UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

TFG del Grado en Ingeniería Informática

título del TFG

Presentado por Nombre del alumno
en Universidad de Burgos — 22 de mayo de 2025
Tutor: nombre tutor

UNIVERSIDAD DE BURGOS
ESCUELA POLITÉCNICA SUPERIOR
Grado en Ingeniería Informática

D. nombre tutor, profesor del departamento de nombre departamento, área de nombre área.

Expone:

Que el alumno D. Nombre del alumno, con DNI dni, ha realizado el Trabajo final de Grado en Ingeniería Informática titulado título de TFG.

Y que dicho trabajo ha sido realizado por el alumno bajo la dirección del que suscribe, en virtud de lo cual se autoriza su presentación y defensa.

En Burgos, 22 de mayo de 2025

Vº. Bº. del Tutor:	Vº. Bº. del co-tutor:
D. nombre tutor	D. nombre co-tutor

Resumen

En este primer apartado se hace una **breve** presentación del tema que se aborda en el proyecto.

Descriptores

Palabras separadas por comas que identifiquen el contenido del proyecto Ej: servidor web, buscador de vuelos, android ...

Abstract

A **brief** presentation of the topic addressed in the project.

Keywords

keywords separated by commas.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

En la actualidad, las tecnologías junto con el Internet de las Cosas se encuentran en constante evolución. Estos continuos avances abren un gran abanico de posibilidades en la automatización de tareas de la vida cotidiana. El uso de sensores conectados a la red local ha sido el factor clave para poder llegar a monitorizar estos diferentes entornos, como pueden ser hogares inteligentes o entornos agrícolas.

Dentro de estos avances tecnológicos surge GreenInHouse2.0, una aplicación móvil orientada a facilitar el cuidado y seguimiento de las plantas en un entorno doméstico. Con ayuda de dispositivos como sensores de humedad, temperatura y luminosidad, la aplicación ayuda al usuario a tener controlados los diferentes parámetros que influyen en el estado de la planta. Todo esto se realiza a través de una interfaz de usuario muy intuitiva que consta de gráficos en los que se pueden ver cuándo los valores se alejan de las zonas óptimas para la planta. Además, también cuenta con un conjunto de hitos que el usuario deberá completar diariamente para que la planta tenga buen cuidado.

Este proyecto muestra que es posible usar una solución tecnológica automatizada para cosas de la vida cotidiana, como puede ser el cuidado de las plantas. Gracias a todas las opciones de personalización y visualización que ofrece la aplicación, los usuarios pueden conocer mejor los cuidados que deben seguir las plantas para que tengan buena salud. Todo ello está adaptado tanto para perfiles más principiantes en el mundo del cuidado de plantas como para gente más experta.

1.1. Estructura de la memoria

La memoria se estructura de la siguiente manera:

1. **Introducción:** Se incluye una breve descripción, se presentan los objetivos generales del proyecto y se explica la estructura tanto de la memoria como de los anexos.

2. **Objetivos del proyecto:** Se definen los propósitos tanto técnicos y funcionales como personales que se han tenido en cuenta para el desarrollo del proyecto.

3. **Conceptos teóricos:** Se explican aquellos conceptos teóricos más importantes usados en la realización del proyecto

4. **Técnicas y herramientas:** Se explican las diferentes tecnologías usadas para el desarrollo del proyecto.

5. **Aspectos relevantes del desarrollo del proyecto:** Se explican las diferentes fases del proyecto junto como cómo se han abordado para el correcto desarrollo del mismo.

6. **Trabajos relacionados:** Se analizan diferentes proyectos similares en el ámbito del cuidado automatizado de plantas.

7. **Conclusiones y líneas de trabajo futuras:** Se explican los diferentes aprendizajes obtenidos junto con mejoras para implementar en futuras versiones de la aplicación.

1.2. Estructura de los anexos

Los anexos se estructuran de la siguiente manera:

1. **A Plan de proyecto:** Se recoge la planificación temporal, los "sprints" realizados y el análisis de viabilidad tanto económica como legal.

2. **B Requisitos:** Se recogen los requisitos tanto funcionales como no funcionales, junto con los casos de uso.

3. **C Diseño:** Se recogen el diseño de datos, el diseño arquitectónico y el diseño procedimental junto a sus respectivos diagramas.

