.UBAfiuba
FACULTAD DE INGENIERÍA

Hanagotchi: aplicación para monitoreo y cuidado de plantas domésticas

Trabajo Profesional de Ingeniería en Informática (TA053/75.99)

Facultad de Ingeniería, Departamento de Computación,

Laboratorio de Sistemas de Información Avanzados

Grupo 38

Alumno:	**Padrón**
Feijoo, Sofía	101148
Firmapaz, Agustín Ezequiel	105172
Pacheco, Federico Jose	104541
Perez Andrade, Violeta	101456

Tutor: Dr. Ierache, Jorge

15 de Julio, 2024

Índice

1. Integrantes

Alumnos:

Feijoo, Sofía	sfeijoo@fi.uba.ar
Firmapaz, Agustín	afirmapaz@fi.uba.ar
Pacheco, Federico Jose	fpacheco@fi.uba.ar
Pérez Andrade, Violeta	viperez@fi.uba.ar

Tutor:

Íerache, Jorge	jierache@fi.uba.ar

2. Resumen / Abstract

Se presenta una aplicación para monitoreo de plantas domésticas en el cual se parte de datos obtenidos mediante la integración con sensores, brindando una interacción afectiva entre usuario y las plantas de su hogar además de poder interactuar con otros usuarios de la aplicación. Hanagotchi proviene de la combinación de la palabra japonesa 花 (Hana), que significa Flor, y la palabra ウォッチ (wo' chi), que proviene del inglés watch (reloj).

The work presented consists on a mobile application for monitoring domestic plants is presented in which data obtained through integration with sensors is presented, providing an affective interaction between the user and the plants of their home in addition to being able to interact with other users of the application. Hanagotchi comes from the combination of the Japanese word 花 (Hana), which means Flower, and the word ウォッチ (wo' chi), which comes from the English watch.

3. Palabras clave / Keywords

IoT (Internet de las Cosas), Floricultura, Cuidado, Socialización, Redes sociales, Afectividad, Domótica, Sensores.

IoT (Internet of Things), Floriculture, Care, Socialization, Social networks, Affectivity, Home automation, Sensors.

4. Agradecimientos

Expresamos nuestra más grande gratitud a la Facultad de Ingeniería de la Universidad de Buenos Aires, por habernos brindado una formación de la mas alta calidad, publica y gratuita, junto con los recursos necesarios para crecer como profesionales. Además, a todos los argentinos y las argentinas que a través de sus aportes al sistema educativo publico nos posibilitaron tanto la oportunidad de estudiar en la facultad como la de poder desarrollar este trabajo.

Agradecemos a la materia Trabajo Profesional de Ingeniería en Informática (TA053/75.99) por haber abierto un espacio que nos ha facilitado el desarrollo del trabajo, y mejorando procesos que en la modalidad tradicional hubiesen sido más complejos de atravesar.

Agradecemos al Dr. Jorge Ierache, por habernos brindado su predisposición y mentoría a lo largo del desarrollo de este proyecto, compartiendo sus conocimiento y resolviendo nuestras dudas durante el mismo.

Un reconocimiento especial a nuestra familia y amigos, que nos brindaron su apoyo no solo durante el desarrollo de este proyecto, sino durante todo nuestro trayecto en la universidad, compartiendo nuestras alegrías y acompañándonos en nuestras angustias. Sin su apoyo, nada de esto hubiese sido posible.

Un agradecimiento especial a la Lic. María Guillermina Ogando, por habernos brindado su ayuda para otorgarle una identidad al proyecto, a través asesorías en materia de diseño.

Agradecemos también al Ing. Agustín Rojas, por habernos brindado su ayuda y expertiz en áreas críticas del proyecto.

Por último, un agradecimiento a todas las personas que, de alguna u otra forma, han formado parte de no solo el desarrollo de este proyecto, sino de todo nuestro arduo recorrido en la universidad.

5. Introducción

Este proyecto propone abordar la problemática de la floricultura dentro de la época de industria 4.0, donde emerge una revolución tecnológica en las áreas de Inteligencia Artificial, Internet de las Cosas, Robótica, entre otros tópicos. Nuevas áreas de estudio ganan popularidad en la sociedad; entre ellas la domótica, cuyo estudio busca aplicar las nuevas tecnologías al control y automatización de viviendas y edificaciones. Es aquí donde surge una conexión entre la tecnología y el cuidado de jardines, donde la aplicación de sensores que permiten mejorar la precisión en la medición de factores (por ejemplo la temperatura o la humedad) ayudan a mejorar la toma de decisiones respecto al cuidado de la flora.

Por otra parte, los dispositivos móviles han ido reemplazando rápidamente a los dispositivos de escritorio en los últimos años. La gran mayoría de herramientas cotidianas que antes se utilizaban en computadoras han migrado hoy en día a los dispositivos móviles. Y es que la accesibilidad que brindan los dispositivos móviles es inigualable, en comparación con las computadoras.

A lo largo de este proyecto daremos una introducción más completa a los conceptos de domótica y de cómo articularla con la floricultura. Analizaremos diversas soluciones brindadas por la industria así como las vacancias que dejan como oportunidad de mejora. En base a esto, se identificarán una serie de problemáticas para luego brindar una solución integral que articule herramientas ya implementadas en otras soluciones, así como presentar una serie de innovaciones respecto a la usabilidad del usuario y la creación de una comunidad de floricultores.

6. Estado del Arte

6.1. Introducción y descripción general

6.1.1. Domótica

La **domótica** es un conjunto de tecnologías aplicadas al control y a la automatización inteligente de una vivienda o edificación, con el objetivo de mejorar la calidad de vida de sus habitantes [1]. Este término proviene de la unión de las palabras "domus"(casa en latín) y robótica", y su propósito es hacer que los espacios sean más cómodos, seguros, eficientes y adaptables a las necesidades y preferencias de las personas que los ocupan.

6.1.2. Jardines inteligentes

Un **jardín inteligente**, también conocido como jardín conectado o **jardín domótico**, es un espacio exterior que utiliza tecnología y automatización para optimizar y mejorar el cultivo, cuidado y gestión de plantas, flores y vegetación en general [2]. El objetivo principal de un jardín inteligente es lograr un equilibrio entre la naturaleza y la tecnología para crear un ambiente más eficiente, estético y sustentable. Los jardines inteligentes permiten ser monitoreados y/o controlados a distancia, principalmente desde aplicaciones móviles u otros dispositivos electrónicos.

Entre las **ventajas** que la tecnología nos permite aprovechar sobre la jardinería encontramos:

1. **Control sobre las tareas**: Las tareas necesarias en un jardín pueden quedar predefinidas y el mantenimiento permanecerá siempre óptimo. Aprovechamiento ecológico: Referido a la optimización de recursos y energía, mediante procesos como riego automatizado y control de iluminación. El riego automatizado permite gastar solo el volumen necesario de agua, pudiendo aplazar el riego si se detectan lluvias. Por otra parte, el control de iluminación permite establecer periodos de luz a los que una planta puede estar expuesta en base a sus requerimientos.

2. **Monitoreo y control de condiciones ambientales**: Integrar sensores para medir la humedad del suelo, la temperatura, la humedad atmosférica, la exposición solar y otros parámetros relevantes para el crecimiento de las plantas. Esta información se utiliza para controlar y adaptar las condiciones ambientales según las necesidades de las plantas.

3. **Seguridad permanente**: Estos jardines cuentan con cámaras y controles de vigilancia incorporados. Avisan a los propietarios cuando hay intrusos o fallos de cualquier tipo.

4. **Integración con aplicaciones y plataformas**: Permite a los usuarios controlar y monitorear su jardín desde aplicaciones móviles o plataformas en línea, brindando información en tiempo real sobre el estado de las plantas y permitiendo la programación de tareas de cuidado.

Existen en el mercado diferentes herramientas, aplicaciones, equipos y servicios que permiten la automatización de tareas referidas a la jardinería.

6.2. Análisis de trabajos existentes

6.2.1. Growing Indoor Plants with Success

Growing Indoor Plants with Success [15] es una publicación académica realizada por Bodie V. Pennisi, especialista en floricultura de la cooperativa de Extensión de la Universidad de Columbia. En ella, el autor presenta una investigación realizada por la cooperativa acerca del cuidado de plantas de interior. Se realiza un análisis detallado de los factores que afectan a las plantas (temperatura, humedad del aire, humedad de la tierra, iluminación y fertilidad de la tierra) desde la perspectiva de interiores, junto con una serie de recomendaciones relacionadas a tópicos relevantes como selección de plantas para interiores, macetas, y cuidados respecto a poda, limpieza, manejo de pestes, entre otras. Finalmente, presenta una tabla con los parámetros ideales para cada factor analizado, para una lista de más de 250 plantas. Cada parámetro se encuentra en formato numérico, lo cual permite fácilmente utilizar estos datos en sistemas expertos.

6.2.2. Flower Care (Monitoreo de condiciones ambientales)

La aplicación mobile **Flower Care** [5], desarrollada por la compañia **HuaHuaCaoCao** (del chino: flores y plantas) permite el monitoreo de diversos factores a través de un sensor que se conecta a la tierra y envía información a la app sobre exposición a la luz, temperatura, humedad y fertilidad de la tierra. La aplicación provee una interfaz simple para el monitoreo de la planta, en tiempo real, la cual permite realizar un informe diario de las lecturas realizadas por el sensor. Además, la aplicación provee un catálogo amplio de plantas con información referente a su cuidado.

Los sensores se conectan al dispositivo móvil mediante Bluetooth, lo cual permite una conexión directa con el dispositivo sin necesidad de intermediarios (aunque demanda estar cerca del sensor para que las lecturas sean inmediatas).

6.2.3. Yoolax (Integración con otros dispositivos)

Actualmente en mercado se encuentra un dispositivo conocido como **Yoolax** [3], una maceta inteligente que cuenta con sensores de temperatura, humedad, luz, ph y hasta niveles de agua, ofreciendo un sistema de interacción con emociones de la planta a través de la visualización de emoticonos en una pantalla incorporada en la maceta, que cambia según el estado en el que se encuentre la planta. Por ejemplo, el emoticono en la maceta puede cambiar según la humedad del suelo indicándonos que tiene "sed" (representando un emoticono triste); también según la cantidad de luz que ha recibido la planta podría cambiar su emoticono con diferentes tonalidades.

Además se ofrece la posibilidad que este dispositivo sea conectado a la aplicación de Yoolax Home [4] mediante WiFi en el cual se puede tener control sobre las escenas de emociones de esta maceta, como también para vincular la maceta con otros dispositivos de Yoolax para notificar al usuario de los cambios o ejecutar acciones según la temperatura, humedad o condiciones de iluminación.

Yoolax también cuenta con una base de datos de plantas que ofrece, mediante la aplicación, instrucciones óptimas para el cultivo de diferentes plantas.

6.2.4. Planta

Planta [8] es una aplicación para Android e iOS para el cuidado de plantas. Con una interfaz sencilla e intuitiva, la aplicación provee una serie de servicios como recordatorios para riego, fertilización, limpieza, etc; identificación de plantas a través del escaneo de fotos, mediciones de luz para determinar el nivel de exposición de luz para cada planta, y asesoramiento online. Además, Planta ofrece recomendaciones de plantas para principiantes según la localización del usuario. Cada planta cuenta además con su propio diario para seguir su progreso, donde el usuario puede agregar fotos y textos breves sobre la evolución de la planta. Cabe destacar que la versión gratuita posee un acceso a uso limitado, mientras que la versión paga permite utilizar las herramientas de la aplicación sin restricciones.

6.2.5. PlantNet

La herramienta de **PlantNet** [9] es otra aplicación móvil que ayuda a identificar plantas mediante fotografías, contando con una enorme base de datos para diferentes especies organizándose por flores según la zona geográfica en la que se encuentre; de esta forma se ofrece un monitoreo de ubicación de las plantas en la que el usuario fue identificando, como también la posibilidad explorar diferentes floras de otras ubicaciones y adicionalmente a contribuir a la creación de esta gran base de datos.

6.2.6. Vera

Vera [11] es una herramienta de cuidado de plantas desarrollada por Bloomscape. Una de sus características más llamativas para los floricultores experimentados es la posibilidad de crear su propio calendario de recordatorios automáticos de cuidado (riego, fertilización, etc), en vez de seguir una guía predeterminada. Además, al igual que Planta, Vera permite llevar un diario del progreso de tu planta, incluyendo fotos y registro de actividades, como la última vez que se ha cambiado el fertilizante, el riego diario, etc. Vera cuenta también con un blog incluido en la aplicación donde se comparten recomendaciones sobre el cuidado de plantas, con artículos escritos por expertos en floricultura.

6.2.7. Blossom

Por último, **Blossom** [10] es una aplicación para cultivo y cuidado de plantas presentando una amplia variedad de servicios. Blossom permite la detección del tipo de planta o enfermedades mediante imágenes capturadas con la cámara del móvil, ofreciendo guías de apoyo y prevención de dichas enfermedades identificadas e incluso un asistente de chat virtual para ayudar con la identificación de manera rápida. También cuenta con un sistema de recordatorios con notificaciones para tareas de cuidado de las plantas basándose en sus necesidades, como por ejemplo recordatorios de recomendaciones de riego o cualquier condición personalizable por el usuario.

Por otro lado, Blossom permite tomar notas a modo de registro del crecimiento de las flores, e incluso armar un blog para describir y compartir los cuidados con consejos, videos y artículos. Además, la aplicación está integrada con la store de Amazon para consumir productos recomendados en la aplicación según el usuario.

6.3. Síntesis de características comunes y diferencias

Luego del análisis de diversos trabajos actuales dentro de las aplicaciones mobile dedicadas a floricultura, puede observarse que cada una de ellas comparten entre sí, en mayor o menor grado:

- **Provisión de base de datos sobre cuidado de plantas**: la gran mayoría de las aplicaciones de cuidado de plantas dan acceso a una base de datos para informar sobre el cuidado de las mismas. En caso de no poder identificar el tipo de la planta del usuario, aplicaciones como PlantNet y Blossom permiten tomar fotos para identificarla.
- **Automatización de recordatorios para el cuidado de la planta**: muchas de las aplicaciones analizadas permiten la notificación de acciones para el cuidado de plantas. Algunas de ellas lo realizan mediante programación de alertas en calendarios, mientras que otras lo hacen mediante lecturas de diversos factores a través de sensores cerca de las

plantas. Estas notificaciones pueden estar asociadas a las necesidades específicas de cada tipo de planta.

- **Bitácora del progreso de cuidado de la planta**: algunas de las aplicaciones analizadas permiten realizar un historial del progreso del cuidado para cada planta, con la posibilidad de escribir notas personalizadas y agregar fotos del progreso. Asesoramiento sobre base de datos: algunas aplicaciones brindan asesoramiento al usuario sobre el cuidado de plantas, ya sea en formato de blog con artículos escritos por profesionales de la floricultura, recomendaciones para principiantes según localización geográfica, guías de prevención para enfermedades, o chats virtuales. Existe una vacancia en esto, y es que ninguna de estas opciones permite conectar usuarios para brindarse asesoramiento entre ellos.

- **Monitoreo de plantas mediante sensores**: aplicaciones como Flower Care [5] y Yoolax [3] precisan de sensores para poder realizar mediciones en las plantas sobre las que trabajan. Flower care provee un sensor Bluetooth que debe clavarse en la tierra, junto con la planta, para realizar las mediciones; esto conlleva al problema de que el usuario debe encontrarse cerca del dispositivo para obtener actualizaciones. Por otra parte, Yoolax integra sus sensores en una maceta, la cual puede conectarse al sistema Yoolax Home [4] mediante WiFi para realizar el monitoreo.

- **Interacción humano/planta mediante muestra de emociones según el estado de la planta**: Yoolax [3] muestra, mediante una interfaz, las mediciones realizadas en una maceta para mostrar las posibles emociones que podría tener la planta. Los emoticones se muestran a través de una banda led que realiza gestos faciales según la emoción a expresar, que varían entre felicidad, tristeza, cansancio o satisfacción. En adición, el usuario puede interactuar con la maceta a través del tacto, ya sea tocando o moviendo la maceta. Por ejemplo, si el usuario toca la maceta, la misma responde con un emoticon sonriente, expresando felicidad.

- **Integración con otros sistemas domóticos**: por último, Yoolax [3] permite la integra-

ción de su maceta inteligente con un sistema domótico. El único problema hallado aquí es que este sistema, Yoolax Home [4], es un sistema que solo permite interactuar con otros dispositivos de Yoolax, excluyendo así una buena cuota del mercado.

Características/Aplicaciones	**Flower Care**	**Yoolax**	**Planta**	**Plant Net**	**Vera**	**Blossom**
Información de cuidado de plantas						
Base de datos de cuidado de plantas	x	x				x
Identificación tipo planta vía fotos			x	x		x
Identificación disease planta vía fotos						x
Integración con sensores						
Sensores IoT	x	x				
Comunicación IoT-App vía Bluetooth	x					
Comunicación IoT-App vía WiFi		x				
Interacción con usuario						
Bitácora de progreso			x		x	x
Interacción amigable humano-planta		x				
Sistema recordatorios		x			x	x

Cuadro 1: Cuadro de características vs. aplicaciones

7. Problema detectado y/o faltante

Se puede determinar como síntesis del estado del arte en función de la tabla [1] mencionado anteriormente que las aplicaciones mencionadas cubren parcialmente los tres principales ejes: información del cuidado de plantas, integración con sensores e interacción con el usuario. Nuestra propuesta intenta satisfacer las vacancias presentadas, particularmente entre las de Yoolax y Blossom.

Nuestro trabajo se apalanca en el tercer eje de características con intenciones de ofrecer una aplicación con una buena interacción amigable humano-planta. No obstante también se mantendrán las características del primer eje relacionadas al cuidado de las plantas, y también abarcara características del segundo eje, como la integración con sensores vía WiFi aunque lo abordamos como objetivo secundario.

7.1. Falta de integración de múltiples sensores en tiempo real

La mayoría de las soluciones provistas por el mercado no ofrecen automatizaciones basadas en lecturas de limitados factores. En estos casos se suele ofrecer información textual sobre el cuidado de plantas, pero esto no permite relacionar estas recomendaciones con el estado de una planta particular. La posibilidad de integrar sensores permitiría no solo la lectura y muestra de las mediciones, sino la posibilidad de inferir nuevas recomendaciones para el usuario, brindadas en tiempo real para un accionar más preciso ya que como podemos ver en la tabla [1] de características y aplicaciones, para la aplicación de Flower Case su comunicación es mediante Bluetooth limitando su uso al estar a un radio cercano del dispositivo, o casos como Yoolax que es via WiFi pero dependiente sobre la red de Yoolax Home.

7.2. Falta de interacción con el usuario

Las aplicaciones suelen únicamente mostrar por pantalla la información recolectada por las mediciones de los sensores sin brindar ninguna interpretación de las mediciones registradas ni recomendaciones al usuario respecto de cómo actuar frente a valores fuera de lo esperado.

7.3. Falta de información del cuidado de las plantas

En la actualidad, un número significativo de individuos adquiere plantas sin poseer un conocimiento sólido acerca de sus requerimientos de cuidado, tales como la cantidad de agua necesaria, la exposición adecuada a la luz natural y la periodicidad de atención que requieren. Este desconocimiento conduce a la adquisición de plantas que lamentablemente no prosperan y, en última instancia, resulta en una inversión insatisfactoria.

8. Solución implementada

8.1. Requerimientos funcionales

8.1.1. Monitoreo de plantas domésticas

La aplicación facilita el proceso a aquellos usuarios que carecen de tiempo y/o experiencia en cuidado de plantas domesticas. La aplicación provee una interfaz amigable que le permite realizar un seguimiento de diversos parámetros relevantes para el cuidado adecuado de las plantas: temperatura, humedad del ambiente, luz y humedad de la tierra (o riego). Estos cambios en los parámetros se reflejan en la aplicación, y se envían notificaciones al smartphone del usuario en caso que la aplicación no se encuentre en ejecución. Adicionalmente, la interfaz provee información acerca del tipo de planta elegido.

8.1.2. Integración con sensores

El sistema integra un sensor que permite la medición de los parámetros previamente mencionados, integrado con una placa. Este sensor se introduce en las macetas para realizar mediciones de los factores referidos a la tierra. El sensor envía mediciones a la placa que las almacena y envía vía Internet a nuestro sistema para su posterior almacenamiento y notificación al usuario. La aplicación provee una interfaz que permite al usuario emparejar un sensor con una planta, para que las mediciones puedan ser exhibidas adecuadamente. En caso de que el usuario no cuente con un sensor, la aplicación puede brindar aproximaciones basadas en lecturas del clima local.

8.1.3. Retroalimentación usuario-planta

La aplicación implementa una interfaz bajo el concepto de Tamagotchi (a lo que nos referiremos como Hanagotchi de aquí en adelante). Esta interfaz busca crear un vínculo emocional entre el usuario y su planta que se caracteriza por la reacción del Hanagotchi frente a diferentes estímulos, tanto provenientes del entorno como del mismo usuario. En base a las mediciones

obtenidas con el sistema de sensores en una planta, el Hanagotchi expresa diferentes emociones a la cual el usuario tendrá la oportunidad de actuar para poder mejorar el estado en que se encuentra. Algunas de estas emociones que puede expresar el hanagotchi según parámetros medibles se visualiza en el Cuadro 3 en formato de emoji, pero emociones como molesto [21], deprimido [22], regado de más [23], feliz [24], abrumado [25], relajado [26], triste [27], riendo [28] e incomodo [29] tienen su figura especial representativa en la aplicación (Anexo [15.2] para más detalle). También en el Cuadro 2 se comparten algunas posibles acciones que puede realizar el usuario para regular los parámetros de la planta. Además se ofrece una retroalimentación desde el lado del usuario en el cual el mismo puede interactuar con su hanagotchi, consultando su estado o brindando comentarios positivos para que este reaccione acorde al comentario.

