetes que permite a los desarrolladores instalar, compartir y administrar

fácilmente módulos y bibliotecas de terceros. El repositorio de paquetes de proporciona una amplia gama de opciones para extender las funcionalidades de las aplicaciones.

- **Desarrollo basado en JavaScript** - Es un lenguaje popular y ampliamente adoptado, lo que significa que hay una gran cantidad de recursos y herramientas disponibles para los desarrolladores.
- **Soporte de tiempo real y aplicaciones en tiempo real** - Node.js es especialmente adecuado para construir aplicaciones en tiempo real, como chats en vivo y aplicaciones de transmisión. Su modelo de eventos y su capacidad para manejar solicitudes de forma asíncrona lo hacen ideal para escenarios en los que la actualización en tiempo real es esencial.
- **Facilidad de creación de API y servicios web** - Node.js proporciona una serie de módulos y herramientas que facilitan la creación de API y servicios web. Esto, combinado con su capacidad para manejar múltiples solicitudes concurrentes, lo convierte en una opción popular para construir servicios backend robustos y escalables.
- **Comunidad activa y abundante documentación** - Su comunidad ofrece una amplia gama de recursos, incluyendo documentación detallada, tutoriales y ejemplos de código. Esto facilita el aprendizaje, la resolución de problemas y la colaboración con otros desarrolladores.

5.2. Estructura

A la hora de desarrollar un backend para la aplicación, necesitamos definir una estructura para organizar los diferentes archivos que contienen toda la lógica del mismo.

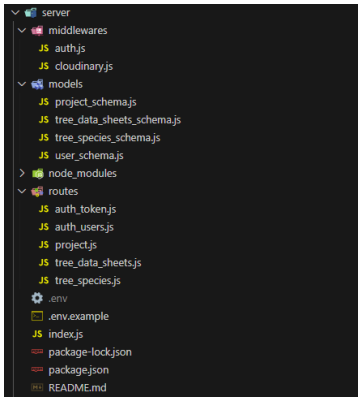


Figura 5.2 – *Estructura de backend de la aplicación*

Se define en una primera instancia el directorio **Server**, el cual será el principal que contendrá todos los subdirectorios y archivos.

- **Middlewares** es un directorio para contener todos los middleware que se han desarrollado. Un middleware es una función o conjunto de funciones que se utilizan para procesar y manipular las solicitudes y respuestas HTTP, puede realizar diversas tareas, como el análisis de datos de la solicitud, la modificación del objeto de solicitud o respuesta... En nuestro caso, tenemos un único middleware en el directorio, que será protagonista en la sección de 5.4.4 (Funcionalidad almacenamiento de imágenes) para tratar el tema de gestión de imágenes en las fichas de datos.

- **Models** contiene los ficheros donde se definen los modelos de datos que se utilizan en la base de datos elegida. Al fin y al cabo, se trata de definir las tablas si se trata de una base de datos SQL o bien definir la estructura de documentos si se trata de una base de datos No SQL. Se centran en los tipos de los campos, si son obligatorios para su almacenamiento, también definir los índices...
- **Routes** es el encargado de contener las rutas o endpoints de la aplicación. Se dividen a su vez en varios archivos para diferencia cada funcionalidad. Es decir, las rutas para la estion de usuario tendrán su archivo propio e independiente del archivo para gestionar las rutas del sistema de gestión de proyectos.
- **.env** es un archivo de configuración que servirá para proteger variables que pueden variar según el entorno de desarrollo. Se almacenan variables tales como credenciales de bases de datos, claves para el uso de APIs, configuraciones de conexión...
- **Index.js** es el archivo principal de NodeJS para poder iniciar el servicio. Aquí se importan los módulos necesarios, se configuran las rutas y se inicia el servidor HTTP para escuchar las solicitudes entrantes.

Figura 5.3 – *Archivo index del backend*

Como se aprecia en la figura de arribar, en nuestro archivo **index.js** se cargan las variables de entorno (línea 13), se configura el uso de una librería con las claves de su API (línea 23), cargamos Middlewares de librerías (línea 32), cargamos las rutas definidas por nosotros (línea 37) y definimos la función asíncrona para el inicio de la aplicación.

- **Package.json** es un archivo de configuración fundamental en una aplicación Node.js. Utilizado para definir y gestionar las dependencias del proyecto, configurar scripts de ejecución, especificar metadatos y realizar otras tareas relacionadas con la administración del proyecto. El formato de este archivo es de tipo JSON.

5.3. Base de datos y dependencias

En esta sección comentaremos la base de datos que se ha elegido para almacenar todos los datos de nuestra aplicación, así como las librerías que se han usado en el backend para poder implementar

la lógica de este mismo.

5.3.1. Base de datos

A la hora de elegir para el backend que tipo y que base de datos necesitamos, se investigó acerca de las diferencias generales entre BBDD SQL y BBDD NoSQL[15] [17]. Tenemos lo siguiente:

- **Estructura de los datos**:
  - SQL. Los datos están basados en un modelo relacional mediante tablas con filas y columnas.
  - NoSQL. Los datos están basados en colecciones de documentos, pares de clave-valor o grafos.
- **Lenguaje**:
  - SQL. Utiliza el lenguaje SQL (Structured Query Language), que es un estándar para realizar consultas y manipulación de datos en bases de datos relacionales.
  - NoSQL. No tiene un lenguaje específico, ya que depende de la base de datos NoSQL utilizada. Por ejemplo, en MongoDB los datos se almacenan en formato JSON y las consultas se basan en un lenguaje similar a JavaScript.
- **Esquemas**:
  - SQL. Presenta un esquema predefinido que define la estructura de las tablas y ayuda a preservar la integridad de los datos. Requiere una definición de esquema antes de insertar datos y proporciona rigidez en la estructura de los datos.
  - NoSQL. Utiliza un esquema dinámico que permite mayor flexibilidad. No requiere una definición de esquema previa y permite tener diferentes documentos con distintos campos en una misma base de datos. Esto brinda mayor agilidad y adaptabilidad a medida que los requisitos cambian con el tiempo.
- **Integridad**:
  - SQL. Ofrece un alto grado de integridad y cumple con los principios ACID (Atomicidad, Consistencia, Aislamiento y Durabilidad). Esto garantiza la consistencia y confiabilidad de los datos, lo cual es especialmente importante en aplicaciones transaccionales.
  - NoSQL. Mantener la integridad puede ser más complejo, especialmente en entornos distribuidos donde puede haber inconsistencia temporal. Algunas bases de datos NoSQL ofrecen garantías de consistencia eventual en lugar de consistencia inmediata.
- **Escalabilidad**:
  - SQL. Escala principalmente de manera vertical, añadiendo más recursos como CPU y memoria a un servidor. No escala de manera tan eficiente en entornos distribuidos con grandes conjuntos de datos.
  - NoSQL. Escala de manera horizontal, lo que significa que se puede agregar más servidores y distribuir los datos entre ellos. Es especialmente adecuado para escenarios donde se requiere una alta escalabilidad y distribución de datos en múltiples ubicaciones.
- **Consultas**:
  - SQL. Es eficiente en consultas estructuradas y complejas gracias a la capacidad de realizar joins entre tablas y aprovechar el poder del motor de consulta relacional.

- NoSQL. Puede requerir más tiempo y esfuerzo, especialmente cuando las consultas son más complejas, ya que no todos los sistemas NoSQL admiten las mismas funcionalidades de consulta. Algunas bases de datos NoSQL ofrecen capacidades de consulta más avanzadas, como índices secundarios y búsqueda basada en texto.

Finalmente viendo las diferencias que hay entre un tipo u otro, el criterio que hace que nos inclinemos por un tipo ha sido la experiencia que tiene el autor del proyecto con el uso de base de datos no relacionales. Esto hace que el desarrollo del backend haya sido más rápido debido al conocimiento previo sobre el uso de NoSQL.

De entre todas las bases de datos NoSQL que nos podemos encontrar, se decidió usar MongoDB por el mismo motivo que se decide usar NoSQL, el conocimiento previo del autor del proyecto. También debemos destacar que **MongoDB** es una base de datos de documentos ampliamente utilizada y muy popular que combina escalabilidad, flexibilidad y rendimiento.

También nos podremos beneficiar de el servicio que nos ofrece **MongoDB Atlas** para poder hostear la base de datos desde la nube y no depender de un sistema local. **Este apartado lo veremos en la sección 6 (Despliegue de aplicación)**.

5.3.2. Dependencias

En esta sección se explicará brevemente las dependencias que se han utilizado en la parte backend de la aplicación.

server > package.json > ...