4. **D Manual del programador:** Se recogen la estructura de directorios junto con una guía para poder continuar con el desarrollo y mantenimiento de la aplicación.

5. **E Manual de usuario:** Se recoge una muestra del funcionamiento de la aplicación para el usuario.

2. Objetivos del proyecto

El objetivo principal de este proyecto ha sido el de desarrollar una aplicación que permita que el cuidado de las plantas por parte de los usuarios se maneje de manera más eficiente, gracias a la posibilidad de monitorizar su cuidado con la ayuda de los sensores.

Para llevar a cabo este objetivo principal se han seguido tres tipos diferentes de objetivos:

2.1. Objetivos Software:

1. Permitir la creación, modificación y eliminación de plantas junto con sus respectivos parámetros asociados.

2. Poder visualizar los diferentes tipos de valores registrados por los sensores, tanto de humedad como temperatura y luminosidad.

3. Poder visualizar en los gráficos un histórico de los datos, seleccionado el rango de fechas deseado

4. Mostrar hitos tanto diarios como mensuales relacionados con el cuidado de la planta creada, como pueden ser el cambio de tierra o agregar fertilizante.

5. Poder configurar el idioma, la foto de la planta y la visualización tanto de las gráficas como de los hitos en función de los que desee el usuario.

6. Mostrar consejos asociados a cada tipo de planta un función de la planta que cree el usuario.

7. Mostrar el estado de cuidado de la planta en forma de porcentaje.
8. Mostrar el estado de los sensores

2.2. Objetivos Técnicos:

1. Desarrollar la aplicación usando Flutter y Dart por su compatibilidad con AndroidStudio y ofrecer buen rendimiento.
2. Garantizar una buena comunicación entre la aplicación y el "backend" mediante peticiones HTTP dentro de la red local.
3. Implementar una arquitectura que en el futuro pueda ser lo mas escalable posible para poder seguir mejorando la aplicación o agregar nuevas funcionalidades, como la agregación de nuevos sensores.
4. Usar herramientas de almacenamiento local, como por ejemplo "SharedPreferences" y de tratamiento de datos como pueden ser las gráficas.
5. Hacer robusta la aplicación frente a posibles errores que pueda dar la aplicación, como por ejemplo por falta de conexión.

2.3. Objetivos Personales:

1. Aprender a programar con Flutter y Dart ya que son herramientas punteras en el ámbito de creación de aplicaciones multiplataforma.
2. Aprender a llevar de la forma más óptima tanto la planificación como la organización de un proyecto creado desde cero.
3. Aprender a integrar hardware con software, mediante el uso de peticiones HTTP dentro de la red local.
4. Mejorar la manera de documentar todo el proceso de creación de un proyecto

3. Conceptos teóricos

GreenInHouse 2.0 es un proyecto que combina la tecnología moderna con el dominio del cuidado de las plantas. Para comprender todo el desarrollo del proyecto es necesario conocer algunos conceptos teóricos relacionados con todas las tecnologías usadas y también otros relacionados con la temática de este proyecto como son las plantas..

3.1. Flutter

Flutter es un framework de código abierto de desarrollo de interfaces multiplataforma creado por Google. Permite construir aplicaciones nativas para Android, IOs, web y escritorio desde un único código fuente.

Al contrario que otros frameworks que actúan como capas intermedias entre el código del desarrollador y los componentes nativos del sistema operativo, Flutter añade directamente los elementos en pantalla gracias al uso de su motor gráfico (Skia). Esto hace que pueda tener un control total sobre el diseño.

Hay varios conceptos importantes usados en el desarrollo de la aplicación que se van a explicar a continuación:

- **Widgets:** Al usar Flutter para la creación de una aplicación, todo se crea mediante el uso de "widgets", desde un simple botón hasta la estructura completa de una pantalla. Gracias a esto, le proporciona la característica de poder reutilizar elementos y poder personalizar toda la interfaz.

  Los "widgets" se van a dividir en dos categorías:

  - StatelessWidget: Hacen referencia a aquellos componentes que no varían en el tiempo, es decir, son fijos. Este tipo de "widgets" pueden ser por ejemplo botones estáticos o textos.
  - StatefulWidget: Hacen referencia a aquellos componentes que van a variar en función de la interacción del usuario con la aplicación o en función del tiempo. Estos son sobre todo los "widgets" más usados en el proyecto, como por ejemplo en la creación de las gráficas.