Parámetro	**Estado de la planta**	**Acciones de regulación**
Temperatura	Mucho calor o frío	▪ Cambiar la ubicación de la planta ▪ Cambiar la exposición a la luz solar
Humedad del ambiente	Poca humedad	▪ Cambiar la ubicación de la planta a una habitación más seca o cerrada
Humedad del suelo	Poco riego o excesivo	▪ Regar la planta con más o menos agua
Iluminación	Iluminación incorrecta	▪ Cambiar la exposición a la luz solar

Cuadro 2: Tabla de Estados y acciones de regulación

Estado	Emoción	Emoji
Poca humedad del suelo	Depresión	😔
Mucha humedad del suelo	Snorkel	🤿
Poca luz o baja temperatura	Displacer	😖
Mucha luz o alta temperatura	Abrumación	😤
Poca humedad del ambiente	Molestia	🙄
Condiciones adecuadas	Relajación	😌
Retroalimentación positiva del usuario	Felicidad	😄
Retroalimentación negativa del usuario	Tristeza	🥺
Cosquilleo	Risa	😆
Cosquilleo (en tristeza)	Malestar	😖

Cuadro 3: Estados con su emoción y emoji representativo

8.1.4. Sistema de bitácoras

La aplicación brinda un sistema de bitácoras donde el usuario es capaz de llevar un historial personalizado sobre la evolución de su planta y los cuidados. Este sistema se basa en un listado de notas que el usuario asigna cada una de sus plantas, las cuales pueden incluir tanto texto como fotografías de la planta para capturar con mayor detalle su estado. La aplicación implementa un sistema de navegación basado en la fecha de creación de las bitácoras, de manera que sea sencillo para el usuario revisar anteriores bitácoras.

8.1.5. Red social

El sistema adopta una plataforma de red social, que permite a los usuarios interactuar entre ellos y ganar seguidores interesados en explorar y admirar las plantas de otros. Los usuarios pueden crear y compartir publicaciones en forma de posteos que pueden incluir fotos, consejos o consultas a la comunidad. Cada una de estas publicaciones puede opcionalmente estar etiquetada con un tag, lo cual facilita a los usuarios la búsqueda de publicaciones bajo temáticas de su interés. Esta característica promueve la discusión y la colaboración entre la comunidad de amantes de las plantas, permitiendo la resolución de problemas a través del intercambio de información entre usuarios.

8.1.6. Recordatorios

El sistema brinda la posibilidad de programar recordatorios personalizados. Estos recordatorios se realizan en formato de notificación y podrán ser configurados para llegar a la hora que el usuario desee. El objetivo de esta funcionalidad es que el usuario pueda manejar recordatorios sobre aquellos aspectos de cuidado de plantas cuyo alcance no se encuentre dentro de nuestro sistema.

8.1.7. Dataset para investigaciones

El sistema es capaz de procesar los datos de las mediciones y fotografías tomadas por los usuarios para generar uno o más datasets de información. El objetivo de esto es preparar el sistema para generar fuentes de información que contengan fotografías del estado de las plantas, retroalimentadas con la información de las mediciones del ambiente realizadas por los sensores en el tiempo previo a la captura de la foto. Si bien no se encuentra dentro del alcance de este proyecto el realizar investigaciones con esta información, sí permite la apertura a futuras investigaciones realizadas por terceros al proyecto.

8.2. Alcances no funcionales

- **Usabilidad**: Se busca que tanto el sistema como la interfaz gráfica (aplicación mobile) cumplan los requerimientos funcionales y se adapten a las necesidades del usuario. Esto implica que el sistema sea intuitivo, que pueda contemplar a las personas con limitaciones físicas, y que posea una experiencia de usuario satisfactoria.

- **Performance**: El sistema debe ser capaz de reaccionar a eventos en un determinado tiempo aceptable. Para medirlo se pueden tomar métricas sobre Throughput, Response Time de una request, etc.

- **Seguridad**: El sistema de ser capaz de prevenir eventos/acciones maliciosos o accidentales fuera del uso esperado. Esto implica cumplir con requisitos como:

  - Confidencialidad: proteger los datos y servicios contra terceros no autorizados.
  - Autenticación: confirmación de que las partes sean quienes afirman ser.
  - Autorización: otorgar los permisos correspondientes al rol del usuario solicitante.
  - Integridad: datos no manipulables sin autorización.

- **Fiabilidad**: La arquitectura del sistema debe ser tolerante a fallos cuando existen fallos parciales en otros componentes y conectores o errores en los datos ingresados, de manera que se minimice el impacto en la experiencia del usuario.

- **Testeabilidad**: cada componente de la arquitectura del sistema debe permitir ser probado para exhibir sus fallos. Idealmente, se debe poder automatizar el proceso de testing mediante la implementación de tests unitarios.

- **Simplicidad**: La arquitectura del sistema debe ser fácil de entender, razonar y explicar, tanto para desarrolladores experimentados como para nuevos desarrolladores que acceden al código del sistema por primera vez.

8.3. Casos de uso

Habiendo mencionado los requerimientos funcionales, realizamos un diagrama de casos (Figura [1]) de uso que permita visualizar la interacción de los distintos tipos de usuarios con nuestro sistema.

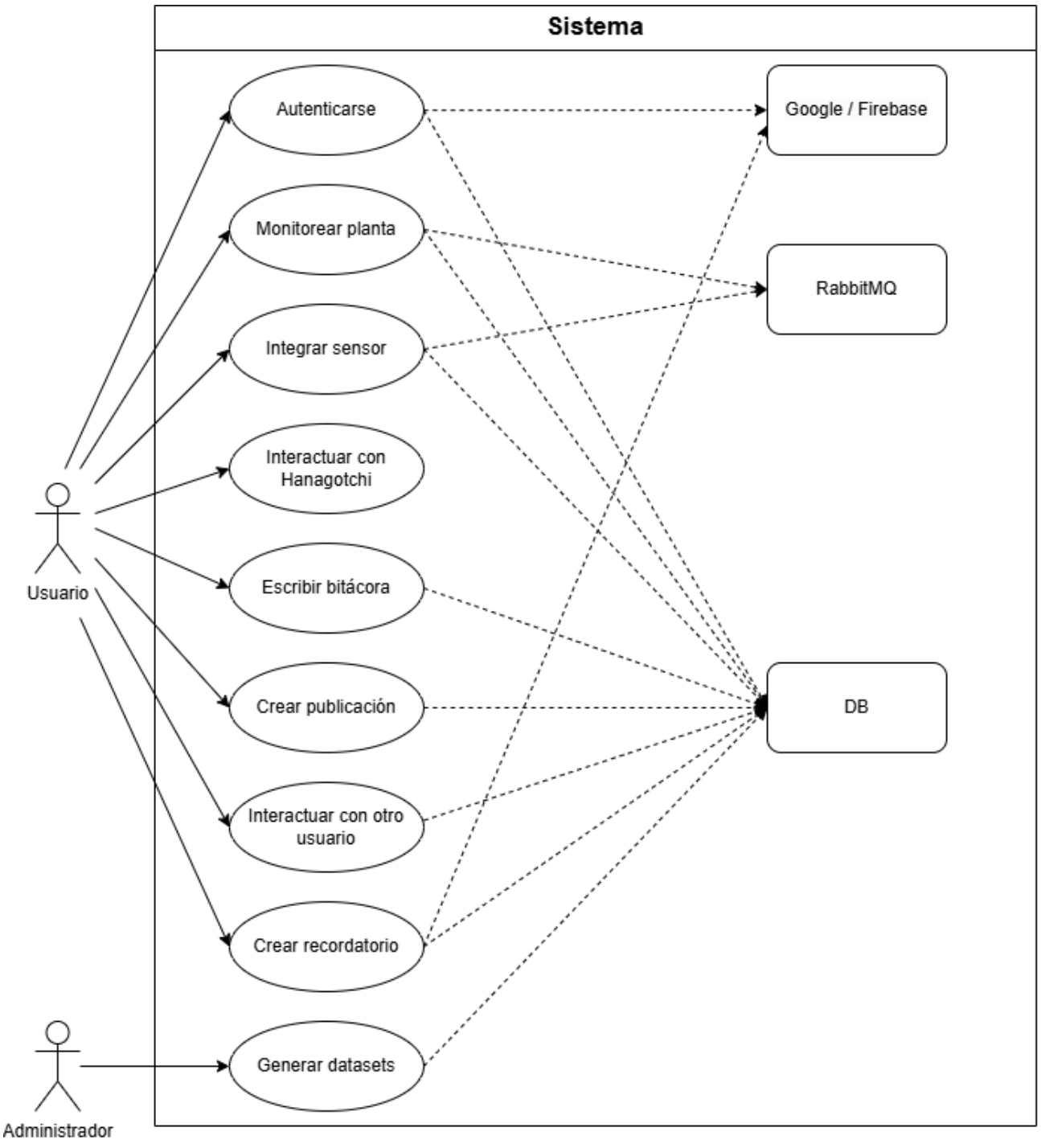

Figura 1: Diagrama de casos de uso principales

A continuación se listan todos los casos de uso del sistema:

- **Registrarse**: como usuario quiero registrarme en el sistema para poder tener una cuenta creada.
- **Completar mis datos**: como usuario quiero completar mis datos en el sistema para personalizar mi cuenta.
- **Iniciar sesión**: como usuario quiero iniciar sesión en el sistema para poder acceder al contenido de la aplicación.
- **Agregar una planta**: como usuario quiero agregar una planta para poder monitorear sus factores ambientales.
- **Eliminar una planta**: como usuario quiero eliminar una planta para dejar de monitorearla.
- **Ver mediciones de planta**: como usuario quiero ver la ultima medición de una planta para controlar que los factores ambientales sean adecuados.
- **Asociar sensor**: como usuario quiero asociar un sensor a una planta para poder monitorear sus factores ambientales a través del sensor.
- **Desasociar sensor**: como usuario quiero desasociar un sensor para no recibir más mediciones del sensor de la planta desasociada.
- **Interactuar con un Hanagotchi**: como usuario quiero interactuar con un Hanagotchi para observar su respuesta emocional frente a los factores ambientales, leer sus recomendaciones y brindarle feedback.
- **Escribir bitácora**: como usuario quiero escribir una bitácora sobre una de mis planta para llevar un control de la evolución del cuidado de la misma.
- **Ver bitácoras escritas**: como usuario quiero ver mis bitácoras escritas y filtrar su busqueda por mes y año.

- **Crear publicación**: como usuario quiero crear una publicación en la Comunidad Hana para interactuar con otros usuarios.
- **Likear publicación**: como usuario quiero dar like a una publicación para demostrar mi interés por la misma.
- **Comentar publicación**: como usuario quiero comentar un publicación para brindar una respuesta al autor de la misma.
- **Ver feed de publicaciones**: como usuario quiero ver mi feed de publicaciones para ver las publicaciones de los usuarios y tags a los que sigo.
- **Seguir usuario**: como usuario quiero seguir a otro usuario para estar al tanto de sus publicaciones.
- **Buscar publicaciones por tag**: como usuario quiero buscar publicaciones por tag para leer publicaciones relacionadas a un tema de mi interés.
- **Suscribirse a un tag**: como usuario quiero suscribirme a un tag para estar al tanto de las publicaciones taggeadas.
- **Crear recordatorio**: como usuario quiero crear un recordatorio en una cierta fecha y hora para que la aplicación me recuerde una tarea personalizada sobre el cuidado de mis plantas.
- **Generar datasets**: como administrador quiero generar datasets de mediciones y fotografías para poder brindar estos datos a terceros que realicen investigaciones.

8.4. Sistema backend

En las siguientes subsecciones se describirá el sistema backend que utiliza la aplicación final. Este sistema es expuesto mediante una API web que respeta la arquitectura REST mediante el protocolo HTTP (en definitiva, una API REST).

8.4.1. Arquitectura

La implementación del sistema backend se llevó a cabo siguiendo una arquitectura de microservicios. En la figura [2] se puede ver que se cuenta con cinco microservicios esenciales, desacoplando las diferentes lógicas de negocio. Cada microservicio tiene una responsabilidad específica, no pueden compartir su contexto entre sí y la única forma de comunicarse entre ellos es vía API REST (no se graficaron los conectores entre ellos para no exceder la complejidad del diagrama). También podemos observar el componente sensor junto a la aplicación móvil.

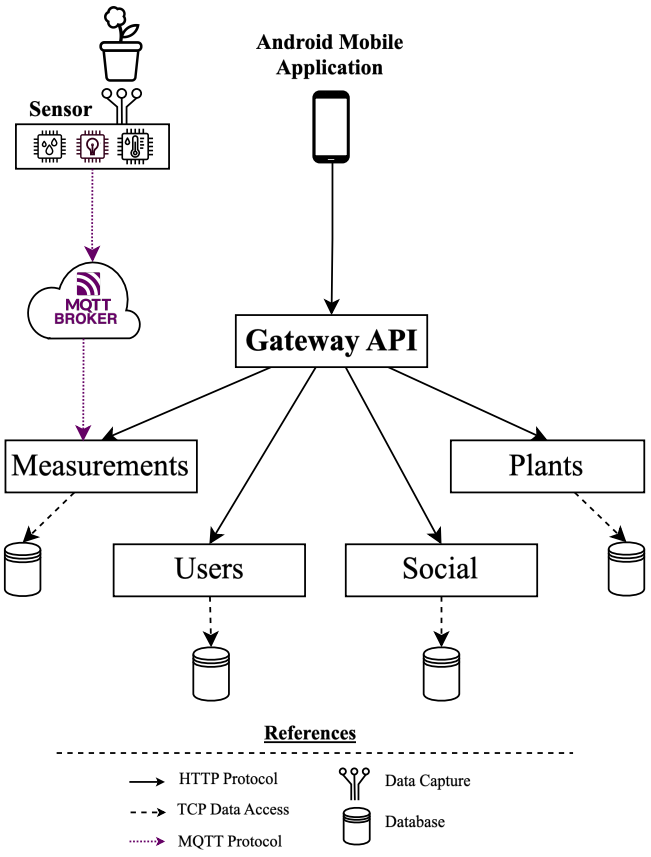

Figura 2: Diagrama de arquitectura general (vista de componentes y conectores) del sistema backend

8.4.2. Diagrama de paquetes

En la imagen de la Figura [3] se incluye un diagrama de paquetes con los que cuenta el microservicio de mediciones en particular, aunque el resto de los microservicios presentan también una estructura análoga.

El diagrama de paquetes muestra la estructura modular del sistema, organizada en diferentes categorías de paquetes. A continuación, se proporciona una explicación detallada de cada uno de estos paquetes y sus archivos componentes.

Models

Este paquete contiene los modelos de datos que representan las entidades principales. Los archivos de este paquete son:

- **base.py**: Archivo que define la clase base para todos los modelos.
- **device_plant.py**: Modelo que representa la relación sensor-planta.
- **measurement.py**: Modelo que representa las mediciones realizadas por los sensores.

Repository

Paquete encargado de la persistencia de datos y las operaciones de base de datos. Contiene:

- **measurements.py**: Archivo responsable de las operaciones de CRUD y otras operaciones de base de datos relacionadas con las mediciones. Implementa esto usando los modelos definidos en el paquete **models**.

Resources

Este paquete cuenta con recursos auxiliares que son usados, en particular por el paquete **service**. Incluye:

- **message_parser.py**: Archivo donde se manejan el análisis y procesamiento de mensajes recibidos del sensor.

- **notification_checker.py**: Archivo que verifica y maneja notificaciones que se le envían al usuario sobre sus plantas.
- **plants_dataset.csv**: Dataset que contiene información sobre plantas.
- **rule_parser.py**: Archivo que analiza y procesa reglas. Utilizando el dataset de plantas, con estas reglas se chequea si una medición está dentro o fuera del rango esperado.

Service

Este paquete contiene la lógica de negocio. Para esto, necesita acceder al repositorio, usando el paquete **repository**, además de comunicarse con otros microservicios, usando el paquete **external**. También, para el manejo de excepciones usa el paquete **exceptions** y el paquete **resources** para el manejo de mediciones. Tiene:

- **measurements.py**: Archivo que maneja la lógica de negocio relacionada con las mediciones.

External

Este paquete contiene las interacciones con microservicos externos. Incluye:

- **Plants.py**: Archivo que maneja la interacción con el servicio de plantas.
- **Users.py**: Archivo que maneja la interacción con el servicio de usuarios.

Exceptions

Este paquete gestiona las excepciones y errores específicos. Contiene:

- **deviating_parameters.py**: Manejo de excepciones para parámetros desviados.
- **empty_package.py**: Manejo de excepciones para paquetes vacíos.
- **invalid_insertion.py**: Manejo de excepciones para inserciones inválidas en la base de datos.
- **logger_messages.py**: Archivo que contiene mensajes de excepciones parametrizables.

- **MeasurementsException.py**: Clase base para excepciones relacionadas con mediciones.
- **row_not_found.py**: Manejo de excepciones para filas no encontradas en la base de datos.

Schemas

Este paquete define los esquemas de datos utilizados para la validación y serialización de los datos enviados por y devueltos al usuario. Incluye:

- **device_plant.py**: Esquema para la relación sensor-planta.
- **measurement.py**: Esquema para las mediciones.
- **plant.py**: Esquema para las plantas.
- **plant_type.py**: Esquema para los tipos de plantas.
- **user.py**: Esquema para los usuarios.

Controller

Este paquete gestiona las requests recibidas y las responses salientes, utiliza los esquemas del paquete **schemas** para chequear y serializar los datos, además de utilizar el paquete de **service**, para aplicar la lógica de negocio previo a enviar la response. Contiene:

- **measurements.py**: Archivo que controla las solicitudes relacionadas con las mediciones.

Router

Este paquete maneja las rutas y la configuración del enrutamiento del sistema, es el archivo principal. Utiliza el paquete **Query_Params** para manejar los parámetros enviados por el usuario. Incluye:

- **main.py**: Archivo principal de configuración de rutas.
- **main_rabbitmq.py**: Archivo que maneja la lógica del consumer de RabbitMQ.

Query_Params

Este paquete define los query parameters que pueden ser usados en las requests. Contiene:

- **QueryParams.py**: Archivo que define y valida los parámetros.

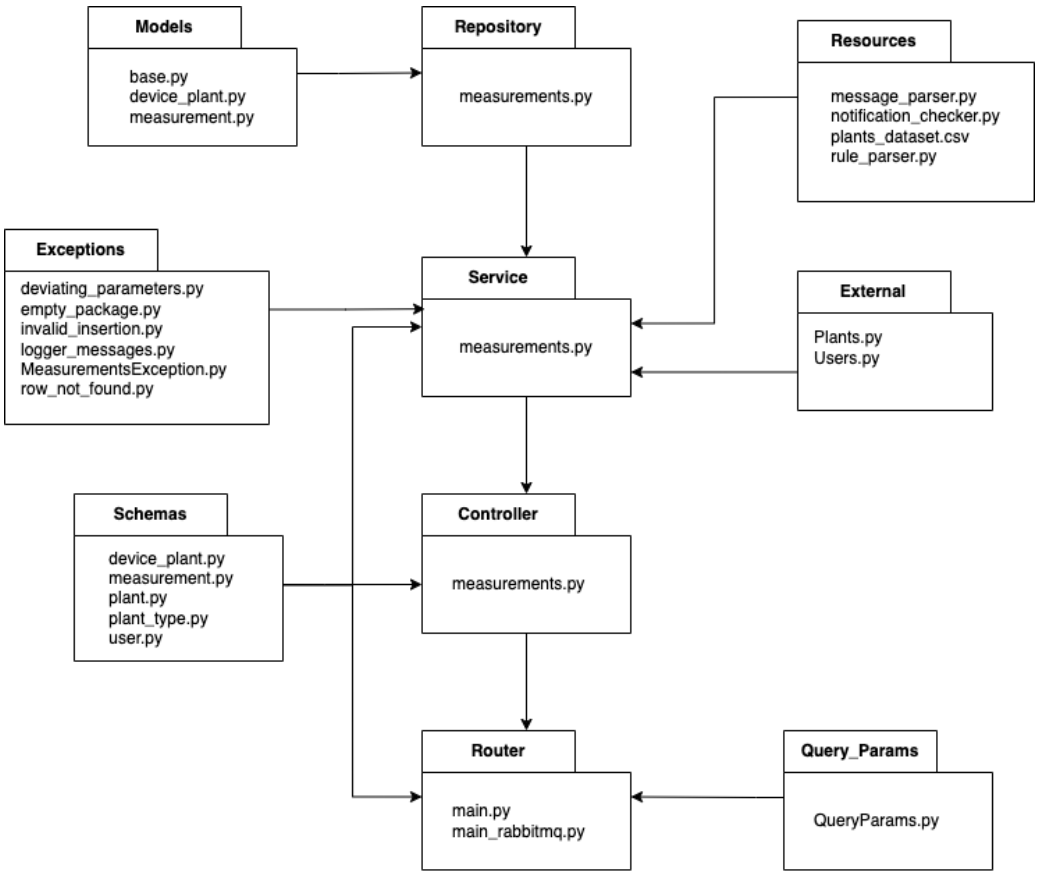

Figura 3: Diagrama de paquetes del microservicio de mediciones

8.4.3. Responsabilidades de los microservicios

Como se mencionó en la sección anterior, el sistema se compone de cinco microservicios. A continuación, se detallan las responsabilidades específicas de cada uno:

- **Usuarios**: Este microservicio gestiona todas las operaciones relacionadas con los usuarios, tales como su creación mediante registro, inicio de sesión, edición de perfil, y la creación, edición y eliminación de notificaciones personalizadas.