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "type": "module",
  Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
  "dependencies": {
    "bcryptjs": "2.4.3",
    "body-parser": "^1.20.2",
    "cloudinary": "^1.36.1",
    "cookie-parser": "^1.4.6",
    "dotenv": "^16.0.3",
    "exceljs": "^4.3.0",
    "express": "^4.18.2",
    "mongoose": "6.7.1",
    "nodemon": "2.0.20"
  }
}
```

Figura 5.4 – *Archivo de dependencias - package.json*

- **bcryptjs**: proporciona funciones de hash y comparación de contraseñas seguras utilizando el algoritmo bcrypt. Es ampliamente utilizado para almacenar contraseñas de forma segura en bases de datos, ya que utiliza un proceso de hash unidireccional que es difícil de revertir. **Aplicaremos su utilidad en el sistema de usuarios para sus contraseñas**.
- **body-parser**: es un middleware de Node.js que analiza los datos enviados en el cuerpo de una solicitud HTTP. Convierte los datos en un objeto JavaScript utilizable y los adjunta al objeto req (solicitud) de Express. Esto permite acceder fácilmente a los datos enviados por el cliente en formato JSON, formularios HTML u otros formatos. **Se intentó aplicar su utilidad para gestionar las imágenes de las fichas de datos** en la sección 5.4.4 (Funcionalidad almacenamiento de imágenes) se explicará que se intentaba conseguir con ella.
- **cloudinary**: una biblioteca que nos proporciona interacción con la plataforma de almacenamiento y manipulación de imágenes en la nube de Cloudinary. Permite cargar, almacenar, transformar

y entregar imágenes y vídeos para su uso en aplicaciones. **Aplicaremos su utilidad en el sistema de proyectos a la hora de gestionar las imágenes de las fichas de datos**.

- **cookie-parser**: un middleware que analiza las cookies enviadas en las solicitudes HTTP y las adjunta al objeto req (solicitud) de Express. Proporciona una forma sencilla de leer y establecer cookies en el lado del servidor.

- **dotenv**: una biblioteca que carga variables de entorno desde un archivo .env. Permite configurar fácilmente variables específicas de entorno en diferentes etapas de desarrollo (como credenciales de API, claves secretas, etc.) sin tener que modificar el código fuente. El archivo .env se utiliza para almacenar estas variables y se carga en el entorno de ejecución de la aplicación.

- **exceljs**: nos permite trabajar con archivos de Excel en Node.js. Permite leer, escribir y modificar datos en hojas de cálculo de Excel, así como formatear y manipular el contenido de las celdas. **Aplicaremos su utilidad en el sistema de proyectos a la hora de exportarlos**.

- **express**: un framework web rápido, minimalista y flexible para Node.js. Permite crear fácilmente aplicaciones web y APIs robustas utilizando enrutamiento, middleware y gestión de solicitudes y respuestas HTTP de manera sencilla. **Aplicaremos su utilidad para todo el sistema de rutas que necesitamos en nuestro backend**.

- **mongoose**: una biblioteca de modelado de objetos de MongoDB para Node.js. Proporciona una capa de abstracción sobre MongoDB que simplifica la interacción con la base de datos y facilita la definición de esquemas, validación de datos y consultas. **Aplicaremos su utilidad para crear y definir los modelos que necesitamos para la aplicación, conectarnos a la base de datos y realizar todo tipo de operaciones con ella**.

- **nodemon**: una herramienta de desarrollo que monitorea los cambios en los archivos de la aplicación y reinicia automáticamente el servidor cuando se detectan cambios. Esto es útil durante el desarrollo, ya que evita tener que reiniciar manualmente el servidor cada vez que se realizan modificaciones en el código. Nodemon mejora la productividad del desarrollador al acelerar el ciclo.

5.4. Implementación de funcionalidades

Una vez que se ha explicado la base de datos elegida y las dependencias que se utilizan en el backend, ahora nos adentraremos en el proceso de desarrollo y programación para llevar a cabo la implementación de las distintas funcionalidades que ha sido diseñadas en la aplicación para poder complementar la parte frontend.

5.4.1. Sistema de usuarios

Para poder terminar de implementar el sistema de usuarios en la aplicación, necesitamos la parte de gestión en el backend para el almacenamiento de los datos de estos mismos. Para comenzar se define el modelo que debe tener la colección de documentos de MongoDB para guardar un usuario registrado. Este modelo se definirá en el archivo **user_schema** dentro del subdirectorio **Models**.

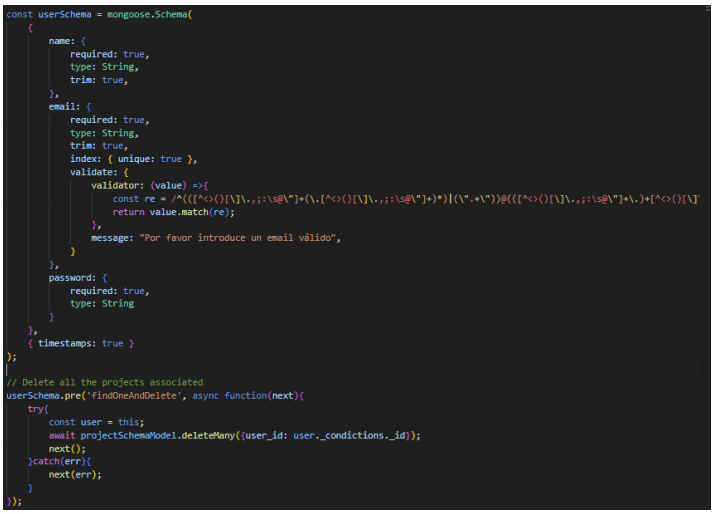


Figura 5.5 – *Modelo de usuario para backend*

En el modelo definimos el tipo de campos que un usuario debe tener almacenados en la base de datos. Estos campos serán, el nombre **(name)**, el correo **(email)** y la contraseña (**password**).

A la hora de almacenar el correo del usuario, metemos una validación en forma de expresión regular que debe cumplir. Está validación será comprobar que el valor del campo sea, efectivamente, una dirección de correo válida. La expresión utilizada se encuentra en la línea 21[19].

A la hora de borrar un usuario tenemos que tener en cuenta que este podrá tener proyectos asociados. Por ello sería un problema, tener en la colección de proyectos, datos que no están relacionados con ningún usuario. Para ello necesitamos hacer uso de otro **pre-hook**. Esta vez, la función se ejecuta antes de borrar un registro usuario, buscando todos los proyectos que este puede tener y eliminándolos de la colección de proyectos. De esta manera evitamos dichos problemas.

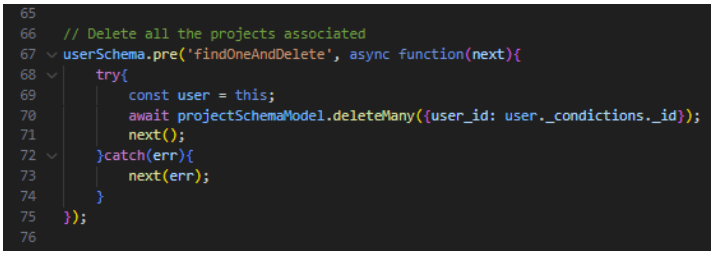


Figura 5.6 – *Función pre-hook utilizado al borrar un usuario*

5.4.1.1. Rutas definidas en sistema de usuarios

A partir del modelo de datos del usuario, necesitamos crear las rutas o endpoints que usaremos en el frontend para poder realizar operaciones CRUD en la aplicación. Estos endpoints los encontramos en el archivo **auth_users** en el directorio **routes** y son los siguientes:

Figura 5.7 – *Rutas para la gestión del usuario*

- Endpoints para obtener los detalles de un usuario concreto mediante el correo electrónico.
- Endpoints exactamente igual al anterior, utilizando el correo y contraseña.
- El endpoint para el registro de usuarios, donde los datos necesarios para el registro se envían a través del cuerpo de la petición, los cuales son proporcionados por el frontend.
- El endpoint destinado al inicio de sesión, en el cual necesitaremos para tanto email y contraseña para posteriormente. Posteriormente, dentro de la lógica del endpoint, se verifica la existencia del correo electrónico y se comprueba si la contraseña es correcta.
- El endpoint de actualización o edición de perfil de usuario, que tiene como objetivo actualizar los campos que se reciben en la petición y que son modificados desde el frontend. Si se modifican las contraseñas, se comprueba previamente que coincida la contraseña con la contraseña de confirmación.

Para mostrar un ejemplo de la lógica de un endpoint, mostraremos el contenido de la ruta de edición de perfil de usuario:

Figura 5.8 – *Endpoint para actualización de perfil*

5.4.2. Sistema de proyectos

Para poder terminar de implantar el sistema de proyectos en la aplicación, necesitamos la parte de gestión en el backend para el almacenamiento de los datos de estos mismos. En este sistema debemos tener listos dos modelos, lo cuales serán, el modelo de proyectos y el modelo de fichas de datos. Estos modelos se definirán en los archivos **project_schema** y **tree_data _sheets_schema** dentro del subdirectorio **Models**.

- **Proyectos**

Figura 5.9 – *Modelo de proyecto para backend*

El tipo de campos que necesitaremos en el modelo a la hora de almacenar un proyecto, serán el nombre **(name)**, la descripción **(description)**, y el ID del usuario al que está asociado **(user_id)**.

A su vez, debemos definir un índice que tiene como objetivo evitar que se creen proyectos duplicados con el mismo nombre y el mismo ID de usuario.

También haremos uso de un **pre-hook** a la hora de borrar un proyecto, ya que un proyecto podrá tener fichas de datos asociadas y por tanto, como pasaba con los usuarios, tenemos que evitar encontrarnos con fichas de datos sin proyectos vinculados, para ello debemos borrar todas las fichas a la vez que el proyecto. **Recalcar que en dicho pre-hook haremos uso de un middleware creado para la gestión de las imágenes de las fichas. Se comentará esta parte en la sección 5.4.4 Funcionalidad almacenamiento de imágenes**.

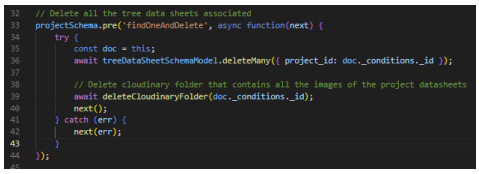


Figura 5.10 – *Función pre-hook utilizado al borrar un proyecto*

5.4.2.1. Rutas definidas en proyectos

A partir del modelo de datos de un proyecto, creamos las rutas o endpoints que usaremos en el frontend para poder realizar operaciones CRUD en la aplicación. Estos endpoints los encontramos en el archivo **projects** del directorio **routes** y son los siguientes:

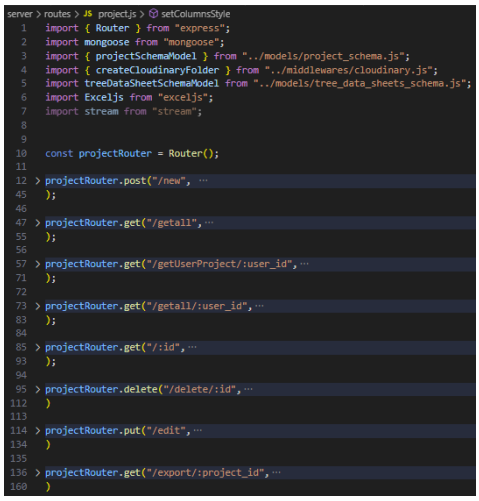
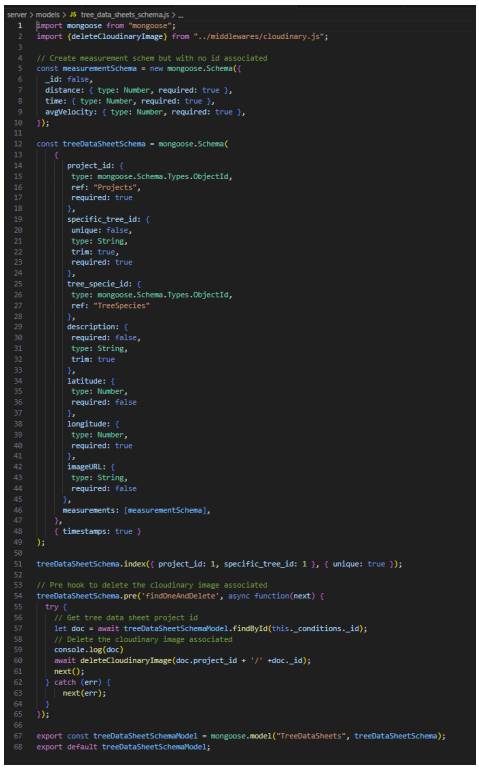


Figura 5.11 – *Rutas para la gestión de proyectos*

- Endpoint para la creación de un nuevo proyecto.
- Endpoint para poder obtener todos los proyectos de la colección de documentos.
- Endpoint para obtener un proyecto concreto de un usuario concreto.
- Endpoint para obtener todos los proyectos que pertenecen a un usuario. **Esto se utilizará para poder obtener el listado de proyectos de un usuario en el frontend**.
- Endpoint para obtener un proyecto a través de su ID de MongoDB.
- Endpoint para borrar un proyecto dado su ID.
- Endpoint para editar un proyecto con los datos que se han recibido desde la aplicación.
- Endpoint para exportar los datos de un proyecto concreto. **Recalcar que dicho endpoint se comentará en la sección 5.4.3 (Funcionalidad exportación de proyectos)**.

- **Fichas de datos**