- **"Hot Reload":** Esta es una de las funcionalidades más destacadas de Flutter, permite que los cambios que se hagan en el código se reflejen de manera instantánea en la interfaz y se puedan ver los cambios sin tener que estar ejecutando la aplicación cada vez que se haga algún cambio. Esta ha sido una de las funcionalidades fundamentales a la hora del desarrollo de la aplicación para probar interfaces, ajustar parámetros y todo el desarrollo de manera más rápida.

- **Pubspec.yaml:** Este es el archivo central de configuración de Flutter, en él se van a definir todas las dependencias externas, assets y configuraciones globales. Se ha usado para poder definir todas las librerías usadas, así como los diferentes recursos usados en la interfaz, como pueden ser las imágenes.

- **Paquetes externos usados:** Pub.dev es un repositorio extenso de paquetes en el que se definen aquellos paquetes que vamos a usar para el desarrollo del proyecto. Algunos de los paquetes más importantes que se han usado en el desarrollo han sido los siguientes:

  - http: Este paquete se define para poder usar las llamadas HTTP que se han usado para la comunicación con la API.
  - shared_preferences: Este paquete se usa para guardar datos localmente, como por ejemplo el nombre de la planta o el día de creación de la misma.
  - intl: Este paquete se usa para la internacionalización de la aplicación y para el formateo de fechas y números en distintos idiomas.
  - syncfusion_flutter_charts: Este paquete se usa para poder generar gráficas avanzadas para permitir a los usuarios poder visualizar el histórico de datos medidos por los sensores.

3.2. Dart

Dart es un lenguaje de programación creado por Google, orientado a objetos y con una sintaxis familiar para desarrolladores acostumbrados a lenguajes como Java o JavaScript entre otros. Este lenguaje nació con el objetivo de poder facilitar la creación de interfaces de usuario fluidas, portables y altamente reactivas. Esto lo convierte en una de las mejores opciones a la hora del desarrollo de aplicaciones modernas.

Unos de los principales conceptos teóricos usados en el desarrollo de la aplicación relacionados con Dart son:

- **Compilación Ahead-of-Time (AOT) y Just-inTime (JIT):** Dart va a permitir dos tipos de compilaciones:
  - JIT: Este tipo de compilación permite cargar los cambios de manera instantánea ("hot reload").
  - AOT: Este tipo de compilación se usa en producción para generar código nativo optimizado. De esta manera se van a reducir los tiempos de arranque y también se va a mejorar el rendimiento general de la aplicación.
- **Soporte nativo para programación asíncrona:** Dart gestiona tareas como son las llamadas a la API o la lectura de archivos. Esto lo lleva a cabo sin tener que bloquear el hilo principal de la aplicación gracias al uso de elementos nativos, como "Future", "async" y "await". Gracias a estos elementos, las interfaces van a funcionar de manera fluida incluso cuando se estén realizando múltiples operaciones en segundo plano.
- **Gestión automática de memoria:** Dart incluye un recolector de basura que se encarga de liberar de manera automática memoria que no se esté usando. Esto evita pérdidas de memoria "memory leaks" y también ayuda a que se mantenga un rendimiento estable en los dispositivos.
- **Multiplataforma:** Dart se usa principalmente para desarrollar aplicaciones móvil, pero también se pueden desarrollar para web, escritorio y servidores. Esto hace que Dart sea una de las mejores opciones para el desarrollo de aplicaciones.

3.3. AndroidStudio

AndroidStudio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones Android, creado por Google y basado en IntelliJ IDEA. En un principio fue diseñado para desarrollar aplicaciones tanto en Java como en Kotlin, aunque a dia de hoy al ser compatible con Flutter y Dart gracias a los "plugings" oficiales es uno de los mejores entornos a la hora de trabajar con este framework.

Unos de los principales conceptos teóricos usados en el desarrollo de la aplicación relacionados con AndroidStudio son:

- **Editor de código con IntelliSense:** Gracias al autocompletado inteligente de este entorno se puede escribir código de manera más eficiente, detectando errores sintácticos y dando sugerencias sobre dichos errores ayudando a identificarlos antes de la compilación. Esto ha hecho que se reduzca el número de fallos en tiempo de ejecución.