- **Social**: Encargado de todas las operaciones de la red social. Esto incluye la creación, edición y eliminación de publicaciones, comentarios y "me gusta". Además, maneja las interacciones entre usuarios, como seguir y dejar de seguir.
- **Plantas**: Este microservicio se encarga de todas las operaciones relacionadas con las plantas, como su creación, edición y eliminación. También gestiona las bitácoras, permitiendo agregar fotos, editar o borrar entradas.
- **Mediciones**: Responsable de gestionar las mediciones y los sensores. Se encarga de la adición de sensores a las plantas y de gestionar las mediciones enviadas por estos sensores, las cuales son leídas y persistidas en la base de datos.
- **Api-Gateway**: Actúa como un intermediario. Su objetivo es brindar un único punto de entrada al sistema. Este microservicio recibe las requests, verifica su validez y, si son válidas, las redirige al microservicio correspondiente.

En el anexo [15.5] se encuentran todos los repositorios con el código fuente implementado por cada microservicio.

8.4.3.1. Ejemplos de casos de uso A continuación se presentan diagramas de actividad que representan algunos casos de uso del sistema. Con estos diagramas se busca clarificar de manera gráfica el flujo de tareas que se ejecutan al detonar el caso de uso puntual del sistema a lo largo de los microservicios descriptos previamente.

Medición de un sensor

En la Figura [4] se analiza el flujo de tareas realizadas cuando el sensor realiza una medición. Al comienzo, nuestros sensores realizan mediciones de los diferentes factores ambientales dispuestos en ello (luz, temperatura, humedad del ambiente y humedad de la tierra). Luego envía estas mediciones a la ESP32, donde se empaquetan bajo un formato específico y se envían al microservicio de mediciones a través de una cola MQTT. Una vez que el microservicio recibe el paquete, valida que este no esté vacío, obtiene el tipo de la planta medida en base al identificador del sensor, y determina si alguno de los factores medidos se encuentra fuera del

rango esperado. En ese caso, se envía una notificación al usuario reportando la planta y su(s) desvío(s). Finalmente, la medición es almacenada en la base de datos.

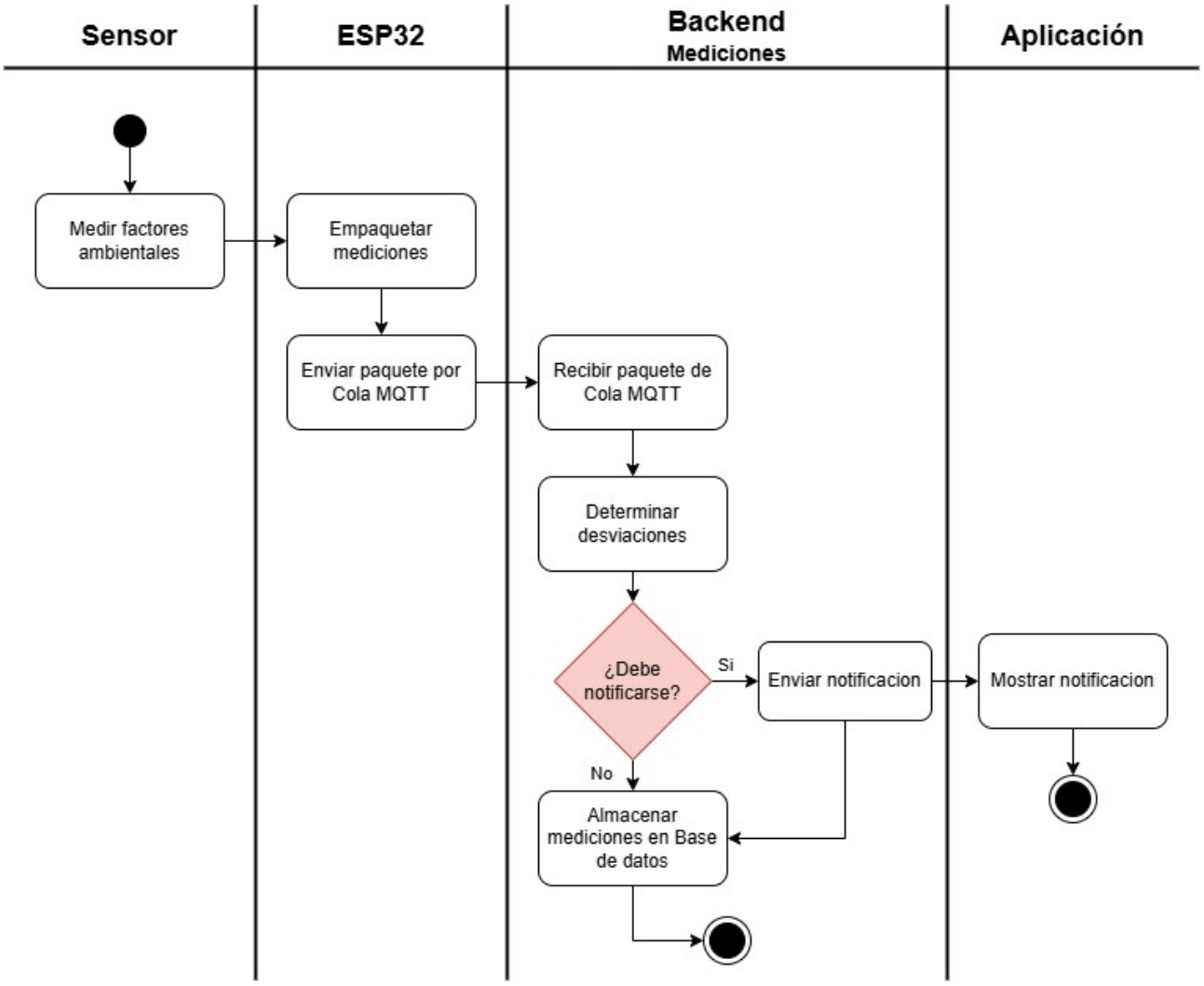

Figura 4: Medición de un sensor.

Creación de una bitácora

La Figura [5] grafica el flujo de tareas que se realizan cuando un usuario crea una bitácora. Una vez creada la bitácora en la aplicación, se enviarán los datos (contenido de la bitácora e identificador de la planta a la cual se le escribe una bitácora). al endpoint correspondiente del sistema backend, junto con un token de acceso único para cada usuario. Una vez allí, el Gateway

interceptará el paquete y validará la integridad del token de acceso. De no poder determinar si el token es válido, se rechazará la creación de la bitácora, y se responderá con un error 401 si el token se encuentra expirado o 403 si no se pudo determinar la integridad del token, finalizando la actividad.

Una vez validado, el Gateway redirige la tarea al microservicio de Plantas, donde lo primero que se realizará es pedir al microservicio de Usuarios la desencriptación del token de acceso y la obtención de los datos del usuario. Luego se determina si el usuario solicitante es dueño de la planta a la cual se le escribe una nueva bitácora; de no ser así, se rechaza su creación y se responde con un error 401. Caso contrario, se acepta la nueva bitácora y se almacena en la base de datos.

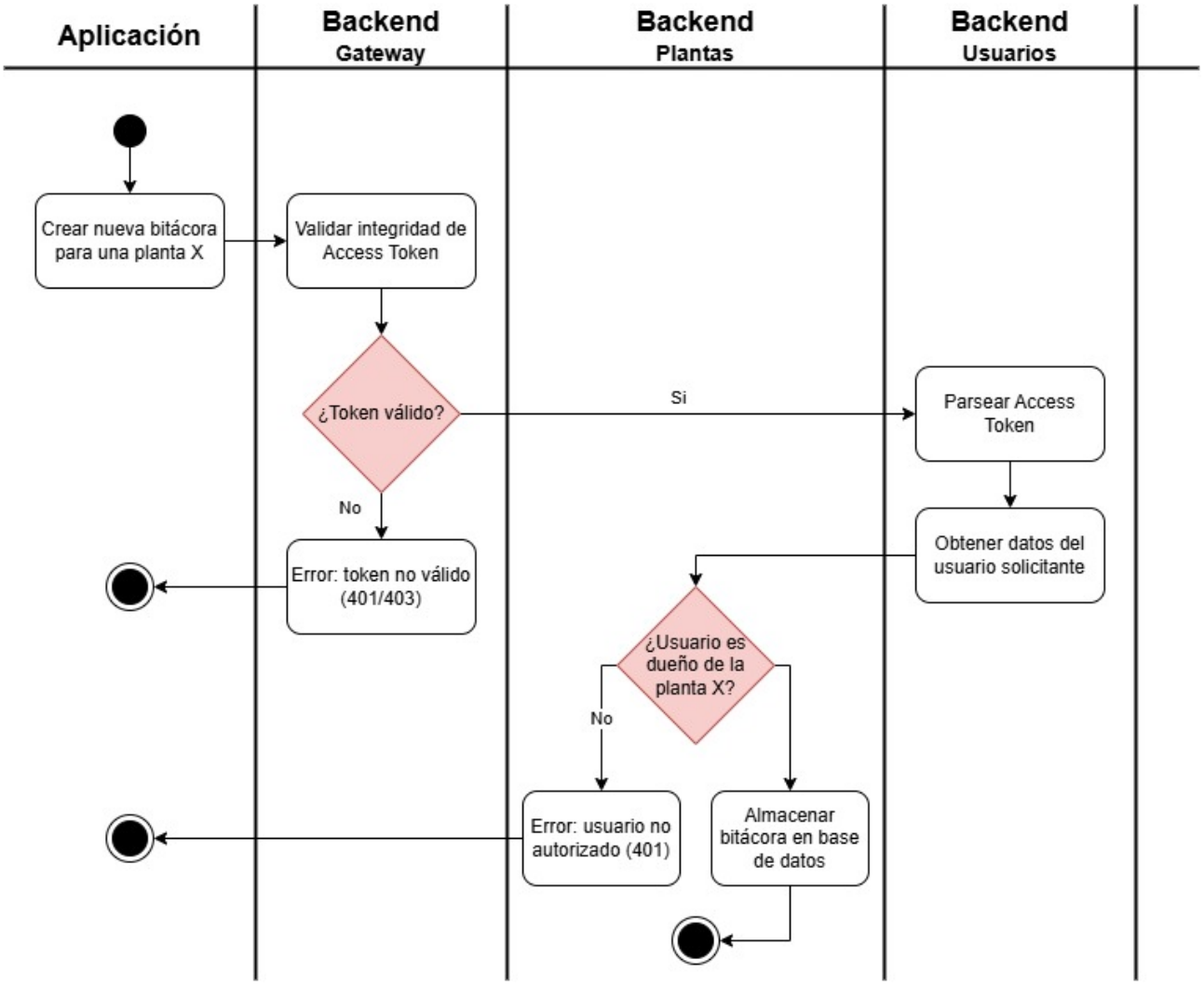

Figura 5: Creación de una bitácora.

8.4.4. Arquitectura de los microservicios

La arquitectura de los microservicios utilizada en el sistema backend está organizada en capas, cada una con una responsabilidad específica. Esta organización permite una separación clara de responsabilidades y facilita la lectura y modificación del código.

En cada microservicio, el **archivo main** es el punto de entrada principal. En él se definen los endpoints de la API REST, que son los puntos donde la API está expuesta para que interactúe el cliente.

La capa de **controlador** actúa como un intermediario entre los endpoints definidos en el archivo main y la lógica de negocio. La función principal del controlador es recibir las requests HTTP de los clientes, parsear los parámetros y devolver la respuesta adecuada.

La capa de **servicio** es la encargada de implementar la lógica de negocio. Procesa los datos y envía la respuesta al controlador, pasando por la capa de repositorio si es necesario.

Por último, el **repositorio** es la capa que maneja la interacción con la base de datos. En esta capa se definen las operaciones ABM y otras consultas necesarias para interactuar con los datos persistidos en la base de datos. Al abstraer las operaciones de base de datos en esta capa, se logra que las demás capas trabajen de forma independiente de la implementación específica de la base de datos.

8.4.5. Tecnologías utilizadas

En la Figura [6] se pueden observar algunas de las principales tecnologías utilizadas en el backend las cuales serán descriptas a continuación.

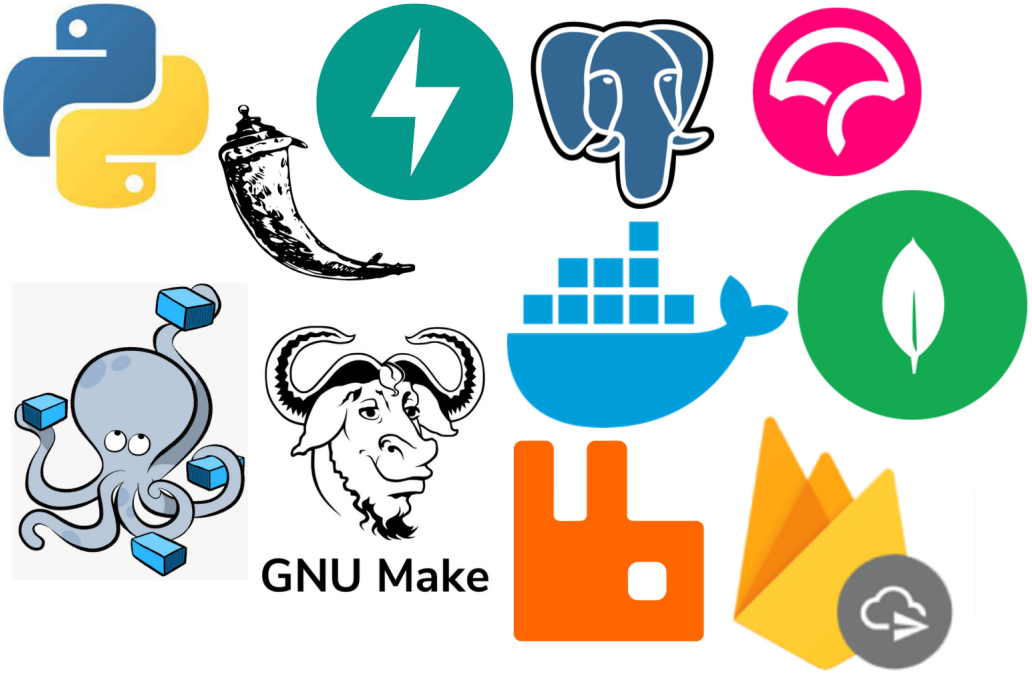

Figura 6: Tecnologías utilizadas

Python y Frameworks

Todos los microservicios fueron desarrollados en python. Para el api-gateway se utilizó el framework Flask, mientras que para el resto de los microservicios se optó por FastAPI. La elección de Flask para el gateway se debe a su flexibilidad y simplicidad en la gestión de rutas, que resultó muy útil para gestionar la lógica de enrutamiento. FastAPI fue elegido para el resto de los microservicios por tener un gran soporte para creación de APIs RESTful. Este framework facilita la implementación de arquitecturas REST al permitir la definición clara y concisa de endpoints REST, maneja las validaciones automáticamente y ofrece un muy buen rendimiento por su integración con ASGI (Asynchronous Server Gateway Interface).

Docker y Docker Compose

Para desplegar y gestionar nuestros microservicios, todo fue orquestado con Docker. Docker nos permite empaquetar cada microservicio con todas sus dependencias en contenedores aislados, asegurando una ejecución consistente en cualquier entorno. Esto mejora la eficiencia en desarrollo y pruebas, ya que cada integrante del equipo puede replicar el entorno de producción en su máquina local. Además, utilizamos Docker Compose para orquestar y gestionar los servicios, ya que requieren múltiples contenedores, como bases de datos o colas de mensajes en RabbitMQ.

Makefile

Cada microservicio cuenta con un Makefile que proporciona instrucciones claras para iniciar y detener los servicios. Esto estandariza y automatiza tareas comunes.

El resto de las tecnologías se describen en las siguientes secciones.

8.4.6. Base de datos

Al trabajar con una arquitectura de microservicios, los bases de datos utilizadas en cada uno deben ser independientes entre sí para buscar el mayor desacoplamiento posible, es decir, cada microservicio debe tener su propia base de datos. Por lo tanto, se eligió una base de datos para cada microservicio adaptándose a las necesidades de cada uno.

Para los microservicios de Usuarios, Plantas y Mediciones se utilizó una base de datos

relacional PostgreSQL [36] para tener una estructuración rígida de los datos. En el microservicio de Red Social se utilizó una base de datos no relacional MongoDB [34] para evitar caer en la estructuración de datos relacional siendo más flexible en la consulta de datos.

En el anexo [15.8] se adjuntan diagramas con los modelos entidad-relación de la base de datos PostgreSQL de los microservicios de usuarios, plantas y mediciones. También se adjunta un breve ejemplo con los documentos de las dos colecciones utilizadas en la base de datos MongoDB del microservicio de red social.

8.4.7. Testing unitario en los microservicios

Para asegurar la calidad y confiabilidad de los microservicios implementados, se realizaron pruebas unitarias en los servicios: usuarios, mediciones, plantas y red social. Estas pruebas se centran en la capa de servicio, que implementa la lógica de negocio de cada microservicio, como se detalla en la sección [8.4.4].

La elección de la capa de servicio para las pruebas unitarias se basa en su rol crítico de manejar la lógica de negocio y garantizar que cada microservicio funcione según lo esperado.

Para medir la efectividad de las pruebas unitarias, se utilizó codecov, una herramienta que evalúa la cobertura de código alcanzada por los tests. Se estableció un umbral mínimo del 70 % de cobertura para cada servicio.

Además, los servicios de api-gateway y el simulador no fueron objeto de pruebas unitarias. El gateway, al ser una capa de intermediación que transfiere requests entre clientes y microservicios, se considero que no tiene una lógica de negocio significativa para ser testada. Similarmente, el simulador, siendo una herramienta interna de propósito específico, se considera fuera del alcance de estas pruebas, enfocándose más en su correcto funcionamiento técnico y de rendimiento interno.

8.4.8. Mecanismos de seguridad - OAuth Google - JWT Bearer

En el sistema, se implementó un mecanismo de autenticación exclusivo a través de Google. Esto significa que todos los usuarios pueden registrarse o iniciar sesión únicamente usando sus

credenciales de Google. Una vez autenticado correctamente, se genera un JWT (JSON Web Token) vinculado al identificador único del usuario. Este token JWT debe obligatoriamente incluirse en el header de todas las requests subsiguientes para verificar la identidad del usuario en todos los servicios del sistema. En caso de no tenerlo, la request será rechazada por el api-gateway.

El uso del JWT proporciona una capa adicional de seguridad al sistema ya que nos garantiza que cada request provenga de un usuario autenticado y autorizado. Esto previene escenarios donde un usuario podría manipular datos de otro usuario, asegurando la integridad de la información y manteniendo la privacidad de los datos de cada usuario.

El flujo de autenticación es el siguiente: el API Gateway valida la presencia y validez del token JWT en el header del request. Después, cada servicio, para sus operaciones de lógica de negocio, verifica si el token incluido en la request pertenece al usuario autenticado, asegurando así que solo el propietario del token pueda realizar acciones específicas como editar datos.

8.4.9. Sistema de notificaciones - Firebase Cloud Messaging (FCM)

El sistema de notificaciones es un componente clave en la aplicación móvil, ya que permite a los usuarios recibir notificaciones sobre diferentes eventos que suceden en la aplicación.

Para esto, se utilizó Firebase Cloud Messaging (FCM) [37], un servicio de mensajería de Google que permite enviar este tipo de mensajes a dispositivos móviles. Además que es una servicio con solución multiplataforma que permite enviar notificaciones a dispositivos Android como iOS.

Como decisión de implementación, se decidió que cada microservicio sea responsable de notificar a los usuarios que correspondan según el evento que se haya producido. Por ejemplo, si un usuario quisiera establecer algún recordatorio para regar una planta, el microservicio responsable de gestionar los recordatorios sería el encargado de notificar al usuario en el momento que corresponda.

8.4.10. Sistema de mensajería - RabbitMQ

En la figura [2] de la arquitectura general se observó una parte clave del sistema backend donde el microservicio de mediciones tiene una diferencial responsabilidad con respecto a los demás microservicios. Será el encargado de recibir, procesar, guardar e incluso notificar las mediciones por parte de los diversos sensores de cada usuario del sistema final.

Para resolver esta responsabilidad fue necesario contar con un sistema de mensajería para la comunicación remota de los sensores a nuestro sistema. Se conoce del mundo IoT que el protocolo estándar y por excelencia es el protocolo MQTT [26], un protocolo ligero con requerimientos mínimos de red facilitando la transmisión de mensajes entre dispositivos IoT y aplicaciones.