```
server > models > JS tree_data_sheets_schema.js > ...
import mongoose from "mongoose";
import {deleteCloudinaryImage} from "../middlewares/cloudinary.js";

// Create measurement schem but with no id associated
const measurementSchema = new mongoose.Schema({
  _id: false,
  distance: { type: Number, required: true },
  time: { type: Number, required: true },
  avgVelocity: { type: Number, required: true },
});

const treeDataSheetSchema = mongoose.Schema(
    {
      project_id: {
       type: mongoose.Schema.Types.ObjectId,
       ref: "Projects",
       required: true
      },
      specific_tree_id: {
       unique: false,
       type: String,
       trim: true,
       required: true
      },
      tree_specie_id: {
       type: mongoose.Schema.Types.ObjectId,
       ref: "TreeSpecies"
      },
      description: {
       required: false,
       type: String,
       trim: true
      },
      latitude: {
       type: Number,
       required: false
      },
      longitude: {
       type: Number,
       required: true
      },
      imageURL: {
       type: String,
       required: false
      },
      measurements: [measurementSchema],
    },
    { timestamps: true }
);

treeDataSheetSchema.index({ project_id: 1, specific_tree_id: 1 }, { unique: true });

// Pre hook to delete the cloudinary image associated
treeDataSheetSchema.pre('findOneAndDelete', async function(next) {
  try {
    // Get tree data sheet project id
    let doc = await treeDataSheetSchemaModel.findById(this._conditions._id);
    // Delete the cloudinary image associated
    console.log(doc)
    await deleteCloudinaryImage(doc.project_id + '/' +doc._id);
    next();
  } catch (err) {
    next(err);
  }
});

export const treeDataSheetSchemaModel = mongoose.model("TreeDataSheets", treeDataSheetSchema);
export default treeDataSheetSchemaModel;
```

Figura 5.12 – *Modelo de fichas de datos para backend*

El modelo de fichas de datos contiene los campos: ID del proyecto asociado **project_id** que será una referencia al modelo de proyectos, el ID del árbol **(specific_tree_id)**, el ID de la especie de árbol **(tree_specie_id)** que será una referencia al modelo de especies de árboles, la descripción que pueda tener dicho árbol **(description)**, el valor de latitud de las coordenadas **(latitude)**, el valor de longitud de las coordenadas **(longitude)**, la URL donde se encontrará almacenada la imagen tomada del árbol si es el caso **(imageURL)** y por último, podrá contener un vector de medidas **(para estas medidas se le define su propio esquema en el mismo fichero, dicho modelo no tendrá un id al no ser necesario)**.


```
// Create measurement schem but with no id associated
const measurementSchema = new mongoose.Schema({
  _id: false,
  distance: { type: Number, required: true },
  time: { type: Number, required: true },
  avgVelocity: { type: Number, required: true },
});
```

Figura 5.13 – *Esquema para definir las medidas de una ficha de datos*

También necesitaremos definir un índice que tiene como objetivo evitar que se creen dichas de datos duplicadas con el mismo nombre y el mismo ID de proyecto.

Por último, implementamos una función **pre-hook** a la hora de borrar una ficha de datos, para poder eliminar la imagen que tiene asociada y que no se queden datos incongruentes en la plataforma donde se gestiona esta funcionalidad. **Se comentará en la sección 5.4.4 (Funcionalidad almacenamiento de imágenes) el manejo de imágenes**.

```
// Pre hook to delete the cloudinary image associated
treeDataSheetSchema.pre('findOneAndDelete', async function(next) {
  try {
    // Get tree data sheet project id
    let doc = await treeDataSheetSchemaModel.findById(this._conditions._id);
    // Delete the cloudinary image associated
    console.log(doc)
    await deleteCloudinaryImage(doc.project_id + '/' +doc._id);
    next();
  } catch (err) {
    next(err);
  }
});
```

Figura 5.14 – *Función pre-hook utilizada al borra una ficha de datos*

5.4.2.2. Rutas definidas en fichas de datos

A partir del modelo de datos de las fichas de datos, implantamos los endpoints que usaremos en el frontend para poder realizar operaciones CRUD en la aplicación. Estos endpoints los encontramos en el archivo **tree_data_sheets** del directorio **routes** y son los siguientes:

```
import { Router } from "express";
import mongoose from "mongoose";
import { treeDataSheetSchemaModel } from "../models/tree_data_sheets_schema.js";
import {cloudinaryMiddleware} from "../middlewares/cloudinary.js";
const treeDataSheetRouter = Router();

treeDataSheetRouter.post("/new",...
);

/**...
treeDataSheetRouter.put("/update/:id",...
)

/**...
treeDataSheetRouter.delete("/delete/:id",...
)

/**...
treeDataSheetRouter.get("/:id",...
);

