Desarrollo de una aplicación de IoT y *machine learning* para el mantenimiento automático de cultivos en siembra vertical

Rodrigo Alejandro Cruz Fagiani

UVG UNIVERSIDAD DEL VALLE DE GUATEMALA

UNIVERSIDAD DEL VALLE DE GUATEMALA
Facultad de Ingeniería

Desarrollo de una aplicación de IoT y *machine learning* para el mantenimiento automático de cultivos en siembra vertical

Trabajo de graduación presentado por Rodrigo Alejandro Cruz Fagiani
para optar al grado académico de Licenciado en Ingeniería Electrónica

Guatemala,

2025

Vo.Bo.:

(f) _____

M. Sc. Carlos Esquit

Tribunal Examinador:

(f) _____

M.Sc. Carlos Esquit

(f) _____

Dr. Luis Alberto Rivera Estrada

(f) _____

Ing. Kurt Emmanuel Kellner

Fecha de aprobación: Guatemala, 13 de febrero de 2025

Prefacio

El desarrollo de esta tesis representa la culminación de mis estudios en ingeniería electrónica, un proceso que ha sido enriquecedor tanto a nivel personal como profesional. A lo largo de este camino, he tenido la oportunidad de profundizar en áreas fundamentales de la electrónica, la programación y los sistemas embebidos, consolidando los conocimientos adquiridos para materializarlos en un proyecto que busca contribuir al campo de la agricultura tecnológica.

Mi trabajo se centra en el diseño y desarrollo de un sistema hidropónico automatizado, utilizando el microcontrolador ESP32 como núcleo de procesamiento para interactuar con sensores y actuadores. A través del protocolo MQTT, el ESP32 se conecta a una Raspberry Pi que actúa como *gateway*, transmitiendo los datos recopilados a una API basada en microservicios, alojada en la web. Este sistema permite el monitoreo en tiempo real de variables críticas como la temperatura, la humedad y el nivel de agua de las torres hidropónicas, además de controlar dispositivos como una bomba sumergible y barras LED para iluminación artificial. Este sistema está diseñado para su implementación en entornos interiores, ofreciendo una solución eficiente y escalable para la producción hidropónica. Entre los microservicios implementados, se destaca uno basado en Machine Learning, diseñado para optimizar el control de recursos, con el objetivo de lograr un funcionamiento más eficiente y reducir el consumo energético.

Quisiera expresar mi más profundo agradecimiento a mis padres, cuyo apoyo a lo largo de mi carrera hizo posible la realización de este proyecto, y a mi novia, por ser un pilar emocional en los momentos más desafiantes durante los diez semestres. Agradezco también a mis maestros, quienes con paciencia y dedicación me transmitieron los conocimientos que ahora aplico, y a la universidad, no solo por lo aprendido, sino también por otorgarme la beca Potencia-T, que me permitió continuar mis estudios durante estos cinco años. Es a ellos a quienes dedico este trabajo, esperando que sea un aporte significativo al desarrollo de tecnologías aplicadas a la agricultura sostenible.

Índice

Lista de figuras

Lista de cuadros

Resumen

Este trabajo consiste en el desarrollo de una plataforma basada en microservicios para el control y monitoreo de torres hidropónicas verticales utilizando tecnologías IoT, sensores y microcontroladores ESP32. Esta plataforma permite a los usuarios gestionar y supervisar el sistema hidropónico mediante una interfaz que muestra los datos esenciales recolectados de los sensores y facilita la configuración remota de los dispositivos. El sistema se compone de una arquitectura modular que emplea tecnologías como Node.js, Docker, Kubernetes y Next.js, garantizando escalabilidad y facilidad de mantenimiento.

El sistema se divide en dos partes principales: los microservicios y los dispositivos IoT. Los microservicios incluyen un servicio de autenticación para la seguridad de los usuarios, un servicio de gestión de dispositivos que permite configurar los parámetros operativos de los dispositivos IoT, un servicio de lecturas que recopila los datos de los sensores, un servicio de comandos para enviar instrucciones a los dispositivos y un servicio de correos electrónicos para notificaciones automáticas. Por su parte, los microcontroladores ESP32 capturan los datos de sensores de humedad, temperatura y luz, y los transmiten al sistema central mediante protocolos HTTP y MQTT, funcionando como el gateway de los dispositivos IoT.

Adicionalmente, se desarrolló un modelo de *machine learning* utilizando la biblioteca *scikit-learn* de *Python*. Este modelo emplea variables del entorno (como temperatura, humedad y niveles de luz) y variables del sistema (intervalos de encendido y apagado de la bomba, tiempo desde la última solución nutritiva, y cantidad de luz artificial) para identificar los intervalos de riego más óptimos. Los resultados obtenidos confirman que tanto la API como el modelo son funcionales y cumplen con los objetivos planteados. Además, la torre hidropónica fue construida exitosamente mediante procesos de manufactura como impresión 3D y corte láser, demostrando la viabilidad del sistema para su aplicación en entornos urbanos.

Abstract

This work consists of the development of a platform for controlling and monitoring vertical hydroponic towers through IoT technology, using sensors and ESP32 microcontrollers. The platform enables users to manage and supervise the hydroponic system via an interface that displays key sensor data and allows for the remote configuration of devices. The communication between components is handled through protocols such as HTTP and MQTT, while the system is built on a microservices architecture to ensure scalability, modularity, and maintainability.

The system can be divided into two main parts: the microservices core and the IoT device edge. The core includes a set of microservices, such as an authentication service for user security, a device management service for configuring IoT devices, a readings service for collecting sensor data, a commands service for sending instructions to devices, and a mailing service for automatic notifications. Additionally, the core leverages a Next.js-based user interface for visualizing real-time data and a graph database for storing device information. On the edge, ESP32 microcontrollers collect environmental data—such as temperature, humidity, and light intensity—and transmit it to the core, serving as the gateway for IoT devices.

Furthermore, a machine learning model was developed using the *scikit-learn* library in Python. This model analyzes environmental and system parameters, such as pump operation intervals and light levels, to determine optimal irrigation intervals that balance plant health and energy efficiency. The suggested irrigation intervals are manually configured via the API. The results demonstrate the successful implementation of both the API and the machine learning model. Additionally, the hydroponic tower was constructed using advanced manufacturing techniques, including 3D printing and laser cutting, validating the practicality of the system for urban agricultural applications.

CAPÍTULO 1

Introducción

El mundo cada vez está más conectado, y la industria busca tener la mayor cantidad de control sobre sus procesos para mantenerlos eficaces y eficientes. A esta incongita llega una solución: el internet de las cosas o IoT (*Internet of Things*). Esta metodología consiste en conectar máquinas a la red para obtener control de los mismos y datos en tiempo real, facilitando así las dos características fundamentales de los procesos industriales. Además, recientemente se ha vuelto tendencia la utilización de *machine learning* para predecir eventos futuros y actuar acorde a estos. Los modelos son ejecutados de forma activa, adecuando actuadores para esperar una retroalimentación del entorno a dichos cambios. De esta forma, los modelos pueden ser utilizados para controlar procesos completos para garantizar la mayor eficiencia con el menor esfuerzo.

Estos paradigmas son fundamentales para un mundo en exponencial crecimiento, donde la demanda de productos crece día a día y la industria debe ponerse responder. Por lo tanto, la escalabilidad es un factor importante, que con metodología tradicionales dificilmente se puede alcanzar y mantener. Un area que encara este dilema es la industria alimenticia. La población cada día busca más alimentos naturales y accesibles, pero con la urbanización a la orden del día, el campo útil para cultivar alimentos naturales directamente en la tierra se reduce. Si el area disponible la reducimos al area útil con suelo apto, esta cifra se reduce aun más. Sin embargo, en varios sitios se ha descubierto y desarrollado el cultivo vertical, que soluciona varios de los retos ya comentados.

El cultivo vertical consiste en crecer alimentos directamente en torres, estanterías o canaletas verticales, las cuales contienen compartimientos para colocar las plantas y colgar las raices sobre una solución nutritiva. Este método reduce la demanda de agua, el espacio necesario, la necesidad de fertilizantes y pesticidas químicos y se aisla de las variables atmosféricas. Aplicando las tecnologías ya comentadas y añadiendo un sistema inteligente que aproveche el IoT y *machine learning*, se puede lograr un proyecto sumamente escalable para suplir dicha demanda, utilizando los recursos disponibles de forma eficiente y responsable.

CAPÍTULO II

Antecedentes

A. Conocimientos previos adquiridos

Dentro del pénsum de la Licenciatura en Ingeniería Electrónica se imparten diversos cursos que cubren varios de los temas abordados en este proyecto de graduación. Los puntos principales son los sistemas IoT, SCADA, manejo de diferentes procesos, *machine learning*, etc. Algunos de los cursos relevantes son Electrónica Digital, Circuitos Eléctricos, Programación de Microcontroladores y, para la realización de las placas necesarias, Simulación y Fabricación de PCBs.

En el curso de Electrónica Digital se aborda todo lo relacionado al funcionamiento de la lógica binaria y como la electrónica puede llevar a cabo funcionalidades complejas. Igualmente se cubren temas como la arquitectura fundamental de un microcontrolador y la comunicación que estos incorporan tales como I2C, SPI y UART. Estos tipos de comunicación son esenciales para utilizar sensores de distintos tipos, como, por ejemplo, sensores de humedad de suelo, temperatura, velocidad del viento, etc. También se aborda lo básico del microcontrolador ESP32, el cual se utiliza de manera elemental en aplicaciones de IoT. Este microcontrolador será utilizado debido a su capacidad de conectarse por medio de wifi, Bluetooth y otras comunicaciones inalámbricas. Para la implementación de las torres hidropónicas se implementaran estos métodos para la recolección de datos sobre el nivel de solución nutritiva en el tanque, temperatura y humedad ambiental, exposición a la luz solar y distintos indicadores tales como LEDs RGB y pantallas. En el último curso de la serie de Electrónica Digital, se abordan temas como la comunicación inalámbrica a través de UDP y TCP, con la Raspberry Pi como la principal protagonista. En esta placa se pueden realizar distintos procesos computacionales, como ejecutar distintos hilos, generar procesos y correr programas de lenguajes de más alto nivel como Python. Estos conocimientos serán esenciales para la construcción de los distintos módulos del proyecto.

El proyecto también contempla la creación de placas especializadas para cada dispositivo. Un curso que cubre el desarrollo de circuitos impresos es Simulación y Fabricación de PCB. En este curso se estudian los softwares de simulación de circuitos y de fabricación

de placas de circuito impreso, como Altium Designer. Además, se enseñan las herramientas necesarias para realizar pruebas de desempeño, chequeo de reglas de diseño y el diseño de componentes electrónicos personalizados e incluirlos en los esquemáticos. También se introduce a la soldadura, tanto de dispositivos con tecnología SMT como THT. Estas habilidades se aplicarán en la fase final de ensamblaje de todos los componentes.

B. Proyectos anteriores del departamento

Dado a que los temas de internet de las cosas y el *machine learning* son temas de actualidad, se han realizado numerosos proyectos al respecto. Entre estos destacan dos, realizados dentro del Departamento de Ingeniería Electrónica, Mecatrónica y Biomédica. Ambos proyectos proponen la utilización de dispositivos IoT para resolver problemáticas distintas. Uno de estos proyectos también incorpora la aplicación de *machine learning*.

El primer proyecto, llevado a cabo por Morales (2024), tuvo como objetivo principal desarrollar una plataforma para la orquestación de dispositivos IoT conformada por microservicios relacionados. Para la implementación, se utilizó la placa de desarrollo ESP32 debido a su variedad de características y su capacidad para utilizar diversos protocolos de comunicación inalámbrica. Este microcontrolador envía mensajes tipo MQTT por medio de un *broker*, que en este caso se utilizó Mosquitto, hacia distintas API alojadas en la nube. Asimismo, se hace mención de la utilización de Django para el desarrollo de estas API y un Docker para contenerlas, logrando así tener microservicios alojados en la nube. Entre los protocolos utilizados para el proyecto están MQTT como el principal, UDP, Bluetooth y HTTP. Para las bases de datos se utilizó Neo4j y PostfreSQL, que son opciones bastante populares. Por último, para concretar la orquestación se iplementó el sobtware de Kubernetes para la orquestación de los contenedores. Para desplegar los datos en una nube pública, se escogió la nube de Microsoft Azure. Este último cuenta con distintos servicios en conjunto con otras tecnologías, como Azure Kubernetes Services y Azure Database for PostgreSQL [1].

El segundo proyecto, realizado por Gil (2023), es similar, aunque con el agregado del aprendizaje automático. Este va enfocado a brindar la opción de conexión IoT a dispositivos que normalmente no lo tienen. Como antecedentes resalta las capacidades de la Raspberry PI para utilizarse como parte de una implementación de aprendizaje automático e IoT y el complemento que puede brindar el ESP32 a estos sistemas con sus protocolos de comunicación inalámbrica. Los protocolos mencionados en este proyecto son ZigBee, Z-Wave, Bluetooth, MQTT, wifi y HTTP. Además, en la metodología destaca la búsqueda de el protocolo de comunicación más eficaz para la aplicación de IoT y el algoritmo de *machine learning* más eficaz para distintas aplicaciones. Otra parte importante de este proyecto es la utilización de un software que funcionara como *home assistant* dentro de la Raspberry Pi, y de esta manera tener mayor control sobre los dispositivos conectados. Por la variedad y diversidad de dispositivos, en este proyecto se exploraron plataformas de IoT tales como ThingsBoards, Thinger.io, Mainflux, OpenRemote y PlatyPush. Todas estas proporcionan una solución para la escalabilidad, manejo de datos y procesos y una experiencia amigable con el usuario [2].

Estos proyecto serán útiles para fijar bases en la investigación del proyecto en cuestión. Se

tomarán algunos de los protocolos de comunicación utilizados en ambos proyectos y se consideraran los algoritmos de aprendizaje automático propuestos en el segundo trabajo. Además se buscará agregar características innovadoras y específicas para lograr una diferenciación de este proyecto más allá del area de aplicación.

C. Avances de IoT en la agricultura

Los países más desarrollados y con la capacidad de cultivar alimentos ya están utilizando estos sistemas para mejorar sus procesos. Un ejemplo muy puntual es la compañía Cropin, que se dedica a implementar soluciones IoT para grandes productores. Como ellos lo explican en su página web, se dedican a diseñar tecnología IoT para monitorear campos de cultivos y automatizar sistemas de riego. Entre sus productos tienen una variedad de robots destinados a diferentes tareas. Tienen robots con la tarea específica de deshierbar, utilizando su gran base de datos y comparando de planta en planta para que así se tengan mejores resultados. Hay máquinas que permiten a los agricultores navegar por los campos de cultivos de forma remota. De estos existen vehículos aéreos al igual que terrestres. En algunos de estos se poseen sistemas de análisis de imágenes para reconocer cuando es un momento óptimo de cosecha o si existe la presencia de plagas. Cuando llega el momento de cosechar, entran otros robots capaces de cortar los frutos por medio de análisis de imágenes y comparando con bases de datos para obtener únicamente aquellos en buen estado. Esto lo realizan de forma cuidadosa y precisa, utilizando la robótica para el control de cada movimiento. Además de toda la robótica aplicada a la agricultura también existen dispositivos más estacionarios capaces de obtener información relevante para los agricultores. Nuevamente, la compañía Cropin posee dispositivos de medición que junto con estaciones de monitoreo del clima, obtienen datos para llevar mejor control sobre los procesos y que así puedan detectarse anomalías con anticipación y actuar acorde. En resumen, este proyecto busca desarrollar tecnología IoT para llevar a que la agricultura cumpla con la futura demanda y se eviten pérdidas en el proceso [3].

D. Avances de *machine learning* en la aricultura

Con la necesidad de aumentar la producción en espacios reducidos, los agricultores se ven retados y obligados a avanzar en la tecnología que utilizan. Parte de estas herramientas es el *machine learning*. Este se implementa con el objetivo de hacer la producción más eficiente y eficaz, en el sentido de obtener la mayor cantidad de cosecha y de buena calidad, con el menor uso de recursos. Esto se realiza por medio de datos históricos obtenidos por medio de sensores y guardados en una base de datos. Se pueden realizar predicciones utilizando distintos enfoques como regresiones, métodos de clasificación y redes neuronales. Los más utilizados dentro de la agricultura son los modelos que utilizan regresiones y los que se enfocan en clasificación.

Se han realizado varios experimentos, como el realizado en *Journal of Physics* escrito por Sundaresan [4]. En este se buscó la implementación de modelos de *machine learning* y en qué aplicaciones puede utilizarse cada uno. Entre los resultados se generaron modelos en los que se podía indicar el mejor cultivo para las condiciones ambientales de cierto lugar.

Además se generaron modelos para indicar en qué momentos es mejor utilizar pesticidas para obtener cultivos sanos y las cantidades de los mismos para no tener cultivos "muertos". Para estos se utilizaron los modelos de clasificación llamados *decision tree* y *KNN algorithm*. Se encontró que ambos poseían alta precisión, aunque el *decision tree* utilizaba menos recursos computacionales.

E. Sistemas hidropónicos en torre

Actualmente, debido a varios factores como lo puede ser espacio, deficientes condiciones climáticas y la falta de agua, se ha visto la iplementación de torres hidropónicas como un método atractivo para cultivar alimentos. Existen dos formas para comenzar en este mundo, y estas son comprando un sistema hidropónico o realizando uno de forma artesanal. Muchos ingenieros y aficionados han tomado distintos caminos y han realizado una gran variedad de ejemplares que pueden tomarse como referencia.

Por parte de la comunidad se han realizado varios prototipos. Entre estos destaca los realizados por creadores de contenido en la plataforma de YouTube. Aquí se puede encontrar varios ejemplos con los materiales utilizados e incluso instrucciones paso a paso de cómo elaborar un sistema hidropónico de torre artesanal. Al observar todos estos ejemplos se obtuvo una lista de puntos de tomar en consideración a la hora de fabricar. Para crear el cuerpo de la torre se puede hacer de dos formas, utilizando impresión 3D o tubos de PVC. En varios ejemplos se utilizó la impresión 3D, aunque este método presentó varios inconvenientes, especialmente el tiempo de impresión y el material a utilizar. Por esta razón se optará por utilizar tubos de PVC cortados ya que estos resisten fácilmente las temperaturas del exterior y, utilizando calor, es fácilmente moldeable. Para el tema del reservorio principal usualmente se utilizan cubetas, tinas plasticas o un sistema de tuberías. Esta parte es esencial ya que es la que contendrá la solución nutritiva. También debe opacar la luz solar para evitar el crecimiento de algas dentro del reservorio y que estas consuman de la solución nutritiva. Otro componente que se debe considerar es la bomba de agua necesaria para llevar la solución del reservorio principal al secundario, ubicado en la parte superior y de donde se comenzará la distribución de la solución de forma equitativa sobre toda la torre. Esta bomba debe tener una capacidad de bombeo suficiente para cumplir con la altura de la torre que en varios de los casos ronda por los 2 metros de altura. Pasando a los últimos elementos encontramos el reservorio secundario que es el encargado de recibir la solución y distribuirla por goteo a través de rejillas agujeradas. Para llevar la solución nutritiva del reservorio principal a este reservorio secundario se utilizan tubos de PVC más delgados o mangueras. Por últmo, en varios prototipos se utiliza un timer para encender y apagar la bomba de forma intermitente en lapsos estáticos. Esto es para ahorrar costos de energía. Sin embargo, en varias ocasiones se presentó la problemática de que, al ser lapsos de tiempo estáticos, en ocasiones cuando el sol era muy intenso o había bastante precipitación era necesario cambiar estos tiempos de forma manual y se volvía algo muy tedioso. En varias ocasiones incluso los cultivos se secaban y detenían su producción. Otro inconveniente que se repetía en varios de los videos era la falta de uniformidad de la distribución de la luz solar, lo que hacía que los cultivos de un lado producieran más que los ubicados en el otro lado de la estructura. En una construcción creada por Loh [5] en su video sobre como hacer una torre hidropónica para el crecimiento de vegetales opta por únicamente utlizar un lado

para asegurar la uniformidad de la luz solar sobre todos los cultivos. Aunque esta solución cumple con su propósito al mismo tiempo elimina el espacio útil que podría utilizarse para cultivar más y aumentar la producción. Los diseños propuestos, retos encontrados y consejos brindados por la comunidad serán tomados en consideración para construir el esqueleto del sistema y asegurar así el mejor funcionamiento.