- **Depuración y emulación de dispositivos:** Gracias al emulador que Android tiene integrado se pueden descargar distintos tipos de dispositivos con distintas versiones de Android para comprobar la correcta visualización y ejecución de la aplicación. Las herramientas de depuración de código también son clave para poder resolver con mayor facilidad los errores tanto lógicos como los de diseño.

- **Integración con Flutter y Dart:** Este entorno cuenta con "plugings" oficiales que permiten trabajar con Flutter de manera nativa. Esto incluye:

  - Creación automática de nuevos proyectos
  - Ejecución eficiente de la aplicación tanto en emulador como en cualquier dispositivo
  - Soporte para "Hot Reload"

- Control de versiones (Git): AndroidStudio también tiene integración con GitHub. Gracias a esto, nos es más fácil poder realizar "commits" y "push" directamente desde el entorno sin tener que descargar los documentos y tener que actualizarlos a mano en GitHub.

3.4. Raspberry PI

La Raspberry Pi es un microordenador compacto de bajo coste desarrollado en Reino Unido por la Fundación Raspberry Pi. Su uso es muy popular, sobre todo en ámbitos educativos, en proyectos de ingeniería y en domótica, entre otros, gracias a sus diferentes usos, su pequeño tamaño y la gran comunidad de soporte que tiene.

En este proyecto, la Raspberry Pi ha tenido un papel clave, tanto como servidor local como recolector de los datos recogidos por los sensores en tiempo real y poniéndolos a disposición de la aplicación mediante el uso de la API.

Unos de los principales conceptos teóricos usados en el desarrollo de la aplicación relacionados con la Raspberry Pi son:

- **Comunicación mediante formato JSON:** Todos los datos con los que se trabajan están en formato JSON (JavaScript Object Notation), ya que es un formato ligero y fácilmente parseable. Por tanto, al hacer las peticiones a la API consultando diferentes rutas, siempre se ha usado dicho formato tanto para crear datos en la base de datos como para modificarlos o para recogerlos para trabajar con ellos.
- **Conectividad en red local:** Para que la aplicación pueda tener conexión con la maceta es necesario que tanto el dispositivo como la maceta estén conectados a la misma red local. De esta manera:
  - Se evita el uso de servicios en nube, eliminando costes extra
  - El manejo de datos se hace dentro de la red local por lo que es más seguro al no exponerlos a internet.
  - Se reducen las peticiones entre el cliente y el servidor por lo que las respuestas son más rápidas.
  - Se simplifica la configuración ya que no va a ser necesario abrir puertos ni instalar certificados SSL para conexiones externas.

3.5. Conceptos relacionados con plantas

Ya que este proyecto trata sobre el seguimiento automatizado del cuidado de las plantas, se van a introducir algunos conceptos básicos sobre botánica para comprender mejor el funcionamiento de la aplicación.

- **Fotosíntesis:** Proceso mediante el cual las plantas convierten la luz solar, el CO2 del aire y el agua del suelo en oxígeno y glucosa que van a usar como fuente de energía. Este proceso tiene lugar en las hojas, en unas estructuras llamadas cloroplastos que es donde se encuentra la clorofila, pigmento esencial para recoger la energía del sol.

  Una iluminación adecuada es imprescindible para que se lleve a cabo bien este proceso, por ello es por lo que se mide la luminosidad ambiental, ya que una deficiencia lumínica puede provocar un crecimiento lento y hojas amarillas, entre otras cosas.

- **Necesidades hídricas:** Dependiendo de la especie de planta va a tener unas necesidades hídricas u otras. Algunas plantas necesitan humedad constante mientras que otras basta con regarlas de manera esporádica. El estrés hídrico (falta de agua) puede causar que una planta se marchite o pierda las hojas, mientras que un exceso de agua puede provocar ciertas enfermedades por hongos o hacer que se pudran las raíces.

  Por tanto es muy importante tener en cuanto tanto el nivel máximo de humedad que una planta puede tener como el nivel mínimo para que empiece a marchitarse.

- **Temperatura y tolerancia térmica:** La temperatura va a influir directamente en el metabolismo de la planta. Al igual que con el agua cada planta tiene un rango óptimo de temperatura que necesita para crecer. Por debajo de ese rango el crecimiento se ralentizaría y por encima las funciones celulares pueden empezar a colapsar. Además también existen temperaturas mínimas críticas que por debajo de ellas se producen heladas y la muerte celular en la planta.