/**...
treeDataSheetRouter.get("/project/:project_id",...
);


export default treeDataSheetRouter;
```

Figura 5.15 – *Rutas para la gestión de fichas de datos*

- Endpoint para la creación de una nueva ficha.
- Endpoint para poder actualizar una ficha de datos a través de su ID con los nuevos valores que se han recibido desde el frontend.
- Endpoint para borrar una ficha de datos dado su ID.
- Endpoint obtener una ficha de datos dado su ID.
- Endpoint para exportar los datos de un proyecto concreto. **Dicho endpoint se comentará en la sección 5.4.3 (Funcionalidad exportación de proyectos)** .

5.4.3. Funcionalidad exportación de proyectos

Como bien se comentó en la sección 4.5.2.3 (Exportar proyecto), cuando un usuario quiere exportar un proyecto y compartirlo a través de otras plataformas, dicho proyecto debe convertirse en un archivo de **Excel**. Esta parte está gestionada en el backend y a continuación explicaremos como se ha conseguido.

Cuando el usuario hace click en el botón de exportar se llama a un servicio para enviar la petición al servidor, este al recibir la petición ejecuta la lógica del endpoint de exportación.

Figura 5.16 – *Endpoint exportación de proyectos*

Figura 5.17 – *Función principal creación Excel*

Dentro de la lógica de este endpoint, llamamos a varios métodos que serán los encargados de construir el Excel gracias al uso de la librería **Exceljs**. Debemos tener en cuenta que antes, se realiza una comprobación en el proyecto para consultar si tiene fichas de datos, si no las tienes se devuelve al frontend una respuesta 500.

Una vez se comprueba que el proyecto tiene fichas de datos, se llama a la función **createExcel**, la cual será la encargada de construir la respuesta de vuelta con el fichero en binario y la cabeceras necesarias. Como por ejemplo el nombre del archivo que será: <**nombre_proyecto**>**.xlsx**. Dentro de esta función se llaman a otras dos de las cuales, la primera (**createNewProjectWorkSheet**), tiene como objetivo crear una página en el archivo destinada a colocar toda la información relativa al proyecto y al usuario asociado. La segunda (**createNewTreeDataSheetWorkSheet**), tendrá como objetivo construir una página en el archivo Excel por cada ficha de datos que contenga el proyecto.

Figura 5.18 – *Función para información de proyecto*

Figura 5.19 – *Función para información de ficha*

Dentro de cada una de estas dos funciones, también se definen los estilos, colores y márgenes que se le desea dar a las páginas. Finalmente un archivo Excel con datos exportados se ilustra con el siguiente diseño final:

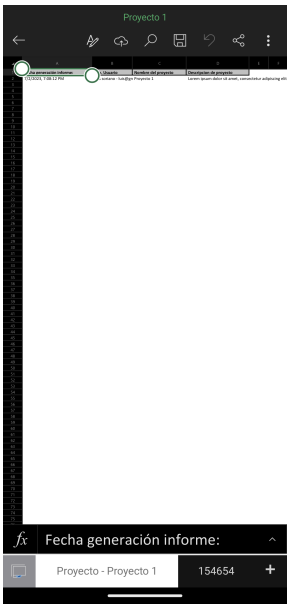


Figura 5.20 – *Diseño página proyecto en archivo*

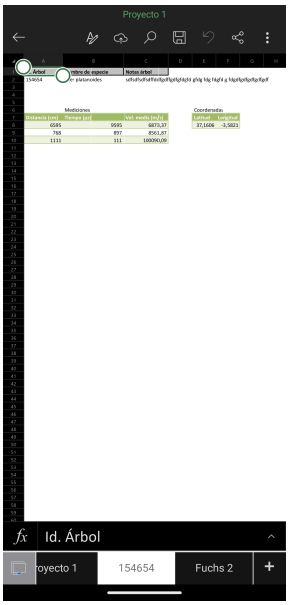


Figura 5.21 – *Diseño página ficha en archivo*

5.4.4. Funcionalidad almacenamiento de imágenes

Durante la sección 4.5.3 (Sistema de fichas de datos), se discutió la gestión de imágenes en el backend del sistema. En relación a la captura de imágenes en las fichas de árboles, se exploraron dos posibles soluciones, aunque como veremos, una de ellas no es viable.

- **Almacenamiento de imagen en MongoDB.** Esta alternativa plantea, que una vez se ha tomado la imagen desde el dispositivo móvil, se envie el binario de dicha imagen a través de un servicio en el frontend, para poder así almacenar en la colección de fichas de datos de MongoDB el binario.

  Para poder llevar a cabo esto, se necesita modificar el máximo de bytes que puede recibir el backend a través de una petición gracias a la librería **body-paser**, ya que por defecto, NodeJS lo establece a 1mb por petición. Este máximo lo establecemos en el archivo **index.js** 5.3 (Archivo index del backend).

  Sin embargo una vez se ha implementado de esta forma la gestión de imágenes, nos damos cuenta que el rendimiento de la base de datos empeora sustancialmente, haciendo que las consultas sobre las colecciones de fichas de datos aumenten hasta los 5-10 segundos. Podemos encontrar información acerca de este problema en la documentación oficial de MongoDB, donde no se debe exceder el límite recomendado por documento añadido[2].

  Por lo tanto, esta alternativa queda descartada por el mal rendimiento que nos aporta en nuestra aplicación.

- **Almacenamiento en plataforma externa.** Esta alternativa ha sido la elegida a implementar finalmente dentro de nuestro backend para poder tratar las imágenes. La plataforma externa que se ha decido utilizar es llamada **Cloudinary**, la cual nos permite a través de su librería para NodeJS controlar las operaciones que necesitamos realizar para almacenar datos.

  Para poder hacer uso de Cloudinary, necesitamos primeramente darnos de alta con una cuenta y crear una API Key para poder implementarla en nuestro backend. La configuración se realiza en

el archivo **index.js** 5.3 (Archivo index del backend). Debemos destacar que en dicha plataforma podremos organizar las imágenes dentro de directorios, lo cual nos conviene de esta forma, para agrupar por proyectos. Es decir, por cada proyecto creado se necesita crear un directorio en la plataforma, para saber donde se debe almacenar una imagen de una ficha de datos concreta.

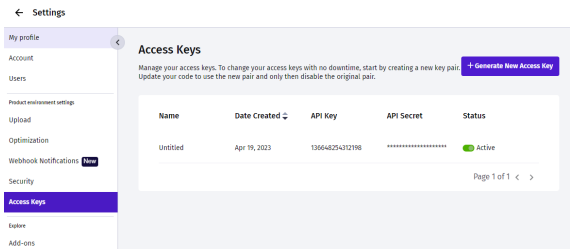


Figura 5.22 – *Cloudinary - Gestión de API key*

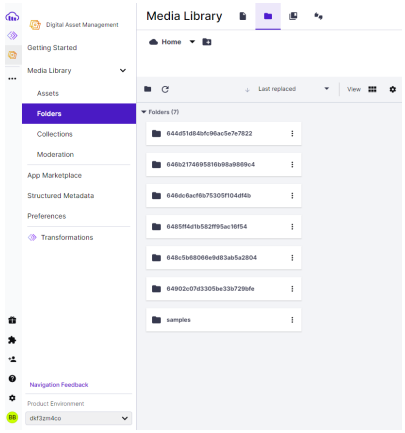
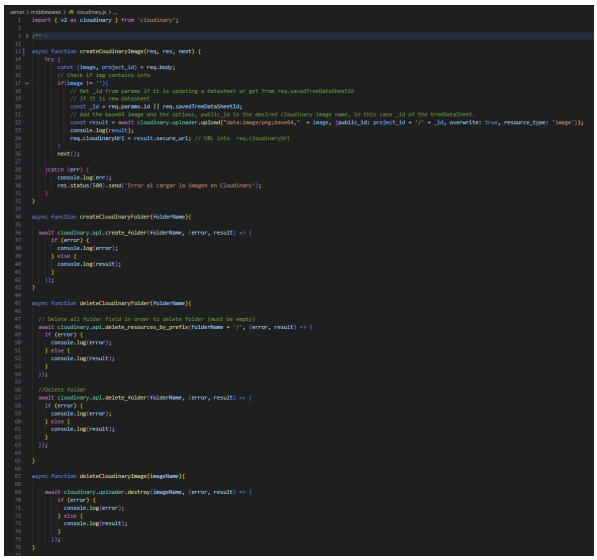


Figura 5.23 – *Cloudinary - Pantalla principal de archivos*

A la hora de realizar operaciones con la librería de Cloudinary, se crea un middleware para poder llevar a cabo todas las operaciones necesarias. Este middleware lo encontraremos en el archivo **cloudinary.js** del directorio **middlwares**.



```
import { v2 as cloudinary } from 'cloudinary';

async function createCloudinaryImage(req, res, next) {
    try {
        const {image, project_id} = req.body;
        // Check if img contains info
        if(image != ''){
            // Get _id from params if it is updating a datasheet or get from req.savedTreeDataSheetId
            // if it is new datasheet
            const _id = req.params.id || req.savedTreeDataSheetId;
            // Add the base64 image and the options, public_id is the desired cloudinary image name, in this case _id of the treeDataSheet.
            const result = await cloudinary.uploader.upload("" + image, {public_id: project_id + '/' + _id, overwrite: true, resource_type: 'image'});
            console.log(result);
            req.cloudinaryUrl = result.secure_url; // URL into  req.cloudinaryUrl
        }
        next();

    }catch (err) {
        console.log(err);
        res.status(500).send('Error al cargar la imagen en Cloudinary');
    }
}

async function createCloudinaryFolder(folderName){

  await cloudinary.api.create_folder(folderName, (error, result) => {
      if (error) {
        console.log(error);
      } else {
        console.log(result);
      }
    });
}

async function deleteCloudinaryFolder(folderName){

  // Delete all folder field in order to delete folder (must be empty)
  await cloudinary.api.delete_resources_by_prefix(folderName + '/', (error, result) => {
    if (error) {
      console.log(error);
    } else {
      console.log(result);
    }
  });