Para implementar el sistema de mensajería se utilizó como broker a RabbitMQ [27], un servicio que actúa como intermediario permitiendo la comunicación mediante el protocolo MQTT.

Debido a esto, al microservicio de mediciones se le adicionará la responsabilidad de actuar como consumidor de los mensajes enviados por los sensores suscribiéndose a un tópico específico de RabbitMQ. Sabiendo que el microservicio de mediciones es una API web, fue necesario contar con un worker que se encargue de consumir los mensajes del broker de RabbitMQ para evitar que la API web se sature (o se quede esperando infinitamente por un mensaje que nunca llega) y no pueda atender a los clientes que interactúan con la API de mediciones. Este worker será un proceso encargado de ejecutar cualquier tarea pesada, ejecutándose de forma aislada del proceso de la API web y con la responsabilidad principal de ser consumidor de los mensajes recibidos por el broker de RabbitMQ.

Este worker recibe cada mensaje con un protocolo simple; como mínimo cada mensaje deberá contener un identificador de dispositivo (que es asignado por nosotros) para saber a qué sensor pertenece la medición, el tiempo en el que se realizó la medición y los parámetros **(humedad del ambiente, temperatura, nivel de riego, luz)** de la medición. Se puede encontrar un breve ejemplo de este mensaje en la sección de anexos [15.6.1]. En base a esto, todos los dispositivos que envíen mensajes al broker de RabbitMQ deberán hacerlo respetando este protocolo simple para que el worker pueda consumirlos y procesarlos.

El procesamiento de los mensajes recibidos en el worker culmina con la persistencia de la medición en la base de datos del microservicio de mediciones, pero antes se analiza cada parámetro de la medición que corresponde a una planta de un determinado usuario. Si algún parámetro (o varios) de la medición supera los límites óptimos, dependiendo del tipo de planta y según la publicación *Growing Indoor Plants with Success (Bodie V. Pennisi, 2009, p.18)* se notificará (mediante al sistema de notificaciones mencionado en la sección anterior) al usuario que su planta se encuentra desviada de los valores óptimos.

La Figura [7] resume todos los componentes finales del sistema mensajería con el microservicio de mediciones subdividido internamente por el worker (que inicialmente se suscribe al tópico de 'mediciones' del broker MQTT para que posteriormente reciba mensajes con las mediciones) y el proceso de la API web tal como se visualiza, para que la aplicación móvil pueda realizarle solicitudes por medio del microservicio del API Gateway como por ejemplo para obtener una ultima medición. En la figura también se destaca un prototipo de sensor totalmente genérico sin entrar en detalles de su implementación *(ver en la sección [8.7])*, asumiendo que el mismo se encarga de publicar el mensaje con la medición a nuestro broker MQTT. El resumen, el flujo representado en la figura empieza con el sensor genérico, publicando un mensaje en el broker MQTT donde el worker del microservicio de mediciones lo recibe por estar suscrito y lo procesa para terminar almacenándolo en la base de datos del microservicio; eventualmente la aplicación móvil podría acceder a esta última medición almacenada, o el usuario final podría ser notificado en caso de existir una medición desviada en el mensaje recibido.

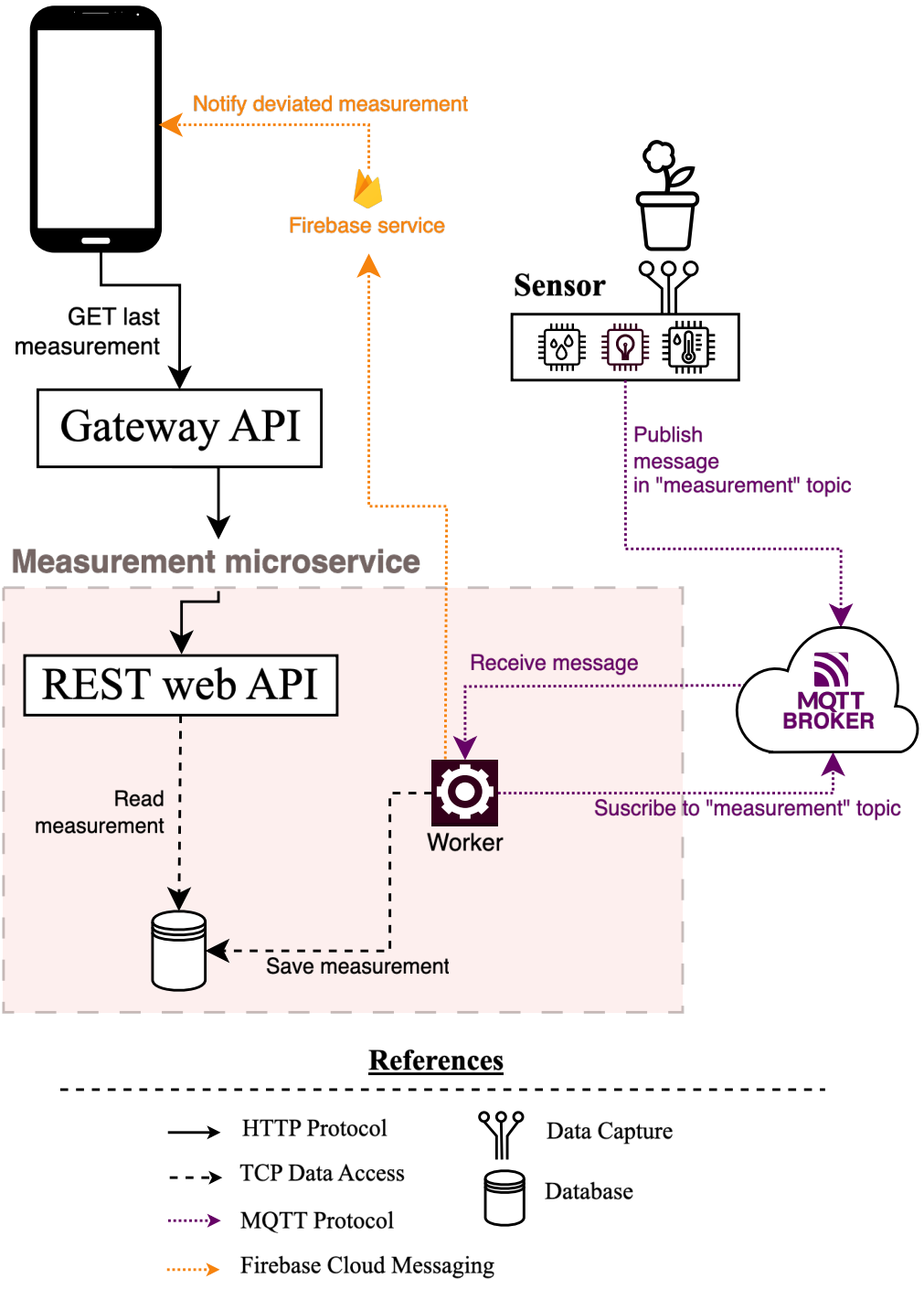

Figura 7: Componentes del sistema de mensajería

8.4.11. Sistema de recordatorios

Para que los usuarios puedan recibir recordatorios personalizados en la aplicación móvil (por ejemplo un usuario podría programar un recordatorio para regar una planta y ser notificado a las 19:00 hs) fue necesario implementar un sistema de recordatorios.

El microservicio de usuarios del backend se encargará de gestionar los recordatorios de los usuarios y de notificar en determinado tiempo estos recordatorios al usuario. Pero esto último no debería interferir con el normal funcionamiento de la API web del microservicio de usuarios. Para resolver esto, de forma similar al sistema de mensajería, se implementó el sistema de recordatorios con un worker que se encargue de identificar y de notificar los recordatorios de los usuarios en la fecha y hora que han indicado.

Pero, ¿cómo sabe el worker qué recordatorios (y cuándo) notificar? El worker hace uso de una cola de mensajes de Redis [28] donde consume *ticks* enviados por otro worker emisor de *ticks*. En concreto, dentro del microservicio de usuarios existirán dos workers.

Este último worker será encargado de programar un *tick* cada 5 minutos (debido al riesgo [8.6.1] al desplegar el sistema backend) enviando estos *ticks* como mensajes a la cola de Redis. Por otro lado, el worker consumidor de *ticks* se encargará de la tarea pesada al buscar en la base de datos los recordatorios que se deben notificar en ese *tick* y de notificar de forma asincrónica a todos los usuarios que programaron recordatorios en dicho *tick*. Al tener 2 workers, evitamos que la tarea de programar *ticks* no se vea interrumpida por la tarea pesada de identificar y notificar todos los recordatorios que existan por usuario; los ticks se encolan, y el worker consumidor los procesa una vez que le toque.

Esta solución permite que el microservicio de usuarios pueda escalar horizontalmente; se podría tener más workers consumidores de *ticks* para distribuir la carga de trabajo.

8.5. Sistema frontend - Aplicación móvil

Como parte de la solución, se ha desarrollado una aplicación movil que será el punto de conexión entre los usuarios y nuestro sistema. Al día de hoy, la aplicación se encuentra disponible solo para dispositivos Android (ver Anexo [15.9]). También se dispone el Anexo [15.5.1] con el

repositorio que incluye el código fuente implementado para la aplicación.

8.5.1. Tecnologías utilizadas

Typescript

Typescript[17] es un lenguaje de programación que a su vez es un superset de Javascript (lo que significa que cualquier código Javascript válido también es código Typescript válido). Así este lenguaje tiene los beneficios de Javascript como su flexibilidad y facilidad de uso, agregando además robustez mediante un sistema de tipos estáticos, el cual permite un control sobre los datos.

React Native

React Native[18] es un framework de desarrollo de aplicaciones móviles de código abierto creado por Facebook. Permite construir interfaces de aplicaciones para iOS y Android utilizando el mismo código base escrito en Typescript, lo cual reduce significativamente el tiempo y los costos de desarrollo.

Npm

Node Package Manager[19] es un gestor de dependencias para Javascript. Utilizamos esta herramienta para instalar, actualizar y desinstalar las librerías utilizadas en el proyecto.

OpenWeather API

La API de OpenWeather[22], proporcionada por la plataforma de servicios meteorológicos OpenWeather, permite a los desarrolladores acceder a datos meteorológicos actuales, pronósticos y datos históricos de ubicaciones en todo el mundo.

En nuestra aplicación hicimos uso de esta API para acceder a parámetros como la humedad y la temperatura en la ubicación del dispositivo móvil.

Expo

Expo[21] es un ecosistema de herramientas que nos permite crear, desarrollar, deployar y distribuir fácilmente aplicaciones React que funcionan en Android, iOS y la web.

Esta herramienta nos facilitó el proceso de compilación de las aplicaciones tanto para entornos de desarrollo como para entornos de producción.

Además, utilizando la aplicación Expo Go[20] fue posible probar las aplicaciones compiladas desde el servidor de Expo directamente en nuestros dispositivos de manera rápida y sencilla.

Firebase Storage

Firebase Storage es un servicio proporcionado por Firebase[23], una plataforma de desarrollo de aplicaciones móviles y web creada por Google.

Firebase Storage nos permitió almacenar de manera segura las imágenes que suben los usuarios a través de la aplicación.

Firebase Authentication

Firebase Authentication es otro servicio proporcionado por Firebase[23] que simplifica el proceso de autenticación de usuarios, proporcionando métodos seguros y fáciles de integrar para autenticar a los usuarios en las aplicaciones.

Ofrece múltiples métodos de autenticación, aunque para nuestra aplicación solo utilizamos autenticación mediante Google.

8.5.2. Arquitectura Funcional

La aplicación está diseñada para brindar a los usuarios una experiencia integral en el cuidado de sus plantas, permitiéndoles monitorear las condiciones ambientales, gestionar bitácoras de evolución y formar parte de una comunidad de aficionados y expertos en floricultura.

A nivel arquitectura, la aplicación se divide en 4 secciones principales, las cuales estan compuestas por una serie componentes "pantalla", los cuales se componen a su vez de componentes mas pequeños, los cuales definen tanto su vista como su lógica de controlador. Esto permite a cada componente ser independiente de la pantalla que se está utilizando, y potencialmente ser reutilizable en cualquier pantalla.

A continuación, se detalla cada una de las funcionalidades implementadas, incluyendo la pantalla principal, la sección de bitácoras y la comunidad Hana, así como otras funcionalidades adicionales.

Gestión de Plantas

La pantalla principal, tal como se ve en la Figura [8], es donde el usuario podrá observar un

resumen de las ultimas mediciones realizadas para cada una de sus plantas.

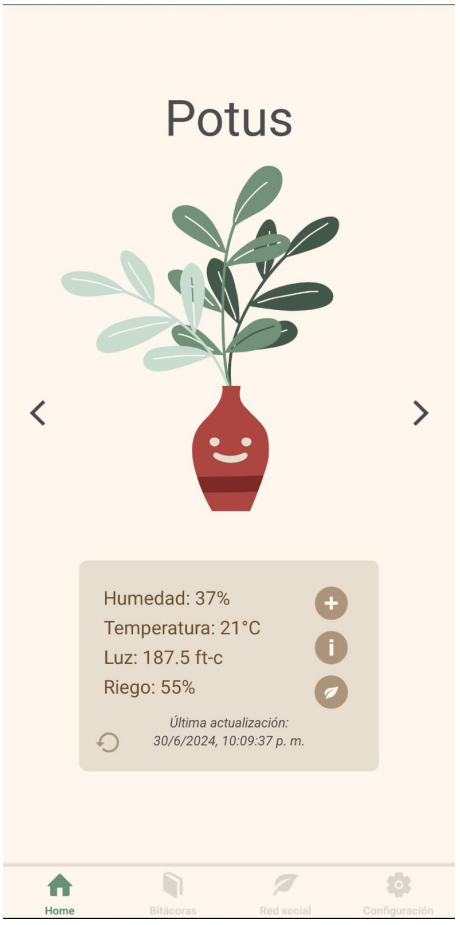

Figura 8: Pantalla principal con Hanagotchi relajado

Entre sus funcionalidades implementadas podemos destacar:

- **Lectura de mediciones**: La pantalla exhibe las ultimas mediciones realizadas por un sensor, brindando información sobre los factores ambientales que impactan en la planta. Si la planta no tiene un sensor asociado, es posible obtener aproximaciones a través de lecturas de la API de OpenWeather [22].
- **Respuesta emocional frente a mediciones**: Cada uno de los Hanagotchis responde a las lecturas recibidas, exhibiendo diferentes emociones. Frente a una emoción "negativa", el usuario deberá accionar para neutralizar la causa y que el Hanagotchi retome su estado relajado. Adicionalmente, el usuario podrá interactuar con su Hanagotchi haciéndole

cosquillas o brindando una retroalimentación positiva. El Anexo [B] muestra diferentes emociones que un Hanagotchi puede adoptar.

- **Información sobre la planta**: La pantalla exhibe dos botones que muestran información tanto de la planta en sí como de su tipo.
- **Acceso directo a creación de bitácoras**: En caso de requerirlo, la pantalla ofrece un acceso rápido a la pantalla de creación de bitácoras para la planta que se encuentra en la pantalla.
- **Crear Hanagotchis**: Los usuarios pueden crear tantos Hanagotchis como deseen. Cada Hanagotchi tendra asociado un nombre, el tipo de planta que lo define y, opcionalmente, un sensor que realice mediciones.
- **Asociar sensores**: Los usuarios podrán asociar un sensor que se encargará de realizar las mediciones de factores ambientales de la planta. Para ello, el usuario debe registrar el numero de serie del sensor e indicar la planta que será medida.

Bitácoras

La sección de bitácoras ofrece al usuario la posibilidad de gestionar notas que registran la evolución de sus plantas a lo largo del tiempo. Entre sus funcionalidades implementadas podemos destacar:

- **Filtrar bitácoras por fecha de creación**: la pantalla de bitácoras exhibe una lista de las ultimas bitácoras escritas por el usuario. Adicionalmente, el usuario puede seleccionar el mes y año de las bitácoras que desea listar, para un mejor manejo de estas.
- **Crear/Editar bitácoras**: Los usuarios pueden escribir nuevas bitácoras para registrar aquellos cambios que noten en la evolución de sus plantas, o aquellos tratamientos que realizan sobre sus plantas (cambio de fertilizante, poda, etc.). Las bitácoras ofrecen la posibilidad de adjuntar fotografías para enriquecer el contenido del registro.

Todas estas funcionalidades pueden verse en la pantalla de Mis Bitácoras, como muestra la Figura [9].

Figura 9: Pantalla de bitácoras

Comunidad Hana

La sección de Red Social (Figura [10]) permite a los usuarios mantenerse conectados con el objetivo de crear una comunidad, tanto de aficionados a la floricultura como de expertos floricultores, que permita compartir intereses comunes entre usuarios y fomente la colaboración entre ellos.

Figura 10: Pantalla de comunidad (Feed)

Entre sus funcionalidades implementadas podemos destacar:

- **Creación de publicaciones**: la aplicación permite la creación de publicaciones con la posibilidad de adjuntar fotografías. Otros usuarios pueden dar like a estas publicaciones, y realizar comentarios a modo de reacción. Adicionalmente, estas publicaciones pueden ser taggeadas mediante el uso de hashtags ('#'), de manera que la publicación aparezca como resultado de búsqueda para otros usuarios.

- **Suscripción a otros usuarios**: los usuarios pueden suscribirse a otros con el objetivo de mantenerse actualizado respecto de las publicaciones del usuario. Al suscribirse, las publicaciones de este usuario aparecerán en su feed.

- **Suscripción a tags**: Los usuarios pueden suscribirse a tags, de manera que las publicaciones taggeadas aparecerán en su feed. Adicionalmente, estas serán listadas en la barra de navegación a modo de acceso directo.
- **Búsqueda de contenido**: La aplicación implementa una barra de búsqueda que permite a los usuarios buscar a otros, además de publicaciones taggeadas con el contenido deseado.

Otras funcionalidades

- **Crear recordatorios**: La sección de recordatorios, como muestra la Figura [11], permite al usuario crear notificaciones personalizadas, que serán recibidas en la fecha y hora que él desee.

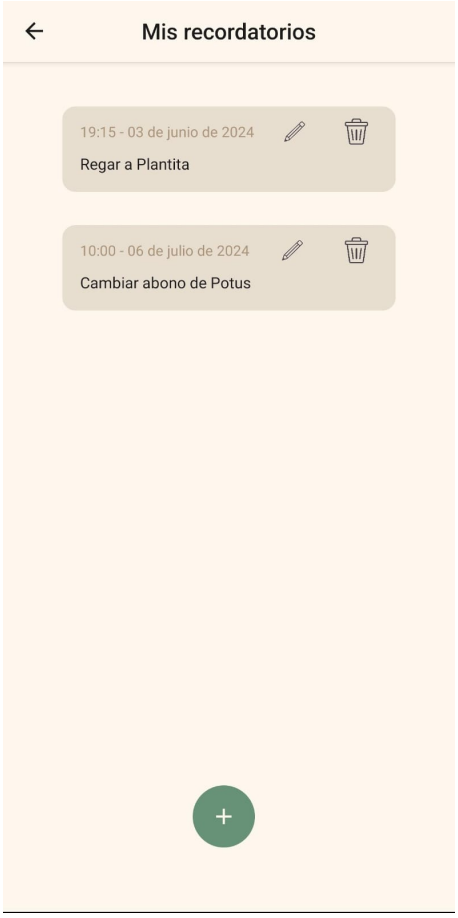

Figura 11: Pantalla de recordatorios

8.6. Infraestructura

En esta sección se detallan las herramientas y servicios utilizados para la implementación de la infraestructura de nuestros servicios y aplicación móvil.

8.6.1. Despliegue de los microservicios backend

Para el despliegue de los microservicios backend se necesitó de una plataforma Cloud donde alojar, logrando que la aplicación móvil pueda consumir estos servicios de manera remota. Se optó por Heroku [29], una plataforma como servicio (PaaS) que permite a los desarrolladores construir, ejecutar y operar aplicaciones en la nube, además de ser un servicio sencillo de utilizar evitando mayores complicaciones en la configuración.

Heroku ofrece un plan académico para estudiantes [33], el cual brinda una cantidad fija de créditos mensuales para utilizar en la plataforma. No solo eso, dispone de Plan Eco [30] para desplegar servicios el cual se puede utilizar con los créditos académicos, siendo suficiente para el despliegue de cada microservicio con sus workers de nuestro proyecto.

La contra de este plan es que mantendrá las instancias como inactivas (o "dormidas") si no se utilizan durante un determinado tiempo, además de tener un límite de horas de uso al mes. Esto se tuvo en cuenta y fue comentado con más detalle en una sección [12.3] de los riesgos.

Para la base de datos relacional de PostgreSQL, se utilizó el servicio de Heroku Postgres [31] el cual también puede ser utilizado con los créditos académicos, aunque para no superar la cuota mensuales de créditos académicos se utilizó una única instancia de motor de base de datos para todos los microservicios backend pero cada microservicio tenía su propio esquema de base de datos con sus datos (sin que ningún otro microservicio pueda acceder a ellos). Por último, en la base de datos no relacional utilizada para el microservicio de red social, se utilizó MongoDB Atlas [35] que brinda una base de datos gratuita para utilizar en proyectos pequeños.

Para el broker del sistema de mensajería, se utilizó el servicio en Heroku de CloudAMQP [32] que actúa como broker de RabbitMQ y admite el protocolo MQTT. Este servicio fue utilizado en su versión gratuita.