En el mundo de las hidroponias verticales existen varias empresas que se dedican a facilitar esta solución para su aplicación en hogares, oficinas o innvernadores. Aunque hacen la implementación de estos sistemas en el hogar más comodo y rápido de comenzar, su costó suele ser muy elevado, llegando a costa alrededor de 600.00 USD por una torre. Estos precios elevados suelen actuar como un bloqueo para las personas que desean entrar en este mundo de una forma más sencilla y sin tanto trabajo manual. Además de los precios elevados estos sistemas traen exactamente los mismos retos que los sistemas elaborados artesanalmente mencionados anteriormente. Existen algunas variante de estas torres que ofrecen luz LED para asegurar la uniformidad del crecimiento, pero para la implementación en hogares, oficinas o lugares son escaso suministro de energía esta opción suele ser costosa. Una compañía que se dedica a ofrecer estos sistemas es Tower Garden. Su producto que se asemeja más a las soluciones artesanales es la linea de Tower Garden HOME. Este paquete incluye la torre modular con ruedas para facilidad de transporte y un paquete de productos para la solución nutritiva, regulador de pH y entre otros. Este paquete tiene un precio de 725.00 USD por unidad. Para que esta solución sea escalable y aplicable para personas con poco acceso a alimentos debido a la infertilidad de la tierra o escasos recursos estos productos no son realistas.

Para el proyecto se consideraran las mejores características de cada uno de los mundos y sus retos. Se busca obtener una solución intermeda, que se base en el uso de materiales locales y la implementación de la electrónica para mejorar la eficiencia y eficacia de los cultivos, haciendo un mejor uso de los recursos dispnibles como son el agua, la energía y el espacio.

CAPÍTULO III

Justificación

La producción de alimentos mediante la agricultura es esencial para nuestra supervivencia, ya sea para consumo humano directo o para alimentar al ganado. Con el crecimiento continuo de la población mundial, la demanda de alimentos se vuelve cada vez más urgente, mientras que la urbanización reduce significativamente el espacio disponible para el cultivo. Estos desafíos imponen una presión considerable sobre los productores, quienes deben aumentar su capacidad de producción y, al mismo tiempo, enfrentar dificultades en el mantenimiento y la calidad del producto final.

A nivel global y nacional, existen regiones donde el cultivo tradicional es inviable debido a la escasez de agua, la infertilidad del suelo o las condiciones climáticas. Frente a estas necesidades, se propone un sistema hidropónico vertical automatizado que emplea una variedad de sensores, actuadores y herramientas avanzadas como el internet de las cosas (IoT) y el *machine learning*. Este sistema está diseñado para facilitar, hacer más eficiente y permitir la escalabilidad de la producción en entornos que carecen de las condiciones esenciales para la agricultura convencional.

La hidroponía vertical se presenta como una solución innovadora que no solo optimiza el uso del espacio, sino que también mejora el control sobre las condiciones ambientales, promoviendo un crecimiento más saludable y rápido de las plantas. El uso de sensores como el DHT22 para monitorear la humedad y la temperatura, el JSN-SR04T para medir la distancia y los niveles de agua, el sensor VEML para luz ambiental y el sensor de efecto Hall para medir la precipitación permiten un control preciso de las variables críticas que afectan el crecimiento de los cultivos.

Para garantizar la escalabilidad del proyecto, se propone la implementación de IoT, lo que permitirá el monitoreo en tiempo real del estado de las torres hidropónicas y su control remoto desde cualquier parte del mundo. Esta conectividad facilita la gestión y supervisión de múltiples unidades, permitiendo a los productores tomar decisiones informadas y rápidas. Además, el uso de *machine learning* es fundamental para la autogestión del sistema. A través de modelos de regresión, clasificación y aprendizaje profundo *deep learning*, el sistema puede analizar datos históricos y en tiempo real para optimizar el uso de recursos como la

solución nutritiva y la energía. Esto no solo reduce el desperdicio y los costos operativos, sino que también mejora la sostenibilidad del sistema. Los algoritmos de *deep learning* también pueden predecir eventos críticos, como cambios en las condiciones climáticas o necesidad de mantenimiento en la torre y así poder enfocar los esfuerzos en las áreas más necesarias.

En resumen, la implementación de un sistema hidropónico vertical automatizado que integre IoT y *machine learning* ofrece una solución prometedora para enfrentar los desafíos actuales de la agricultura. Este enfoque innovador no solo mejora la eficiencia y la productividad, sino que también abre nuevas posibilidades para la agricultura en áreas urbanas y regiones desfavorecidas, contribuyendo significativamente a la seguridad alimentaria global.

CAPÍTULO IV

Objetivos

A. Objetivo general

Desarrollar un sistema de monitoreo y control de variables agricultura en un sistema hidropónico vertical mediante la integración de elementos de internet de las cosas y *machine learning*, utilizando el microcontrolador ESP32 y la plataforma Raspberry Pi.

B. Objetivos específicos

- Diseñar un sistema hidropónico vertical automatizado utilizando el método de riego por goteo.
- Elaborar un control central utilizando la computadora Raspberry PI para la recopilación de datos e implementación de modelos de *machine learning* para la predicción de eventos.
- Desarrollar sistemas periféricos secundarios donde se midan distintas variables relacionadas a los cultivos en el sistema utilizando el ESP32, especificamente la temperatura ambiental, humedad y luz.
- Realizar un sistema de comunicación inalámbria de los sistemas periféricos con el control central utilizando el protocolo MQTT entre los ESP32 y la Raspberry PI.
- Implementar el IoT para el monitoreo constante y control remoto del sistema utilizando distintos protocolos de comunicación.

CAPÍTULO V

Alcance

El alcance de este proyecto es proporcionar una solución que integre variedad de componentes, protocolos y algoritmos para automatizar un sistema hidropónico vertical. Esta solución se desea alcanzar mediante la implementación de IoT y *machine learning* que permitirán al usuario la facilidad de interactuar y mantener cultivos. Entre los protocolos principales a utilizar estan MQTT, HTTP y TCP, para la comunicación inalámbrica de los componentes. Para la comunicación por cable, se utilizará I2C, SPI, UART y otros protocolos de bus bidiraccional.

El sistema físico a construir para las pruebas es un prototipo a escala reducida, aunque se tiene como visión su fácil escalabilidad. La implementación tendrá los sensores DHT22, JSN-SR04T, VEML7700 y KY-003 para monitorear distintas variables ambientales clave como humedad, temperatura, iluminación, nivel de solución nutritiva, precipitación, entre otras. Como componentes principales de procesamiento se utilizarán el ESP32-WROOM Devkit en conjunto con la Raspberry Pi. El ESP32 es un dispositivo que por sus capacidades de comunicación inalámbrica y compatibilidad con distintos protocolos de comunicación es una opción popular para esta clase de aplicaciones. Por otro lado, la Raspberry Pi es una computadora con gran poder de procesamiento con variedad de aplicativos previamente desarrollados que facilitan la implementación de sistemas de control remoto.

Además de la parte física, se hará un enfoque extra en el desarrollo de software para el procesamiento de datos y el despliegue de los mismos. Se explorarán distintos servicios, tales como Docker, para alojar microservicios en la nube y que estos se encuentren en constante ejecución. Con respecto al *machine learning*, se desarrollarán algoritmos implementando regresiones para optimizar el ciclo de riego y la autogestión en general, basándose en datos históricos y en tiempo real. Con todas estas características, el sistema tendrá una variedad de características para sobreponerse a los retos que presenta la agricultura hoy en día.

CAPÍTULO VI

Marco teórico

A. Internet de las cosas (IoT)

1. Definición

El internet de las cosas (IoT, por sus siglas en inglés) ha experimentado un auge en los últimos años. Con la llegada de redes 5G, estos dispositivos se han vuelto cada vez más comunes. Este término se refiere a la capacidad de los dispositivos electrónicos para comunicarse desde cualquier parte del mundo a través de internet, con el objetivo de ofrecer un control más eficiente y eficaz en diversas aplicaciones. Más allá de simplemente recopilar y mostrar datos, este paradigma permite la integración de inteligencia en los sistemas, permitiendo predecir y gestionar de manera automática los eventos. Para su funcionamiento, se emplean sensores y actuadores que trabajan en conjunto, optimizando recursos, previniendo incidentes y mejorando los resultados a corto, mediano y largo plazo.

La arquitectura del internet de las cosas es bastante sencilla desde una perspectiva general. Participan cinco elementos principales: los dispositivos IoT, el*gateway*, internet, el control remoto y el almacenamiento y análisis de los datos recopilados. Los dispositivos IoT pueden ser cualquier aparato conectado a internet, como refrigeradores, tostadoras, aspersores o bombillas. Estos pueden conectarse directamente a la red mediante protocolos o a través de un intermediario, conocido como*gateway*. Este utiliza protocolos como HTTP, MQTT, CoAP o XML para transmitir la información a través de internet (mencionados en profundidad más adelante). El internet se encarga de gestionar la comunicación entre puntos remotos mediante paquetes de datos. Cuando estos paquetes llegan a un servidor, se traducen y almacenan en bases de datos para un acceso posterior. Estos datos pueden ser analizados en la nube mediante API, generando información adicional que puede presentarse al usuario o utilizarse para hacer el sistema más robusto. Finalmente, los datos de los dispositivos IoT pueden ser accesibles en tiempo real desde cualquier dispositivo conectado a la red [6].

El IoT se está aplicando en numerosos campos. Uno de los más antiguos es la automatización del hogar, donde se utilizan redes de dispositivos inteligentes como bombillas, interruptores y enchufes para controlar electrodomésticos desde cualquier lugar. Hoy en día, incluso se han incorporado refrigeradores y tostadoras inteligentes que pueden ser gestionados remotamente. Otro sector donde el IoT ha ganado relevancia es la medicina. Aquí, los dispositivos pueden incluir desde marcapasos hasta botones de emergencia o sensores de temperatura, que monitorean la salud de los pacientes. Si algún sensor detecta valores anormales, se puede actuar rápidamente para proporcionar asistencia. Uno de los avances más recientes es la cirugía a distancia, donde un médico en cualquier parte del mundo puede operar a través de un dispositivo robótico en tiempo real, facilitando intervenciones críticas y permitiendo la colaboración con especialistas de otras regiones.

2. Protocolos de comunicación

MQTT (*message queuing telemetry transport*)

MQTT es un un protocolo desarrollado por IBM para el transporte de mensajería de tipo cliente-servidor con una arquitectura de publicación/subscripción. Se caracteriza por su ligereza, simplicidad, de código abierto y fácil de implementar en diversas aplicaciones. Debido a estas propiedades, se utiliza en comunicación máquina a máquina (M2M) y en el internet de las cosas (IoT), los cuales son contextos donde se requiere una pequeña huella de código y/o el ancho de banda de la red es escaso [7].

En cuanto a la arquitectura de MQTT, podemos identificar tres componentes clave: los dispositivos IoT, como sensores y actuadores, el *broker* MQTT y una aplicación, que puede ser móvil o web. Esta arquitectura facilita la comunicación entre diversos clientes y dispositivos. Aunque podría parecer similar a algunos protocolos donde hay un maestro y varios esclavos, en MQTT solo existen clientes. El *broker* simplemente se encarga de distribuir los mensajes según corresponda, utilizando un esquema de publicación y suscripción. Cualquier cliente en la red MQTT puede suscribirse a un tópico o tema. Cuando uno de los clientes realiza una publicación, el *broker* recibe la información y la distribuye a todos los clientes suscritos a ese tema o tópico.

Entre las características más destacadas de MQTT, encontramos el desacoplamiento tridimensional, que abarca tres aspectos fundamentales. El primero es el desacoplamiento espacial, lo que significa que los clientes no necesitan conocerse entre sí ni realizar un *handshake* como en otros protocolos. Al publicar, no es necesario especificar un destino concreto, ni siquiera que haya uno presente en ese momento. De igual manera, un cliente puede suscribirse a un tema sin la necesidad de que haya publicaciones activas. El segundo aspecto es el desacoplamiento temporal, lo que implica que la comunicación no necesita ser en tiempo real. Por ejemplo, un sensor puede publicar un evento y este podrá ser procesado posteriormente sin depender del instante exacto de la publicación. Por último, está el desacoplamiento en sincronización, lo que significa que los clientes no se bloquean al realizar publicaciones o recibir mensajes, garantizando una comunicación eficiente.

Los clientes MQTT pueden ejecutar varias operaciones básicas. La primera es *connect*, que permite la conexión al *broker*. Otra es *publish*, que permite publicar mensajes sobre

un tópico, los cuales pueden contener cualquier tipo de información, como temperaturas, distancias o eventos. También existen las acciones *subscribe* y *unsubscribe*, que permiten a los clientes interactuar con otros clientes suscritos al mismo tópico o retirarse del tema si es necesario. Además, MQTT soporta subtópicos, que ayudan a organizar la información publicada y recibida de manera jerárquica, utilizando la estructura "tópico/subtópico/subsubtópico...". Para visualizar esto, podemos utilizar un ejemplo aplicado al sistema hidropónico. Si un cliente (como puede ser la Raspberry Pi) se suscribe al tópico "torres/torre1/sensores/temperatura", este cliente quedaría escuchando por todos los eventos enviados a este tema referido a la temperatura en cierta torre. MQTT también soporta çomodines", que ayudan a suscribirse a varios subtópicos a la vez. Primero, se cuenta con el comodín multinivel, implementado con el uso del caracter "#". Este comodín permite suscribirse a todos los subtópicos anidados dentro de un tópico. Continuadno con el ejemplo anterior, si la Raspberry Pi como cliente se suscribe a "torres/#", esta escucharía por todos los mensajes enviados por todas las torres hidropónicas. El segundo comodín es el mononivel, implementado con el caracter -". Este comodín permite al cliente suscribirse a un subtópico específico dentro de varios subtópico. Un ejemplo es si la Raspberry Pi se suscribe al tópico "torres/+/sensores/temperatura". Con esto, se estaría recibiendo todos los eventos de temperatura de todas las torres hidropónicas [8].

Finalmente, MQTT ofrece dos características adicionales muy útiles. La primera es la retención de mensajes: si esta opción está habilitada, el broker enviará el último mensaje publicado a cualquier cliente nuevo que se suscriba a ese tópico. Esto es útil para transmitir el estado actual de un tópico. La segunda es la sesión persistente, que permite que, si un cliente se desconecta y vuelve a conectarse, reciba todos los mensajes enviados durante su desconexión [9].

En conclusión, MQTT es un protocolo eficiente, eficaz y que permite una variedad de funcionalidades para tener una red robusta de dispositivos. Por esta razón, el proyeccto basará la transmición de información en la red a través de MQTT. Para la aplicación de dicho protocolo se utilizará el ESP32 y la Raspberry Pi 4 con el *broker* MQTT Mosquitto.

HTTP

HTTP es un protocolo que permite realizar peticiones de recursos y datos como por ejemplo archivos HTML. Este protocolo es el más uitilizado en la web para el intercambio de datos. Funciona en la capa de aplicación y se transmite por medio de TCP. Además, tiene una estructura cliente-servidor, donde un cliente es el que solicita por datos y el servidor el que los provee. El cliente suele ser un navegador web tales como Google Chrome, Firefox o Safari. El servidor es algún equipo físico que se encuentra en otra parte del mundo conectado por medio de internet que almacena distintos tipos de archivos. Entre estos podemos encontrar los archivos HTML, CSS, JavaScript, imagenes, videos y demás. Básicamente este protocolo permite la existencia de la navegación en internet.

Partiendo del lado del cliente, como ya se mencionó, es el ente que realiza la solicitud y usualmente son navegadores web. Actualmente se está ampliando el uso de los *bots* que también hacen esta clase de peticiones. En estas existen varios métodos, como los que se listan a continuación para realizar la comunicación con el servidor.

- *GET*: este método es el que más se utiliza en comparación con los demás. Su principal función es solicitar a un servidor algún recurso como lo puede ser un archivo HTML para desplegar una página web. Al momento de estar navegando se hacen múltiples solicitudes *GET* para ir obteniendo imagenes, videos y sripts para cargar páginas.
- *HEAD*: a diferencia de *GET*, este método únicamente responde con la información de cabecera. Estos datos pueden ser utiles para conocer el tamaño del archivo HTML que debe solicitarse y así ayudar a optimizar las solicitudes GET que se realicen.
- *POST*: se encarga de enviar datos al servidor. Un ejemplo de esto es cuando se está rellenando un formulario de inicio de sesión y se envía lo escrito en los campos.
- *PUT*: este método es muy similar al método *POST*, ya que envía datos al servidor. Sin embargo, este se utiliza principalmente para actualizar datos ya existentes en el servidor.

La evolución del protocolo HTTP ha sido clave para mejorar el rendimiento y la seguridad en la web. Desde la primera versión, HTTP/1.0, que estableció el concepto de comunicación basada en solicitudes y respuestas entre cliente y servidor, hasta la actual versión, HTTP/3, el protocolo ha pasado por varias mejoras. HTTP/1.1 introdujo conexiones persistentes, lo que permitió mantener abierta una conexión TCP para varias solicitudes, mejorando la eficiencia en la carga de múltiples recursos. HTTP/2, lanzado en 2015, optimizó aún más el rendimiento mediante la multiplexación, permitiendo que múltiples solicitudes y respuestas viajaran por la misma conexión. La versión más reciente, HTTP/3, abandona TCP en favor de QUIC, un protocolo de transporte diseñado para reducir la latencia y hacer la comunicación más rápida y segura.

El protocolo HTTPS es una extensión de HTTP que agrega una capa de seguridad mediante el uso de SSL (secure sockets layer) o TLS (transport layer security). Esta capa adicional encripta los datos intercambiados entre el cliente y el servidor, asegurando que la información transmitida, como contraseñas o datos personales, no pueda ser interceptada por terceros. HTTPS se ha vuelto indispensable en la mayoría de los sitios web, especialmente en aquellos que manejan información confidencial. Los navegadores modernos muestran un candado al lado de la barra de direcciones para indicar que el sitio utiliza HTTPS, y cada vez más, los motores de búsqueda priorizan sitios seguros en los resultados de búsqueda.