  Podemos distinguir:

  - Plantas termófilas: Plantas que requieren calor, como por ejemplo plantas tropicales.
  - Plantas mesófilas: Plantas que están adaptadas a climas templados.
  - Plantas criófilas:Plantas que toleran el frío.

- **Luz y fotoperiodismo:** La luz, aparte de afectar a la fotosíntesis, también afecta a procesos como pueden ser la germinación, el crecimiento y la floración de las plantas. Muchas plantas responden al fotoperiodo (duración del día y la noche) y se clasifican en:

  - Plantas de día corto: Plantas que florecen cuando las noches son largas
  - Plantas de día largo: Plantas que requieren más horas de luz
  - Plantas neutrales: Planras indiferentes al fotoperiodo

- **Suelo y sustrato:** El sustrato es el medio en el que se van a desarrollar las raíces. Además de sostener a la planta al suelo tiene la función de aportar tanto agua como aire y nutrientes a esta. Existen diferentes tipos:

  - Sustratos universales: Sustratos válidos para la mayoría de las especies.
  - Sustratos específicos: Sustratos específicos para un tipo de planta, como pueden ser los cactus o las orquídeas.
  - Sustratos inertes: Sustratos como por ejemplo la perlita que se usan en hidroponía que es una técnica de cultivo que permite cultivar plantas usando disoluciones minerales en lugar de suelo agrícola.

  El sustrato con el tiempo se degrada y va perdiendo los nutrientes y la aireación, por lo que se recomienda cambiarlo de forma periódica.

- **Fertilización y nutrientes:** Las plantas necesitan tanto macronutrientes como micronutrientes para un desarrollo correcto. La falta de alguno de estos nutrientes puede provocar síntomas en la planta, como deformaciones o falta de floración. La fertilización puede ser:

  - Orgánica: En este tipo de fertilización se usa el compost o el humus de lombriz.
  - Química: En este tipo de fertilización se usan abonos NPK balanceados.

4. Técnicas y herramientas

Para la realización de este proyecto se han empleado una serie de técnicas y herramientas que han permitido la realización del mismo de manera eficiente. Se van a explicar a continuación todas estas metodologías usadas durante la realización del proyecto junto con las técnicas y herramientas y una comparación con otras opciones disponibles en el mercado.

4.1. Metodología ágil basada en Sprints

Durante el desarrollo del proyecto se ha seguido una metodología que ha permitido llevar una correcta organización y llegar a los objetivos planteados. Para ello, antes del desarrollo se consideraron distintas metodologías como han podido ser las tradicionales y las ágiles.

Las metodologías tradicionales, como puede ser la metodología de *Waterfall*, se basan en un desarrollo secuencial en el que para pasar a otra fase primero se tiene que haber completado la anterior. En cambio, las metodologías ágiles se centran en un desarrollo iterativo e incremental, en el que se pueden realizar entregas parciales y recibir retroalimentación de las mismas para poder ir mejorándolas a largo plazo.

- **Metodología usada:** Se ha optado por usar una metodología ágil inspirada en Scrum. EL trabajo se ha dividido en "sprints" de dos semanas, en los que se planificaban unos objetivos al final de la reunión y en la siguiente se ponía en común esos avances para poder recibir retroalimentación y mejorar algunas cosas que no estuvieran del todo correctas.

  Cada sprint ha seguido los siguientes pasos:

  - Planificación de las tareas a desarrollar
  - Desarrollo de dichas tareas
  - Documentación de las tareas desarrolladas
  - Revisión de las tareas terminadas y planificación del siguiente "sprint"

  Este enfoque ha permitido tener un control del correcto avance del proyecto, gracias a la buena organización de esta metodología utilizada.

- **Metodología alternativa:** Como alternativa, se valoró la metodología tradicional en cascada, en la que se realiza toda la planificación del proyecto desde un principio. Esta metodología no se adapta bien al tipo de proyecto desarrollado, ya que no permite modificar decisiones ni implementar nuevas funcionalidades, por lo que es poco flexible.

  Por tanto, la elección de la metodología ágil ha sido la más óptima para que el proyecto se gestionara mejor y fuera más flexible.