8.6.2. Proceso de integración y distribución continua (CI/CD)

En el proyecto, todos los repositorios cuentan con un proceso de integración y distribución continua (CI/CD). La única forma de realizar un push a la rama *master* es mediante un *pull request* previamente aprobado por uno o más integrantes del equipo.

Cada commit desencadena la ejecución de dos trabajos (*jobs*): uno que ejecuta *pylint* para asegurar la calidad del código y otro que corre las pruebas unitarias (*tests*) verificando que la cobertura de código supere el umbral definido. Si estos trabajos no se completan satisfactoriamente, no es posible realizar el *merge* a *master*.

Adicionalmente, cada commit en la rama *master* se despliega automáticamente en Heroku, garantizando que la última versión del código esté siempre en producción.

8.6.3. Despliegue de aplicación móvil

Expo Application Services (EAS) con un conjunto de servicios cloud-based para aplicaciones creadas con Expo o React Native [24]. Estos servicios permiten agilizar y facilitar el proceso de desarrollo de las aplicaciones, desde su creación hasta su publicación en tiendas de aplicaciones.

Para este proyecto, se utilizo principalmente **EAS Build**. Este servicio permite la construcción de binarios de aplicación hosteada en los servidores de EAS, lo que nos ha permitido prescindir del gasto de recursos de nuestras computadoras para su construcción, además de facilitar la distribución de estos binarios entre desarrolladores. Opcionalmente, EAS build permite la posibilidad de construir la aplicación junto con un cliente de desarrollo, el cual brinda herramientas de debugging útiles para el desarrollo de la misma.

Se contrató el paquete gratuito, el cual nos ha brindado hasta 30 builds mensuales.

8.7. Hardware de sensorización para plantas

Presentaremos a continuación una propuesta de solución a la falta de integración de múltiples sensores en tiempo real vía WiFi [7.1] proponiendo un hardware de sensorización para que

el usuario final pueda obtener lecturas de mediciones de sus plantas con la aplicación móvil desarrollada, ofreciendo la posibilidad de que el usuario tenga la libertad de ubicar este hardware en cualquier lugar de la red WiFi de su hogar. En la Figura [12] se puede observar cómo queda conformado el hardware de sensorización final que incluye dos componentes claves: un sensor (Xiaomi) junto a una pequeña placa microcontroladora (ESP32, conectado a una batería como fuente de alimentación de 5V). Estos componentes serán comentados con más detalle en las próximas subsecciones.

Figura 12: Hardware de sensorización

8.7.1. Sensor (Xiaomi) para lectura de mediciones

El objetivo principal del proyecto se centró en el desarrollo de software, por lo que se buscó un enfoque sencillo en términos de hardware. Se tomó la decisión de usar el sensor Xiaomi Flower Care - HHCCJCY01 [39] observado en la Figura [13], debido a su simplicidad y a la facilidad de integración sin necesidad de cableado complejo. Este sensor dedicado para plantas nos permite obtener mediciones precisas de parámetros ambientales, como temperatura, conductividad de la tierra y luz, utilizando tecnología Bluetooth para la transmisión de datos.

Figura 13: Sensor Xiaomi Flower Care - HHCCJCY01

8.7.2. Microcontrolador ESP32

El microcontrolador ESP32 (Figura [14]) es una placa de desarrollo compacta y potente, que cuenta con un procesador dual-core, capacidades integradas de WiFi y Bluetooth, y diversos periféricos que permiten su uso en una gama muy amplia de aplicaciones de IoT.

Figura 14: Microcontrolador ESP32

Este dispositivo fue elegido para el proyecto debido a algunas de las siguientes ventajas:

- **Conectividad:** La posibilidad de conexión de WiFi y Bluetooth facilita la comunicación con otros dispositivos y sensores, permitiendo la recolección y transmisión de datos de forma inalámbrica y sencilla.
- **Versatilidad:** Su capacidad para manejar múltiples tareas simultáneamente y soportar varios protocolos de comunicación lo hace ideal para aplicaciones complejas.

- **Bajo costo:** El ESP32 es una opción económica en comparación con otras soluciones similares, lo que lo hace accesible.
- **Facilidad de programación:** Utilizando el software ESPHome [38], es posible flashear el ESP32 con archivos de configuración YAML, simplificando enormemente el proceso de configuración y programación.

Para que funcione, es necesario que el ESP32 esté conectado de forma constante a una fuente de alimentación de 5V. Las especificaciones técnicas del sensor adquirido se pueden ver en el anexo [15.7.2].

8.7.2.1. Software del microcontrolador Como se menciona en la sección anterior, el ESP32 se controla utilizando el software ESPHome, el cual permite flashear la placa con archivos de configuración YAML sin necesidad de escribir código a bajo nivel. Esta herramienta puede utilizarse para configurar la conexión WiFi [43] y otros parámetros necesarios para las operaciones del microcontrolador. Además, se utiliza para indicar parámetros de configuración como por ejemplo la dirección MAC Bluetooth del sensor a al cual la placa debe conectarse. Para ver el archivo de configuración que se utilizó referirse a [15.7].

8.7.3. Diagrama de conexión y recorrido final de mediciones

En anteriores secciones se presentó un diagrama [7] con los componentes esenciales del sistema de mensajería, donde también se podía observar el recorrido del mensaje con un sensor genérico, pasando por el worker del microservicio de mediciones hasta llegar a la aplicación móvil. En este caso, en base a los 2 componentes de hardware mencionados anteriormente, se presenta un diagrama de componentes definitivo [15] con el recorrido del mensaje desde el hardware hasta el microservicio de mediciones.

En la Figura [15] se omiten detalles del microservicio de mediciones y de la aplicación móvil. Lo único que varía este diagrama con respecto a la Figura [7] es el agregado de los componentes de hardware con la placa controladora ESP32 y el sensor Xiaomi. Vemos como el sensor emite mediciones por Bluetooth para que los reciba la placa ESP32, y luego la misma

termine encargándose de enviarlo a nuestro broker MQTT. Estas funcionalidades serán mayor detalladas en las siguientes dos subsecciones.

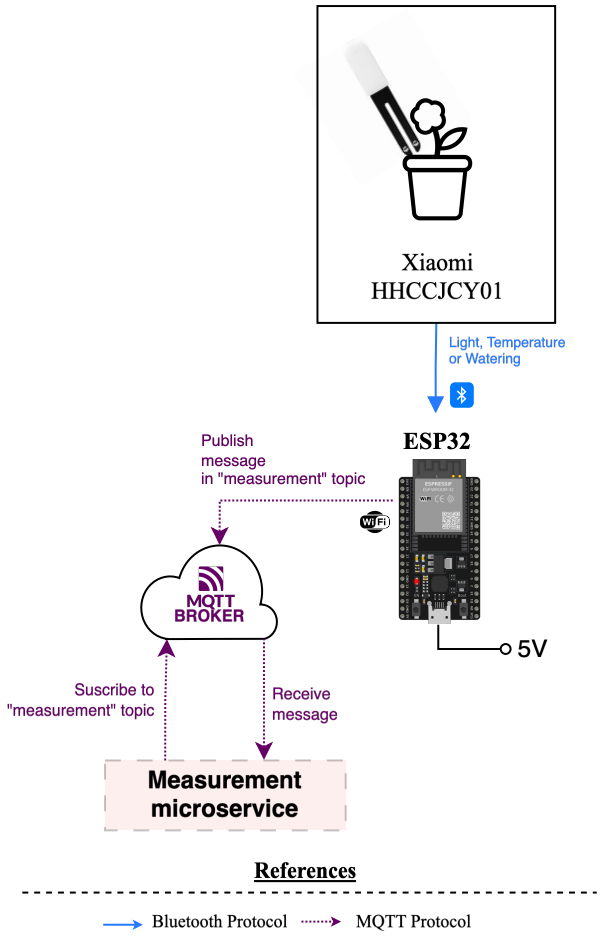

Figura 15: Componentes del hardware de sensorización

8.7.3.1. Recepción de mediciones vía Bluetooth Gracias a que el ESP32 es un microcontrolador con soporte para Bluetooth (BLE), desde el software ESP Home es posible configurar al ESP32 para que pueda recibir las mediciones del sensor Xiaomi vía Bluetooth.

Esto es posible mediante el módulo tracker de BLE [41] que trae el software. Este módulo es pasivo, no necesita conectarse al sensor, sino que trackea los paquetes BLE de aquellos dispositivos que sean indicados en la configuración del software. Dado que este software tiene compatibilidad con el sensor Xiaomi [40] adquirido, es posible agregar este sensor en la configuración especificando la MAC Address del sensor.

8.7.3.2. Publicación de mediciones al broker RabbitMQ vía WiFi Teniendo en cuenta que el ESP32 es un microcontrolador con soporte para WiFi, desde el software ESP Home es posible configurar a qué red WiFi conectar el ESP32 [43]; pero también es importante mencionar que el software ESP Home también dispone de un módulo para poder publicar a un broker MQTT [42] todas las mediciones tomadas con los sensores.

Desde la configuración del software, se puede indicar el host de nuestro broker contratado junto a sus credenciales para que el ESP32 se pueda conectar a él, logrando publicar las mediciones al broker mediante el protocolo MQTT.

Cabe mencionar que con este módulo, también es posible indicar el tópico al cual se publicaran las mediciones. En nuestro caso, los mensajes con las mediciones fueron publicados al tópico de *measurements*.

8.7.4. Emparejamiento del sensor con usuarios de la aplicación

El proceso de emparejamiento del sensor con los usuarios de la aplicación se realiza mediante la configuración del ESP32 para publicar paquetes con un número de serie determinado. El número de serie lo registra el usuario en la aplicación, lo que permitie el soporte de múltiples dispositivos con la simple adición de más pares sensor-ESP32.

Para facilitar la configuración de la red WiFi del hogar en el ESP32, el software controlador es capaz de generar una red WiFi hotspot [44]. Este hotspot permite al usuario conectarse a la red temporal generada por el ESP32, desde la cual se abre una interfaz de usuario como la que se ve en la Figura [16]. En esta UI se muestran todas las redes WiFi disponibles en el entorno. El usuario puede elegir la red deseada, ingresar la contraseña correspondiente y, de esta manera, conectar el ESP32 a la red WiFi.

Para guiar al usuario en este proceso, en la pantalla de la aplicación se incluyen instrucciones, mostrando las credenciales de la red hotspot generada por el ESP32 y guiando al usuario a través de cada paso necesario para completar la configuración de la red.

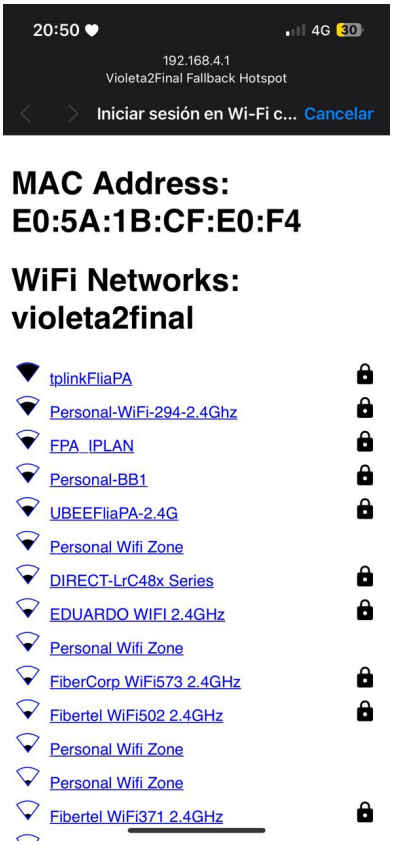

Figura 16: Interfaz de usuario para la configuración del ESP32, mostrando las redes WiFi disponibles

8.8. Sensor virtual (simulador)

Con el fin de probar fácilmente nuestro backend y la integración del mismo con la aplicación, creamos un sensor virtual. Este sensor virtual tiene la capacidad de crear y enviar paquetes con la misma estructura que los enviados por el sensor real (es decir, con los parámetros temperatura, humedad, luz, humedad de la tierra, timestamp y id del sensor) pero con los parámetros simulados.

8.8.1. Simulación a tiempo real - OpenWeather

Una manera posible de utilizar el sensor virtual es inicializarlo para que el mismo cree y envíe un paquete cada cierto período de tiempo. Este período puede ser una cantidad fija de

segundos, minutos u horas, o un cierto diferencial significativo en algún parámetro medido.

En este modo, la temperatura y la humedad son recolectados OpenWeather API[22] en base a la ubicación indicada y la humedad de la tierra y la luz son simulados.

8.8.2. Test de paquete parametrizado

También es posible enviar un paquete con uno o más parámetros personalizados. Por ejemplo, podría enviarse un paquete que tenga específicamente una temperatura de 21°C y una humedad del 10 % y el resto de los parámetros simulados.

8.8.3. Test de paquete personalizado según tipo de planta

Por último, el sensor virtual también permite enviar un paquete que presente desviaciones. Indicando qué tipo de planta es la que está asociada al sensor que está siendo simulado, el sensor virtual puede crear un paquete que presente los valores fuera de los rangos esperados para los parámetros que le hayan sido indicados.

8.9. Generación de datasets

Con el pasar del tiempo de vida del sistema, este ira recolectando de manera automática información relevante respecto al ciclo de vida y cuidado de las plantas. Esta información es sumamente valiosa debido a que expresa el detalle de los diversos parámetros ambientales que la planta ha registrado desde su ingreso a la aplicación.

Se desarrolló un script en Python que permite la generación de datasets a trave de la lectura y preprocesamiento de los datos almacenados en nuestra base:

- **Dataset de mediciones**: contiene datos referidos a las lecturas realizadas por los sensores: identificador de planta, tipo de planta, factores ambientales, hora de lectura.
- **Dataset de fotografías**: recopila las fotografías tomadas por los usuarios en las bitácoras escritas, junto con los identificadores de la bitácora y la planta, y la hora de captura de la fotografía.

- **Dataset de bitácoras**: recopila los comentarios escritos en cada bitácora para complementar las observaciones que puedan realizarse sobre las fotografías.

Estos datasets serían generados por los administradores del sistema, bajo evaluación de peticiones de diversas personas o entes interesados.

8.10. Diagramas de actividad

A continuación se presentan diagramas de actividad que representan algunos casos de uso del sistema. Con estos diagramas se busca clarificar de manera gráfica el flujo de tareas que se ejecutan al detonar el caso de uso puntual del sistema.

8.10.1. Medición de un sensor

En la Figura [17] se analiza el flujo de tareas realizadas cuando el sensor realiza una medición. Al comienzo, nuestros sensores realizan mediciones de los diferentes factores ambientales dispuestos en ello (luz, temperatura, humedad del ambiente y humedad de la tierra). Luego envía estas mediciones a la ESP32, donde se empaquetan bajo un formato específico y se envían al microservicio de mediciones a través de una cola MQTT. Una vez que el microservicio recibe el paquete, valida que este no esté vacío, obtiene el tipo de la planta medida en base al identificador del sensor, y determina si alguno de los factores medidos se encuentra fuera del rango esperado. En ese caso, se envía una notificación al usuario reportando la planta y su(s) desvío(s). Finalmente, la medición es almacenada en la base de datos.

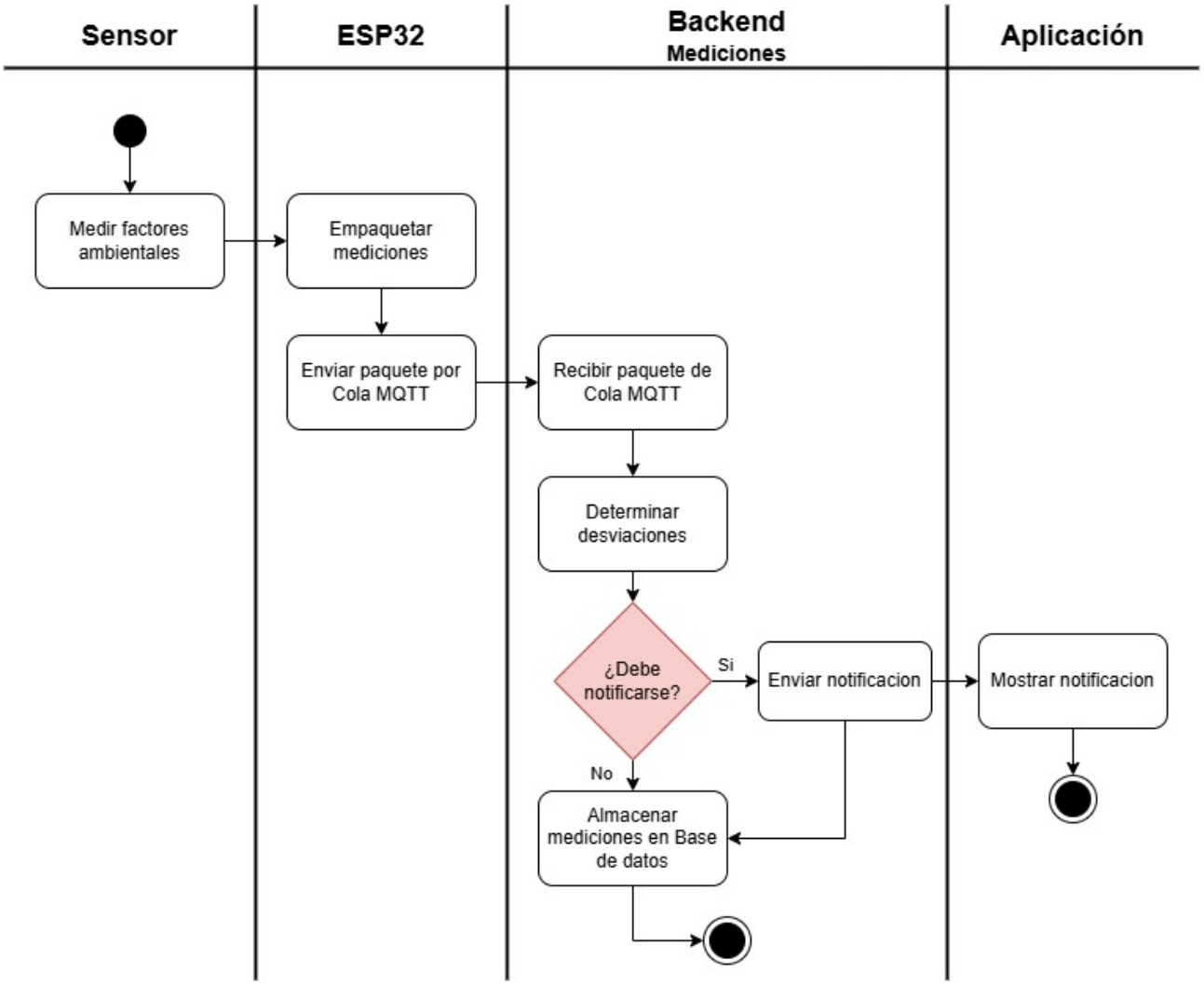

Figura 17: Medición de un sensor.

8.10.2. Creación de una bitácora

La Figura [18] grafica el flujo de tareas que se realizan cuando un usuario crea una bitácora. Una vez creada la bitácora en la aplicación, se enviarán los datos (contenido de la bitácora e identificador de la planta a la cual se le escribe una bitácora). al endpoint correspondiente del sistema backend, junto con un token de acceso único para cada usuario. Una vez allí, el Gateway interceptará el paquete y validará la integridad del token de acceso. De no poder determinar si el token es válido, se rechazará la creación de la bitácora, y se responderá con un error 401 si el

token se encuentra expirado o 403 si no se pudo determinar la integridad del token, finalizando la actividad.

Una vez validado, el Gateway redirige la tarea al microservicio de Plantas, donde lo primero que se realizará es pedir al microservicio de Usuarios la desencriptación del token de acceso y la obtención de los datos del usuario. Luego se determina si el usuario solicitante es dueño de la planta a la cual se le escribe una nueva bitácora; de no ser así, se rechaza su creación y se responde con un error 401. Caso contrario, se acepta la nueva bitácora y se almacena en la base de datos.