Los códigos de estado en HTTP juegan un papel crucial en la comunicación entre cliente y servidor, proporcionando información sobre el resultado de las solicitudes. Estos códigos se dividen en varias categorías: las respuestas informativas *(1xx)*, las respuestas de éxito *(2xx)*, como el común *200 OK*, las redirecciones *(3xx)*, y los errores *(4xx)* y *(5xx)*. Un código de error típico es el *404 Not Found*, que indica que el recurso solicitado no existe en el servidor. Por otro lado, los errores *5xx* como el *500 Internal Server Error* reflejan problemas en el servidor. Estos códigos permiten a los clientes y desarrolladores conocer el estado de sus solicitudes y gestionar posibles problemas en la comunicación.

El manejo de *cookies* en HTTP es fundamental para mantener sesiones y personalizar la experiencia de los usuarios en la Web. Las *cookies* son pequeños fragmentos de datos que los servidores envían a los clientes y que se almacenan en el navegador. Estos datos pueden incluir información de autenticación, preferencias del usuario o datos de seguimiento. Aunque

son esenciales para muchas funcionalidades, las *cookies* también han suscitado preocupaciones sobre la privacidad, ya que permiten rastrear el comportamiento de los usuarios en diferentes sitios web. Para gestionar esto, los navegadores ofrecen opciones para bloquear o eliminar *cookies*, y las leyes de privacidad, como el GDPR, imponen requisitos sobre su uso y almacenamiento [10].

B. *Machine learning*

1. Definición

En las últimas décadas, *machine learning* (ML) ha revolucionado el campo de la inteligencia artificial (IA), permitiendo a las máquinas aprender de los datos sin la necesidad de programación explícita. Este concepto, introducido por Samuel (1959), sentó las bases para el desarrollo de algoritmos capaces de detectar patrones, hacer predicciones y recomendaciones al procesar datos y experiencias. A medida que estos algoritmos reciben más datos, se ajustan automáticamente para mejorar su rendimiento, lo que ha impulsado su aplicación en una amplia variedad de sectores.

Uno de los avances más importantes dentro de ML es el surgimiento de *deep learning*, una rama especializada que se destaca por su capacidad de procesar datos no estructurados, como imágenes y texto, con una mínima intervención humana. Las redes neuronales profundas, inspiradas en la forma en que las neuronas interactúan en el cerebro humano, permiten a los modelos de *deep learning* analizar datos en múltiples capas. Este enfoque les permite reconocer características cada vez más complejas, desde identificar formas simples hasta reconocer objetos en imágenes, lo que ha permitido grandes avances en áreas como la visión por computadora y el procesamiento del lenguaje natural.

El auge de las redes neuronales profundas también ha sido clave en el desarrollo de modelos generativos, como ChatGPT. Estos modelos utilizan redes llamadas *transformer networks*, que permiten a las IA generar contenido basado en entradas como texto o imágenes. Los transformers, entrenados con vastos conjuntos de datos, son capaces de aprender relaciones complejas y generar respuestas coherentes a partir de nuevas entradas. Este tipo de IA ha transformado la interacción humana con la tecnología, haciéndola más accesible y dinámica.

Los avances en *machine learning* también han tenido un impacto significativo en el ámbito empresarial. Empresas de todo el mundo están adoptando estos algoritmos para mejorar la eficiencia operativa y optimizar la toma de decisiones. Sectores como el bancario y el de servicios públicos han experimentado mejoras sustanciales en la identificación de fraudes, la predicción de mantenimiento preventivo y la optimización del uso de recursos energéticos. En la NBA, por ejemplo, se han utilizado modelos predictivos para analizar el rendimiento de los jugadores y tomar decisiones estratégicas más precisas durante los juegos. Para que las empresas aprovechen al máximo el poder de ML, es crucial integrar estos algoritmos en el núcleo de sus arquitecturas tecnológicas. Esto implica replantear los desafíos tradicionales como problemas de aprendizaje automático y asegurarse de que los datos relevantes se utilicen de manera efectiva. Además, es esencial adoptar estrategias centradas en el talento humano, fomentando una cultura organizacional que apoye la adopción y el uso adecuado

de herramientas basadas en IA [11].

2. Algoritmos de aprendizaje

Para lograr estos aprendizajes, existen diversos algoritmos para entrenar modelos, dependiendo de la aplicación. Una clasificación básica distingue entre algoritmos de aprendizaje supervisado y no supervisado. En el aprendizaje supervisado, los datos ingresan al algoritmo con etiquetas que definen las categorías o resultados esperados, mientras que en el aprendizaje no supervisado, el algoritmo debe identificar automáticamente patrones en los datos sin etiquetas.

Uno de los grupos más populares de algoritmos de *machine learning* es el de la regresión. Estos modelos se basan en encontrar correlaciones entre variables dependientes e independientes. El objetivo es identificar un patrón que permita predecir el comportamiento futuro de la variable dependiente en función de las independientes. Algunos de los algoritmos más destacados son los siguientes:

- **Regresión lineal**: utiliza ecuaciones lineales para modelar las relaciones entre una variable dependiente y una o más variables independientes. El objetivo es encontrar una línea recta que mejor se ajuste a los datos. La regresión lineal tradicional compara una variable dependiente con una independiente, y se conoce más específicamente como regresión lineal simple, con la ecuación de una pendiente mostrada en la Ecuación 1. Si se incluyen múltiples variables independientes, se tiene una regresión lineal múltiple, modelada por la ecuacion 2. Las ecuaciones para la regresión lineal simple y múltiple se muestran a continuación:

$$y = \beta_0 + \beta_1 X_1 + \epsilon \tag{1}$$

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_n X_n + \epsilon \tag{2}$$

- **Regresión polinómica**: amplía la regresión lineal al incluir términos polinómicos para capturar relaciones más complejas entre las variables. Esta técnica es útil cuando los datos siguen una tendencia no lineal, y se utiliza en campos como la economía, la biología y la física. La ecuación es:

$$y = \beta_0 + \beta_1 X_1 + \beta_2 X_2^2 + \cdots + \beta_n X_n^n + \epsilon \tag{3}$$

- **Regresión logística**: se utiliza principalmente para problemas de clasificación. A diferencia de los modelos de regresión lineal, la regresión logística modela la probabilidad de un resultado binario (por ejemplo, éxito o fracaso) utilizando la función logística [12]:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X)}} \tag{4}$$

Otro grupo destacado es el de los algoritmos de clasificación. Estos modelos agrupan los datos en categorías discretas, en lugar de predecir valores continuos [13]. A continuación se describen algunos de los algoritmos de clasificación más utilizados:

- **Clasificación binaria**: este algoritmo clasifica los datos en dos categorías. Un ejemplo común es la clasificación de correos electrónicos como spam o no spam.
- **Clasificación multiclase**: similar a la clasificación binaria, pero permite agrupar los datos en más de dos categorías. Por ejemplo, un sistema de correo electrónico podría clasificar los correos en categorías como primarios, sociales y promociones.
- **Árbol de decisión**: este algoritmo utiliza un diagrama de flujo, donde cada nodo interno representa una prueba sobre un atributo de los datos y las ramas son los resultados de la prueba. Los nodos hoja representan la clasificación final. Los árboles de decisión son fáciles de interpretar y se utilizan en una variedad de aplicaciones empresariales y científicas [14].

El *machine learning* ha revolucionado el campo de la inteligencia artificial al permitir que las máquinas aprendan y mejoren sus capacidades a partir de datos sin intervención humana explícita. Desde sus inicios, esta tecnología ha avanzado significativamente, con el desarrollo de *deep learning* y modelos generativos que transforman cómo interactuamos con la tecnología. Los algoritmos de aprendizaje supervisado y no supervisado, como la regresión y la clasificación, ofrecen herramientas versátiles para resolver una amplia gama de problemas, desde la predicción de valores hasta la categorización de datos.

La adopción efectiva de *machine learning* en diversas industrias ha llevado a mejoras sustanciales en la eficiencia y en la toma de decisiones. No obstante, para maximizar su impacto, es crucial seleccionar los algoritmos adecuados y considerar las implicaciones éticas de su uso. A medida que la tecnología continúa evolucionando, su capacidad para enfrentar desafíos complejos y optimizar procesos seguirá impulsando avances significativos en la ciencia y la tecnología.

C. Microservicios en la nube

En la actualidad, la información se espera al instante, y más cuando se trata de una plataforma que se enfoca en brindar algún servicio, una red social o un sitio web. A medida que el tiempo avanza, las arquitecturas de los mismos escala exponencialmente y su mantenimiendo es mucho más complejo. Además, la caída de una página monolítica por cualquier situación (como por un pico de demanda) significa la pérdida total de la funcionalidad. Por estas y vairas otras razones resulta esencial el uso de los microservicios.

1. Definicion y principios

Una plataforma basada en microservicios es simple de explicar, aunque se implementación es mucho más complejo. Esta arquitectura busca generar una aplicación a partir de varios microservicios que se intercomunican para solventar algunas solucitud del usuario. Los microservicios se busca que sean lo más independientes uno del otro para que, en caso uno de ellos deje de funcionar, los demás continúan operando sin problema, asegurando una mejor experiencia de usuario.

Entre las características de los microservicios está el componente autónomo, que permite que un servicio escale para suplir la demanda sin afectar los otros servicios. Tampoco es necesario que haya código compartido entre servicios, sino que se hagan consultas estructuradas a través de la API para asegurar consistencia. Otro punto importnate es que cada servicio tiene una tarea especializada que se enfoca en resolver una sola tarea. Esto significa que, a medida de que un servicio crece y se vuelve más copmlejo se puede dividir en servicios más pequeños. Esta arquitectura también fomenta el trabajo en equipos pequeños e independientes, ayudando así a la agilidad de la producción y el proceso de cada servicio es bien comprendido. Como ya se mencionó, otro principio es la escalabilidad. Si un servicio comienza a exigir más recursos, con el uso de contenedores y orquestación de los mismos, se puede suplir esta demanda abriendo nuevas instancias del mismo. Por último, las aplicaciones basadas en microservicios son de implementación sencilla, permite libertad a los desarrolladores de utilizar distintos *framworks*, tecnologías, bases de datos, etc., se genera código reutilizable a lo largo de la aplicación y la independencia de los servicios aumenta la resistencia a errores [15].

2. Herramientas y tecnologías para la implementación de microservicios

Con la continua popularidad de la arquitectura de microservicios se han creado variedad de herramientas, cada una con sus aportes positivos. Entre las tecnologías que se abordarán tenemos todo lo relacionado a Javascript, que comprende un conjunto de librerías y *framworks*, además del uso de contenedores de Docker para crear entornos individuales para cada servicio y el Kubernetes para su orquestación.

Next JS: *server-side rendering*

Next JS es un *framewrok* de React para construir aplicaciones web modernas. Esta se ejecuta utilizando Node JS, que también es enesencial para agilizar la creación de código. Este *framewrok* junta React, una librería de Javascript muy popular (mejor explicada más adelante) y *server-side rendering* (SSR) para entregar una mejor experiencia al usuario. El SSR permite a las aplicaciones realizar todas las consultas necesarias a distintos servicios para desplegar interfaces de usuario completamente renderizadas. Next JS fue creado por la compañía Vercel, que aborda este tema en los acercamientos tradicionales, donde el explorador se enecarga de renderizar todo el contenido [16].

React JS es una librería de Javascript que permite a los desarrolladores crear interfaces de usuario con piezas individuales llamadas componentes. Estos componentes de React JS son funciones de Javascript, por lo que pueden utilizarse libremente a lo largo de la aplicación. A estos también se les puede enviar datos y que automáticamente definan qué es lo que debe aparecer en pantalla. Incluso, mientras el usuario maneja, los componentes pueden ir cambiando dependiendo de sus interacciones. Todo esto puede ser implementado utilizando el *framwork* de Next JS [17].

Docker: virtualización de contenedores

Un contenedor es un entorno de ejecución virtualizado y aislado, independiente de otros procesos que se ejecutan en la misma máquina. Estos contenedores se crean a partir de una imagen, que es una plantilla predefinida que incluye todas las dependencias necesarias, como *frameworks* y bibliotecas, para garantizar la correcta ejecución del software dentro del contenedor. Los contenedores se ejecutan dentro de *namespaces*, los cuales son útiles para organizar y gestionar grupos de contenedores dentro de un clúster.

Existen muchas imágenes disponibles en la web, como Ingress NGINX, una imagen comúnmente utilizada para equilibrar la carga de tráfico, redirigiendo solicitudes a diferentes servicios. Otra imagen utilizada dentro del proyecto es la de NATS Streaming Server, que su funcionamiento es muy parecido al del protocolo MQTT. Con esta imagen, los contenedores pueden suscribirse a ciertos temas y enviar/recibir mensajes al respecto. Sin embargo, si se requiere una configuración personalizada, es posible crear una nueva imagen desde cero utilizando un archivo de configuración, que en Docker se conoce como Dockerfile.

Docker es una plataforma ampliamente utilizada para la virtualización de contenedores, permitiendo a los desarrolladores crear y gestionar distintos entornos de forma sencilla. Entre las principales ventajas de Docker, se destaca su gratuidad, la disponibilidad de imágenes públicas a través de Docker Hub y el respaldo de una gran comunidad de desarrolladores. Además, los contenedores pueden ejecutarse en cualquier sistema operativo o infraestructura en la nube, una vez finalizado su desarrollo.

Docker ofrece compatibilidad con una amplia variedad de lenguajes de programación, como Python, JavaScript, TypeScript, Java, entre otros, lo que lo convierte en una opción muy versátil. Para los propósitos de este trabajo, Docker es la opción más adecuada debido a sus diversas propiedades y facilidades [18].

Kubernetes: orquestación de contenedores

Kubernetes es una plataforma de orquestación de contenedores que automatiza el despliegue, la gestión y el escalado de aplicaciones en contenedores. Permite a los desarrolladores y administradores gestionar aplicaciones distribuidas de manera eficiente en clústeres de servidores, asegurando alta disponibilidad y escalabilidad. Algunas de las características principales de Kubernetes son:

- **Automatización de despliegues:** Kubernetes facilita el despliegue de aplicaciones mediante *Deployments*, que permiten definir la configuración de la aplicación, el número de réplicas y las actualizaciones progresivas. Los *Deployments* aseguran que las aplicaciones se ejecuten correctamente y permiten revertir cambios en caso de errores.
- **Autoescalado:** Kubernetes puede ajustar automáticamente el número de réplicas de una aplicación en función de la carga de trabajo actual. Esta capacidad permite que las aplicaciones manejen aumentos repentinos en el tráfico sin intervención manual.
- **Balanceo de carga:** a través de los diferentes tipos de servicios que ofrece, Kubernetes facilita el balanceo de carga entre los contenedores que conforman una aplicación. Un

ejemplo es el servicio ClusterIP, que expone un conjunto de pods internamente en el clúster, proporcionando una dirección IP estática y un puerto por el cual los otros servicios pueden acceder a ellos.

- **Recuperación automática:** Kubernetes monitorea continuamente el estado de los contenedores y, si alguno falla, lo reinicia o lo reemplaza automáticamente para asegurar que la aplicación siga disponible. Esta capacidad de *self-healing* es fundamental para garantizar alta disponibilidad en entornos de producción.
- **Gestión de secretos y configuraciones:** Kubernetes permite manejar la configuración de las aplicaciones de manera dinámica mediante el uso de ConfigMaps y Secrets, los cuales separan la configuración del código de la aplicación. Esto facilita la actualización de parámetros sin necesidad de reconstruir las imágenes de los contenedores.
- **Escalabilidad horizontal:** Kubernetes facilita la escalabilidad horizontal mediante el uso de *ReplicaSets*, que aseguran que el número correcto de *pods* esté corriendo en todo momento, garantizando un rendimiento adecuado ante diferentes cargas de trabajo.
- **Servicios de red:** Kubernetes abstrae la complejidad de las redes internas a través de su modelo de servicios. Además del ClusterIP, Kubernetes ofrece otros tipos de servicios como NodePort, que expone un puerto en cada nodo del clúster, y LoadBalancer, que permite distribuir la carga a nivel de red y es comúnmente utilizado en entornos de producción.
- **Namespace:** los *namespaces* permiten dividir un clúster de Kubernetes en múltiples entornos lógicamente aislados, facilitando la gestión de recursos y la segregación de equipos o proyectos dentro de una misma infraestructura.

Kubernetes ha emergido como una herramienta clave en el desarrollo de aplicaciones basadas en microservicios, gracias a su capacidad para gestionar entornos de producción con alta eficiencia y flexibilidad. Su integración con contenedores como los de Docker permite un flujo de trabajo más ágil, reduciendo la complejidad en la operación de grandes clústeres distribuidos.

D. Microcontroladores, sensores y actuadores utilizados

1. ESP32

El microcontrolador ESP32 es un chip de 32 bits que opera a 1.024 MHz, desarrollado por Espressif Systems. Este chip ha ganado gran popularidad en el desarrollo de sistemas que requieren comunicación inalámbrica, destacándose por su versatilidad y capacidades. Su voltaje de operación es de 3.3V, aunque puede alimentarse con 5V a través de un pin dedicado o mediante puerto USB. Entre las opciones de comunicación más utilizadas se encuentran wifi y bluetooth, aunque el ESP32 es capaz de soportar otros protocolos. El módulo cuenta con un total de 36 pines GPIO con diversas funcionalidades, incluyendo alimentación, ADC, comunicación UART, I2C, SPI y PWM. Gracias a estas características,

el ESP32 se considera un componente esencial para implementar soluciones IoT de bajo costo.

El ESP32 utiliza un microprocesador Tensilica Xtensa de 32 bits, que puede alcanzar una frecuencia de 240 MHz. Además, está diseñado con un modo de bajo consumo energético, permitiendo realizar conversiones analógicas y lecturas de pines digitales incluso cuando se encuentra en el estado de *deep sleep*. Para complementar su eficiencia energética, el microcontrolador incluye bluetooth V4.2 con soporte para bluetooth *low energy* (BLE). A esto se suma su antena wifi integrada, que le permite enviar y recibir datos a través de routers y redes inalámbricas [19].

Una de las características distintivas del ESP32 es su soporte para el protocolo ESP-NOW, desarrollado por Espressif. Este protocolo permite una comunicación de largo alcance que, según Espressif, puede llegar hasta los 200 metros en campo abierto. Su arquitectura maestro-esclavo facilita una comunicación rápida y eficiente, que además puede ser encriptada para mayor seguridad. Un aspecto destacado del protocolo es su capacidad para conectar y comunicar bidireccionalmente con múltiples dispositivos de manera simultánea, haciendo del ESP32 una opción ideal para aplicaciones que requieran redes distribuidas y confiables.

El ESP32 se destaca por su flexibilidad en cuanto a modos de bajo consumo, lo que lo hace ideal para aplicaciones que requieren eficiencia energética. Entre los modos de bajo consumo se encuentran el *light sleep* y el *deep sleep*, que permiten apagar ciertos módulos internos para reducir el consumo de energía. En el modo *light sleep*, el microcontrolador suspende la CPU mientras los periféricos como el wifi y el *bluetooth* pueden seguir funcionando, lo que es útil cuando se necesita mantener conectividad pero no procesamiento activo. Por otro lado, en el modo *deep sleep*, casi todos los componentes del ESP32 se apagan, dejando activo solo un reloj de baja frecuencia y ciertos pines para despertarlo cuando sea necesario. Este modo es particularmente efectivo en aplicaciones de monitoreo o sensorización que solo requieren transmitir datos en intervalos largos.