  //Delete folder
  await cloudinary.api.delete_folder(folderName, (error, result) => {
    if (error) {
      console.log(error);
    } else {
      console.log(result);
    }
  });
}

async function deleteCloudinaryImage(imageName){

    await cloudinary.uploader.destroy(imageName, (error, result) => {
        if (error) {
            console.log(error);
        } else {
            console.log(result);
        }
    });
}
```

Figura 5.24 – *Middleware para controlar operaciones en Cloudinary*

Se explicarán las funciones implementadas en el middleware por orden de uso:

- **Crear directorio.** Esta función llamada **createCloudinaryFolder** dentro del código, se le tiene que pasar por parámetro el nombre de la carpeta que deseamos crear, para luego,

hacer uso de la función **create_folder** que nos brinda la librería. El nombre que a nosotros nos interesa pasarle es el ID del proyecto de MongoDB, ya que este middleware es llamado una vez se ha creado el registro de dicho proyecto en la base de datos. Este uso del middleware se puede consultar en el endpoint de creación de proyecto 5.11 (Rutas para la gestión de proyectos).

- **Subir imagen a directorio.** Esta función, llamada dentro del código **createCloudinaryImage**, se le tiene que pasar por parámetro la petición que se manda desde el frontend para obtener el binario de la imagen a subir, y el nombre del proyecto para poder almacenar la imagen en el directorio correspondiente. Una vez tengamos estos datos, se hace uso de la función **upload** de la librería, he indicamos el nombre del archivo que será el ID de la ficha de datos de MongoDB y el directorio de destino. Una vez se ha subido la imagen, la función **upload** nos devuelve una respuesta de la operación y, si todo ha ido bien, nos devuelve la URL de almacenamiento de la imagen la cual la necesitamos para guardar en nuestro registro de ficha de datos en el campo **imageURL** definido en el modelo. Este uso del middleware se puede consultar en el endpoint de creación de ficha de datos y en el endpoint de actualización de ficha de datos 5.15 (Rutas para la gestión de fichas de datos).

- **Eliminar directorio.** Esta función llamada **deleteCloudinaryFolder** dentro del código, se le tiene que pasar por parámetro el nombre de la carpeta que deseamos borrar. Luego, haremos uso de la función **delete_resources_by_prefix** que nos brinda la librería para eliminar previamente cualquier fichero que se encuentre dentro del directorio. Una vez realizado esto, hacemos uso de la función **delete_folder** para borrar el directorio previamente vaciado. Este uso del middleware se puede consultar en el **pre-hook** 5.10 (Función pre-hook utilizado al borrar un proyecto) del modelo de datos de los proyectos a la hora de borrar uno de estos.

- **Eliminar imagen de directorio.** Esta función llamada **deleteCloudinaryImage** dentro del código, se le tiene que pasar por parámetro el nombre del directorio concatenado con el nombre de la imagen que deseamos borrar. Luego, haremos uso de la función **destroy** que nos ofrece la librería para llevar a cabo esta operación. Este uso del middleware se puede consultar en el **pre-hook** 5.14 (Función pre-hook utilizada al borra una ficha de datos) del modelo de las fichas de datos a la hora de borrar una de estas.

5.4.5. Seguridad en los datos

En el backend, a la hora de tratar los datos que se han recibido desde el frontend, debemos definir algunos mecanismos de seguridad para estos mismos.

En cada una de las rutas que se definen para los sistemas implementados, se definen una serie de comprobaciones para evitar que falten datos o que estos son incorrectos. Por ejemplo, a la hora de crear un proyecto es necesario recibir en la petición el ID del usuario que ha creado el proyecto y el nombre de dicho proyecto ya que son campos obligatorios. Si esto no se cumple, el servidor devolverá al frontend un error en la respuesta.

Capítulo 6

Despliegue de aplicación

Desde el inicio del desarrollo de la aplicación, se ha buscado una solución que permita el despliegue tanto de la base de datos como del backend en servidores en la nube, en lugar de depender de una máquina local. Esta decisión tiene como objetivo facilitar el trabajo, ya que la implementación en una máquina local limitaría la posibilidad de probar la aplicación en dispositivos móviles físicos.

Además, durante la fase intermedia del desarrollo de la aplicación, se inició el proceso de despliegue en dispositivos iOS y Windows. Esta medida se tomó con el propósito de crear una aplicación multiplataforma que brinde una experiencia de uso más accesible a los usuarios interesados en el software.

A continuación, se presentarán las herramientas que se utilizarán para llevar a cabo todo el despliegue que engloba el proyecto y se explicará todo el proceso seguido para lograrlo.

6.1. MongoDB Atlas

A la hora de implementar la base de datos en la nube, se ha optado por elegir el servicio que nos brinda **MongoDB Atlas**. Gracias a su licencia gratuita tenemos la posibilidad de tener nuestra base de datos en una plataforma en línea sin coste alguno.

Para ello, los pasos que seguimos son, crear una cuenta en su web oficial y una vez creada la cuenta debemos comenzar con la configuración del entorno que hosteará la base de datos.

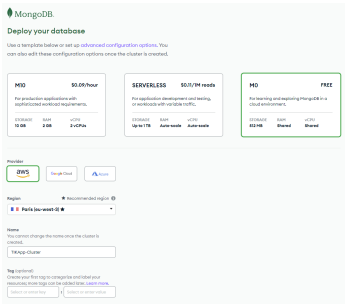


Figura 6.1 – *Creación de entorno en MongoDB Atlas*

Se elige el plan **M0**, que tendrá un coste de 0 €/mes con ciertas limitaciones como pueden ser, **512MB de almacenamiento**, **RAM compartida y uso de CPU compartido**.

Una vez realizado este paso, tardará unos minutos en estar listo el entorno para poder empezar a crear colecciones. Sin embargo, necesitaremos generar la URL de acceso que utilizará la librería **Mongoose** para poder realizar la conexión a la base de datos y tener acceso desde nuestro backend. Esto lo haremos desde el apartado de conexión una vez dentro del entorno y nos mostrará la URL a utilizar, en nuestro caso es la siguiente:

```
mongodb+srv://<username>:<password>@tikapp.rs7pd7q.mongodb.net/?retryWrite
s=true&w=majority
```

En la cual en el **username** y **password** indicados debemos establecer un usuario creado con acceso a la base de datos con los permisos que deseemos. A esta URL se le creará una variable en el fichero **.env** del backend para poder ser utilizada.

6.2. Render

Actualmente existen múltiples posibilidades para realizar un despliegue de backend para el entorno de NodeJS[1]. Para este proyecto se ha elegido **Render**, que ya ha sido mencionado en la sección 3.4.3 (Software).

Para comenzar, necesitamos crear una cuenta para configurar el entorno y una vez dentro, crearemos un nuevo servicio web. Debemos tener en cuenta que necesitamos tener el backend en un sistema de control de versiones en la nube, ya sea utilizando **GitHub** o **GitLab**. Esto es, ya que cualquier cambio que se realice en el backend y se suba dicho cambio al control de versiones, Render automáticamente lo detecta para poder actualizar el entorno en la nube con la nueva implementación y así disfrutar de los cambios en casi tiempo real.

Después de seleccionar el repositorio de donde accederá a los archivos, necesitamos indicar que configuración queremos 6.2 (Configuración de entorno en Render), esto es, el nombre del entorno, región donde estará ubicado el servidor, rama del repositorio donde queremos controlar los cambios, el entorno de ejecución del servicio web y el comando para compilar. También necesitamos elegir que tipo de instancia vamos a crear, que en nuestro caso será aquella que nos ofrece **512MB de RAM y 0.1 de CPU sin costo** alguno.

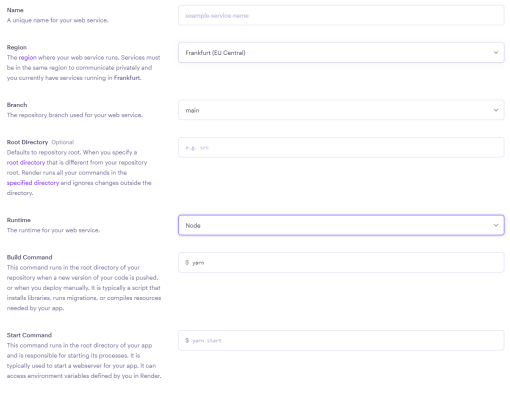


Figura 6.2 – *Configuración de entorno en Render*

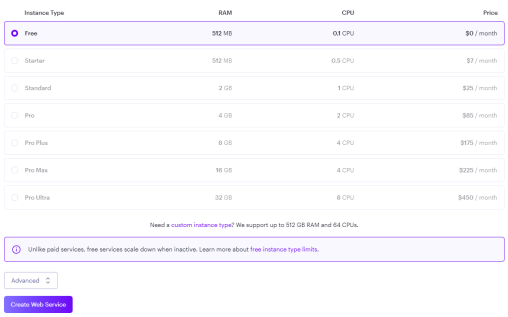


Figura 6.3 – *Selección de entorno en Render*

Una vez tenemos el entorno listo, nos queda cargar las variables de entorno que se definen en el archivo **.env** y utilizar, en las rutas de nuestro frontend, la URL que Render nos ha brindado para nuestro entorno en la nube.

6.3. iOS

Para poder cumplir el objetivo de implementar una aplicación multiplataforma, debemos ser capaces de compilar el software para su uso en dispositivos iOS. Durante el desarrollo, se descubre que para poder compilar la aplicación en dicho sistema operativo es necesario utilizar el programa **XCode**, el cual es exclusivo para equipos MacOS. Debido a la condición del autor del proyecto de no disponer de dicho equipo con MacOS, se busca alguna alternativa para poder llevar a cabo el objetivo.

Esta alternativa las encontramos en la plataforma online **Codemagic.io**, la cual nos ofrece en máquinas virtuales poder compilar código de varios tipos de aplicaciones diferentes, entre ellas flutter. A esta máquinas virtuales nos podremos conectar mediante **VNC**, el cual es un protocolo de escritorio remoto que permite a usuarios controlar y acceder a un equipo de forma remota a través de una red.

Para poder hacer uso de esta plataforma debemos crear una cuenta y conectar el repositorio en el cual mantenemos el código de la aplicación que necesitemos compilar. A la hora de configurar la máquina virtual, debemos indicar de que tipo de aplicación se trata y si es una creada a partir de flutter, también debemos indicar en que diferentes dispositivos queremos compilar, en nuestro caso únicamente iOS.

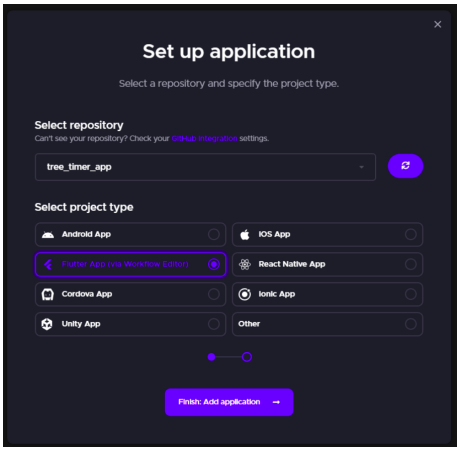


Figura 6.4 – *Configuración de máquina virtual*

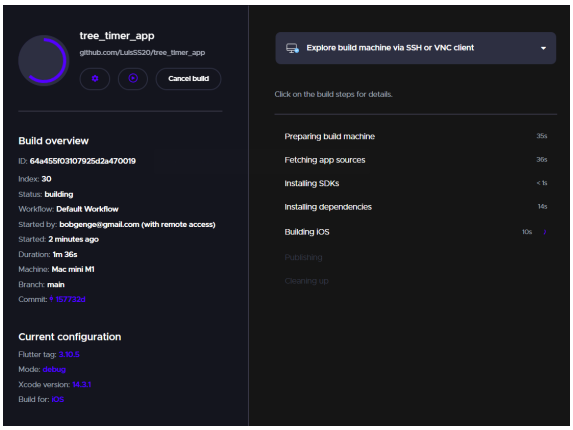


Figura 6.5 – *Compilación de aplicación en iOS*

Una vez finalizada la compilación, será necesario establecer una conexión con la máquina virtual utilizando el protocolo **VNC** para poder probar la aplicación en el sistema operativo iOS. Dentro de la máquina virtual, se iniciará el programa XCode, que Codemagic habrá instalado previamente durante el proceso de compilación. A continuación, ejecutaremos la aplicación dentro del simulador proporcionado por XCode para poner a prueba todas las funcionalidades implementadas en la aplicación. Esta etapa nos permitirá asegurar el correcto funcionamiento y la compatibilidad de la aplicación en el entorno de iOS.

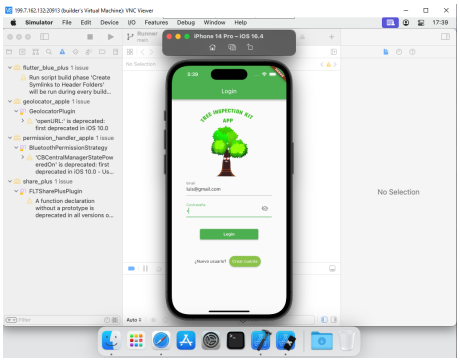


Figura 6.6 – *Pantalla de login en iOS*

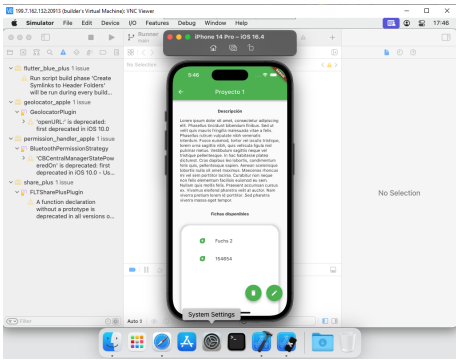


Figura 6.7 – *Pantalla de proyecto en iOS*

6.4. Windows

Un objetivo a cumplir que se ha definido como secundario, es poder compilar la aplicación en Windows de manera que esta pueda ser ejecutada en equipos con dicho sistema y permitir al usuario realizar consultas acerca de los proyectos que tienen creados.

Para lograr esto, es necesario configurar el proyecto de Flutter para la plataforma de Windows[5] mediante el comando **flutter config –enable-windows-desktop**. Una vez completada esta configuración, se ejecuta el comando **flutter create –platform=windows .**, el cual generará los archivos necesarios para la compilación en Windows. El siguiente paso consiste en realizar la compilación utilizando el comando **flutter build windows**, lo cual generará el ejecutable de la aplicación junto con los archivos necesarios en el directorio **buildwindowsrunnerRelease**.

Con estos pasos, la aplicación estará lista para funcionar en el sistema operativo Windows. Sin embargo se quiso dar un paso adicional para crear un instalador de la aplicación en Windows y automatizar su creación con cada cambio detectado en la plataforma de control de versiones, GitHub, haciendo uso de las conocidas como **GitHub Actions**. A continuación se procede a explicar todos los pasos que se han seguido:

6

- **Uso de InnoSetup.** Al surgir la necesidad de crear un instalador con los archivos que se generan a la hora de compilar la aplicación para windows, nuestro objetivo ahora, es encontrar una herramienta que nos permita crear dicho instalador y, lo más importante, que nos permita utilizarla mediante comandos de terminal y sea válida para usar en el apartado **GitHub Actions**.

  Nuestra solución viene dada por **InnoSetup** el cual nos ofrece todo lo que buscamos. Para ello nos descargamos el software desde su página oficial y probamos a crear un instalador manualmente. Los pasos a seguir son realmente sencillos, debido a la interfaz del software en todo momento nos guía. Durante los pasos debemos configurar **nombre de la aplicación**, **versión** de la misma, **destino para la instalación** de la aplicación, **permitir al usuario cambiar el destino**, **lenguajes disponibles** para el instalador y **archivos que necesita el instalador** para llevar a cabo la operación de instalación. Una vez completada la instalación, nos muestra por pantalla un script. Este script lo guardamos en un directorio llamado **installers_scripts**, el cual se comentó en la sección 4.4 (Estructura), ya que será necesario para la automatización dentro de GitHub Actions.

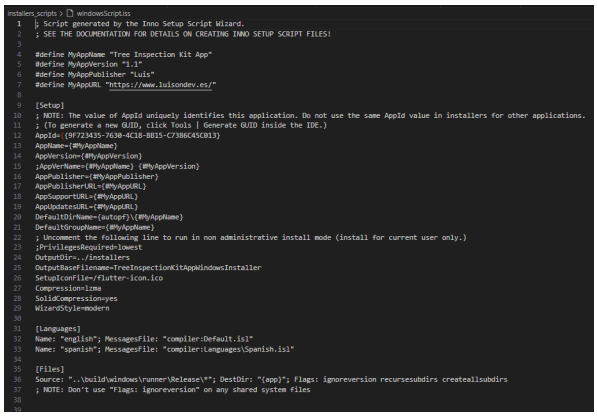


Figura 6.8 – *Script de creación de instalador windows*

- **GitHub Actions.[12]** Para terminar de automatizar la creación de un instalador con cada actualización que ocurra en el repositorio, necesitaremos hacer uso de **GitHub Actions**. Es una plataforma de automatización de tareas integrada en GitHub que nos permite automatizar diferentes flujos de trabajo en el desarrollo de software.