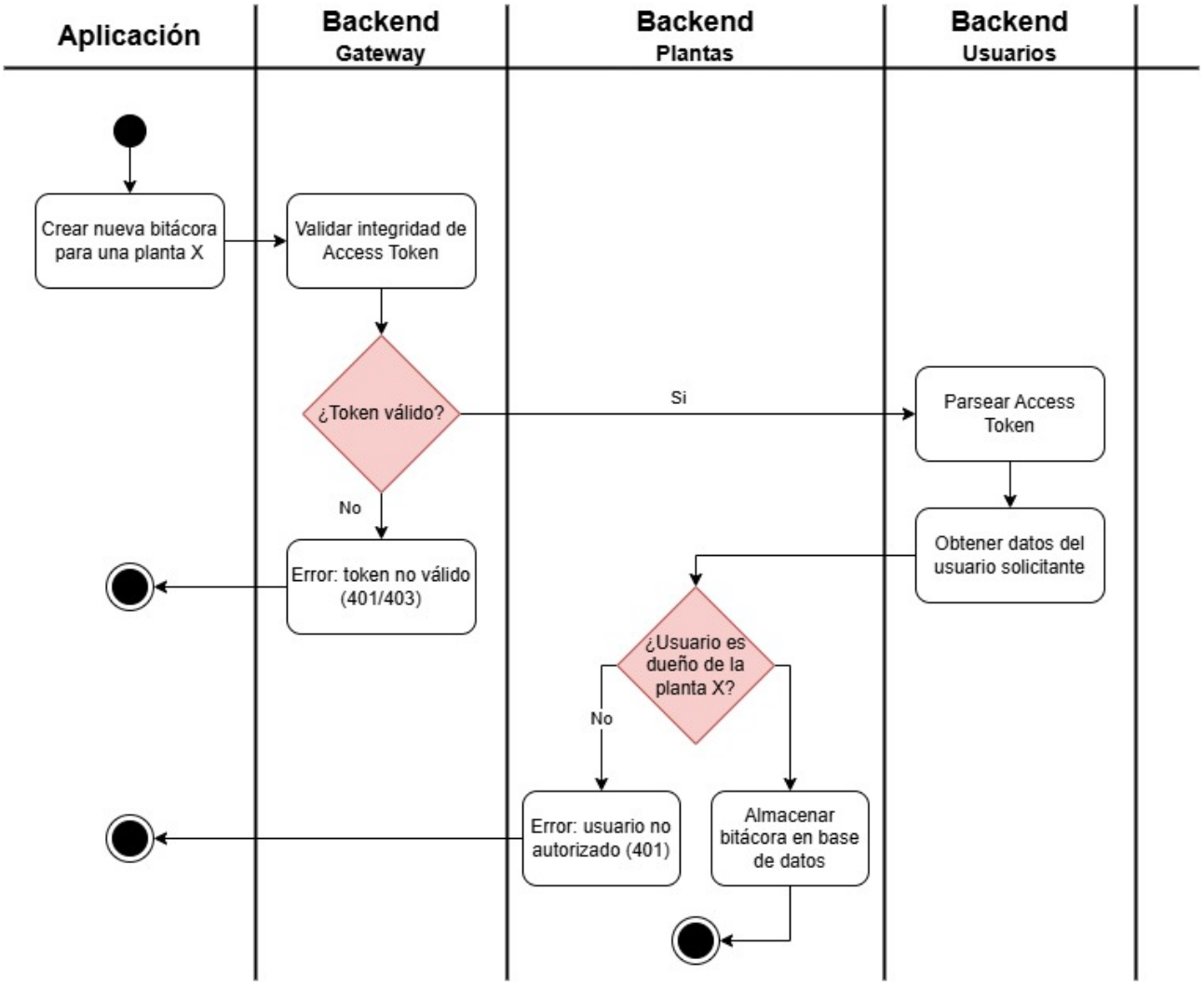

Figura 18: Creación de una bitácora.

9. Metodología aplicada

9.1. Metodología de trabajo

Durante la etapa de desarrollo se empleo un enfoque de desarrollo ágil como metodología de trabajo, la cual es resultado de una adaptación simplificada de la metodología Scrum. De acuerdo a esta metodología, los ciclos iterativos de trabajo fueron divididos en sprints cuya duración fue de 2 semanas (con excepciones según el contexto).

Al comienzo de cada sprint, se realizó una ceremonia de **Planificación**, con el objetivo de determinar las tareas a realizar durante esa iteración de trabajo. Previo al comienzo del *sprint*, uno de los integrantes dedicaba tiempo al estudio de los requerimientos y la preparación de los mismos para su debate con el equipo. De esta manera se lograba reducir los tiempos de planificación. Durante estas reuniones se realizaba un recorrido sobre las tareas registradas por el planificador, debatiendo tanto sus requerimientos como la importancia de la tarea para la iteración corriente. De esta manera, las tareas eran refinadas respecto de sus requerimientos o removidas de la planificación del sprint. Luego se realizaba una estimación del esfuerzo y tiempo que conllevaría la realización cada una de las tareas, para su posterior asignación a los integrantes del equipo, según sus preferencias o conocimiento sobre los requerimientos de la tarea.

Una vez por semana se realizaron ceremonias en formato *Weekly*, con el objetivo de poner en común el trabajo realizado durante la semana anterior, y avisar sobre qué se avanzará la semana actual. Además, se aprovechó dicho espacio para resolver aquellos problemas que pudiesen estar impidiendo el desarrollo de ciertas tareas, dada la presencia de todo el equipo.

A su vez, cada dos semanas se realizaron reuniones con el tutor del equipo para presentar avances y realizar demostraciones del funcionamiento del sistema.

9.2. Gestión de las tareas del proyecto

Para la gestión del backlog se utilizo Jira, una herramienta diseñada para la planificación de proyectos que permite la organización y seguimiento de tareas y proyectos [25]. Entre sus beneficios encontramos:

- La capacidad de organizar las tareas en PBI's (Product Backlog Increment), pudiendo atribuir a cada tarea, entre otras cosas, story points, categorías, prioridades, y dependencias con otras tareas.
- La articulación de cada tarea con una feature-branch en el repositorio del proyecto.
- La generación automática de gráficos de trabajo realizado, trabajo pendiente, velocidad de trabajo, entre otros.

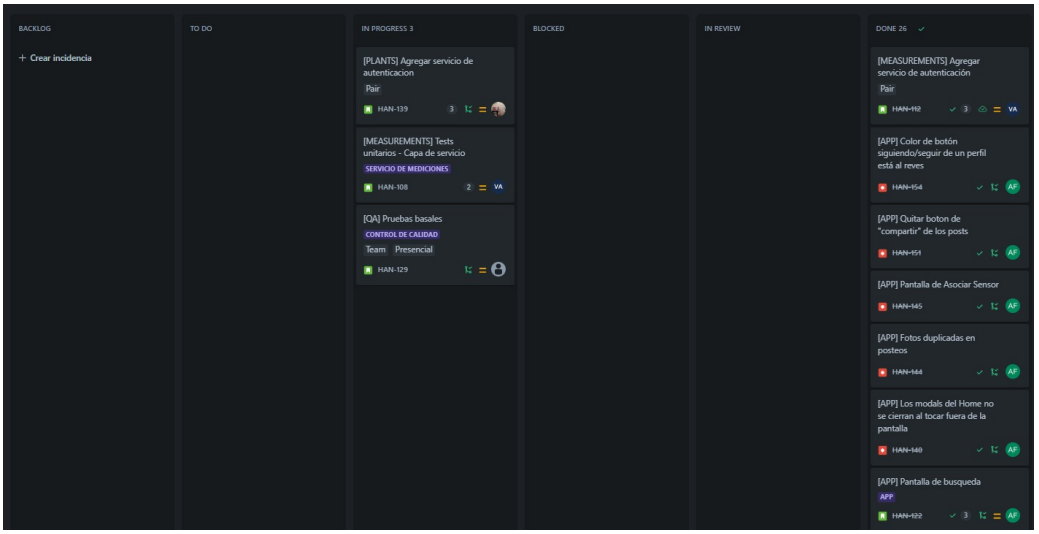

Figura 19: Tablero Kanban utilizado durante la gestión del trabajo.

Una vez planificadas las tareas para el corriente sprint, las información de cada tarea era bajada a incidencias en Jira, las cuales eran dispuestas en un tablero Kanban, listas para que cada integrante pudiese tomarlas. Aquellas tareas planificadas para el sprint corriente eran dispuestas en las columna *TO-DO*, y una vez comenzado su desarrollo, eran movidas a la columna *IN PROGRESS*. En ciertas ocasiones, algunas tareas no eran posibles de comenzar,

debido a dependencias con otras tareas o a imprevistos durante su desarrollo; en este caso, estas tareas eran movidas a la columna *BLOCKED*, y era el integrante asignado el encargado de hacer un seguimiento de sus dependencias para saber cuando desbloquearla. Una vez finalizada la tarea, esta era movida a la columna *IN REVIEW* donde se notificaba al resto del equipo de que el desarrollo había sido finalizado y que se requería una revisión del trabajo. Una vez validado el trabajo, este era movido a la columna DONE.

9.3. Gestión del desarrollo

Para la gestión del desarrollo del software se utilizo Github como herramienta de control de versiones, y se optó por utilizar la metodología de *feature-branching*, la cual maneja una rama principal que contiene el código efectivo del componente, junto con ramas individuales que almacenan el desarrollo individual de cada tarea. Siguiendo esta metodología para asegurar la integridad de la rama principal, se realizaban los siguientes pasos:

1. Al comenzar cada tarea, se abre una nueva rama partiendo de la principal, donde se trabajara en la tarea asignada. El nombre de esta rama será el numero de la incidencia asociado al ticket en Jira.

2. Una vez finalizada la tarea, el autor abre un *Pull Request*, donde se escribe una descripción de los cambios a incorporar. Para mantener un estándar en la descripción de los mismos, se estableció un template. Estos cambios deben ser aprobados por uno o dos integrantes del equipo.

3. Una vez aprobados los cambios, el autor realiza un merge de los cambios, junto con una ultima prueba para asegurar que la nueva funcionalidad incorporada funcionan acorde a lo descripto en sus requerimientos.

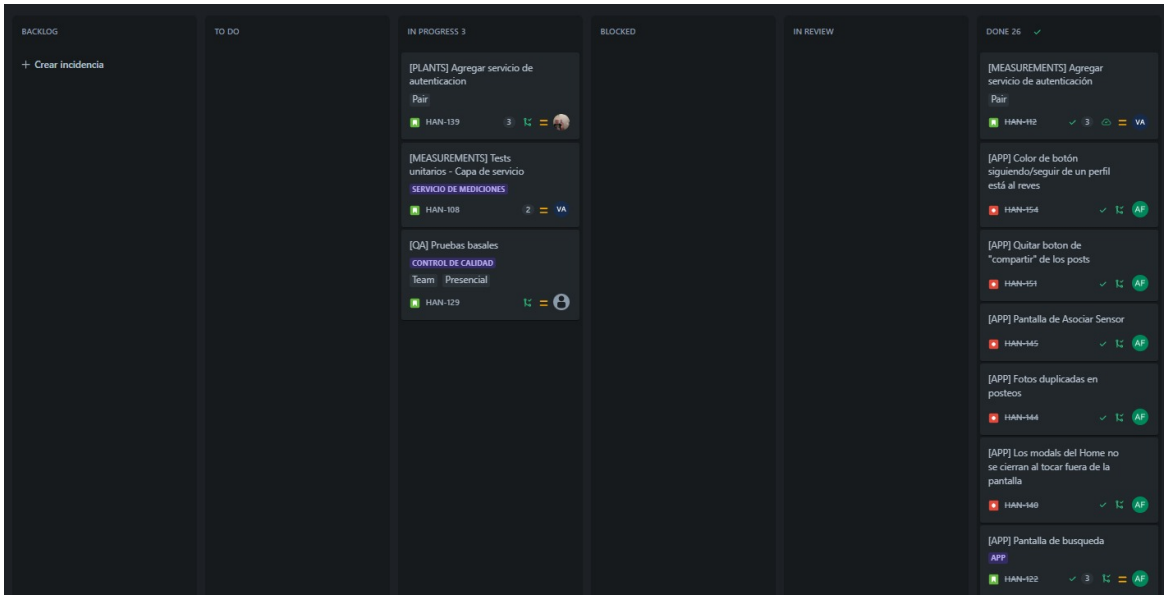

Figura 20: Template utilizado en los pull requests.

10. Experimentación y validación

10.1. Pruebas basales

Para validar el correcto funcionamiento de la aplicación y su integración con todos los microservicios desarrollados se llevaron a cabo pruebas basales. Estas pruebas consistieron en utilizar todas las funcionalidades de la aplicación tanto con el sensor virtual como con el prototipo, y validar que el comportamiento en todos los escenarios sea el esperado.

En el Anexo C se encuentran los links a los videos de las pruebas realizadas. Esto nos permitió confirmar el correcto funcionamiento de los casos de uso y así prepararnos para continuar con las pruebas de campo.

10.2. Pruebas de campo

Las pruebas de campo nos permitieron evaluar el funcionamiento de la aplicación con usuarios reales y en tiempo real. Para llevarlas a cabo se les fué entregado el prototipo de sensorización (el sensor y la placa ESP32) y la aplicación a diferentes usuarios. Estos usuarios utilizaron

la aplicación durante dos ó tres días seguidos, tiempo que les permitió familiarizarse con la misma y obtener asistencia en el cuidado de sus plantas.

Con estas pruebas se buscó validar el correcto funcionamiento de los casos de uso y requerimientos funcionales planteados anteriormente. Además, se buscó evaluar criterios no funcionales del sistema, como la usabilidad y simplicidad de la aplicación, y la robustez del sistema al exponerse a situaciones de uso reales. Por último, las pruebas nos permitieron recibir observaciones de parte de los usuarios relacionadas a las funcionalidades implementadas y problemas que surgieron durante las mismas. Aquellas observaciones brindadas por los usuarios pueden ser encontradas en el Anexo [15.4].

10.3. Análisis de los resultados

Los resultados de las pruebas demostraron que los usuarios pudieron desempeñarse satisfactoriamente al utilizar la aplicación. A lo largo de las pruebas, los usuarios pudieron validar exitosamente tanto aquellas funcionalidades con resultados inmediatos (ej: gestión de bitácoras, manejo de publicaciones de la red social), como aquellas cuyos procesos y resultados no eran inmediatos (ej: detección y corrección de factores ambientales fuera de rango para las plantas con sensor). Adicionalmente, se destacó la buena usabilidad de las interfaces de usuario, y la facilidad en la navegación de la aplicación. Todo este feedback puede encontrarse en el Anexo [15.4].

Durante las pruebas se reportaron problemas relacionados con el volumen de datos enviado por el prototipo de sensorización. Esto permitió observar que nuestro sistema almacenaba mediciones altamente similares en un rango de tiempo diminuto, generando redundancia en los datos. Además, se observo también como los servicios de terceros contratados (específicamente el motor de base de datos), no eran suficientes para las pruebas llevadas a cabo. El problema logró ser mitigado rápidamente, y el detalle del mismo se encuentra en [12.3]

Finalmente, las observaciones recolectadas por los usuarios nos permitieron identificar problemas que durante el período de desarrollo no pudieron ser identificados, que van desde detalles como errores de ortografía o de visualización en la interfaz, hasta problemas mas grandes, los

cuales atraviesan flujos basales del sistema, como incongruencias a la hora de notificar y exhibir desvíos en los factores ambientales, o reportes de una alta tasa de recepción de notificaciones en un período de tiempo corto. Adicionalmente, los usuarios también propusieron nuevas funcionalidades a incorporar en el sistema, como informar al usuario sobre cuáles son los valores óptimos para cada factor ambiental en cada tipo de planta y la creación de un tutorial inicial para comprender mejor el funcionamiento de la aplicación. Estas observaciones nos permitirán definir nuevos flujos de desarrollo a futuro, que permitirán tanto mejorar aspectos existentes como expandir la aplicación con nuevas características y funcionalidades.

11. Cronograma de actividades realizadas

Como se ha descrito anteriormente, el proyecto fue desarrollado a lo largo de 11 sprints de desarrollo, donde cada sprint mantuvo una duración de 2 semanas, salvo excepciones. El detalle de estas tareas pueden ser observadas en el Anexo A [15.1]. A continuación se presenta el cronograma de actividades realizadas, detallando la cantidad de horas dedicadas a diversos aspectos del sistema.

Total de horas dedicadas: 1350 hs.

11.1. Pre-desarrollo

- Estudio de dominio: 24 hs.
- Planificación de tareas: 88 hs.
- Investigación de tecnologías: 36 hs.
- Diseño preliminar de las vistas: 40 hs.

Total de horas dedicadas a la etapa de pre-desarrollo: 188 hs.

11.2. Desarrollo

- **Backend** (512 hs. dedicadas):

- Servicio de Mediciones (156 hs. dedicadas):
  - Recolección y validación de datos: 80 hs.
  - Integración con sensores: 24 hs.
  - Otras tareas: 52 hs.
- Servicio de Plantas (120 hs. dedicadas):
  - Gestión de plantas: 40 hs.
  - Gestión de bitácoras: 40 hs.
  - Otras tareas: 40 hs.
- Servicio de Usuarios (100 hs. dedicadas):
  - Autenticación: 28 hs.
  - Gestión de usuarios: 24 hs.
  - Gestión de notificaciones personalizadas: 20 hs.
  - Otras tareas: 28 hs.
- Servicio de Red Social (124 hs. dedicadas):
  - Gestión de perfil social: 44 hs.
  - Gestión de publicaciones: 64 hs.
  - Otras tareas: 16 hs.
- API Gateway (24 hs. dedicadas):
  - Creación de gateway: 16 hs.
  - Validación de session tokens: 8 hs.

- **Aplicación Móvil** (264 hs. dedicadas):
  - Configuración de herramientas: 16 hs.
  - Registro e inicio de sesión: 16 hs.
  - Sección Home: 28 hs.

  - Sección Bitácoras: 28 hs.
  - Sección Configuración: 24 hs.
  - Sección Red Social: 60 hs.
  - Sección Alarmas: 12 hs.
  - Interacción humano - planta: 28 hs.
  - Arreglos varios: 12 hs.
  - Otras tareas: 40 hs.
- **Sensores** (68 hs. dedicadas):
  - Implementación de simulador: 36 hs.
  - Integración de sensores con placa ESP32: 16 hs.
  - Integración de cliente MQTT con placa ESP32: 16 hs.
- **Otras tareas**:
  - Generación de datasets de los datos recolectados: 8 hs.
  - Configuración de DBMS: 4 hs.
  - Deployment de microservicios en servidores cloud: 12 hs.
  - Agregado de retries en microservicios: 4 hs.

Total de horas dedicadas a la etapa de desarrollo: 884 hs.

11.3. Post-desarrollo

- Pruebas y ajustes: 70 hs.
- Confección del informe final: 80 hs.
- Confección del artículo académico: 32 hs.
- Presentación: 96 hs.

Total de horas dedicadas a la etapa de post-desarrollo: 278 hs.

12. Riesgos materializados y Lecciones aprendidas

12.1. Notificaciones personalizadas

12.1.1. Contexto

Una de las funcionalidades a desarrollar, presentada en el anteproyecto, fue la implementación de un sistema de recordatorios personalizados. Este sistema permitiría al usuario configurar recordatorios de cuidados relacionados a sus plantas.

12.1.2. Riesgo

No poder llevar a cabo la realización de la tarea, debido a una alta complejidad de implementación que demandaría mucho mas tiempo del estimado para la misma.

12.1.3. Materialización

Luego de comenzar el sexto sprint, el equipo comenzó a realizar investigaciones relacionadas al tema. Luego de dos sprints (un mes) de investigación, las mismas concluyeron en que la única solución viable era la implementación de un nuevo microservicio dedicado únicamente a la gestión de recordatorios, cuya estimación de esfuerzo era demasiado alta como para llevarla a cabo. Por otra parte, se encontró que Heroku ofrecía un servicio pago de cola de mensajería basado en Redis que nos permitiría programar la ejecución de tareas para que un worker ejecute el envió de notificaciones.

12.1.4. Plan de respuesta

Se tomo la decisión de contratar el servicio ofrecido por Heroku, lo cual nos proporcionó una solución rápida y eficiente frente a la alternativa de implementar un nuevo microservicio.

12.1.5. Lecciones aprendidas

El equipo debió realizar una investigación previa antes de planificar la realización del sistema de recordatorios, para comenzar su desarrollo con un mayor conocimiento de herramientas y técnicas

12.2. Hardware de sensorización para plantas

12.2.1. Contexto

El sensor Xiaomi adquirido para la lectura de mediciones posee la limitación de la conexión que solo funciona a través de Bluetooth. Para solventar, se decidió comprar un microcontrolador ESP32 que funcionase como intermediaria entre el sensor y nuestro sistema de mensajería del backend, formateando las lecturas en paquetes y añadiendo lógica necesaria para su envío al broker mediante la red WiFi.

12.2.2. Riesgo

La lógica del sensor fue implementada bajo archivos de configuración gracias a un software de terceros. Debido a esta lógica de alto nivel, existía el riesgo de una limitación en el manejo de sus módulos e incluso con los recursos de capacidad y cómputo de la ESP32.

12.2.3. Materialización

Al integrar el sensor con el ESP32, ocurrió que la placa no puede formar mensajes con las mediciones de todos los factores ambientales, sino que envía mensajes con la medición de un solo factor, provocando que el servicio de mediciones solo reciba paquetes parciales. Por otra parte, la placa envía las mediciones al broker sin tener en cuenta si algún factor fue significativo respecto de la última medición enviada, llegando a generar una ráfaga de mensajes en el broker con mediciones redundantes.

12.2.4. Plan de respuesta

Se debió implementar lógica adicional en el worker consumidor del microservicio de mediciones para poder armar los paquetes completos con todas las mediciones capturadas por cada sensor de cada usuario. También se tuvo que agregar lógica especial para controlar la ráfaga de mediciones recibidas, evitando que el usuario final reciba notificaciones redundantes de las mediciones de su planta.

12.2.5. Lecciones aprendidas

Implementar la misma funcionalidad, pero considerando la alternativa de manipular el microcontrolador ESP32 con un software propio, buscando controlar los parámetros de cada mensaje enviado a nuestro sistema de mensajería. También se podría haber considerado utilizar otra placa que actúe como intermediario entre el sensor y el sistema de mensajería del backend en el que tengamos mayor dominio.