En términos de seguridad en las comunicaciones, el ESP32 proporciona varias capas de protección para asegurar la transmisión de datos. En lo que respecta a wifi, el módulo es compatible con los estándares de encriptación WPA/WPA2, y también soporta TLS (Transport Layer Security), lo que permite la encriptación de extremo a extremo en las comunicaciones a través de Internet. Además, al utilizar el protocolo ESP-NOW, es posible encriptar la comunicación entre dispositivos, lo que añade una capa adicional de seguridad en redes locales. Estas características hacen del ESP32 una solución confiable para aplicaciones que requieren el envío de información sensible, como el monitoreo remoto de infraestructuras o sistemas de control industrial.

El soporte para sensores y actuadores es otro de los puntos fuertes del ESP32, ya que su capacidad de manejar múltiples interfaces de comunicación lo convierte en un microcontrolador ideal para proyectos de automatización y sensorización. Gracias a sus pines GPIO, es posible conectar una amplia variedad de sensores que midan parámetros como temperatura, humedad, presión, luminosidad, entre otros. Además, su compatibilidad con protocolos como I2C, SPI y UART permite la conexión de módulos más avanzados, como cámaras o pantallas. De la misma manera, el ESP32 puede controlar actuadores como relés, motores y luces, lo que lo convierte en un componente central en aplicaciones IoT para el hogar inteligente o en sistemas industriales automatizados [20].

2. Raspberry Pi

La Raspberry Pi 4 es una computadora de bajo costo diseñada para fomentar el aprendizaje en el ámbito de la programación y otras disciplinas relacionadas. Aunque fue concebida originalmente para la enseñanza secundaria, sus capacidades rápidamente la convirtieron en una plataforma popular a nivel global, con aplicaciones en educación, investigación y desarrollo de proyectos tecnológicos. Esta placa ha sido y sigue siendo desarrollada por la Raspberry Pi Foundation. A pesar de que la versión más reciente es la Raspberry Pi 5, cuyas unidades aún son limitadas, para este proyecto se utilizará el modelo Raspberry Pi 4 Model B.

Cada versión de la Raspberry Pi cuenta con múltiples modelos que ofrecen mejoras progresivas y capacidades adaptadas a las necesidades del usuario. En el caso de la Raspberry Pi 4 Model B, esta placa está equipada con un procesador ARM Cortex-A72 de 64 bits que opera a 1.8 GHz, y puede adquirirse con 2GB, 4GB o 8GB de memoria RAM LPDDR4-3200, lo que proporciona un excelente rendimiento para una variedad de aplicaciones. Además, incluye una amplia gama de puertos como Gigabit Ethernet, dos puertos micro-HDMI, dos puertos USB 2.0 y dos USB 3.0, lo que permite una conectividad versátil tanto para dispositivos externos como para redes locales. Entre sus características inalámbricas se encuentran bluetooth y wifi, lo que facilita su integración en redes y proyectos IoT.

Una de las principales ventajas de la Raspberry Pi 4 es su conjunto de pines GPIO, que permiten la conexión directa y el control de una amplia variedad de sensores, actuadores y módulos, lo que la convierte en una plataforma excelente para proyectos de automatización, robótica y monitoreo. Adicionalmente, la placa viene preinstalada con un sistema operativo basado en Linux, llamado Raspberry Pi OS, aunque es posible instalar otros sistemas operativos compatibles, dependiendo de las necesidades del proyecto. Gracias a sus capacidades de procesamiento, su bajo costo y su portabilidad, la Raspberry Pi se ha consolidado como una solución ideal en múltiples escenarios, desde el desarrollo educativo hasta la investigación científica y la creación de prototipos tecnológicos [21].

Además de sus capacidades como plataforma educativa y de desarrollo, la Raspberry Pi 4 ha demostrado ser una herramienta poderosa en el ámbito de la inteligencia artificial (IA) y machine learning (ML). Con soporte para bibliotecas como TensorFlow, Keras y PyTorch, es posible desarrollar e implementar modelos de aprendizaje profundo y otras técnicas de IA directamente en la Raspberry Pi. A pesar de sus limitadas capacidades en comparación con sistemas más robustos, es adecuada para ejecutar modelos preentrenados o realizar inferencias ligeras en aplicaciones como el reconocimiento facial, clasificación de imágenes y procesamiento de lenguaje natural (NLP). El bajo costo y su capacidad para integrarse con otros módulos la convierten en una opción atractiva para prototipos y soluciones de IA de bajo consumo energético.

En el campo del *machine learning*, la Raspberry Pi 4 es útil para el procesamiento de datos en tiempo real en aplicaciones industriales y de monitoreo. Es posible entrenar modelos en sistemas más potentes y luego desplegarlos en la Raspberry Pi para que se encargue de tareas como la predicción de fallos en equipos, la optimización de procesos o la toma de decisiones autónomas en sistemas robotizados. Su capacidad para manejar redes neuronales ligeras permite su integración en proyectos de *edge computing*, donde los datos se procesan directamente en el dispositivo en lugar de depender de servidores remotos.

Por otro lado, el papel de la Raspberry Pi 4 en el desarrollo de soluciones para el internet de las cosas (IoT) es fundamental. Gracias a sus módulos integrados de wifi y bluetooth, y sus pines GPIO, la placa es ideal para conectarse y gestionar una amplia gama de sensores y actuadores. En aplicaciones como el monitoreo ambiental, domótica, o sistemas de riego automatizados, la Raspberry Pi 4 actúa como un nodo central que recopila datos de sensores, los procesa y envía la información a la nube o toma decisiones locales. Su capacidad para ejecutar servidores web y aplicaciones de base de datos también permite que se convierta en el controlador principal de sistemas IoT, manejando comunicaciones, almacenamiento y análisis de datos de forma eficiente.

3. Sensor de temperatura y humedad ambiental: DHT22

El sensor DHT22 es un dispositivo ampliamente utilizado en proyectos electrónicos para el control ambiental. Su capacidad para medir tanto la temperatura como la humedad relativa del aire lo convierte en una herramienta esencial para monitorear y mantener condiciones óptimas en diversos entornos. Este sensor es conocido por su fiabilidad y facilidad de uso, lo que lo hace ideal para aplicaciones en sistemas de climatización, agricultura inteligente y otros proyectos de control ambiental.

Figura 1: Módulo sensor de temperatura y humedad DHT22

4. Sensor de luz ambiental: VEML7700

El sensor VEML7700 es un sensor de luz ambiental que mide la intensidad de la luz en su entorno. Este sensor es crucial para aplicaciones que requieren ajustar automáticamente la iluminación en función de la luz disponible, como sistemas de iluminación inteligente, pantallas de dispositivos electrónicos que ajustan el brillo automáticamente y otras aplicaciones que dependen de la detección precisa de la luz ambiental. Su capacidad para ofrecer mediciones precisas lo convierte en una opción confiable para una amplia gama de proyectos.

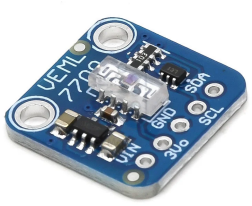

Figura 2: Módulo sensor de distancia JSN-SR04T

5. Sensor de distancia: JSN-SR04T

El JSN-SR04T-V3.0 es un sensor de distancia ultrasónico que se utiliza para medir distancias sin contacto físico. Este sensor emite ondas ultrasónicas y mide el tiempo que tardan en rebotar en un objeto y regresar. Es ideal para aplicaciones como la detección de obstáculos en robots, la medición de nivel de líquidos en tanques y otros proyectos que requieren mediciones precisas de distancia. Su capacidad de operar en diversas condiciones ambientales lo hace especialmente útil en entornos exteriores e industriales.

Figura 3: Módulo sensor de distancia JSN-SR04T

6. Sensor de efecto Hall: KY-003

El módulo KY-003 es un sensor magnético basado en un interruptor Hall, que detecta la presencia de campos magnéticos. Es utilizado en aplicaciones que requieren la detección de la proximidad y la posición de objetos magnéticos, como en contadores de velocidad, detectores de posición y otros sistemas de detección de proximidad. Este sensor es fácil de integrar en diversos proyectos electrónicos y es ideal para aplicaciones en automatización industrial, sistemas de seguridad y proyectos de robótica. En este caso, se elaborará con este sensor un medidor de precipitación.

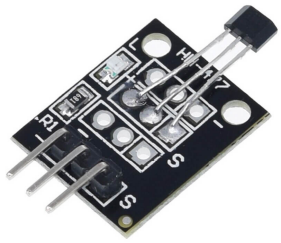

Figura 4: Módulo sensor magnético KY-003

E. Cutlivos hidropónicos

Actualmente se está popularizando dentro de la industria agrícola los sistemas hidropónicos. Lo que busca principalmente este método de cultivo es dejar a un lado el cultivo directamente en el suelo y utilizar otras formas de llevar los nutrientes esenciales a las raíces de las plantas. Su objetivo principal es poder convertir espacios no convencionales en áreas de cultivo, sin dejar a un lado las necesidad de la planta como lo son luz, temperatura, agua y los nutrientes esenciales. Estos nutrientes se suplen a las plantas por medio de una solución nutritiva que se forma principalmente de fósforo, potasio y nitrógeno.

Dentro de la hidroponía se utilizan distintos tipos de riego, los cuales se mencionaron anteriormente. Estos son por medio de inundación y por goteo. En el primer método se utiliza un tanque principal con la solución nutritiva con el cual se llena un tanque secundario o una red de tuberías hasta que esta se inunde. Luego de cierto tiempo, el agua retorna al tanque principal para ser utilizada nuevamente y así disminuir el consumo. Como se puede observar en la Figura 5, el funcionamiento únicamente consta de dos bombas, una de agua y otra de aire. La de agua se encarga principalmente de asegurar el flujo constante de la solución nutritiva, mientras que la de aire tiene el fin de oxigenar el agua que retorna para asegurar una mejor calidad de la misma. Este método es bastante eficiente en cuanto al consumo de recursos, aunque requiere de grandes espacios donde se puedan colocar estos sistemas horizontales.

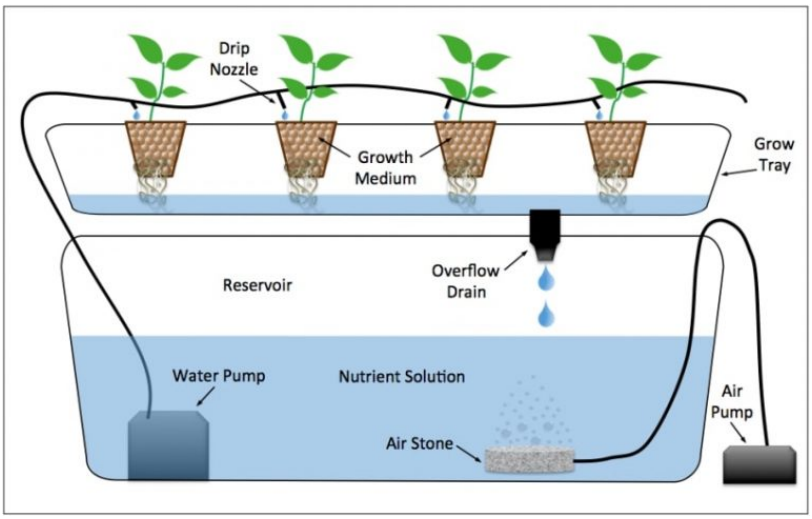

Figura 5: Sistema hidropónico por inundación

El segundo método es utilizando el método de goteo. En este método se utilizan armazones verticales donde se tiene un suministro en la parte inferior. Por medio de una bomba de agua, la solución se lleva a un deposito superior de donde la solución va goteando sobre las raíces. Además de que utiliza el riego por goteo que es más controlado, esta se puede realizar de forma vertical, de modo que se abre la posibilidad de utilizar menos espacios para desarrollar más plantas. Estos sistemas también se conocen por ser modulares, ya que pueden colocarse en torre cuantas plantas se desee. Un ejemplo se muestra en la Figura 6. Por sus vent qajas en cuanto al uso de espacios reducidos y la capacidad de utilizar riego por goteo se propone la realización de un sistema automatizado para el cultivo de plantas en hidroponias utilizando el método de goteo, implementando distintos sensores juntos con *machine learning* e IoT para hacer un sistema más inteligente, eficiente y eficaz.

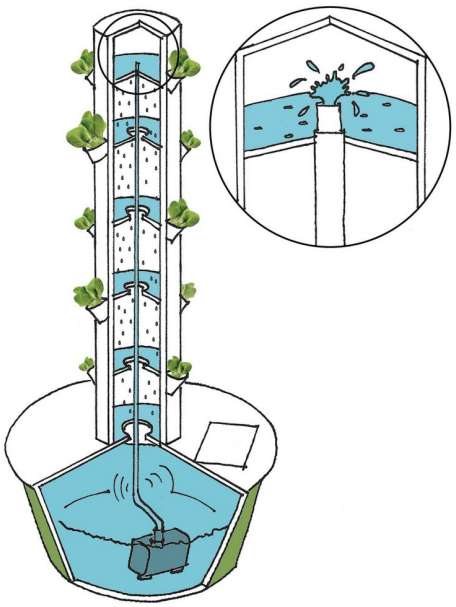

Figura 6: Sistema hidropónico por goteo vertical

CAPÍTULO VII

Construcción de la estructura

A. Construcción de la estructura

El proyecto está diseñado para operar en un entorno abierto, donde estará en constante contacto con tierra, aire y lluvia. Por temas de tiempo y recursos, se realizará la iteración de este prototipo para uso en interiores, aunque el objetivo anterior se mantendrá durante todo el desarrollo y fabricación de la estructura. Debido a las condiciones mencionadas al principio, es fundamental que el diseño considere la impermeabilidad, evite la deformación causada por la luz solar y resista diversas condiciones climáticas sin deteriorarse. La robustez y durabilidad del proyecto son esenciales para su funcionamiento óptimo a largo plazo. Para superar estos desafíos, es crucial centrarse en tres aspectos principales: la selección de materiales, el diseño de las piezas y el proceso de ensamblaje. La elección de materiales adecuados garantizará que los componentes puedan soportar las condiciones ambientales adversas. Un diseño de piezas bien pensado evitará deformaciones y daños y facilitará el acople entre ellas, mientras que un proceso de ensamblaje meticuloso asegurará que la estructura alcance todos los objetivos. Esta sección se dividirá en los tres apartados mencionados: selección de materiales, diseño de piezas y proceso de ensamblaje. Cada apartado detallará las consideraciones y estrategias específicas adoptadas para asegurar la resistencia y funcionalidad de la estructura en un entorno abierto.

1. Selección de materiales

Para esta parte se estudiaran las propiedades de los materiales disponibles para evaluar la utilidad que pueden tener estos en la estructura. El material preferido es el plástico debido a la fácil disponibilidad y su accesiblidad. Sin embargo, este material viene en varias presentaciones, cada una con propiedades distintas. Se combinarán diferentes tipos de materiales

según el propósito de cada pieza.

Para comenzar, consideramos el cuerpo de la estructura. Para lograr una orientación vertical, se optara por un cuerpo cilíndrico, lo que también facilita el acople de los módulos y mantiene una simetría. Por estas razones y las mencionadas anteriormente, se utilizarán tubos de PVC, un material robusto y resistente. El PVC se utiliza en aplciaciones de transporte de liquidos bajo tierra, condiciones incluso más exigentes que las requeridas para este proyecto. Entre sus características destacan las siguientes:

- Su vida útil puede sobrepasar los 50 años.
- Contiene estabilizadores UV que reducen la degradación y la decoloración por la luz solar.
- Requiere un mantenimiento mínimo en comparación a otros materiales como la madera, que necesitan de pintado y sellado de forma regular.
- Esta comprobado que este material absorbe menos del 0.04 % del líquido que circula.
- Tiene una baja conductividad térmica (alrededor de 0.19 $W/m * K$), significativamente menor que la de otros materiales como los metales. Además, no se deforma en temperaturas desde los $-30°C$ hasta los $70°C$.
- Es ligero, con una densidad de 1.4 g/cm^3.
- Es más económico que otros materiales que podrían sustituirlo.
- Es un termioplástico, lo que significa que es fácilmente deformable con altas temperaturas.
- Es útil en aplicacionies de máxima higiene, como el transporte de agua potable.
- La facilidad del reciclaje del PVC lo hacen un material sostenible, además de que se producen bajas emisiones de CO2 en su fabricación

En el desarrollo de este proyecto, se empleará la tecnología de impresión 3D por extrusión utilizando el material PETG (polietileno tereftalato glicol). La impresión 3D por extrusión, también conocida como FDM (Modelado por Deposición Fundida), es una técnica avanzada que permite la creación de piezas tridimensionales mediante la deposición de capas sucesivas de material fundido. Este método es ideal para producir componentes personalizados con alta precisión y complejidad geométrica, lo que resulta especialmente útil en el diseño y fabricación de piezas específicas para la estructura del proyecto.

El PETG ha sido seleccionado como el material principal para la impresión 3D debido a sus excelentes propiedades mecánicas y térmicas. El PETG combina la durabilidad del ABS con la facilidad de impresión del PLA, ofreciendo una resistencia superior a impactos y a la tracción, lo que garantiza la robustez de las piezas impresas. Además, el PETG es altamente resistente a productos químicos y a la humedad, lo que lo hace adecuado para aplicaciones en exteriores y en ambientes exigentes, sin mencionar que se utilizarán soluciones químicas para entregar nutrientes a los cultivos. Su baja contracción y deformación durante el enfriamiento aseguran que las piezas mantengan sus dimensiones y forma con precisión, lo que es crucial para el ensamblaje y funcionamiento del proyecto.

2. Diseño de piezas

La estructura debe cumplir con varios objetivos. Primero, debe ser modular, lo que significa que puede armarse y desarmarse como desee el usuario. Además, permite modificar la altura dependiendo de las necesidades particulares. Debido a esto, la estructura debe ser de fácil acople. Se estudiarán los diferentes métodos de acople para emplear en la estructura. Por último, debe ser robusta y sostenerse firmemente sin importar las condiciones climáticas. Esta última característica se buscará al momento de diseñar la base de la estructura y las tolerancias de los acoples.

Para comenzar, utilizando una broca sierra, se abrieron agujeros al costado del tubo de forma alternante para maximizar el área utilizable. Este proceso se realizó con un taladro de banco junto con una prensa para asegurar la pieza y realizar el corte más limpio posible. Posterior a realizar los agujeros, se lijaron los cortes para luego colocar las piezas que sostendrán los recipientes en su lugar. Por motivos de estética, se aplicó una capa de pintura primer en aerosol de color blanco para asegurar que la estructura reflejará la mayor cantidad de la luz solar y se mantendrá una temperatura favorable para el crecimiento de los cultivos (Figura 7).