4.2. Herramientas de desarrollo

Durante el desarrollo de este proyecto se han usado diferentes herramientas que han ayudado tanto a la construcción del código, como al diseño de interfaces y depuración, entre otros. La elección de estas herramientas se ha llevado a cabo teniendo en cuenta la compatibilidad entre las mismas y lo óptimas que son para este tipo de proyecto en concreto.

- **Android Studio:** ha sido el entorno de desarrollo principal usado para el desarrollo del proyecto. Se trata del IDE oficial para desarrollo Android, el cual está basado en IntelliJ IDEA, y tiene soporte nativo tanto para Flutter como para Dart gracias a sus extensiones oficiales. Unas de las principales funcionalidades usadas son:

  - Editor de código con autocompletado y sugerencias inteligentes (IntelliSense)
  - Herramientas de análisis de rendimiento las cuales pueden detectar si se producen cuellos de botella o procesos costosos.
  - Sistema para poder emular dispositivos Android y poder probar así las aplicaciones en diferentes versiones y tamaños de pantalla.
  - Gestor de dependencias y paquetes integrado con pubspec.yaml.

- **Entorno alternativo:** Visual Studio Code es un editor de texto muy popular y compatible con Flutter a través de extensiones. Este entorno consume menos recursos, por lo que es más rápido, pero tiene limitaciones con ciertas herramientas visuales. Android Studio es una opción más completa, ya que presenta emuladores y herramientas gráficas como son los "widgets".

- **Dart:** es un lenguaje de programación moderno, desarrollado por Google, orientado a objetos y con una sintaxis clara. Este lenguaje es la base sobre el cual se construye Flutter. Su uso ha sido primordial en este proyecto, ya que se ha empleado para comunicaciones con otros dispositivos, como para crear la lógica o la interfaz.

  Unas características destacadas de Dart son:

  - **Compilación Ahead-of-Time (AOT):** permite que el código se compile antes de su ejecución, lo que mejora el rendimiento de la app en dispositivos móviles.
  - **Programación asíncrona:** Facilita la gestión de tareas que puedan llevar mucho tiempo como las peticiones a la API, sin que se bloquee la interfaz del usuario.
  - **"Null safety":** es una característica moderna que obliga a que se controle la posibilidad de que hayan valores nulos. Esto hace que el código sea más seguro y que tenga menos posibilidades de tener errores.

- **Flutter:** es un framework de desarrollo de interfaces multiplataforma creado por Google. Usa Dart como lenguaje de programación y permite que se puedan crear aplicaciones para Android, iOS, web y escritorio desde el mismo código. Su ventaja frente al resto es que gracias a los "widgets" permite personalizar toda la interfaz.

  Los elementos más importantes usados en el proyecto han sido:

  - **"Widgets":** Se pueden dividir en:
    - **StatelessWidget:** usado para elementos que no van a cambiar en el tiempo como puede ser un botón o texto escrito.
    - **StatefulWidget:** usado para elementos que varían en el tiempo como pueden ser las gráficas.
  - Sistema de navegación entre pantallas: se ha usado el sistema "Navigator" de Flutter para la gestión en los cambios entre pantallas de la aplicación.

  - **"Hot Reloado":** es una herramienta que permite visualizar los cambios en tiempo real sin tener que volver a ejecutar la aplicación para ver si se han aplicado los cambios.
  - Dependencias externas:
    - **http:** Usado para realizar las peticiones contra la API.
    - shared_preferences: Usado para guardar datos de la planta activa, como puede ser el nombre o la fecha de plantación, entre otras.
    - **intl:** Usado para el formateo de fechas.
    - **syncfusion_flutter_charts:** Usado para la visualización de las gráficas.

- **Alternativa:** React Native es también un framework de creación de aplicaciones multiplataforma, pero está basado en JavaScript y React. Su arquitectura va a depender del puente entre el código JavaScript y el sistema nativo, por lo que esto puede hacer que afecte al rendimiento. Además, a diferencia de Flutter, no cuenta con un motor propio de renderizado, que haría que la aplicación fuera más fluida.

4.3. Backend:

- **Raspberry Pi:** empleada como servidor local para la recogida de datos de los sensores y guardados en una base de datos a la que se accede a través de una API. Se usó esta opción en el anterior proyecto GreenInHouse por su bajo coste y consumo, su facilidad para la integración de sensores físicos y porque Linux es el mejor entorno de desarrollo.