12.3. Despliegue del sistema backend

12.3.1. Contexto

Se tuvo la necesidad de desplegar los microservicios con sus workers y base de datos en una plataforma Cloud para que la aplicación móvil pueda interactuar con estos servicios de forma remota. Recordando en [8.6.1], nos inclinamos por el plan eco académico de Heroku para el despliegue de los microservicios junto a sus workers. También se utilizó la base de datos de PostgreSQL en Heroku con único motor para todos los microservicios donde cada uno tenía su propio esquema; salvo para el microservicio de red social que utilizó una base de datos gratuita en MongoDB Atlas.

12.3.2. Riesgo

Encontrar limitaciones de recursos con los planes eco de Heroku o MongoDB Atlas gratuito al momento de utilizar los microservicios en producción. Además que el uso de una única base

de datos para 4 microservicios podría saturar los recursos del mismo.

12.3.3. Materialización

Durante el periodo de pruebas de campos con usuarios reales, se encontraron problemas de saturación en los servidores, afectando la performance del sistema, terminando de impactar en la usabilidad de los usuarios. La aplicación móvil solía emitir errores al inicio de la aplicación hasta que los microservicios se "despertabanz comenzaban a funcionar correctamente. Además hasta que el microservicio no se encontraba activo, los workers de los mismos se encontraban apagados imposibilitando la lectura de nuevas mediciones a tiempo real o en el envío de notificaciones para el sistema de recordatorios. Por último, al compartir el sensor a diferentes usuarios, la base de datos de PostgreSQL estuvo al punto de llegar a los límites de recursos en cuanto a la cantidad de filas máximas que podía almacenar.

12.3.4. Plan de respuesta

Se decidió actualizar los planes de las instancias de Heroku incrementando el costo de recursos. También se mejoró el plan de la base de datos de PostgreSQL en Heroku para poder almacenar una cantidad considerable de mediciones. Por último se decidió que el worker del sistema de recordatorios emita ticks cada 5 minutos en lugar de cada segundo para evitar que tenga una carga de trabajo excesiva.

12.3.5. Lecciones aprendidas

A pesar de que los microservicios que utilizan PostgreSQL interpreten que tienen su propia base de datos, en realidad todos comparten la misma base de datos pero con diferentes esquemas en el que ningún otro microservicio podría acceder a los datos de otro. Sin embargo, un único motor de base de datos para todos los microservicios no escalaría a mayor cantidad de usuarios, cada microservicio debería tener la propia para que la base de datos no sea el cuello de botella. También se aprendió que los planes iniciales de Heroku no son suficientes para un despliegue en producción a mayor escala, pero para un despliegue en pruebas o para un grupo reducido de

usuarios es suficiente. Si se quisiera escalar a mayor cantidad de usuarios, se debería considerar un plan superior de Heroku o buscar otra plataforma de despliegue que ofrezca más recursos que incluya hasta la posibilidad de escalar horizontalmente.

13. Conclusiones

La domótica es un campo de estudio floreciente, el cual tiene un gran potencial de desarrollo cuando se habla de la automatización del hogar. Por su parte, la aplicación de la tecnología en el cuidado de plantas es un campo naciente que permite asistir a los usuarios en materia de floricultura. Nuestra cotidianidad se encuentra atravesada por la tecnología, y es nuestro deber como futuros ingenieros en informática tanto innovar en soluciones a problemáticas actuales como facilitar aún más el acceso a estas tecnologías.

Durante el proyecto se abarcaron de forma transversal diversos tópicos estudiados en la carrera, como redes, multicomputing y virtualización, manejo de bases de datos, gestión de proyectos, análisis de datos, e ingeniería de software. A su vez, la carrera nos permitió desarrollar habilidades para estudiar y aplicar nuevos conceptos y herramientas relacionadas a IoT e integración de servicios de terceros; además de habilidades para la resolución de conflictos que fueron surgiendo a lo largo del proceso de trabajo, permitiéndonos desarrollar soluciones creativas que respeten los estándares de calidad aprendidos.

Las pruebas de campo nos permitieron adquirir una perspectiva mucho más amplia sobre el dominio del negocio planificado inicialmente. La ejecución de estas pruebas nos brindó feedback valioso respecto a qué es lo que los usuarios esperaban al usar nuestro sistema, permitiéndonos observar funcionalidades de gran valor que, como equipo, no habíamos tenido en cuenta durante del desarrollo del mismo.

Encontrándonos en el último tramo del proyecto, destacamos la importancia de la inclusión de los usuarios en etapas previas, y la planificación de un buen MVP que permita corregir el curso del trabajo en base a las devoluciones de las experiencias de los usuarios. Si bien las pruebas realizadas fueron satisfactorias, alcanzando los resultados esperados, se detectaron muchas posibilidades de mejora en diversos aspectos, tanto técnicos como de negocio.

Todo esto permite la apertura de múltiples lineas de trabajo e investigación a futuro, las cuales podrán incorporar tanto nuevas funcionalidades a desarrollar como mejoras en aquellas existentes, en respuesta de las necesidades que surjan del mercado. Este proyecto permite continuar con su desarrollo en áreas diversas, como la aplicación de conceptos de computación afectiva para el procesamiento de emociones del Hanagotchi, la incorporación de modelos de Machine Learning para una mejor precisión en el cuidado de plantas, y el desarrollo del sensor, en materia de hardware, para la incorporación de mediciones de otros factores ambientales, entre otras cosas.

En resumen, el Trabajo Profesional ha sido una oportunidad para poner a prueba todos los conocimientos adquiridos a lo largo de nuestra trayectoria académica. La posibilidad de autogestionar un proyecto propio desde su concepción hasta su materialización ha resultado en un aprendizaje continuo, que no hace mas que enriquecernos como futuros profesionales, creando experiencias y permitiéndonos explorar nuevos campos de estudio.

14. Desarrollos futuros

14.1. Futuras líneas de investigación

14.1.1. Investigaciones en materia de botánica

Actualmente nuestro sistema almacena un historial de las mediciones realizadas en cada planta registrada bajo un sensor, junto con las fotos que los usuarios realizan en cada bitácora. Con el objetivo de poder proporcionar estos datos a terceros para futuras investigaciones, se ha preparado un script que permite exportar estos datos.

Una propuesta que el equipo realiza es la utilización de estos datos para validar y/o refinar los resultados de las investigaciones realizadas en la publicación *Growing Indoor Plants with Success*, en el cual se basó el servicio de mediciones de nuestro sistema. La posición de los datasets permite incluso el entrenamiento de modelos de Machine Learning que puedan automatizar la estimación de factores ambientales a través del procesamiento de imágenes.

14.1.2. Interacción con el Hanagotchi a través de la detección de emociones por voz

La presencia de la Inteligencia Artificial en nuestra cotidianeidad es cada vez mas marcada. Durante el desarrollo, se planteo la idea de incorporar una I.A. que permita mejorar la experiencia de usuario a través de una mejora en la interacción con los hanagotchis. Está I.A. permitiría la inferencia de emociones a través del procesamiento de la voz. De esta manera, un usuario podría hablarle a su Hanagotchi, expresando como se siente respecto a su estado. Por su parte, el Hanagotchi, entendiendo las emociones del usuario, sería capaz de brindar una reacción acorde a ello.

14.1.3. Reconocimiento por imágenes para planta

Se plantea la idea de que por medio de reconocimiento de imágenes que se capturen a la planta, y por medio de Inteligencia Artificial se puedan detectar diferentes tipos de enfermedades. Por ejemplo, detectar a través de fotografía si planta cuenta con alguna bacteria, esta seca, o si esta condiciones normales.

14.1.4. Integración del hardware de sensorización a través de la aplicación móvil

Actualmente el emparejamiento del sensor a la aplicación móvil [8.7.4] es muy poco amigable para el usuario final. Se propone como alternativa de investigación buscar una forma para que desde la aplicación desarrollada se pueda reconocer al microcontrolador ESP32 directamente para configurar la red WiFi al que se conectará, sin pasar por la red hotspot WiFi que el mismo se genera. Incluso pudiendo generar un primer emparejamiento con el microcontrolador mediante Bluetooth para que se integre desde la aplicación y se pueda indicar a qué WiFi debe conectarse el ESP32.

14.1.5. Añadir otros factores de mediciones al sistema

Se propone sumar, a la lista de Luz, Humedad, Temperatura y Riego, otros tipos de factores ambientales. Estos permitirían enriquecer la recolección de datos para mejorar su análisis y

poder extraer nuevas inferencias en base a ellos. Como ejemplo, se propone añadir lecturas de:

- Fertilidad de la tierra.
- Orientación de impacto de la luz.

14.2. Futuras líneas de trabajo

14.2.1. Funcionalidades adicionales y arreglos

Durante la ejecución de las pruebas de campo se recolectaron observaciones realizadas por los usuarios, que involucran tanto problemas identificados como propuestas de funcionalidades nuevas. A continuación, se listan aquellas que consideramos mas relevantes como futura linea de trabajo:

1. Exhibir los valores óptimos para cada factor ambiental cuando se notifique sobre un desvío en ellos.
2. Crear un acceso directo para añadir plantas y sensores desde la pantalla principal.
3. Facilitar la identificación de si una planta tiene asignado un sensor o no.
4. Agregar un tutorial para los nuevos usuarios.
5. Reducir la redundancia en las mediciones almacenadas.
6. Reducir la tasa de notificaciones enviadas por periodo de tiempo.
7. Permitir que un Hanagotchi pueda ser compartido por mas de un usuario.
8. Añadir mas información sobre cuidados de plantas para cada tipo.

14.2.2. Desarrollo de la aplicación mobile para dispositivos iOS

La carencia de dispositivos de sistema operativos MacOS/iOS para el desarrollo y prueba de la aplicación mobile impidió el desarrollo de la aplicación para dichos sistemas operativos. En

consecuencia, el equipo concentro entonces su desarrollo en el funcionamiento de la aplicación para dispositivos Android. Sin embargo, dado que la aplicación se encuentra desarrollada utilizando React Native y Expo Application Service, la migración de la aplicación a dispositivos iOS demandaría un bajo esfuerzo, a la vez que permitiría ampliar el alcance de usuarios de nuestro sistema.

14.2.3. Opción de adaptar componentes propios

Como se mencionó anteriormente, el desarrollo del trabajo se centro en el software y por eso hay mucho trabajo a futuro que se puede hacer en lo que al hardware refiere. Hoy día, el producto brindado es el conjunto de aplicación, con sensor y placa viniendo la placa ya flasheada con lo necesario. Se busca en un futuro poder brindar el producto parcial de modo tal que se pueda acomodar a las necesidades del usuario.

Por ejemplo, si un usuario ya cuenta con una placa ESP32 propia, se busca poder brindarle únicamente la aplicación y el sensor y que pueda usar su placa. Se asume que una persona con una placa tiene un mínimo conocimiento informático, tan mínimo como ser capaz de correr un comando en una terminal. De este modo, se le proporcionará al usuario con instrucciones sobre como flashear su propia placa ESP32 con lo necesario para el funcionamiento de la aplicación.

Similarmente, se busca poder brindar el producto que el usuario necesite, es decir los otros dos casos en los que el usuario cuente con un sensor pero no una placa o que cuente con ambos.

14.2.4. Integración de otro tipos de sensores al microcontrolador ESP32

A partir del software de terceros ESP Home [38] utilizado para el microcontrolador ESP32, es posible agregar una gran variedad de otros dispositivos hardware [45] que son compatibles con el software para lograr medir desde otro sensor alguno de los 4 parámetros requeridos en el sistema. Sabiendo que nuestro sensor Xiaomi no media la humedad del ambiente, por ejemplo este parámetro podría ser calculado incorporando el sensor de aire SCD4X [46] al ESP32 que es compatible con ESP Home.

14.2.5. Presentación de estadísticas sobre las lecturas de las plantas

Se propone la implementación de una sección de estadísticas, donde el usuario pueda visualizar mediante gráficos la evolución de los diversos factores medidos de aquellos Hanagotchis asociados a un sensor. De esta manera, el mismo usuario tendrá una mejor comprensión del estado y cuidado de su planta a través de gráficos que muestren la evolución de cada uno de los factores medidos.

Referencias

[1] Domótica (3 de Agosto de 2023). En Wikipedia. https://es.wikipedia.org/wiki/Dom%C3%B3tica

[2] Escuela y posgrado de Ingeniería y Arquitectura (11 de Agosto de 2020). "¿Qué es un jardín inteligente? Escuela y posgrado de Ingeniería y Arquitectura" . https://postgradoingenieria.com/que-es-jardin-inteligente/

[3] Yoolax Smart Plant Pot (2023). https://www.yoolax.com/products/yoolax-smart-plant-pot

[4] Yoolax Home (2023). https://www.yoolax.com/pages/work-with-yoolax-home

[5] Beijing HHCC Plant Technology Co., Ltd (2023, 22 de julio), Flower Care [Aplicación móvil]. https://play.google.com/store/apps/details?id=com.huahuacaocao.flowercare&hl=es&gl=US

[6] Bodie V. Pennisi (2009). "Growing Indoor Plants with Success" . https://secure.caes.uga.edu/extension/publications/files/pdf/B%201318_6.PDF

[7] Planterina (2023). "A Beginner's Guide to Calathea Care | All you Need to Grow!" https://planterina.com/blogs/plant-care/calathea-care

[8] About Planta app (2023). https://getplanta.com/about

[9] PlantNet (2023). https://identify.plantnet.org/es

[10] Blossom [Aplicación móvil]. (24 de octubre de 2023). https://apps.apple.com/us/app/blossom-plant-care-guide/id1487453649?mt=8

[11] Bloomscape (2023). Vera [Aplicación móvil]. https://bloomscape.com/vera/

[12] Bonnie L. Grant (2 de Noviembre de 2020). Guía de jardinería inteligente: aprenda sobre jardinería con tecnología. DiverseGarden. https://www.diversegarden.com/garden-how-to/info/gardening-with-technology.htm

[13] Huahuacaocao (2023, 18 de Octubre), Flower Care™ Smart Monitor. Huahuacaocao. http://www.huahuacaocao.com/product

[14] Otitelecom (2017, 18 de Diciembre). "En el mundo hay más celulares que humanos" . https://otitelecom.org/telecomunicaciones/mundo-mas-celulares-humanos/

[15] Bodie V. Pennisi (Julio de 2022). "Growing indoor plants with success" , Georgia Cooperative Extension. https://secure.caes.uga.edu/extension/publications/files/pdf/B%201318_6.PDF

[16] Tamagotchi (1996). https://tamagotchi.com/

[17] Typescript https://www.typescriptlang.org/

[18] React Native https://reactnative.dev/

[19] Node package manager https://www.npmjs.com/

[20] *Expo Go*. Expo https://expo.dev/go

[21] Expo https://expo.dev/

[22] OpenWeather API https://openweathermap.org/api

[23] Firebase https://firebase.google.com/?hl=es

[24] Expo Application Services (Julio de 2024), "Expo Application Services" . https://docs.expo.dev/eas/

[25] "Bienvenido a Jira" , Atlassian Jira https://www.atlassian.com/es/software/jira/guides/getting-started/introduction#what-is-jira-software

[26] MQTT https://mqtt.org

[27] RabbitMQ https://www.rabbitmq.com

[28] Redis Queue [Redis Glossary] https://redis.io/glossary/redis-queue

[29] Heroku https://heroku.com

[30] *Usage and Billing*. Heroku Dev Center. https://devcenter.heroku.com/articles/usage-and-billing#dyno-usage-and-costs

[31] *Heroku Postgres*. Heroku Dev Center. https://devcenter.heroku.com/articles/heroku-postgresql

[32] *Cloud AMQP*. Heroku. https://elements.heroku.com/addons/cloudamqp

[33] *Heroku is for students*. Heroku. https://www.heroku.com/students

[34] MongoDB https://www.mongodb.com/

[35] *MongoDB Atlas*. MongoDB. https://www.mongodb.com/es/atlas

[36] PostgreSQL https://postgresql.org/

[37] *Firebase Cloud Messaging*. Firebase (2022, 25 de Agosto). https://firebase.google.com/docs/cloud-messaging?hl=es-419

[38] ESPHome https://esphome.io/

[39] *Flower Care™ Smart Monitor*. Huahuacaocao. http://www.huahuacaocao.com/product

[40] *Xiaomi Mijia BLE Sensors - HHCCJYC01*. ESPHome. https://esphome.io/components/sensor/xiaomi_ble#hhccjcy01

[41] *ESP32 Bluetooth Low Energy Tracker Hub*. ESPHome. https://esphome.io/components/esp32_ble_tracker.html

[42] *MQTT Client Component*. ESPHome. https://esphome.io/components/mqtt.html

[43] *WiFi Component*. ESPHome. https://esphome.io/components/wifi.html

[44] *Captive Portal*. ESPHome. https://esphome.io/components/captive_portal

[45] *ESPHome Components*. ESPHome. https://esphome.io/index.html#esphome-components

[46] *SCD4X CO2, Temperature and Relative Humidity Sensor*. ESPHome. https://esphome.io/components/sensor/scd4x

15. Anexos

15.1. Anexo A - Lista de tareas

Link a documento de las tareas del proyecto:

https://docs.google.com/spreadsheets/d/1lRuUwjMkowdw3APNGpE4Eg_GkW3z8tOrBPY1FchpmGo/edit?gid=242542650#gid=242542650

15.2. Anexo B - Posibles emociones de los Hanagotchis

Hanagotchi molesto: adopta esta emoción cuando se mide un bajo porcentaje de humedad en el ambiente.

Figura 21: Hanagotchi molesto.

Hanagotchi deprimido: adopta esta emoción cuando se mide un bajo porcentaje de humedad en la tierra.

Figura 22: Hanagotchi deprimido.

Hanagotchi regado de más: adopta esta emoción cuando se mide un alto porcentaje de humedad en la tierra.

Figura 23: Hanagotchi con mucha agua.

Hanagotchi feliz: adopta esta emoción cuando recibe comentarios positivos por parte del usuario.

Figura 24: Hanagotchi feliz.

Hanagotchi abrumado: adopta esta emoción cuando se mide una alta temperatura o un alto nivel de luz.

Figura 25: Hanagotchi abrumado.

Hanagotchi abrumado: adopta esta emoción cuando se encuentra en condiciones ambientales ideales.

Figura 26: Hanagotchi relajado.

Hanagotchi triste: adopta esta emoción cuando recibe comentarios negativos por parte del usuario.

Figura 27: Hanagotchi triste.

Hanagotchi riendo: adopta este comportamiento cuando el usuario le hace cosquillas.

Figura 28: Hanagotchi riendo.

Hanagotchi incómodo: adopta esta emoción cuando se mide una baja temperatura o un bajo nivel de luz. También se adopta cuando se encuentra previamente triste, y el usuario intenta realizarle cosquillas.

Figura 29: Hanagotchi incómodo.

15.3. Anexo C - Pruebas basales

15.3.1. Login

Se prueba el caso de uso en el cual el usuario se registra a la aplicación por primera vez.

Link al vídeo de la prueba: https://drive.google.com/file/d/1ArmSZ82kuHImuF1WoLyfQuW2zfqPGM_5/view?usp=sharing

15.3.2. Editar mi perfil

Se prueba el caso de uso en el cual el usuario ya registrado edita los datos de su perfil.

Link al vídeo de la prueba: https://drive.google.com/file/d/1ArSbiCZwDQWV4gbqORHID81KJAq_LAfO/view?usp=sharing

15.3.3. Agregar una planta

Se prueba el caso de uso en el cual el usuario crea una planta nueva.

Link al vídeo de la prueba: https://drive.google.com/file/d/1AG-S5SPWyR4OOsDq9pSI87Dgu5Vt_IEi/view?usp=sharing

15.3.4. Ver el estado de mis plantas

Se prueba el caso de uso en el cual el usuario revisa el estado actual de sus plantas creadas y se encuentra con que una de ellas tiene una exposición solar muy alta.

Link al vídeo de la prueba: https://drive.google.com/file/d/1A7_zNjBlh9NqaLNLMlJhF9xKDUvmW-vx/view?usp=sharing

15.3.5. Dar feedback a la planta

Se prueba el caso de uso en el cual la aplicación le pide al usuario feedback respecto de cómo ve a la planta.

Link al vídeo de la prueba: https://drive.google.com/file/d/1Ahl8o2YLqevkxFyRtCnqCMC-apb6vBfO/view?usp=sharing

15.3.6. Eliminar una planta

Se prueba el caso de uso en el cual el usuario elimina una de sus plantas creadas.

Link al vídeo de la prueba: https://drive.google.com/file/d/1ALbo7LHnjZc5Kn0U5LWt2tEUevSc3wx8/view?usp=sharing

15.3.7. Agregar una bitácora

Se prueba el caso de uso en el cual el usuario redacta una bitácora para una de sus plantas y le agrega imágenes.

Link al vídeo de la prueba: https://drive.google.com/file/d/1AXg48H1jZ0Osyibx8PN7IBpCfuSnme8z/view?usp=sharing

15.3.8. Editar una bitácora

Se prueba el caso de uso en el cual el usuario edita una de sus bitácoras ya creadas.

Link al vídeo de la prueba: https://drive.google.com/file/d/1AXqDKzE4eTDRG8vzdYsyJkV-6Dswe3Kf/view?usp=sharing

15.3.9. Ver mi usuario de red social

Se prueba el caso de uso en el cual el usuario ve su perfil de red social, lo modifica, ve sus propios posteos y agrega uno nuevo.