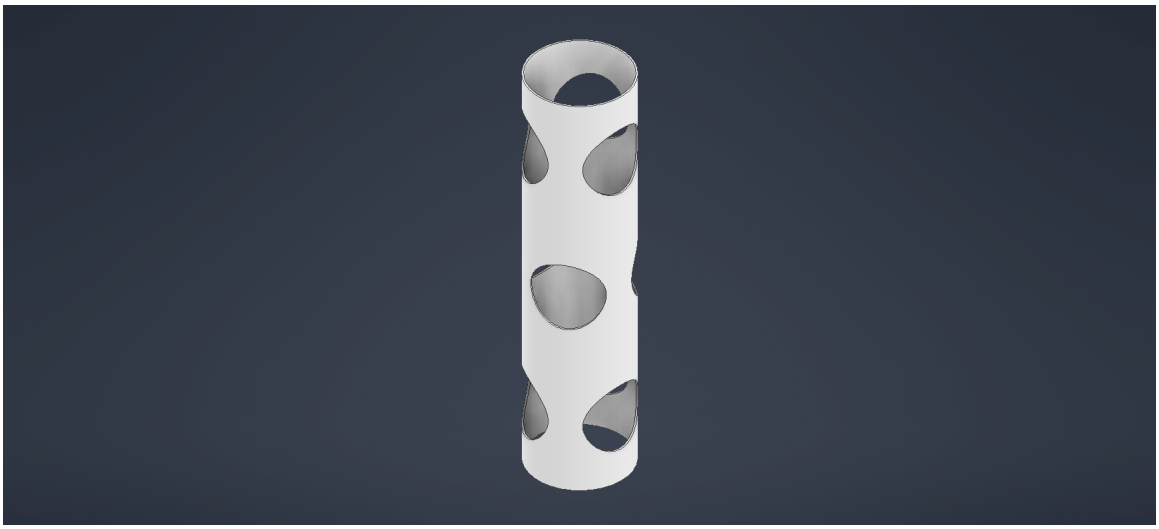

Figura 7: Módulo PVC para el cuerpo central de la estructura

Luego, se procedió al diseño en 3D de las piezas más específicas. Utilizando un Bernier, se midieron los diámetros exterior e interior del tubo para crear las primeras piezas que servirán para el acople de los módulos. Estos utilizarán un tipo de unión por deslizamiento, en la que una pieza se desliza dentro de la otra y al girar estas piezas quedan aseguradas entre sí. La pieza principal (Figura 8) está pensada para ubicarse en la parte inferior de cada módulo y la secundaria (Figura 9) en la parte superior del modulo. La parte primaria tiene las rejillas en donde entran los seguros de la parte secundaria. Además, la parte primaria posee agujeros disitrbuidos para asegurar un riego uniforme sobre las raices de los cultivos. A continuación se observan las piezas mencionadas.

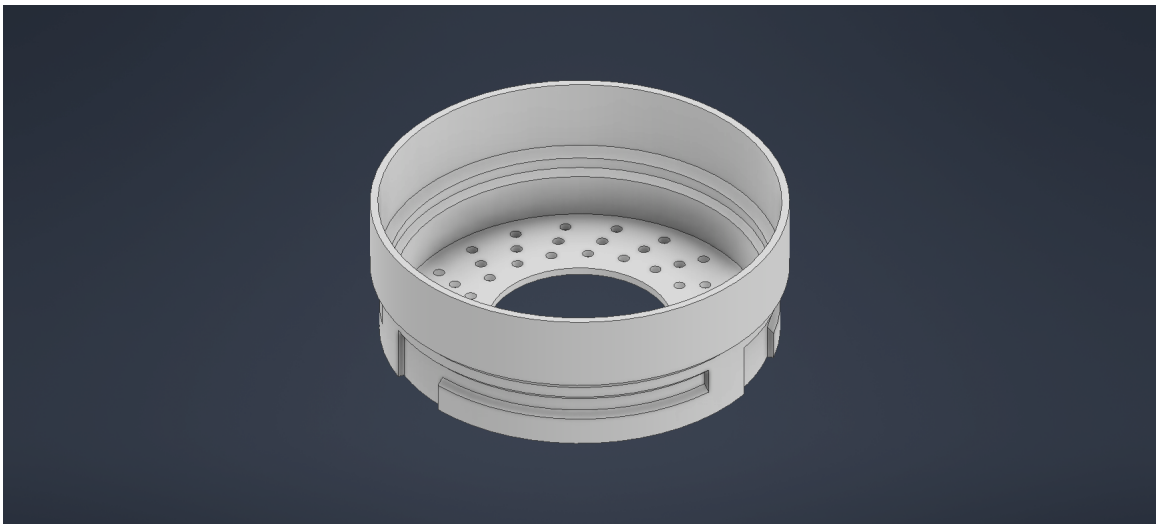

Figura 8: Acople modular primario

Figura 9: Acople modular secundario

Con la estructura y sus conexiones ya definidas, se procedió a diseñar un sistema que permita asegurar el tubo de PVC interior (Figura 10), el cual transporta la solución nutritiva hasta la corona de la torre. Este diseño debía cumplir tres requisitos clave: el tubo debía permanecer firmemente en su lugar, permitir su fácil remoción para facilitar la limpieza de raíces y sustrato, y no comprometer la presión de la solución nutritiva durante su paso por el sistema.

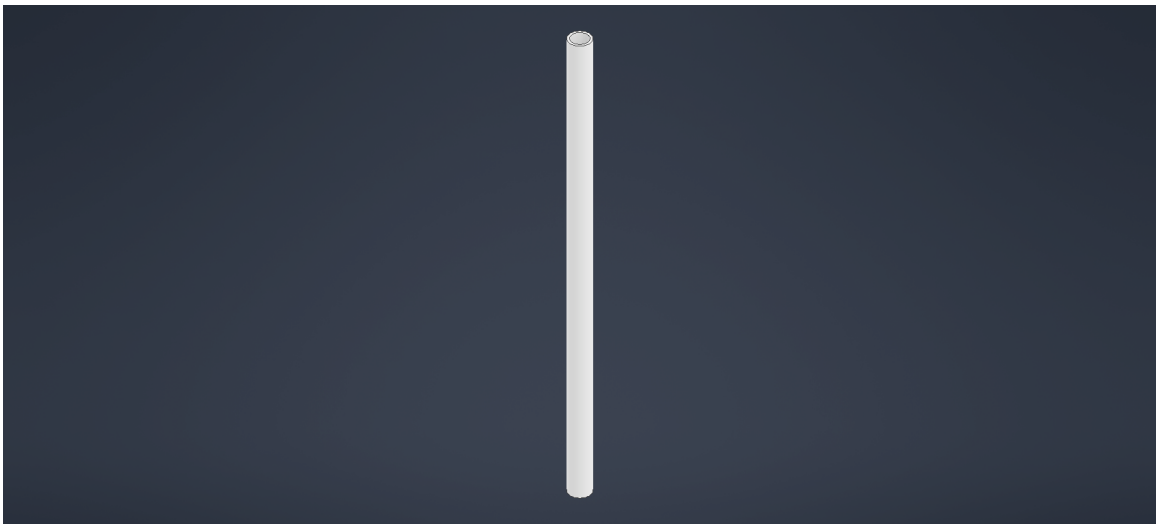

Figura 10: Tubo central PVC

Para cumplir con estas especificaciones, se diseñaron piezas personalizadas que conectan el tubo interior a los acoples primario y secundario mencionados previamente. El acople del tubo interior primario, mostrado en la Figura 11, incluye agujeros para su conexión con el acople secundario (Figura 12). Además, este acople está equipado con tres dientes que aseguran el tubo en su posición, evitando movimientos indeseados, mientras permiten una fácil extracción cuando sea necesario. Por último, para agregar aun más soporte al tubo central se diseñó la pieza en la Figura 13, que posee tres alas movibles para permitir el fácil acceso dentro del módulo, al mismo tiempo que sostiene el tubo central firmemente.

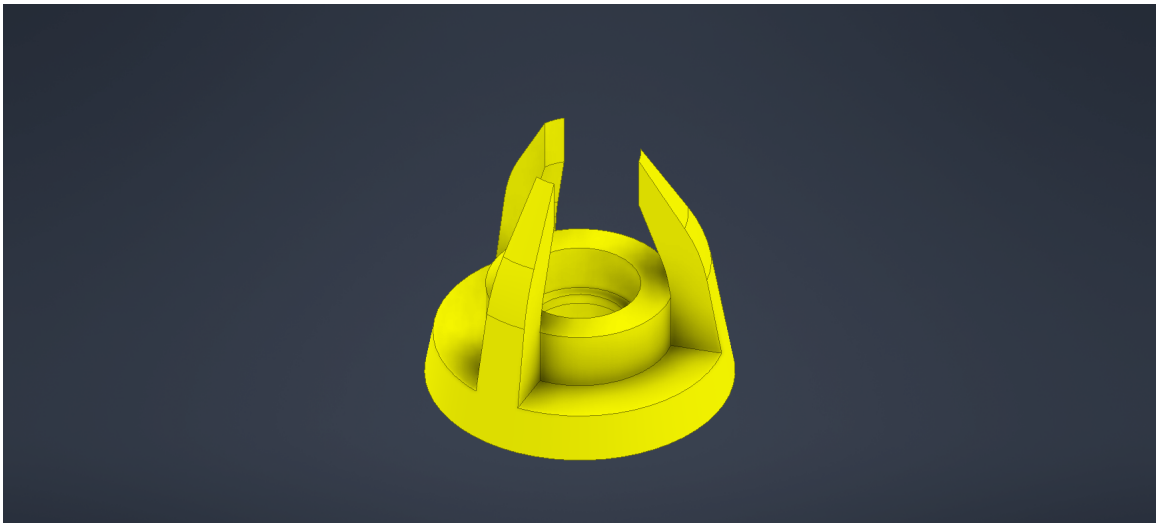

Figura 11: Fijador del tubo interior primario adherido al acople modular primario

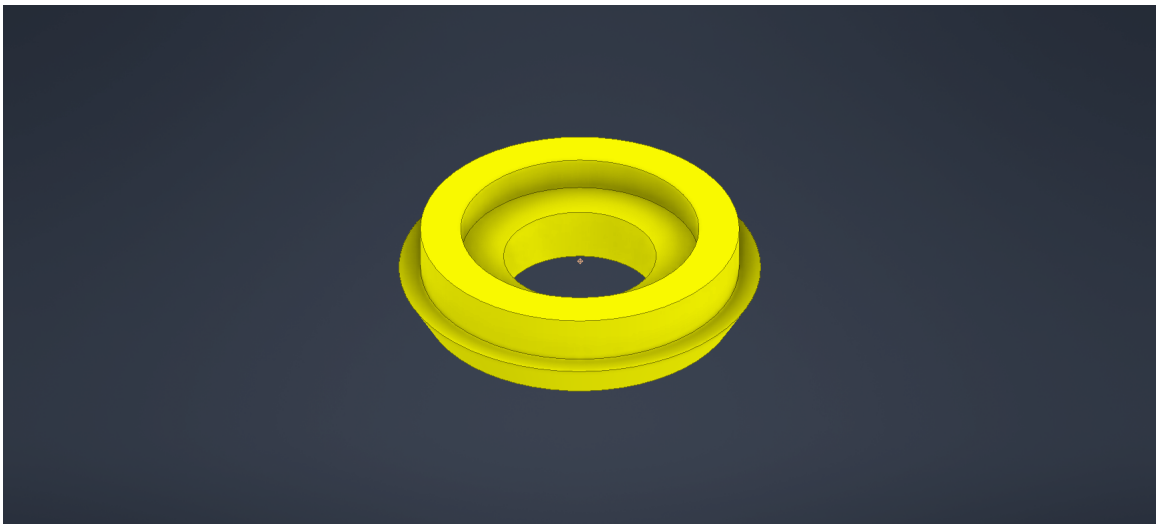

Figura 12: Fijador del tubo interior secundario adherido al acople modular primario

Figura 13: Fijador del tubo central con 3 alas movibles.

Al momento de diseñar la base de la estructura se tomó en consideración que debe ser pesada y resistente, ya que puede llegar a medir 2 metros la estructura. La base se dividió en dos grupos de piezas, el cuerpo de la base y las patas. La base se buscará que sea lo suficientemente robusta, aumentando el *infill* en la impresión. También se buscará hacer un compartimiento para colocar plasticina o algún material que pueda agregar peso a la estructura para agregar estabilidad (Figura 14). Las patas se hicieron largas y se dejaron agujeros para atornillas a la tapa del contenedor de la solución (Figura 15).

Al diseñar la base de la estructura, se tomó en cuenta que debía ser lo suficientemente pesada y resistente para soportar una torre de hasta 2 metros de altura. Para lograr esto, se dividió la base en dos conjuntos principales: el cuerpo de la base y las patas. El cuerpo de la base se diseñó para ser robusto, optimizando el *infill* durante la impresión 3D para aumentar su resistencia. Además, se añadió un compartimiento que permite colocar plastilina u otro

material que añada peso a la estructura, proporcionando mayor estabilidad (Figura 14).

Figura 14: Cuerpo de la base

Las patas fueron diseñadas para ser largas, con el fin de aumentar el área de apoyo y mejorar la estabilidad. Se incluyeron también agujeros en las mismas para atornillarlas de manera segura a la tapa del contenedor de la solución nutritiva (Figura 15), asegurando una fijación firme a la base.

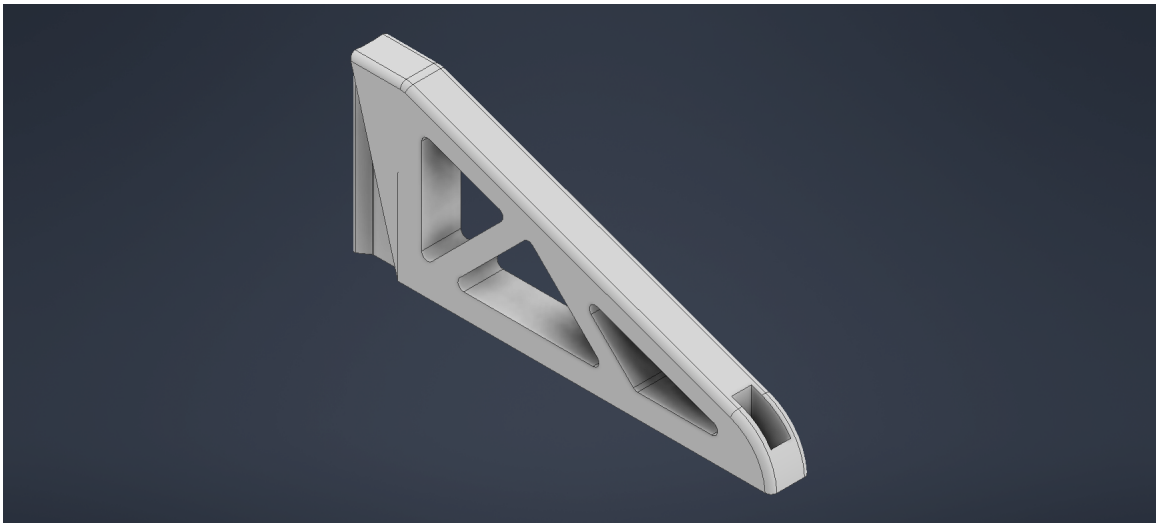

Figura 15: Patas de la base

La parte inferior se dividirá en tres piezas principales, sin contar el contenedor de plástico. La primera es la tapadera, que deberá ser rígida y lo suficientemente amplia para soportar la torre sobre el contenedor. A continuación, se diseñarán apoyos internos para colocar dentro del contenedor, de manera que la tapadera pueda descansar de forma segura sobre él. Por último, se fabricarán bases para las barras LED, utilizando tubos de PVC más delgados. La combinación de estas tres piezas, junto con los tubos de PVC, tornillos y el contenedor de

plástico, completará el desarrollo de las piezas individuales. A partir de aquí, se procederá al ensamblaje de la torre, el cual se diseñó para ser sencillo y eficiente.

3. Proceso de ensamble

Debido a que todos los elementos son modulares, todas las piezas encajaron fácilmente y sin complicaciones. Para armar los módulos, se utilizaron pegamentos como Super Bonder y epoxi para asegurar la impermeabilidad de la estructura. En cada módulo se colocó un acople modular primario en la parte inferior y uno secundario en la superior. Luego, se instalaron los sujetadores de recipientes en cada uno de los orificios del cuerpo del módulo. En el acople primario, se ensambló el sujetador del tubo central utilizando ambas partes previamente mencionadas. Además, justo antes de introducir el acople secundario, se agregó el fijador del tubo central. El resultado fue un módulo como se muestra en el dibujo CAD de la Figura 16. Asimismo, en la Figura 17 se presenta el módulo construido.

Figura 16: Módulo armado en Inventor

Figura 17: Módulo aramado

Luego de ensamblar los módulos, se procedió a armar la base y la corona de la torre. Estas no requirieron ningún tipo de pegamento, ya que su unión se realizó mediante los acoples previamente mencionados. Para la base, se ensamblaron las patas y el cuerpo de la base deslizándolos hasta que encajaron en su lugar, como se muestra en la imagen 3D de la Figura 18 y en la Figura 19.

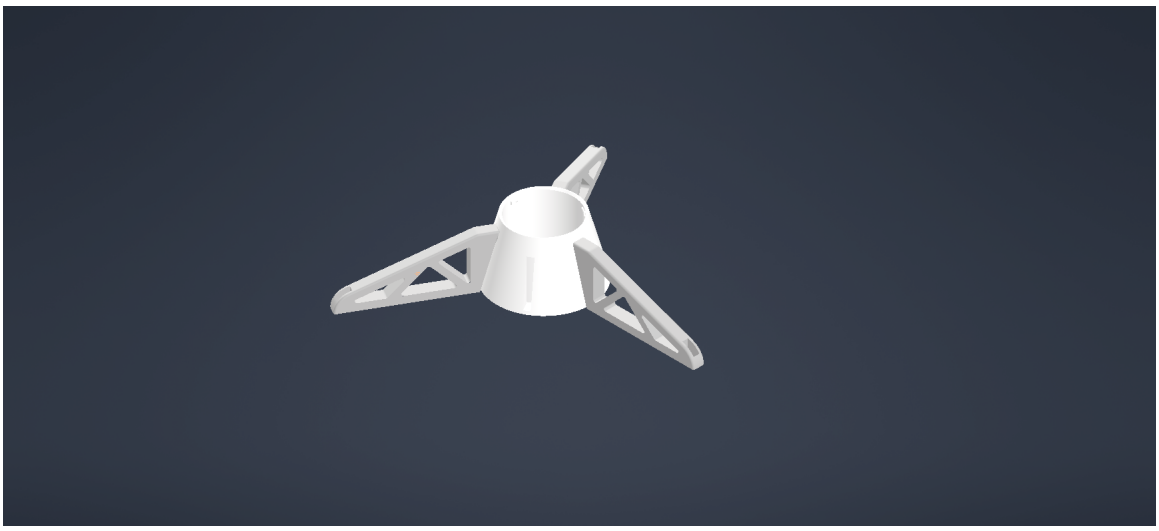

Figura 18: Base armada en Inventor

Figura 19: Base armada

La corona de la torre se acopló únicamente uniendo la tapa al cuerpo de la corona. Esta contiene aros para permitir sujetar elementos en el futuro, si así se desea. Asimismo, las fotografías tanto del diseño en 3D como de la pieza real se presentan en las figuras 20 y 21, respectivamente.

Figura 20: Corona armada en Inventor

Figura 21: Corona armada

Estos tres componentes se unieron igualmente por medio de sus acoples. El único procedimiento es encajar y luego girar las piezas para asegurarlas en su lugar. El resultado de esta sección se muestra en la Figura 22.

Figura 22: Torre sin contenedor armada

Esta estructura luego se colocó sobre una tapadera de acrílico blanco, sostenida por piezas atornilladas a un contenedor. El resultado final de toda la construcción se observa en la Figura 23.