- **Firebase:** hubiera sido una alternativa al ser un servidor en la nube pero se optó por tener los datos de forma local y acceder a ellos a través de la API reduciendo costes y teniendo el control total sobre el entorno y los datos.

4.4. Control de versiones y gestión de tareas

Para la buena gestión del código y de la documentación del proyecto, se ha usado Git como sistema de control de versiones junto con un repositorio para la documentación y otro para el código creados en GitHub. Aparte, se ha utilizado también la herramienta de Zube, una herramienta de gestión

de proyectos la cual se integra con GitHub y permite organizar el trabajo con la creación de tarjetas como si fuera un Kanban.

- **Git y GitHub:** han permitido llevar un control del desarrollo de la aplicación. Algunas de las ventajas han sido:
  - **Historial de los cambios:** Cada "commit" que se hace, queda registrado en GitHub por lo que se puede revisar los cambios hechos y hacer un seguimiento.
  - **Seguridad y respaldo en la nube:** GitHub actúa como un repositorio don de se van a almacenar todas las versiones del proyecto.
  - **Explicación de "commits":** Mediante los mensajes escritos al realizar cada "commit", se puede hacer un seguimiento de todas las decisiones tomadas.
- **Zube:** Es una herramienta de gestión de proyectos que permite organizar las tareas usando un sistema de tarjetas ("issues"), donde cada tarjeta representa una de estas tareas a realizar. Estas tarjetas se van a definir en un "sprint" para llevar un mejor control en cada reunión. Al estar integrado con GitHub, estas modificaciones de creación de tarjetas o "sprints" se van a ver reflejadas en el repositorio vinculado, facilitando el seguimiento del desarrollo del proyecto.

  Ventajas de Zube:
  - **Organización Visual por tareas:** Las tareas se separan por "sprints" para saber qué tareas hay que tener terminadas para cada reunión.
  - **Integración con GitHub:** Cada tarea creada en Zube va a estar vinculada a un "commit" del repositorio para poder llevar así un seguimiento del progreso.
- **Alternativas a Zube:** Existen algunas alternativas como Trello, Jira o GitLab Boards aunque Zube ha sido sin duda la mejor solución por su integración con GitHub y su sencillez a la hora de usarlo en un proyecto individual.

5. Aspectos relevantes del desarrollo del proyecto

Este apartado pretende recoger los aspectos más interesantes del desarrollo del proyecto, comentados por los autores del mismo. Debe incluir desde la exposición del ciclo de vida utilizado, hasta los detalles de mayor relevancia de las fases de análisis, diseño e implementación. Se busca que no sea una mera operación de copiar y pegar diagramas y extractos del código fuente, sino que realmente se justifiquen los caminos de solución que se han tomado, especialmente aquellos que no sean triviales. Puede ser el lugar más adecuado para documentar los aspectos más interesantes del diseño y de la implementación, con un mayor hincapié en aspectos tales como el tipo de arquitectura elegido, los índices de las tablas de la base de datos, normalización y desnormalización, distribución en ficheros3, reglas de negocio dentro de las bases de datos (EDVHV GH GDWRV DFWLYDV), aspectos de desarrollo relacionados con el WWW... Este apartado, debe convertirse en el resumen de la experiencia práctica del proyecto, y por sí mismo justifica que la memoria se convierta en un documento útil, fuente de referencia para los autores, los tutores y futuros alumnos.

6. Trabajos relacionados

Este apartado sería parecido a un estado del arte de una tesis o tesina. En un trabajo final grado no parece obligada su presencia, aunque se puede dejar a juicio del tutor el incluir un pequeño resumen comentado de los trabajos y proyectos ya realizados en el campo del proyecto en curso.

7. Conclusiones y Líneas de trabajo futuras

Todo proyecto debe incluir las conclusiones que se derivan de su desarrollo. Éstas pueden ser de diferente índole, dependiendo de la tipología del proyecto, pero normalmente van a estar presentes un conjunto de conclusiones relacionadas con los resultados del proyecto y un conjunto de conclusiones técnicas. Además, resulta muy útil realizar un informe crítico indicando cómo se puede mejorar el proyecto, o cómo se puede continuar trabajando en la línea del proyecto realizado.

Bibliografía