  La forma de trabajar con la herramienta es bastante sencilla, necesitamos crear un archivo **.yml** en el directorio **.github_Workflows** de nuestro repositorio y definir dentro todas las tareas que queremos que se lleven a cabo. Con la siguiente figura se explicaran las tareas:

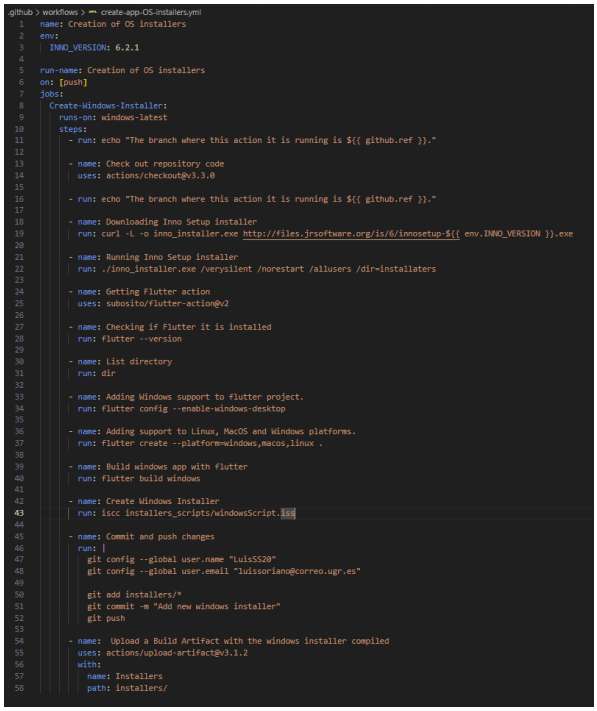


Figura 6.9 – *Archivo de tareas para instalador Windows*

1. Primeramente debemos definir el nombre del **workflow** y una variable para indicar que versión de **InnoInstaller** queremos instalar en el contenedor. **Líneas 1-3**

2. Seguido indicamos cuando queremos que se ejecute el workflow, en nuestro caso con cada push en todas las ramas del repositorio. **Línea 6**
3. Definimos en que contenedor queremos que se ejecuten las tareas, en nuestro caso Windows con la última versión. Esto es muy importante ya que no nos valdría un contenedor con otro S.O. **Línea 9**
4. Realizamos un checkout del repositorio en el contenedor utilizando un action de la comunidad. **Líneas 13-14**
5. Una vez tengamos el repositorio, el siguiente paso será descargar e instalar el software de **InnoSetup** para tener disponible su uso mediante terminal. **Líneas 18-22**
6. Ahora necesitaremos instalar flutter en el contenedor para posteriormente realizar una compilación del proyecto. **Líneas 24-25**
7. Procedemos a configurar el proyecto de flutter para windows y realizar una compilación utilizando los mismos comandos que vimos en la sección 6.4 (Windows). **Líneas 33-40**
8. Ya está todo listo para generar el instalador por lo que debemos hacer uso mediante la terminal del software instalado InnoSetup[18] junto con el script que generamos 6.8 (Script de creación de instalador windows) y guardamos en el directorio **installers_scripts**. **Líneas 42-43**
9. Por último debemos subir el instalador creado al repositorio para poder tener acceso a él y que pueda ser instalado en cualquier equipo con el sistema operativo Windows instalado. El instalador **.exe** se encontrará en el directorio **installers** del repositorio. **Líneas 45-58**

Una vez seguido todos los pasos anteriores, como se ha comentado, cualquier cambio en el repositorio hará que se genere un nuevo instalador de Windows para así tener la aplicación actualizada al día en dicha plataforma.

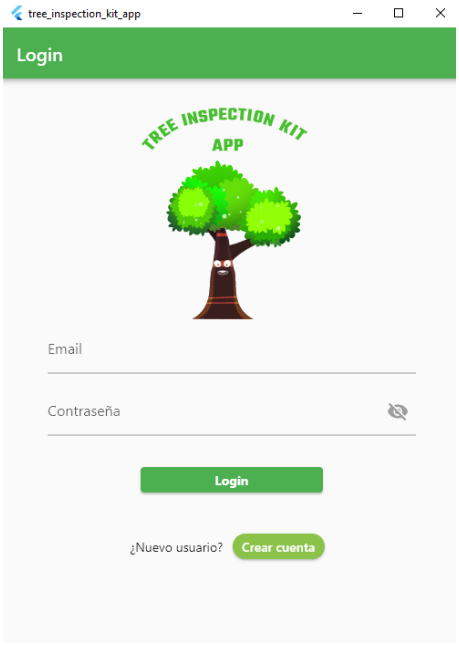


Figura 6.10 – *Pantalla de login en Windows*

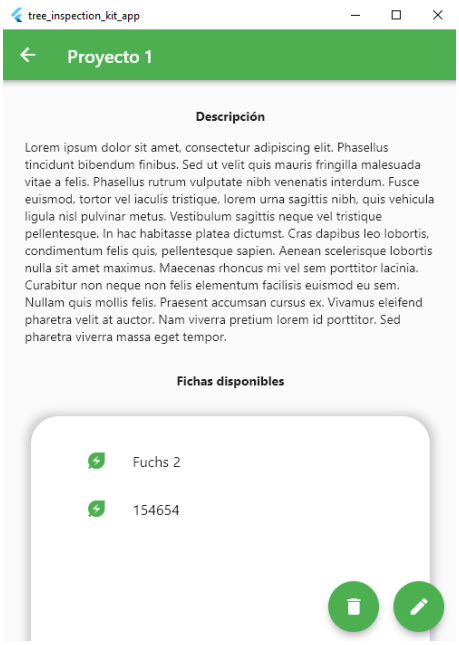


Figura 6.11 – *Pantalla de proyecto en Windows*

Capítulo 7

Conclusiones y trabajos futuros

Llegados a este último capítulo, abordaremos las conclusiones que se han podido sacar tras el desarrollo del proyecto de la aplicación móvil, así como posibles trabajos futuros que se podrían implementar en el software.

7.1. Conclusiones

El propósito principal del desarrollo del proyecto ha sido la creación de una aplicación móvil para complementar al dispositivo TIK, evitando así utilizar aplicaciones software de terceros como la ya vista en la sección 2.1 (Caso de estudio: Microsecond Timer App). El desarrollo de la misma se desglosó en varios objetivos, ya vistos en la sección 1.2 (Objetivos y metas del proyecto).

Como objetivos primarios tenemos los siguientes:

- Crear aplicación cliente con el rol de frontend.
- Desplegar servidor en la nube con el rol de backend.
- Aplicación funcional tanto en Android como en iOS.

Conforme al primer objetivo, se ha logrado satisfacer el mismo gracias al uso del framework Flutter, permitiéndonos crear todas las pantallas que se han visto necesarias y así poder entregar a los usuarios una aplicación con varios sistemas funcionales como los que se han podido ver anteriormente: sistema de usuarios, sistema de proyectos, sistema de fichas, exportación de proyectos, cambio de idiomas... Gracias al uso de Flutter nos podemos dar cuenta que es un framework muy potente y muy resolutivo a la hora de creación de aplicaciones tanto móviles como web.

En relación al segundo objetivo, se ha implementado un backend gestionado en la nube permitiendo así a los usuarios poder hacer un uso completo de toda la funcionalidad que podemos encontrar dentro de la aplicación. Este servidor se encarga de gestionar todas las operaciones que el usuario realiza con los distintos sistemas implementados dentro del software. Destacamos la importancia de diferentes plataformas externas que permiten a todo tipo de desarrolladores, implementar en la nube servidores a un bajo coste o incluso con un gasto cero asociado.

El tercer objetivo se ha cumplido permitiendo que tanto usuarios de dispositivos Android como de dispositivos con iOS puedan hacer uso de la aplicación. Debido a las limitaciones de recursos para poder probar y compilar la aplicación en un dispositivo con iOS, se necesitó hacer uso de plataformas

externas y máquinas virtuales para lograrlo. Esto nos permite darnos cuenta que gracias a abrir nuestra aplicación a diferentes plataformas y sistemas operativos se tiene un mayor alcance en el mercado.

También fueron establecidos unos objetivos secundarios:

- Ofrecer una interfaz intuitiva al usuario.
- Incorporar elementos animados a la aplicación para dar dinamismo a esta.
- Aplicación funcional a modo de consulta en el sistema operativo Windows.

Conforme al primer objetivo secundario, se ha conseguido establecer una interfaz bastante sencilla a la vez que intuitiva para el usuario. Esto hace que no se sienta abrumado por la cantidad de elementos que aparecen por pantalla, ya que se ha conseguido que esta sea lo más limpia posible para ofrecer la mejor experiencia.

En relación al segundo objetivo secundario, se muestran varias animaciones a la hora de realizar ciertas acciones por parte del usuario dentro de la aplicación, dando así la impresión de una aplicación cuidada y que esta dotada de un mayor dinamismo para no parecer aburrida. Podemos extraer de aquí, la importancia de cuidar ciertos detalles internos de la aplicación para así poder captar la atención de los usuarios.

Respecto al tercer objetivo secundario, se ha dado la posibilidad a los usuarios de poder instalar la aplicación en sus equipos con el sistema operativo Windows para así poder consultar toda la información almacenada de sus proyectos hasta el momento. Con esto brindamos al usuario un abanico mayor de uso de la aplicación y ampliamos soporte en diferentes plataformas.

7.2. Trabajos futuros

La aplicación que ha sido creada para gestionar el dispositivo TIK se ha presentado como una aplicación final que con tiempos, recursos y conocimientos limitados por parte del estudiante, autor del proyecto.

El desarrollo de la aplicación para gestionar el dispositivo TIK ha sido un reto debido a los tiempos ajustados, los recursos limitados y los conocimientos por parte del estudiante, autor del proyecto. A pesar de estos factores, se ha logrado crear un prototipo en fase final que se encuentra completamente funcional. Sin embargo, es importante tener en cuenta que se requiere un mayor testeo y la participación de múltiples usuarios para obtener una retroalimentación sólida y mejorar aún más la aplicación. Algunos posibles trabajos futuros que se tendrían por delante serían:

- **Funcionalidad de conexión bluetooth a dispositivo TIK completa**. Una vez se haya desarrollado el módulo de bluetooth para el dispositivo TIK se tendría que completar la funcionalidad bluetooth en la aplicación, para así poder conectar ambos dispositivos y permitir la captura de datos a través de este sistema sin necesidad de que el usuario lo haga manualmente.