Figura 23: Torre con contenedor armada

CAPÍTULO VIII

API basada en microservicios

Una parte fundamental de este proyecto de graduación es desarrollar una API basada en microservicios para monitorear y controlar torres hidropónicas. Entre los requisitos clave, la aplicación debe poder mostrar los dispositivos, sus estados (como variables ambientales, estado de conexión y características), definir ciclos de iluminación y riego, así como activar de manera instantánea la bomba de agua y ajustar el nivel de iluminación de las barras LED en los cultivos.

Dividir la API en microservicios ofrece varias ventajas, como la capacidad de escalar según el tráfico variable, facilitar el mantenimiento y añadir nuevas funcionalidades de manera más ágil. Además, tener las funcionalidades separadas y no depender de un solo sistema permite que, en caso de falla de un microservicio, el resto de la API permanezca operativa mientras se reinicia el servicio afectado.

A. Arquitectura, configuración y despliegue general de microservicios

La API está compuesta por varios microservicios desarrollados en Node JS, empleando tanto JavaScript como TypeScript. Este último aporta el beneficio de un sistema de tipos estáticos, lo cual reduce errores y hace que el código sea más fácil de leer y mantener. Para el manejo de rutas y la validación de datos recibidos, se usa el framework Express, que también facilita la construcción de APIs ligeras y eficientes.

Como ya se mencionó, cada microservicio se despliega en un contenedor independiente utilizando Docker. Cada contenedor tiene un archivo `Dockerfile`, que es un *script* de configuración que define todos los pasos necesarios para construir el contenedor de cada microservicio. Dentro del `Dockerfile` se especifican:

- La `imagen base` de Docker que se utilizará (por ejemplo, node:14-alpine para aplicaciones ligeras).
- Los archivos de la aplicación que deben copiarse al contenedor, por ejemplo, el código fuente y archivos de configuración necesarios.
- Las dependencias adicionales y comandos de `npm` o `yarn` para instalar los módulos necesarios.
- El puerto de exposición para la API o servicio, permitiendo que el contenedor escuche y responda a las solicitudes en el puerto designado.
- El comando de `CMD` para ejecutar el servicio (como `npm start`), asegurando que la aplicación inicie cuando el contenedor esté activo.

Además de los `Dockerfile` individuales para cada microservicio, la infraestructura de la API se define mediante Kubernetes (K8s) utilizando archivos de configuración en formato `.yaml`. Estos archivos especifican las configuraciones esenciales para el despliegue y la gestión de contenedores. En cada archivo `.yaml` se encuentran configuraciones para:

- Deployment: define el *deployment* de cada microservicio, incluyendo la cantidad de réplicas, la estrategia de actualización y la imagen Docker a utilizar.
- Servicio (*service*): configura la comunicación entre los microservicios. Por ejemplo, `ClusterIP` permite la comunicación interna dentro del clúster, especificando el puerto y protocolo de comunicación.
- Variables de entorno: permite definir valores necesarios para la configuración de los microservicios, como rutas de base de datos o credenciales.
- Secretos: gestiona información sensible como claves API y contraseñas. Kubernetes los almacena de forma segura y los inyecta en los contenedores a través de variables de entorno.

Los archivos de Kubernetes organizan y gestionan la infraestructura de la API, permitiendo que cada microservicio escale de forma independiente según la demanda y proporcionando una comunicación efectiva y segura entre ellos.

En cada microservicio también se incluye un archivo `package.json` que contiene todas las dependencias necesarias, además de información clave sobre la API, como la versión, el autor, y scripts ejecutables definidos para simplificar tareas comunes con `npm`. Entre los comandos configurados se encuentran los scripts `start` (para iniciar la aplicación), `build` (para construir el proyecto), `pub` (para publicar librerías en NPM) y `test` (para ejecutar pruebas). Todo el código fuente de cada microservicio está organizado en la carpeta `src`, lo que facilita una estructura consistente y permite una fácil navegación y mantenimiento del código.

B. Microservicios con imágenes en Dockerhub

1. Microservicio de ingress NGINX

Ingress NGINX es un controlador de ingreso para Kubernetes que utiliza el servidor web NGINX para gestionar y enrutar el tráfico de red hacia los diferentes servicios dentro de un clúster. Su función principal es actuar como un punto de entrada para las solicitudes HTTP y HTTPS, controlando cómo se distribuyen y acceden los servicios internos de Kubernetes. Ingress NGINX permite definir y gestionar reglas de enrutamiento HTTP, proporcionando funcionalidades esenciales como enrutamiento basado en URL, balanceo de carga, terminación SSL/TLS, y autenticación y seguridad.

El uso de ingress NGINX es particularmente valioso en aplicaciones multicomponente, donde se compone de múltiples microservicios (por ejemplo, un frontend que se comunica con varios servicios backend). En estos casos, ingress NGINX permite enrutar las solicitudes a diferentes servicios dependiendo del dominio o la ruta de la URL, lo que facilita una arquitectura más modular y escalable. Además, en escenarios donde algunos servicios deben ser accesibles desde internet y otros no, ingress NGINX permite definir reglas de acceso específicas, asegurando que solo las partes adecuadas de la aplicación se expongan al público.

En el contexto de aplicaciones de microservicios, ingress NGINX juega un rol crítico como puerta de enlace para gestionar el acceso externo a los servicios internos de Kubernetes. Al proporcionar un único punto de entrada, simplifica el acceso y reduce la complejidad de configuración, centralizando también la administración de seguridad. Esto es especialmente útil en sistemas que requieren balanceo de carga y protección SSL/TLS, ya que ingress NGINX facilita la gestión de certificados SSL de forma centralizada y distribuye el tráfico entrante entre las instancias de cada servicio. De este modo, ingress NGINX no solo mejora la eficiencia del uso de recursos, sino que también asegura una alta disponibilidad para los usuarios de la aplicación [22].

2. Microservicio de Nats Streaming Server

NATS Streaming Server es una imagen disponible en Docker Hub que actúa como un bus de eventos, esencial para el funcionamiento de aplicaciones basadas en microservicios. Esta imagen implementa un servicio de tipo publicación-suscripción, donde se encuentran tanto *listeners* como *publishers* que se suscriben a distintos "temas"para la comunicación de eventos, similar a lo que ofrece MQTT. Esta arquitectura permite que los microservicios sincronicen eficazmente sus bases de datos, dado que cada uno de ellos es individual e independiente. Además, NATS agrega funcionalidades que garantizan que todos los eventos se procesen correctamente.

Para implementar NATS Streaming Server (en adelante, NATS), es necesario configurar un cliente por microservicio. Este cliente existe de forma global dentro del mismo y se utiliza cada vez que se desea publicar o escuchar un evento. Para facilitar la implementación, se creó un archivo llamado `nats-wrapper.ts`, que contiene la configuración inicial del objeto. Este objeto se inicializa una vez que la imagen se levanta (en el archivo `index.ts`), permitiendo su utilización en la comunicación dentro de los archivos ubicados en la carpeta

events de cada microservicio.

Un ejemplo de funcionamiento, que se ampliará más adelante en otra sección, ocurre al crear un dispositivo. Cuando un dispositivo se registra mediante el microservicio de *devices*, se publica un evento de tipo `device:registered`, que actualiza las bases de datos de los microservicios que requieren esta información para funcionar, como el microservicio de *commands*. Esta base de datos contiene información esencial, como la IP, el ID, el nombre y otros datos necesarios para enviar órdenes al dispositivo.

C. Microservicios con imágenes propias

1. Microservicio de autenticación

El microservicio de autenticación de usuarios se encarga de gestionar todas las operaciones relacionadas con los usuarios en el sistema. Entre las funcionalidades principales, se incluyen la creación de usuarios, autenticación, cierre de sesión y verificación de cookies del usuario actual. Además, proporciona servicios para la creación y validación de *tokens* de verificación de correo electrónico durante el registro, así como *tokens* para restablecer contraseñas olvidadas.

A continuación, se describen los modelos utilizados en el microservicio de autenticación, los cuales incluyen dos entidades clave para gestionar las funcionalidades mencionadas.

Cuadro 1: Modelo copia de token

Nombre	**Tipo de dato**	**Descripción**
value	cadena	Representa el valor del *token* generado para el usuario.
userId	cadena	Identificador del usuario asociado al *token*.
createdAt	fecha	Fecha y hora de creación del *token*.
expiresAt	fecha	Fecha y hora de expiración del *token*.
used	booleano	Indica si el *token* ha sido utilizado. Por defecto, es `false`.
version	número	Número de versión del documento, utilizado para controlar la concurrencia.

Cuadro 2: Modelo de usuario

Nombre	Tipo de Dato	Descripción
firstName	Cadena	Primer nombre del usuario, campo obligatorio.
lastName	Cadena	Apellido del usuario, campo obligatorio.
username	Cadena	Nombre de usuario único, requerido para la identificación del usuario.
email	Cadena	Dirección de correo electrónico única del usuario, campo obligatorio.
password	Cadena	Contraseña del usuario, almacenada como hash para mayor seguridad.
verified	Booleano	Indica si el usuario ha verificado su cuenta, con un valor predeterminado de `false`.
version	Número	Número de versión del documento, usado para control de concurrencia.

El microservicio cuenta con varias rutas que implementan las funcionalidades mencionadas:

- **POST a `/api/users/signup`:** verifica la información de la solicitud para la creación de un usuario, comprobando si el *email* y el nombre de usuario ya están registrados. De no existir, el usuario se crea en el modelo correspondiente, generando también un *token* asociado. Posteriormente, se publica un evento de creación de usuario y se retorna la información del nuevo usuario junto con una respuesta 201 *CREATED*.
- **POST a `/api/users/signin`:** valida la información de la solicitud de autenticación, buscando al usuario en la base de datos. Tras encontrarlo, se compara la contraseña proporcionada con la almacenada y se verifica el estado de *verificación* del usuario. Al autenticarse, se genera una cookie con un JWT que permite acceso a las rutas protegidas, y se retorna junto al usuario y una respuesta 200 *OK*.
- **POST a `/api/users/signout`:** elimina la cookie de sesión y retorna una respuesta 200 *OK*.
- **GET a `/api/users/currentuser`:** utiliza un *middleware* para verificar la autenticidad de la cookie, devolviendo el usuario autenticado con una respuesta 200 *OK*.
- **POST a `/api/users/user-verify/:id`:** busca el *token* proporcionado en la URL y, si está activo y sin uso, localiza el usuario asociado, marcando el *token* como *usado* y estableciendo el atributo *verified* en *true*. Luego, se crea la cookie de autenticación y se retorna una respuesta 200 *OK*.

Este microservicio no cuenta con *listeners*, sino con un único *publisher* para la creación de usuarios. Este publica en el tema `user:created` la información del nuevo usuario con el propósito de enviar un *token* de verificación de correo electrónico a través de una URL, permitiendo al usuario confirmar su cuenta accediendo a la ruta */api/users/user-verify/:id*.

2. Microservicio de comandos

Este proyecto incluye la capacidad de enviar comandos a los actuadores desde la API, con la proyección de que en futuras iteraciones esta API pueda desplegarse en la nube y accederse desde cualquier ubicación en el mundo. En esta iteración, como se mencionó previamente, se empleará un despliegue local, facilitando la demostración del funcionamiento esperado de todo el sistema. Es crucial comprender este concepto para entender el funcionamiento de este microservicio.

Este microservicio cuenta con las siguientes rutas:

- **POST a `/api/commands/send-command`:** recibe el *payload* con los datos necesarios para enviar a los actuadores y la ID del dispositivo de destino. Una vez recibida la información en el *endpoint*, se verifica que el dispositivo exista, que esté en línea, y que el usuario autenticado sea el propietario del dispositivo. En caso afirmativo, se localiza la IP del dispositivo *gateway* (la Raspberry Pi) y, utilizando la biblioteca Axios, se envía una solicitud HTTP al *gateway* para que este publique un mensaje dirigido al dispositivo identificado, junto con la instrucción deseada. Si la operación es exitosa, se devuelve una respuesta `200 OK`.

Como es evidente, este microservicio depende del microservicio de dispositivos para obtener información actualizada. En este caso, el microservicio de comandos escucha eventos como *device:registered* y *device:updated*, permitiendo mantener una base de datos local con información relevante de cada dispositivo, lo cual es útil en caso de que el microservicio de dispositivos se encuentre inactivo. A continuación, se muestra el modelo de la base de datos MongoDB utilizado para registrar los dispositivos.

Cuadro 3: Modelo copia de dispositivo

Nombre	Tipo de dato	Descripción
type	cadena	Tipo de dispositivo (*tower/station/gateway*), campo obligatorio.
status	cadena	Estado actual del dispositivo (*online/offline/maintenance*).
gatewayIp	cadena	IP de la Raspberry encargada de la comunicación MQTT con dicho dispositivo.
userId	cadena	Identificador del usuario asociado al dispositivo.

Además del *listener*, el microservicio incluye un *publisher* encargado de notificar cuando se ha enviado un comando a un dispositivo. Esta notificación resulta útil para el microservicio historiador, el cual se dedica a registrar los eventos relevantes en el sistema. El *publisher* publica sus eventos en el tema *command:executed*. Aunque este es un microservicio pequeño y sencillo, su funcionalidad es esencial para lograr el control remoto de los dispositivos.

3. Microservicio de dispositivos

Este microservicio tiene la responsabilidad de gestionar el registro de todos los dispositivos, incluyendo tanto su información como su estado actual. Para cumplir con esta tarea, dispone de varios *endpoints* que permiten la creación y actualización de los registros de dispositivos. Además, cuenta con *listeners* que facilitan la actualización automática de algunas variables en la base de datos.

Este microservicio utiliza un modelo de base de datos único, centrado en los dispositivos, que es más extenso en comparación con el microservicio de comandos, y contiene información adicional acerca de los dispositivos. A continuación, se presenta el modelo de dispositivos dentro de este microservicio.

Cuadro 4: Modelo de dispositivo

Nombre	Tipo de Dato	Descripción
type	cadena	Tipo de dispositivo (*tower/station/gateway*), campo obligatorio.
name	cadena	Nombre asignado al dispositivo, obligatorio.
status	cadena	Estado actual del dispositivo (*online/offline/maintenance*).
userId	cadena	Identificador del usuario asociado al dispositivo.
gatewayIp	cadena	IP de la Raspberry Pi encargada de la comunicación MQTT con dicho dispositivo.
payload	objeto	Datos adicionales enviados desde el dispositivo, opcional.
lastUpdated	fecha	Fecha de la última actualización del dispositivo.
version	número	Número de versión del dispositivo, utilizado para el control de versiones.

Un aspecto clave de este microservicio es que es el único encargado de incrementar la versión de los registros. Este mecanismo asegura que el número de versión sea consistente en todos los microservicios que dependen de dicha información. El incremento de versiones se gestiona mediante un *plugin* para MongooseDB denominado mongoose-update-if-current. Al guardar un nuevo registro, se asigna un valor inicial de versión cero, y con cada actualización posterior, este valor se incrementa. Cada vez que se actualiza la información de un dispositivo, se emite un evento de tipo *device:updated*, lo que permite que los microservicios que requieran los datos más recientes reciban esta notificación.

A continuación, se detallan los *endpoints* disponibles en este microservicio.

- **GET a `/api/devices`:** una petición *GET* que devuelve todos los registros de dispositivos almacenados en la base de datos relacionados al usuario actual.
- **POST a `/api/devices/via-gateway`:** una petición, especial para realizarse desde el *gateway*, que registra un nuevo dispositivo en la base de datos. Requiere de toda la información desplegada en la tabla 4. Después de crear el registro, se publica un evento de tipo *device:registered*.

- **GET a `/api/devices/:id`:** una petición *GET* que devuelve un registro de dispositivo en particular, basado en el *id* proporcionado en la URL.
- **PUT a `/api/devices/:id`:** se pueden actualizar cierta información acerca del dispositivo, como puede ser la IP del gateway, nombre y status. Este publica en el tema *device:updated*.

Otro aspecto importante a considerar en este microservicio es la sincronización de los registros en las bases de datos. Para ello, el microservicio dispone de un *listener* encargado de escuchar las lecturas recibidas desde el microservicio de lecturas (que se detallará más adelante). Este *listener* actualiza los atributos *payload* y *lastUpdated* de los registros correspondientes. Una vez se ha procesado la información, procede a emitir una publicación en el tema *device:updated* para asegurar que todas las bases de datos reciban la actualización de los registros. Este mecanismo garantiza que las bases de datos estén sincronizadas y operen correctamente. De no implementarse este procedimiento, podrían producirse incongruencias en las versiones de los registros, lo que afectaría el funcionamiento adecuado de la API.

4. Microservicio de lecturas

Este microservicio tiene la responsabilidad de recibir las lecturas directamente desde los dispositivos a través del *gateway*. Las lecturas, que contienen datos relevantes sobre el estado y las condiciones de los dispositivos, se almacenan en una base de datos especializada que está diseñada específicamente para gestionar este tipo de registros. Esta base de datos permite un acceso rápido y eficiente a las lecturas, facilitando su posterior análisis y procesamiento. La sección de *payload* contiene variables de sensores, como puede ser nivel de tanque, cantidad de luz, temperatura y humedad. El modelo se meustra en la tabla 5.

Cuadro 5: Modelo de lectura

Nombre	Tipo de dato	Descripción
userId	cadena	Identificador del usuario que realizó la lectura.
payload	objeto	Contiene los datos de la lectura, puede ser de tipo *TowerPayload* o *StationPayload*.
device	id de object	Referencia al dispositivo que generó la lectura, es una referencia al modelo `Device`.
timeStamp	fecha	Marca temporal que indica cuándo se registró la lectura, asignada automáticamente al guardar el registro.

Para una gestión más organizada y eficiente de la información, este microservicio emplea un modelo de dispositivos derivado del modelo principal utilizado en el microservicio de dispositivos. Este modelo derivado está diseñado para guardar información pertinente, que para este microservicio, únicamente es el tipo de dispositivo, su nombre y el id del usuario al que está asociado. Este modelo funciona principalmente para saber la estructura del *payload* dependiendo de el tipo de dispositivo.

Al igual que el microservicio de comandos, las rutas en este microservicio son pocas, pero de gran utilidad. Estas se listan a continuación:

- **GET a `/api/readings/:deviceId`:** Una petición *GET* que devuelve todos los registros de lecturas almacenadas en la base de datos relacionados al dispositivo en la URL.
- **POST a `/api/readings`:** Una petición *POST* para crear una nueva lectura. Únicamente espera el ID del dispositivo que lo manda y el *payload* con la información de la lectura. Luego, se procesa esta información y se guarda en la base de datos. Por último, se publica un evento de tipo *reading:recieved* para alimentar al microservicio de historiador y actualizar el registro en la base de datos del microservicio de dispositivos.

5. Microservicio de mensajería

Para complementar y agregar mayor funcionalidad autónoma, se creó el microservicio de mensajería. Este microservicio no cuenta con *endpoints*, únicamente con una variedad de *listeners*. Se utiliza la libería de `nodemailer` para enviar correos electrónicos a los usuarios en variedad de ocasiones. Para la etapa de desarrollo, esta libería utiliza `Ethereal Email` para visualizar los correos. Estos pueden llevar una estrctura HTML y se muestra al presionar un URL de visualización previa.