Link al vídeo de la prueba: https://drive.google.com/file/d/1C2rMMrHHyzMkmYsGGUMCzshClsZvughF/view?usp=sharing

15.3.10. Seguir a un usuario

Se prueba el caso de uso en el cual el usuario a través del buscador encuentra a un usuario y comienza a seguirlo.

Link al vídeo de la prueba: https://drive.google.com/file/d/1BzPuDRfGRYMxlmXAB6S4uKPKanYrv77u/view?usp=sharing

15.3.11. Búsqueda por tags

Se prueba el caso de uso en el cual el usuario realiza una búsqueda por tags y observa todos los posteos presentes que contienen ese tag.

Link al vídeo de la prueba: https://drive.google.com/file/d/1Bx0La9BM3ZTFUGXfzSrqwb5BQM815auE/view?usp=sharing

15.3.12. Crear un recordatorio

Se prueba el caso de uso en el cual el usuario crea un recordatorio para dentro de pocos minutos, y luego recibe la notificación push del mismo.

Link al vídeo de la prueba: https://drive.google.com/file/d/18SxAmig2rqTHtpnX8UVAZXfKhqyxU3tC/view?usp=sharing

15.3.13. Eliminar un recordatorio

Se prueba el caso de uso en el cual el usuario elimina un recordatorio previamente creado.

Link al vídeo de la prueba: https://drive.google.com/file/d/1AXDGRdJXSK0oZ8sLv9EBsa8ozOCX_WgG/view?usp=sharing

15.3.14. Editar un recordatorio

Se prueba el caso de uso en el cual el usuario edita un recordatorio previamente creado.

Link al vídeo de la prueba: https://drive.google.com/file/d/1ASTjHtXWxb37JDZ1Zov1AVfebJm-KsId/view?usp=sharing

15.3.15. Aumento de temperatura con hardware de sensorización

La prueba consistió en aumentarle la temperatura intencionalmente a la planta que está siendo monitoreada por el sensor para validar que al cabo de unos segundos se registra ese cambio de temperatura en la aplicación.

Link al vídeo de la prueba: https://drive.google.com/file/d/1MrYvCVJqsDUE0-3zsA_mcil2qfhy_DQ0/view

15.3.16. Aumento de luz con hardware de sensorización

La prueba consistió en incrementarle la exposición a la luz intencionalmente a la planta que está siendo monitoreada por el sensor para validar que al cabo de unos minutos se registra ese cambio de cantidad de luz recibida en la aplicación.

Link al vídeo de la prueba: https://drive.google.com/file/d/17vBdryEifFNgxXn5PlapZosf1N1ypxVr/view

15.3.17. Aumento de agua con hardware de sensorización

La prueba consistió en regar la planta que está siendo monitoreada por el sensor para validar que al cabo de unos segundos se registra ese cambio en la humedad de la tierra en la aplicación.

Link al vídeo de la prueba: https://drive.google.com/file/d/1eKjg6AF2XdKTlQBYqEz5kUTDkb6lFuHI/view

15.4. Anexo D - Pruebas de campo

Link al drive con las pruebas de campo:

https://drive.google.com/drive/u/1/folders/1IgIXTlqfnwGfBew0MMNeUcsoKyReFy0Y

15.5. Anexo E - Repositorios de código fuente

15.5.1. Aplicación móvil

Link al repositorio de GitHub:

https://github.com/Hanagotchi/app-mobile

15.5.2. Microservicio de API Gateway

Link al repositorio de GitHub:

https://github.com/Hanagotchi/api-gateway

15.5.3. Microservicio de Mediciones

Link al repositorio de GitHub:

https://github.com/Hanagotchi/measurements

15.5.4. Microservicio de Plantas

Link al repositorio de GitHub:

https://github.com/Hanagotchi/plants

15.5.5. Microservicio de Usuarios

Link al repositorio de GitHub:

https://github.com/Hanagotchi/users

15.5.6. Microservicio de Red Social

Link al repositorio de GitHub:

https://github.com/Hanagotchi/social

15.5.7. Sensor virtual (Simulador)

Link al repositorio de GitHub:

https://github.com/Hanagotchi/simulator

15.5.8. Generación de datasets

Link al repositorio de GitHub:

https://github.com/Hanagotchi/dataset-generation-process

15.6. Anexo F - Sistema de mensajería

15.6.1. Protocolo simple de los mensajes

En la Figura 30 se observa un ejemplo del protocolo simple que deben cumplir los mensajes del sistema mensajería. El nivel de humedad (`humidity`) y el nivel de riego (`watering`) son valores enteros porcentuales que varían entre 0 a 100; la luz (`light`) es un valor flotante en unidad de foot-candle (ft-c) y por último la temperatura (`temperature`) que también es valor flotante en unidades de grado Celsius (°C). El identificador del sensor (`id_device`) es un valor alfanumérico que identifica al sensor que envía el mensaje al broker de RabbitMQ. Por último, el tiempo (`time_stamp`) es la fecha y hora en la que se registró la medición.

```
{
    "temperature": 25.0,
    "humidity": 37,
    "light": 547.1,
    "watering": 55,
    "id_device": "sensor_1",
    "time_stamp": "2024-07-15T00:00:00-03:00"
}
```

Figura 30: Protocolo simple que deben cumplir los mensajes del sistema de mensajeria

15.7. Anexo G - Sensor

15.7.1. Archivo de configuracion ESP32

Link al YAML:

https://drive.google.com/file/d/1wcajEtUGNscmM1Ar_uPdQiFlUIjfi7TC/view?usp=drive_link

15.7.2. Especificaciones técnicas placa ESP32

Link a especificaciones técnicas:

https://tienda.starware.com.ar/producto/placa-desarrollo-espressif-esp32-ch9102x-dual-core-wifi-bluetooth/

15.8. Anexo H - Base de datos

15.8.1. Base de datos relacional

En la Figura 31 se puede observar el modelo con las dos tablas que conforman el esquema de la base de datos relacional (PostgreSQL) del microservicio de usuarios. La primera tabla es la de usuarios (**users**) que representa a los usuarios de la aplicación. Cada usuario tiene un identificador único (`id`) siendo este numérico y auto-incremental como clave primaria. Se almacena el nombre del usuario (`name`), el email (`email`), el género (`gender`) y la foto (`photo`) como valores de tipo `string`, siendo el mail un valor único y no nulo que debe cumplir con el formato de mail. Se guarda la fecha de nacimiento (`birthdate`) como un valor de tipo `Date`, la ubicación (`location`) con tipo `JSONB` que debe contener las coordenadas de latitud y longitud del usuario. También se almacena el apodo (`nickname`) y la biografía (`biography`) como valores de tipo `string`, pero el apodo debe ser único. Por último, se guarda el token del dispositivo móvil del usuario (`device_token`) como un valor de tipo `string` que sirve para enviarle notificaciones push.

La segunda tabla es la de alarmas que representan los **recordatorios** que un usuario puede crear en la aplicación. Cada recordatorio tiene un identificador único (`id`) siendo también numérico y auto-incremental como clave primaria; también se almacena el identificador del usuario (`id_user`) que creó el recordatorio como un valor entero que será una clave foránea que referencia al identificador de la tabla de usuarios. Se guarda la fecha y hora (`date_time`) como valor de tipo `timestamp` para notificar junto al contenido del recordatorio (`content`) como `string`. En base a esta relación, vemos que un usuario podría tener 0 o N alarmas, y una alarma siempre estará asociada a un usuario.

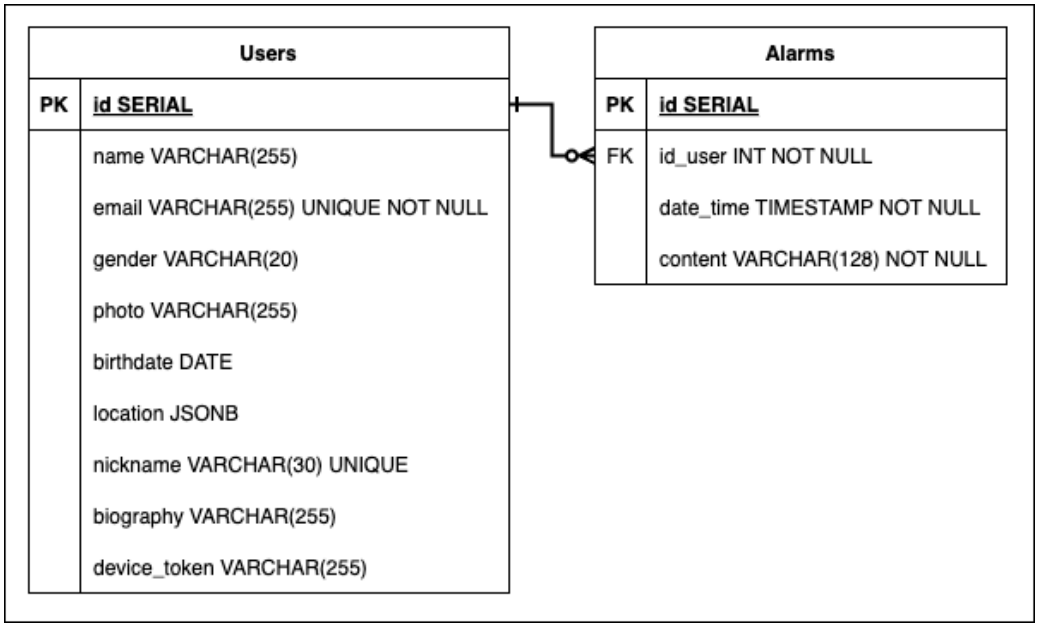

Figura 31: Microservicio de Usuarios. Modelo entidad-relación de la base de datos.

Para el esquema de la base de datos del microservicio de measurements se crearon dos tablas que se pueden observar en la Figura 32.

La tabla de **device_plant** que contiene la información de los sensores asociados a las plantas registradas por cada usuario. Esta tabla almacena el identificador del dispositivo (`id_device`) como clave primaria, el identificador de la planta (`id_plant`) del usuario (que es el identificador de la planta que se encuentra en la tabla de plantas del microservicio de plantas), el tipo de planta (`plant_type`) que representa el nombre científico de la planta (y también se encuentra en la tabla de plantas del microservicio de plantas) y el identificador del usuario (`id_user`) propietario de la planta. El identificador de la planta será único para cada usuario, por ende un usuario no puede tener mas de una planta con el mismo sensor. Además el tipo de planta y el identificador del usuario son valores numéricos que no pueden ser nulos logrado que no exista un sensor sin planta asociada (y sin usuario asociado).

La segunda tabla de **measurements** contiene todas las mediciones para una planta. Esta tabla contiene el identificador de la medición (`id`) como clave primaria; el identificador de la planta (`id_plant`) que es el identificador de la planta que pertenece al usuario y que se encuentra en la tabla de plantas del microservicio de plantas; el tipo de planta (`plant_type`);

el timestamp (`time_stamp`) que representa el momento en el que se realizó la medición y por último los 4 factores que se miden con los sensores: la temperatura (`temperature`), el nivel de humedad del ambiente (`humidity`), la luz (`light`) y el nivel de riego (`watering`).

Los niveles de humedad y riego son valores enteros porcentuales que varían entre 0 a 100; la luz es un valor flotante en unidad de foot-candle (ft-c) y por último la temperatura también es valor flotante en unidades de grado Celsius (°C).

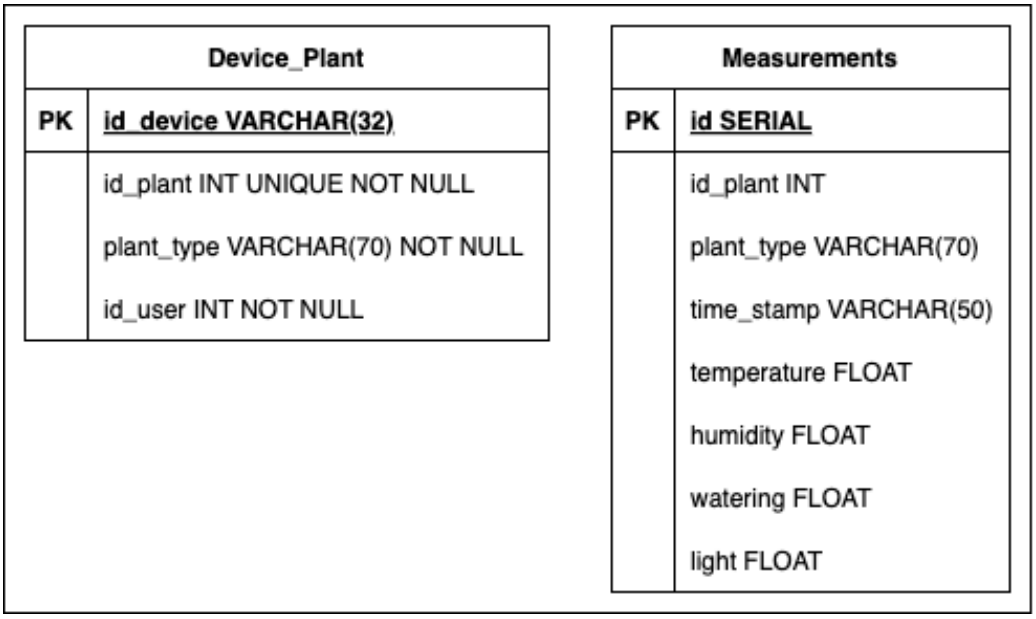

Figura 32: Microservicio de Mediciones. Modelo entidad-relación de la base de datos

Por último, en la Figura 33 para el microservicio de plantas se pueden observar las 4 tablas que se crearon para su esquema en la base de datos.

La primera tabla es la de **plant_types** que contiene los tipos de plantas que se pueden registrar en la aplicación (basados en los tipos de plantas que se pueden encontrar en la publicación *Growing Indoor Plants with Success (Bodie V. Pennisi, 2009, p.18)*). Cada tipo de planta tiene un nombre científico (`botanical_name`) como clave primaria; un identificador único (`id`) que es un valor entero; el nombre común (`common_name`) por el cual se conoce a la planta; una descripción (`description`) de la misma; texto con recomendaciones de cuidados (`cares`) para brindar al tipo de planta y por último un link a una foto (`photo_link`) de la planta.

La segunda tabla es la de **plants** que contiene las plantas registradas por los usuarios.

Cada planta tiene un identificador único (`id`) como clave primaria, el identificador del usuario (`id_user`) que es asignado en el microservicio de usuarios, el nombre de la planta (`name`) que es elegido por el usuario y el nombre científico de la planta (`scientific_name`) que es una clave foránea de la tabla de planta. Una tipo de planta puede estar asociado con 0 o N plantas; una planta siempre estará asociada a un tipo de planta.

La tercera tabla es la de **logs** que contiene las bitácoras de las plantas. Cada bitácora tiene un identificador único (`id`) como clave primaria, un título (`title`) que es un valor de tipo `string`, la fecha de creación (`created_at`) y la fecha de actualización (`updated_at`) que son valores de tipo `timestamp`, y el identificador de la planta (`plant_id`) que es una clave foránea al identificador de la tabla de plantas. Esto último implica que una planta puede tener varias bitácoras asociadas o ninguna, pero una bitácora siempre estará asociada a una planta.

Por último se muestra la tabla de **logs_photos** que contiene las fotos asociadas a las bitácoras. Cada foto tiene un identificador único (`id`) como clave primaria, el identificador de la bitacora (`log_id`) que es una clave foránea al identificador de la tabla de bitácoras y un link a la foto (`photo_link`) que es un valor de tipo `string`. Esta relación implica que puede existir 0 o N fotos asociadas a una bitácora y una foto de bitácora siempre estará asociada a una bitácora.

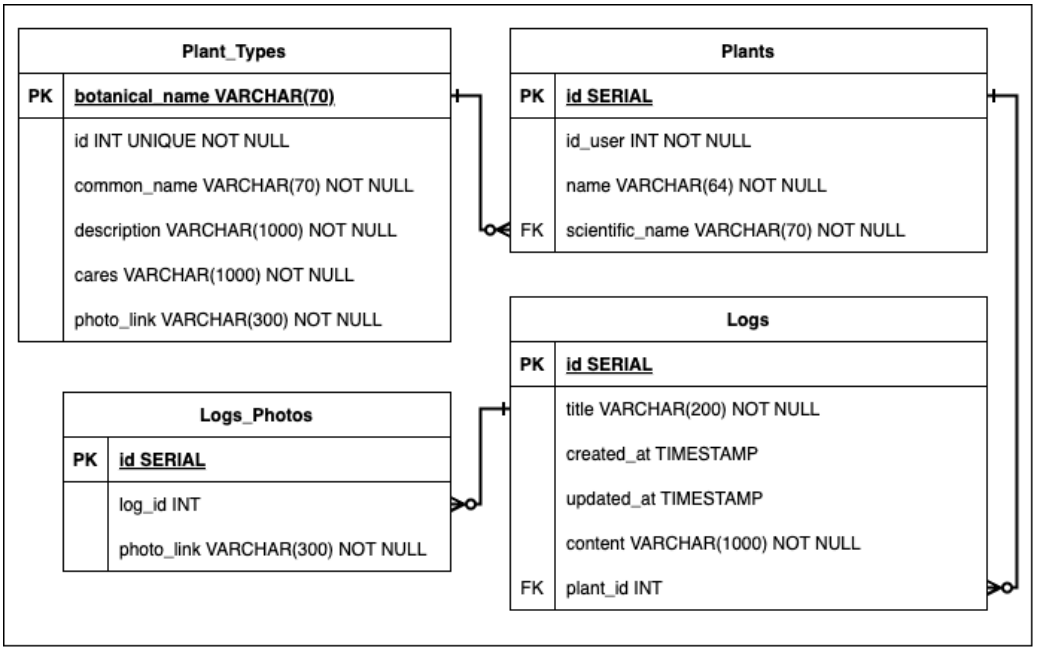

Figura 33: Microservicio de Plantas. Modelo entidad-relación de la base de datos.

15.8.2. Base de datos no relacional

En la Figura [34] se puede observar un breve ejemplo de cómo queda conformado un documento de la colección de **publicaciones** en la base de datos no relacional (MongoDB) utilizada en el microservicio de red social. Vemos que contamos con un identificador (`_id`) auto-generado por MongoDB en formato `ObjectId`; el identificador del usuario que realizó la publicación (`author_user_id`) que es un valor entero obtenido del modelo de usuarios del microservicio de usuarios; el contenido de la publicación (`content`) que es de tipo `string`; la cantidad de likes que recibió (`likes_count`) como valor entero; los identificadores de los usuarios que dieron like (`users_who_gave_like`) que de la misma forma son valores enteros referentes a los identificadores de los usuarios asignados en el microservicio de usuarios; la fecha de creación (`created_at`) y la fecha de actualización (`updated_at`) que son valores de tipo `Date`; las etiquetas (`tags`) que se le asignaron a la publicación como un vector de `strings`; un vector con los links de las fotos (`photo_links`) que son de tipo `string` pero que deben responder a un formato de URL; un vector con los comentarios (`comments`) que se realizaron en la publicación y la cantidad de

comentarios (`comments_count`) siendo un valor entero. Cabe destacar que cada comentario es guardado como un documento embebido con el identificador del comentario (`id`) también autogenerado por MongoDB, el identificador del usuario que realizó el comentario (`author`) junto al contenido del comentario (`content`) y la fecha de creación del comentario (`created_at`).


```
{
  "_id": {
    "$oid": "668b3c1c688a0666d6884856"
  },
  "author_user_id": 26,
  "content": "Suculentas giveaway! Comenta si queres una",
  "likes_count": 2,
  "users_who_gave_like": [
    25,
    11
  ],
  "created_at": {
    "$date": "2024-07-07T22:08:44.122Z"
  },
  "updated_at": {
    "$date": "2024-07-07T22:13:57.837Z"
  },
  "tags": [],
  "photo_links": [
    "https://firebasestorage.googleapis.com/v0/b/hanagotchi.
     appspot.com/o/social%2F26%2Fposts%2F1720400921110?alt=
     media&token=f9286d3b-c3d5-4ad6-bdeb-2bb54be7b82a"
  ],
  "comments": [
    {
      "id": "2d85855c-e332-4360-bc05-7370c5b0b364",
      "author": 25,
      "content": "buenisimo!!",
      "created_at": "2024-07-07T22:13:57.717080"
    }
  ],
  "comments_count": 1
}
```


Figura 34: Modelo de un ejemplo de documento de la colección de Publicaciones

Por último, en la Figura 35 se observa el documento de la colección de **usuarios de la red social**. El mismo busca contener la información mínima y necesaria para el contexto de la red social ya que dentro del microservicio de usuarios (observado en la Figura 31) se almacena la información completa de los usuarios. En este caso, se tiene un identificador (`_id`) que es un valor entero asociado al identificador del usuario asignado en el microservicio de usuarios; también se cuenta con dos vectores, uno con los identificadores de los seguidores (`followers`) y otro con los identificadores de los usuarios seguidos (`following`) del usuario. Por último, se

tiene un vector de strings con las etiquetas (`tags`) de interés del usuario.

```
{
  "_id": 13,
  "followers": [25],
  "following": [26],
  "tags": ["plantas"]
}
```

Figura 35: Modelo de un ejemplo de documento de la colección de Usuarios

15.9. Anexo I - APK para dispositivo Android

Link a la apk para dispositivos Android:

https://drive.google.com/file/d/15vdHFSBEPEqSU9hqYrJigPj_KkAFq3ZL/view?usp=sharing