- **Modo sin conexión**. Debido a la utilización que el usuario haría con la aplicación, podemos entender que el uso de esta sería en un entorno de campo o rural. En dichos lugares podríamos tener problemas de conexión a internet, por lo tanto la aplicación no podría conectarse con los servidores. Una forma de solucionar dichos problemas, sería crear un apartado offline donde el usuario pudiera hacer uso de la aplicación con total normalidad y cuando se conectara a una red, subir todos los cambios no guardados en los servidores.

- **Mejoras en los despliegues**. Dado que el despliegue que se ha realizado ha sido en entornos en la nube con planes de suscripción gratuitos, las capacidades de estos entornos están limitados. Sería un aspecto a tener en cuenta escalar dichos entornos para ofrecer mayores prestaciones a los usuarios.
- **Función recuperación de contraseña**. Dotar de un sistema de recuperación de contraseña para los usuarios previamente registrados en la aplicación.
- **Sistema de autenticación basado en tokens**. Dotar de un sistema autenticación basado en tokens para mejorar y agregar un plus en la seguridad de la aplicación.

7.3. Valoración personal

El trabajo de este proyecto me ha permitido aprender de una gran cantidad de tecnologías de las cuales puedo sacar provecho y partido en diversos desarrollos personales.

El haber trabajado en el proyecto de esta aplicación móvil, ha hecho despertar en el desarrollador que hay en mí el interés por esta rama. Juntando el desarrollo de backend, he podido experimentar un circuito cerrado de lo que sería una implementación completa. Así mismo, el despliegue de todo el conjunto ha hecho darme cuenta de la facilidad que tenemos los desarrolladores para poder en práctica nuestro conocimiento y lanzarlo a la red.

Apéndice A

Instalación de Flutter sobre Windows

Si el lector desea instalar en su equipo Windows las herramientas necesarias para la compilación y ejecución de la aplicación, se desarrolla esta pequeña guía para ayudar con el proceso. Si el lector necesita una ayuda extra, puede consultar la documentación oficial para seguir el proceso completo[10].

- El primer paso a realizar será descargar una versión del SDK de Flutter[9]. El proyecto ha sido desarrollado con la versión **3.10.5**, por lo que se recomienda descargar dicha versión

- Se debe extraer el contenido del archivo comprimido del SDK en cualquier directorio deseado del sistema. Por ejemplo en **C:\src\flutter**.

- El siguiente paso y que es muy importante será actualizar las variables de entorno del sistema para incluir en la variable **PATH** el directorio **bin** que se encuentra en el directorio donde hemos extraido el contenido del SDK.

  Por lo tanto, se debe abrir el apartado **Editar variables de entorno del sistema** en Windows y agregar dicha ruta a la variable **PATH**

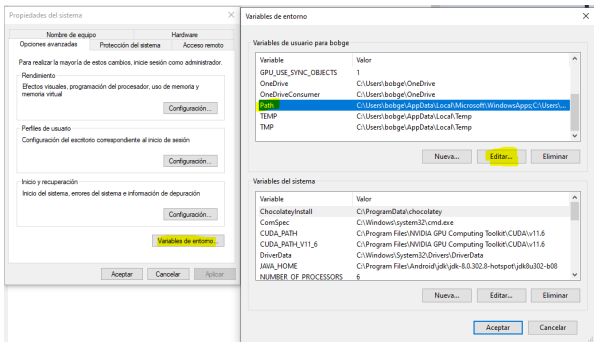


Figura A.1 – *Editar variables de entorno*

- Se necesitará comprobar que se tienen instaladas todas las dependencias que son necesarias a la hora de ejecutar Flutter. Para ello ejecutaremos en una terminal el comando **flutter doctor** y se observa si falta alguna dependencia.

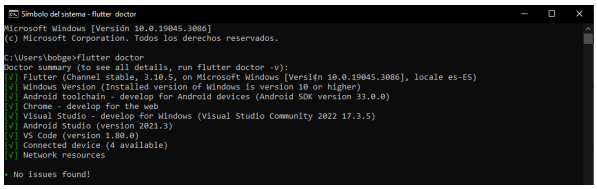


Figura A.2 – *Comprobando instalación de dependencias de flutter*

Si se desea probar la aplicación en el mismo sistema Windows con un emulador Android, se debe cumplir con las depencias de **Android tootlchain (Android SDK)** y **Android Studio**

- Se necesitará instalar Android Studio. Una vez completada la instalación, se debe iniciar y seguir el **Asistente de configuración de Android Studio**. Esto instalará la última versión del SDK de Android, sus herramientas de línea de comandos y las herramientas de compilación, que son necesarias para Flutter al desarrollar en Android.

  Para poder configurar un dispositivo virtual en Android Studio y probar a ejecutar el proyecto, se deben realizar los siguientes pasos:

  - Habilita la aceleración de la máquina virtual en tu equipo.
  - Inicia Android Studio, haz clic en el icono de **Administrador de Dispositivos** y selecciona **Crear dispositivo en la pestaña Virtual**...

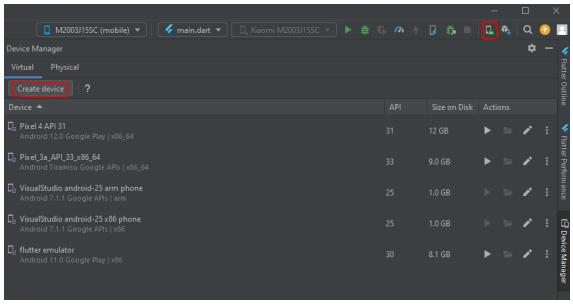


Figura A.3 – *Administrador de dispositivos*

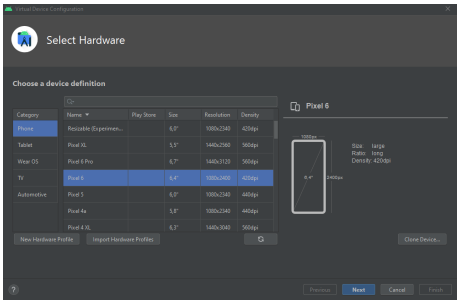


Figura A.4 – *Crear dispositivo virtual*

También se puede acceder desde el **Menú de los Tres Puntos / Más Acciones >Administrador de Dispositivos Virtuales** y **seleccionar Crear dispositivo...**

- Elige una versión de dispositivo y selecciona Siguiente.
- Selecciona una o más imágenes del sistema para las versiones de Android que deseas emular y selecciona Siguiente.
- Verifica que la configuración del AVD sea correcta y selecciona Finalizar.

Bibliografía

[1] Yogesh Chavan. *How to Deploy Your Node.js Application for Free with Render*. 2022. URL: https://www.freecodecamp.org/news/how-to-deploy-nodejs-application-with-render/.

[2] Mongo DB. *Reducir tamaño de documentos*. 2023. URL: https://www.mongodb.com/docs/atlas/schema-suggestions/reduce-document-size/.

[3] Abhishek Doshi. *Flutter en profundidad - Árboles*. 2021. URL: https://abhishekdoshi26.medium.com/deep-dive-into-flutter-trees-542f7395df5c.

[4] Claire Drumond. *¿Qué es scrum?* 2023. URL: https://www.atlassian.com/es/agile/scrum.

[5] Flutter. *Add desktop support to an existing Flutter app*. 2023. URL: https://docs.flutter.dev/platform-integration/desktop#add-desktop-support-to-an-existing-flutter-app.

[6] Flutter. *Características de Flutter*. 2023. URL: https://docs.flutter.dev/resources/faq#capabilities.

[7] Flutter. *Documentación google maps en flutter*. 2023. URL: https://pub.dev/packages/google_maps_flutter.

[8] Flutter. *Documentación para la creación de widgets*. 2023. URL: https://docs.flutter.dev.

[9] Flutter. *Flutter SDK archive*. 2023. URL: https://docs.flutter.dev/release/archive?tab=windows.

[10] Flutter. *Instalación Flutter en Windows*. 2023. URL: https://docs.flutter.dev/get-started/install/windows.

[11] Flutter. *Internacionalizando aplicaciones Flutter*. 2023. URL: https://esflutter.dev/docs/development/accessibility-and-localization/internationalization.

[12] GitHub. *Add desktop support to an existing Flutter app*. 2023. URL: https://docs.github.com/en/actions/using-workflows/workflow-syntax-for-github-actions.

[13] Philips Hue. *Documentación acerca de librería bluetooth Flutter Reactive BLE*. 2020. URL: https://pub.dev/packages/flutter_reactive_ble.

[14] Immune Technology Institute. *Las razones de la demanda de ingenieros informáticos: Su sueldo medio*. 2021. URL: https://immune.institute/blog/razones-demanda-ingenieros-informaticos-sueldo/.

[15] Anastasia Kushnir. *What Is the Best Database for Node.js?* 2023. URL: https://bambooagile.eu/insights/the-best-database-for-node-js/.

[16] Universidad Alfonso X el Sabio. *¿Cuánto gana un ingeniero informático?* 2021. URL: https://www.uax.com/blog/ingenieria/cuanto-cobra-un-ingeniero-informatico.

[17] Robert Sheldon. *What Is the Best Database for Node.js?* 2021. URL: https://www.red-gate.com/simple-talk/databases/nosql/how-to-choose-between-sql-and-nosql-databases/.

[18] JR Software. *Inno Setup Command Line Compiler Execution.* 2023. URL: https://jrsoftware.org/ishelp/index.php?topic=compilercmdline.

[19] Stackoverflow. *Validación email con aceptación de caracteres latinos.* 2015. URL: https://es.stackoverflow.com/questions/142/validar-un-email-en-javascript-que-acepte-todos-los-caracteres-latinos.