A continuación se listan los *listeners* mencionados, listados por tópico en el que escuchan:

- **`auth:reset-password`:** este se encarga de enviar un correo de restablecimiento de contraseña a los usuarios que lo solicitan. El token se crea del lado del servicio de autenticación y se envía al usuario. Este correo contiene un enlace que redirige a un formulario de reinicio de contraseña.
- **`auth:send-verification`:** este se encarga de enviar un correo para validar una cuenta de usuario. Este correo también contiene un enlace que envía a una página que autentica automáticamente al usuario y le permite iniciar sesión.

6. Microservicio de cliente

El microservicio de cliente es el responsable de renderizar la interfaz gráfica de usuario de la aplicación mediante *server-side rendering (SSR)*, implementado con Next JS. El uso de SSR permite que las páginas se generen en el servidor antes de ser enviadas al cliente, optimizando así el rendimiento inicial y mejorando la indexación en motores de búsqueda.

Este microservicio contiene exclusivamente el *frontend* de la aplicación y está estructurado en las siguientes carpetas principales:

- **api:** incluye funcionalidades generales para gestionar la comunicación con otros microservicios. Esta carpeta contiene, por ejemplo, un cliente de Axios configurado específicamente para enviar peticiones HTTP a través de ingress NGINX, el cual dirige y distribuye las solicitudes hacia los *pods* correspondientes en el clúster de Kubernetes.

- **components:** almacena componentes React reutilizables. Estos componentes están diseñados para modularizar la interfaz gráfica, mejorando la consistencia y facilitando el mantenimiento del código.
- **hooks:** incluye *hooks* personalizados que encapsulan lógica reutilizable. Uno de los *hooks* destacados es `use-request`, que se encarga de realizar las peticiones HTTP hacia otros microservicios y manejar los posibles estados de error de manera centralizada.
- **pages:** cada archivo dentro de esta carpeta corresponde a una ruta específica de la aplicación. Esta estructura permite que Next JS gestione automáticamente el enrutamiento, facilitando la generación de nuevas páginas y una navegación fluida.
- **public:** contiene recursos estáticos, como imágenes, que se sirven directamente al cliente, optimizando así la carga de estos elementos visuales.

El flujo de comunicación en el microservicio de cliente sigue un diseño que maximiza la eficiencia y mantiene la seguridad en un entorno de microservicios. Este proceso se describe a continuación:

1. **Inicio de la solicitud desde el cliente:** cada vez que un usuario navega en la aplicación, el cliente (*frontend*) envía solicitudes HTTP para acceder a datos o realizar acciones. Estas solicitudes están encapsuladas mediante `use-request` para asegurar el manejo adecuado de errores y respuestas.
2. **Enrutamiento de solicitudes a través de ingress NGINX:** Las solicitudes HTTP generadas por el cliente no se envían directamente a los microservicios específicos. En su lugar, se dirigen a ingress NGINX, el cual actúa como un *proxy reverso* y un equilibrador de carga. Ingress NGINX determina a qué microservicio (o *pod*) debe redirigir la solicitud según el tipo de petición y el endpoint solicitado.
3. **Distribución interna en Kubernetes:** dentro del clúster de Kubernetes, ingress NGINX enruta las solicitudes a los pods específicos que ejecutan los microservicios correspondientes. Esta configuración permite la escalabilidad, ya que Kubernetes puede gestionar la cantidad de réplicas de cada microservicio según la demanda.
4. **Procesamiento y respuesta de microservicios:** Los microservicios correspondientes procesan la solicitud recibida, acceden a las bases de datos u otros servicios según sea necesario y devuelven una respuesta a ingress NGINX.
5. **Respuesta al cliente:** ingress NGINX recibe la respuesta de los microservicios y la reenvía al frontend del cliente, completando así el flujo de comunicación. En este punto, el *hook* `use-request` recibe la respuesta y la utiliza para actualizar el estado de la interfaz de usuario, mostrando los datos solicitados o confirmando las acciones realizadas.

El microservicio de cliente, basado en React JS y Next JS, emplea varias estrategias para optimizar la experiencia del usuario. Las páginas renderizadas en el servidor son especialmente beneficiosas para el rendimiento y SEO, ya que permiten una carga más rápida y mejor

accesibilidad para motores de búsqueda. Además, la estructura modular con componentes y *hooks* personalizados facilita el mantenimiento y la escalabilidad de la aplicación.

En cuanto a la interfaz gráfica, la página inicial, accesible mediante la ruta / (index), ofrece una breve descripción del proyecto e incluye un encabezado con botones de navegación. La visibilidad de estos botones depende del estado de autenticación del usuario: si el usuario ha iniciado sesión, puede acceder al *dashboard* directamente desde esta página inicial al hacer clic en el botón *Dashboard* en la barra de navegación. Las figuras 24 y 25 muestran la página de inicio en ambos escenarios de autenticación.

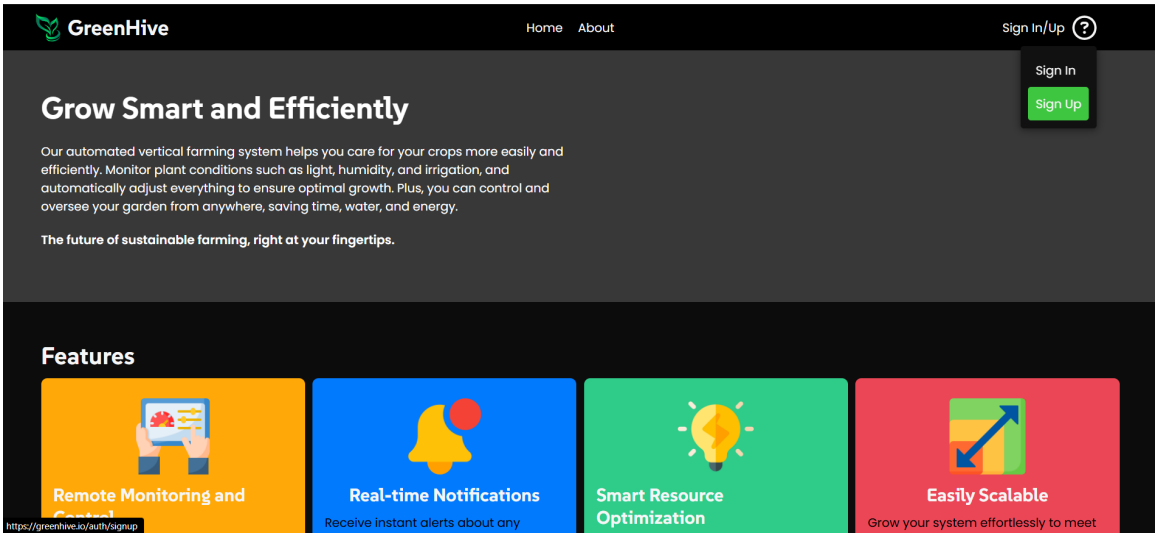

Figura 24: Página inicial sin autenticación

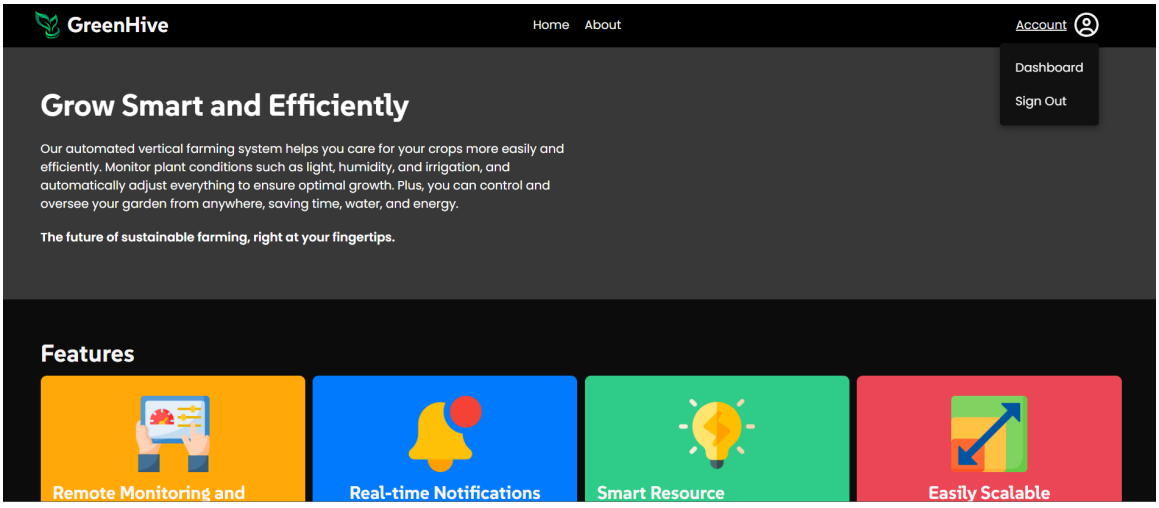

Figura 25: Página inicial con autenticación

El procedimiento de autenticación incluye tres rutas principales: *sign-up*, *sign-in* y *sign-out*. Al acceder a la ruta `auth/signup`, se mostrará el formulario ilustrado en la Figura 26, donde el usuario debe ingresar la información necesaria para utilizar la API. Los campos

obligatorios incluyen un correo electrónico único (no registrado previamente) para enviar notificaciones, nombre, apellido, nombre de usuario y una contraseña para proteger la información confidencial. Es esencial que todos los campos se completen con datos válidos, ya que la información en el *body* de la solicitud se validará. En caso de no cumplir con los requisitos, se devolverá un error *400 Bad Request*.

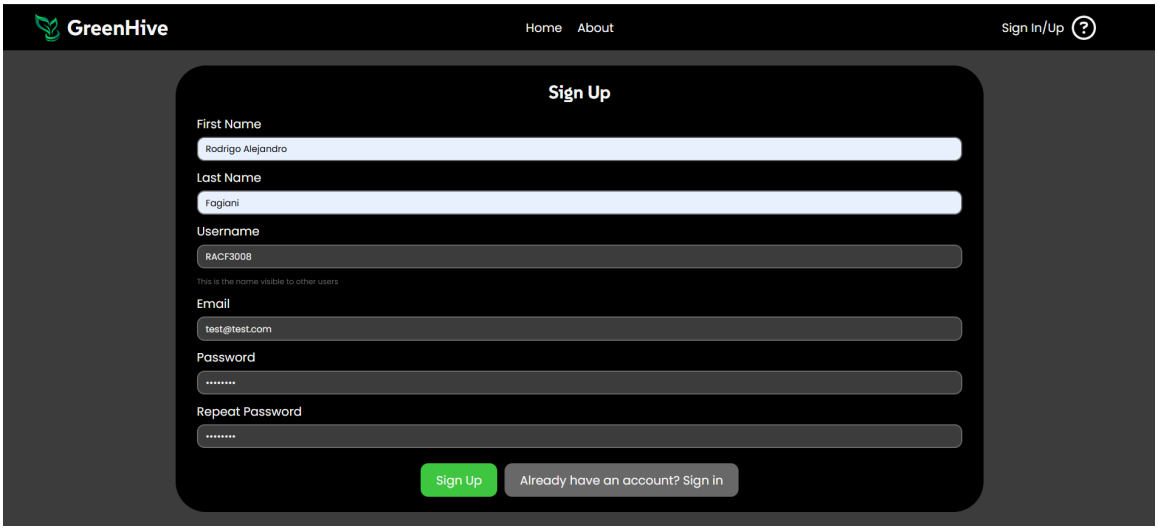

Figura 26: Formulario de creación de usuario

Como ya se mencionó en el microservicio de mensajería, al crear el usuario se envía un correo al usuario a la dirección indicada en el formulario. A continuación se puede observar una vista pervia del crreo en cuestión en la Figura 27.

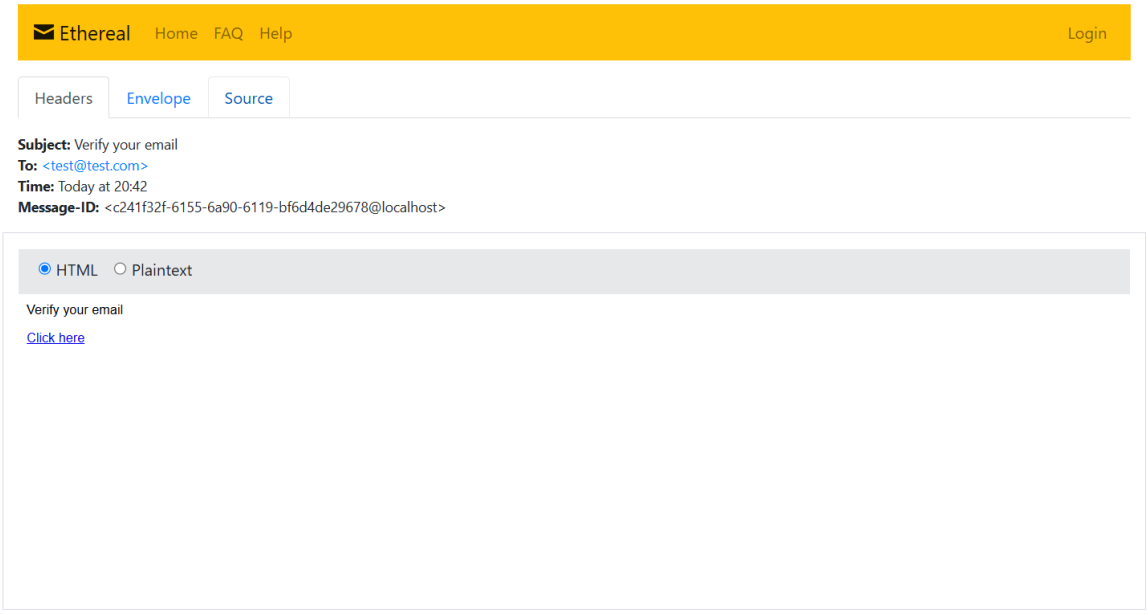

Figura 27: Correo de verificación de usuario

Al presionar en el enlace, el usuario tendrá total acceso a la aplicación, y podrá ingresar a cada una de las rutas del cliente. La primera, y la más importante, es el panel de control o *dashboard*. Al ingresar a la ruta, se hace una consulta previa tipo *GET* al microservicio de dispositivos, específicamente a la ruta de `/api/devices`. Al obtener la información de los dispostivos existentes, se mostrarán todos en tarjetas con información resumida del estado actual de los dispositivos. Estas tarjetas también sirven como botones para ir a rutas con información más detallada sobre cada dispositivo individual. En la Figura 28 se muestra un ejemplo de la pantalla del panel de control.

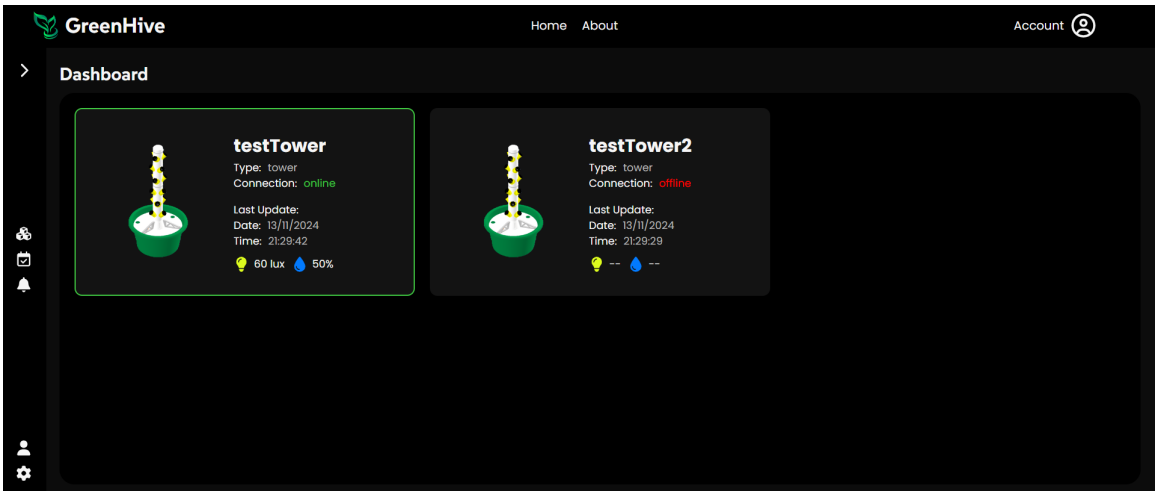

Figura 28: Listado de dispositivos

El panel de control de cada torre individual se presenta en la Figura 29. Esta interfaz proporciona información detallada sobre el dispositivo y ofrece una variedad de actuadores para controlar la torre. Entre las funcionalidades disponibles se incluyen:

- Un botón para encender y apagar la bomba de agua.
- Un botón para encender y apagar las luces LED.
- Un deslizador para ajustar la intensidad de las luces.

Cuando se actualiza el estado de alguno de estos elementos, se envía un comando al microservicio de comandos, el cual utiliza una solicitud HTTP para comunicarse con la Raspberry Pi. Posteriormente, el código del puente (que se detallará más adelante) transmite un evento mediante MQTT a todos los ESP32, incluyendo la información del comando y el ID del dispositivo correspondiente. De este modo, se garantiza la actualización en tiempo real del estado de los dispositivos.

Además, el panel incluye una tabla de lecturas históricas que muestra la fecha, hora y valores registrados. Estas lecturas están organizadas en páginas para facilitar la navegación y comprensión, especialmente cuando se maneja un gran volumen de datos. La información se obtiene del microservicio de lecturas mediante una solicitud GET a la ruta `/api/readings/:deviceId`, que devuelve las lecturas asociadas a un dispositivo específico.

Por último, el panel incorpora una sección para configurar temporizadores que definen los intervalos de encendido y apagado de la bomba de agua. También se muestra el tiempo restante para el cambio de estado de la bomba de agua. El temporizador opera principalmente en el ESP32 para garantizar la funcionalidad de la torre hidropónica incluso en ausencia de conexión a la plataforma. Para sincronizar el tiempo actual, se realiza una solicitud HTTP a la Raspberry Pi, la cual solicita el tiempo actual del temporizador al ESP32. Asimismo, se incluyó un botón de pausa y reanudación que, mediante el mismo proceso, comunica al ESP32 si debe pausar o reanudar el temporizador.

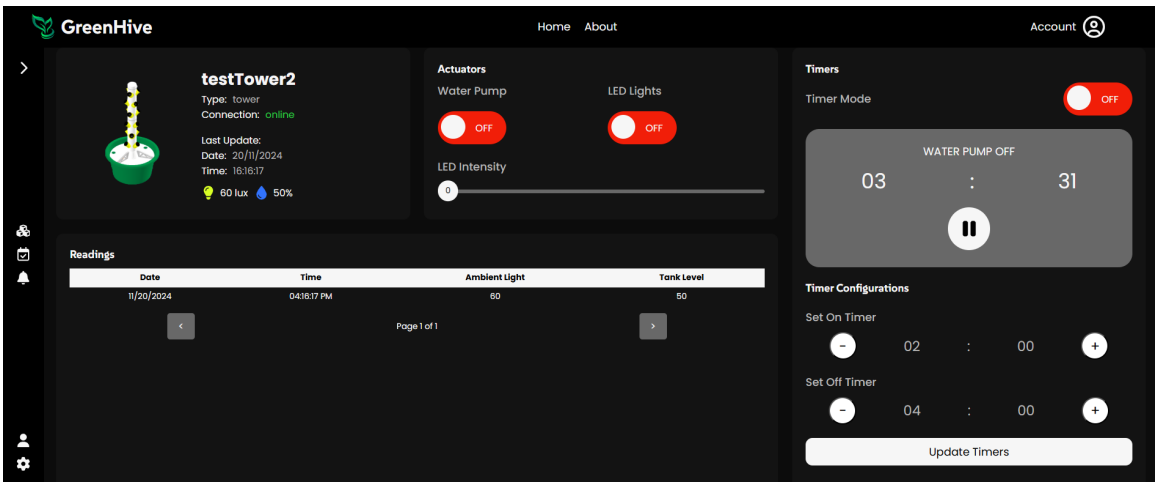

Figura 29: Panel de control

Estas son las secciones destacadas de la aplicación. Se pleaneó agregar más microservicios, como un historiador, un microservicio de alertas y demás, para agregar más funcionalidad. Sin embargo, para esta iteración se incluyó lo esencial, dejando las puertas abiertas para futuro seguimiento de este proyecto.

CAPÍTULO IX

Circuitos y conexión de sensores

A. Circuito de torre hidropónica

El circuito de la torre hidropónica integra diversos sensores y módulos para garantizar su funcionamiento óptimo. Entre los componentes principales se encuentran:

- **Sensor de luz ambiental VEML7700:** utilizado para medir la intensidad de luz ambiental y ajustar parámetros en función de las necesidades de las plantas.
- **Sensor ultrasónico JSN-SR04T:** empleado para monitorear el nivel de agua en el tanque y prevenir situaciones críticas como el vaciado total.
- **Módulo LED WS2812B:** diseñado para la indicación de alertas mediante un sistema de códigos de color que facilita la interacción con el usuario.
- **Tira LED 5050:** incorporada para brindar luz artificial a los cultivos hidropónicos, en caso se coloque el sistema en un lugar con poca cantidad de luz natural.
- **Módulo de relé:** controla el flujo de corriente hacia la bomba, permitiendo encenderla o apagarla según sea necesario, al conectar o desconectar la fase.
- **Tomacorriente:** utilizado para conectar la bomba. Este diseño evita el daño a la espiga de la bomba, permitiendo su reutilización en otros proyectos futuros.

La Figura 30 ilustra el diseño del circuito de la torre hidropónica.

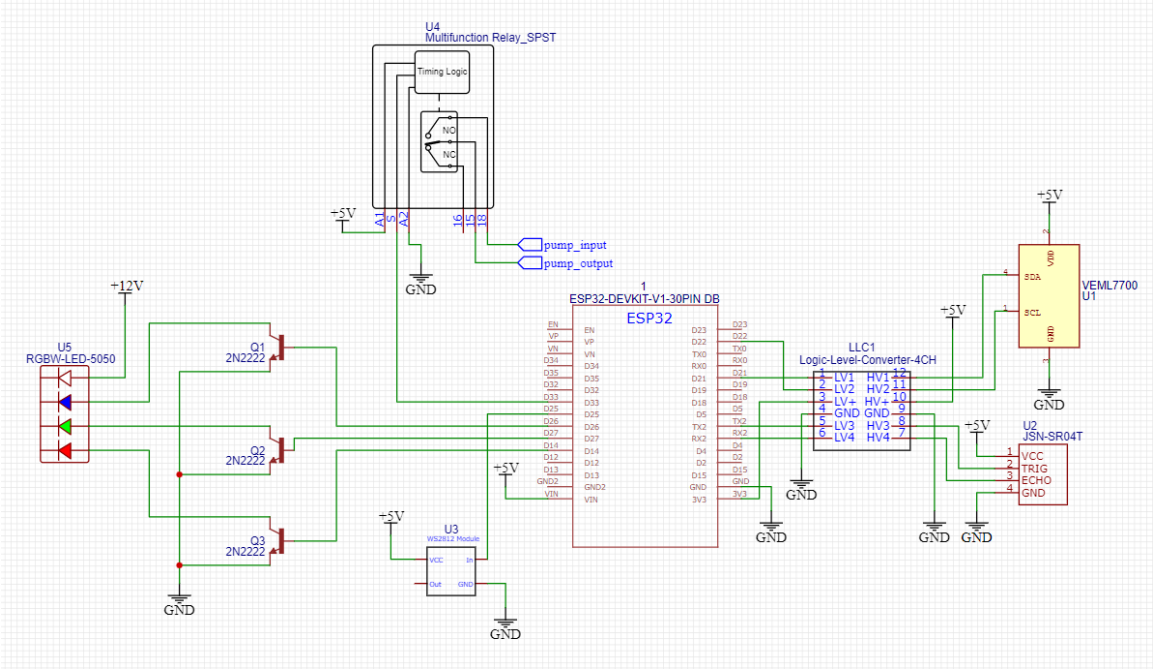

Figura 30: Circuito de torre hidropónica

El módulo LED WS2812B proporciona una retroalimentación visual al usuario mediante el uso de colores que representan diferentes estados y alertas. A continuación, se detalla el significado de cada color:

- **Cyan:** indica que la API ha enviado un mensaje y se está produciendo un cambio en el comportamiento de la torre. Este color asegura al usuario que el sistema funciona correctamente.
- **Magenta:** señala desconexión con el servidor MQTT. En este caso, es necesario verificar la conexión a internet del servidor y la IP configurada en el ESP32, ya que esta puede cambiar tras una reconexión.
- **Azul:** indica desconexión de la red wifi. Se recomienda comprobar la disponibilidad de la red, las credenciales configuradas en el microcontrolador y la proximidad del dispositivo al router.
- **Amarillo:** alerta de atención preventiva, como un nivel de agua más bajo de lo esperado. Aunque la torre sigue funcionando, se recomienda una revisión para prevenir problemas mayores.
- **Rojo:** indica una situación crítica que detiene el funcionamiento de los actuadores para evitar daños. Por ejemplo, si el nivel de agua es insuficiente para cubrir completamente la bomba sumergible, esta se apaga automáticamente para prevenir daños permanentes.

La intensidad de iluminación de la tira LED se controla mediante el deslizador en el panel de control de torre mencionado anteriormente. Además, se puede controlar si esta se

activa o no. Para modular la intensidad se utilizan señales PWM a través de los pines RGB de la tira LED. Debido a que esta opera a un voltaje de 12V y consume mayor cantidad de corriente, se agregaron transistores NPN que actúan como drivers, uno para cada canal de color.

Para los sensores JSN-SR04T y VEML7700 se utilizó un convertidor de nivel lógico. Este dispositivo está diseñado para soportar hasta cuatro canales, cada uno con dos pines: uno de bajo voltaje y otro de alto voltaje. Además, cuenta con entradas dedicadas para la alimentación de alto y bajo voltaje, permitiendo convertir las señales de 3.3V del ESP32 a los 5V necesarios para operar correctamente estos sensores.

Se emplearon dos canales del convertidor para la comunicación I2C con el sensor VEML-7700, y los otros dos para la comunicación UART con el sensor JSN-SR04T. En el caso del JSN-SR04T, fue necesario realizar una conexión con soldadura en la placa para activar el modo 2, que habilita la comunicación UART y proporciona lecturas más precisas.

La conexión con el módulo WS2812B se realizó de forma directa, ya que este opera correctamente con señales de 3.3V y no requiere un consumo elevado de corriente. Este módulo cuenta con tres pines principales: un pin de datos, uno de alimentación y otro de referencia, además de un pin de salida de datos que permite conectar múltiples módulos en cascada. En este caso, se utilizó un único módulo para simplificar el diseño. Sin embargo, la incorporación de módulos adicionales en tres puntos de la estructura podría mejorar significativamente la visibilidad de las alertas.

Por último, el ESP32 se alimenta mediante un voltaje de 5V, obtenido a partir de un adaptador de corriente. Este diseño asegura un circuito completamente funcional para las torres hidropónicas, con todos los componentes trabajando en sincronía.

B. Circuito de estación de variables ambientales

Como segundo dispositivo se implementó una estación de variables ambientales. Está unicamente consta del circuito, sin niguna pieza impresa o encubrimiento. En las fases iniciales se planeó incorporar un sensor de cantidad de presipitación utilizando el sensor de efecto hall KY-024. Al comenzar la lluvia, el agua se centraría en un mecanismo que haría moverse de un lado para otro una pieza con un imán incorporado. De esta forma, midiendo el intervalo entre cambios del sensor, se mediría la precipitación. Sin embargo, por falta de espacio de trabajo se optó por hacer el proyecto para estar en interiores. Por esta razón, se descontinuó esta parte del proyecto y se optó por agregar las tiras LED de luz artifical.

Por el caso explicado anteriormente, el circuito de la estación de variables ambientales cuenta únicamente con la conexión al módulo sensor DHT22 para medición de temperatura y humedad. Este utiliza su propio protocolo de comunicación, ya que únicamente tiene un pin de datos a parte del de alimentación y referencia. El circuito de este dispositivo se muestra en la Figura 31.

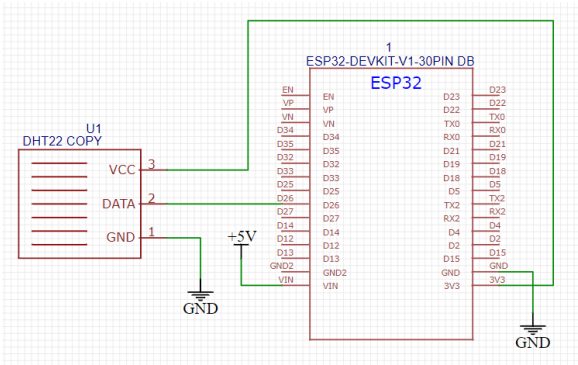

Figura 31: Circuito de estación de variables ambientales

CAPÍTULO X

Programación de ESP32 para IoT

A. Código de torres hidropónicas

1. Librerías, constantes, variables, *setup* y *loop*

El código de las torres hidropónicas debe cumplir con varios requisitos. Además, se diseñó para que cada proceso evitara obstaculizar el funcionamiento general, por lo que se utiliza ampliamente la función *millis()*. Los requisitos principales son:

- Obtener lecturas de los distintos sensores a intervalos regulares.
- Operar los actuadores inmediatamente después de recibir el comando (lo más cerca posible del tiempo real).
- Enviar las lecturas a la API, pasando previamente por el *gateway*.
- Mostrar los estados del sistema mediante el módulo LED RGB.

Se emplearon diversas librerías para:

- Manipulación de datos JSON.
- Adición de puertos seriales (UART para el sensor JSN-SR04T).
- Control del módulo LED WS2812B.
- Lectura del sensor VEML7700.
- Conexión MQTT y sifi.

Se definieron constantes para especificar pines de conexión, direcciones IP, contraseñas de red y valores de intervalo. Posteriormente, se crearon las instancias de los objetos correspondientes:

- Neopixel (para el módulo WS2812B).
- Sensores VEML7700 y JSN-SR04T.
- Lectura de JSON.
- Clientes wifi y MQTT.

Adicionalmente, se organizaron las variables de funcionamiento según el componente al que pertenecen. El código está completamente documentado.

En la configuración inicial, se estableció la comunicación serial a 115200 baudios. Además, se configuraron ciertos pines de salida, como el pin 2 correspondiente al LED incorporado del ESP32. Se inicializó el módulo WS2812B mediante el objeto *pixels* y se estableció la conexión a wifi utilizando la función *connectTowifi*, que se detallará posteriormente. También se configuró el cliente MQTT para la comunicación con el *gateway* y se inicializaron los sensores.

En el *loop* principal, se realizan las siguientes acciones:

- Se verifica el estado de los clientes wifi y MQTT.
- Se comprueba si el dispositivo ya posee un ID registrado en la API. En caso de no contar con uno, se envía un mensaje al tópico *towers/register-need*. Esto genera una solicitud HTTP desde el *gateway* a la API, cuya respuesta retorna al dispositivo MQTT con el nuevo ID.
- Se revisa el estado del dispositivo. Si este se encuentra en una condición crítica, se hace parpadear el LED.
- Si no existen errores, advertencias o desconexiones de wifi o MQTT, se ejecuta el *sensor_loop*, que contiene la lógica de los sensores.

Finalmente, se llama a *client_loop* para asegurar la funcionalidad del cliente MQTT.

```
#include <ArduinoJson.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_VEML7700.h"
#include <HardwareSerial.h>
#include <Preferences.h>
#include <PubSubClient.h>
#include <WiFi.h>
#include <Wire.h>

// ================================================
// DEFINITIONS
// ================================================
// Actuators
#define LEDPIN 2
#define PUMPPIN 27
#define STRIPPIN 33         // GPIO pin connected to the MOSFET gate
// Communication pins
#define RXPIN 16
#define TXPIN 17
#define PIXEL_PIN 25
// Other constants
```

```
#define PWM_FREQ 5000
#define PWM_RESOLUTION  8
#define PIXEL_QTY 1

#define WIFI_CHECK_INTERVAL 5000 // 5 seconds

#define SENSOR_PERIOD 5000 // Sensor measurement period (5 seconds)

// ================================================
// OBJECT INSTANCING AND VARIABLES
// ================================================
Adafruit_NeoPixel pixels(PIXEL_QTY,
                         PIXEL_PIN,
                         NEO_GRB + NEO_KHZ800); // Initialize NeoPixel object

HardwareSerial jsnSerial(2); // Serial communication for JSN sensor

Preferences preferences;

WiFiClient wifiClient;
PubSubClient client(wifiClient); // MQTT client

StaticJsonDocument<256> jsonDoc; // JSON document to parse incoming MQTT messages

// LED Strip constants

// Device variables
String device_id = "";  // Device ID for the device
bool has_id = false;    // Flag to indicate if device ID is present
unsigned long id_req_prev = 0;  // Timer for ID request interval

// WiFi variables
unsigned long wifi_last_check = 0;  // Last WiFi check time
bool is_wifi_connected = false;    // WiFi connection status

// MQTT variables
unsigned long mqtt_time_prev = 0; // Time of last attempt for MQTT connection
unsigned long f_to_z_now = 0; // Timer for notification blink

// Sensor variables
unsigned long sensor_time_prev = 0; // Last sensor reading time
unsigned int tank_level = 0;  // Tank water level reading

// Actuator variables
int brightness = 0;  // LED intensity (0-100)
bool led_on = false;  // LED state (on/off)
bool pump_on = false;  // Pump state (on/off)

// Timer variables for pump
unsigned long off_time = 15 * 60 * 1000;
unsigned long on_time = 15 * 60 * 1000;
unsigned long timerDuration = 0;  // Duration in milliseconds
unsigned long timerStartTime = 0; // Start time in milliseconds
bool timerRunning = false;        // Timer state
unsigned long current_timer_duration = 0; // Current timer duration
unsigned long elapsedMillis = 0; // Current timer duration

// Sistema
String ssid = "";
String password = "";
String ip = "";
bool auto_mode = false; // Auto mode flag (on/off)
uint8_t status = 6;    // Error severity status
// 6 - normal
// 5 - message from API
// 4 . no connection to MQTT
// 3 - no connection to WiFi
// 2 - warning
// 1 - error

// ================================================
// INITIAL SETUP
// ================================================
void setup() {
  Serial.begin(115200);  // Start serial communication for debugging
  pinMode(LEDPIN, OUTPUT);  // Set LED pin as output
  pinMode(PUMPPIN, OUTPUT); // Set pump-relay pin as output
  digitalWrite(PUMPPIN, LOW);

  pixels.begin();  // Initialize NeoPixel
  pixels.clear();
  pixels.show();   // Clear and update NeoPixel display

  preferences.begin("network", false);
  ssid = preferences.getString("ssid", "N/A");
  password = preferences.getString("password", "N/A");
  ip = preferences.getString("ip", "N/A");

  connectToWiFi();  // Connect to WiFi

  client.setServer(ip, 1883); // Set MQTT server IP and port
  client.setCallback(callback);  // Set the callback function for incoming MQTT messages
```

```
114
115   jsnSerial.begin(9600, SERIAL_8N1, RXPIN, TXPIN); // Initialize serial communication for JSN sensor
116
117   // Configure the LEDC channel and attach it to the pin
118   ledcAttach(STRIPPIN, PWM_FREQ, PWM_RESOLUTION);
119
120   startTimer(off_time);
121 }
122
123 // ================================================
124 // MAIN LOOP
125 // ================================================
126 void loop() {
127   // Check WiFi connection status
128   manageWiFiConnection();
129
130   // If MQTT server is disconnected, try to reconnect
131   if (!client.connected()) {
132     connect_mqttServer();
133   }
134
135   // LED strip control
136   if(led_on) {
137     ledcWrite(STRIPPIN, brightness);
138   } else {
139     ledcWrite(STRIPPIN, 0);
140   }
141
142   // Call the timer handler to manage pump states
143   // timerHandler();
144
145   if (pump_on) {
146     digitalWrite(PUMPPIN, false);
147   } else {
148     digitalWrite(PUMPPIN, true);
149   }
150
151   // If device ID has not been assigned yet, request it from the server
152   if (!has_id && client.connected()) {
153     unsigned long id_req_now = millis();
154
155     // Send an ID request every 5 seconds
156     if (id_req_now - id_req_prev >= 5000) {
157       id_req_prev = id_req_now;
158       client.publish("devices/register-need", "id-request");
159     }
160   }
161
162   // Handle different status codes and control NeoPixel LED
163   switch (status) {
164     case 1:
165       blink(pixels.Color(255, 0, 0), 0, 200, 1);  // Red blink for error severity 1
166       break;
167     case 2:
168       blink(pixels.Color(255, 255, 0), 10000, 40, 3);  // Yellow blink for error severity 2
169       break;
170     case 3:
171       blink(pixels.Color(0, 0, 255), 10000, 40, 2);  // Blue blink for error severity 3
172       break;
173     case 4:
174       blink(pixels.Color(255, 0, 255), 10000, 40, 2);  // Purple blink for error severity 4
175       break;
176     case 5:
177       if (f_to_z_now + 5000 > millis()) {
178         blink(pixels.Color(0, 255, 255), 10000, 40, 1);  // Cyan blink for normal operation
179       } else {
180         status = 6;
181       }
182       break;
183     default:
184       break;
185   }
186
187   // If device is connected and in normal state, perform sensor readings
188   if ((status == 5 || status == 0) && device_id) {
189     sensor_loop();  // Handle sensor data collection and MQTT publishing
190   }
191
192   // Continuously check for incoming MQTT messages
193   client.loop();
194 }
```

Cuadro 6: Librerías, constantes, variables, *setup* y *loop*

2. *Loop* de sensores

El resto del código se ha dividido en módulos para facilitar su organización y funcionamiento. Estos módulos se organizan alfabéticamente durante la compilación. Sin embargo, en esta explicación se abordarán en función de su importancia.

En primer lugar, se encuentra el módulo relacionado con los sensores, el cual está documentado de manera detallada. Este módulo consta de un bucle principal y una función auxiliar. El bucle principal, denominado `sensor__loop`, se encarga de comunicarse con los sensores para obtener sus lecturas. Posteriormente, envía un mensaje MQTT al canal `devices/readings` con la información recopilada, en un intervalo de 5 segundos.

Además, este módulo incluye una verificación del nivel de solución en el tanque para evitar que se desborde o quede vacío, lo cual podría causar daños a la bomba.

Como complemento, se utiliza una función auxiliar denominada `get_tank_level`, la cual envía un mensaje a través de UART para calcular el nivel del tanque. El resultado se traduce en un valor numérico que se retorna y utiliza dentro del `sensor__loop`.

```
1  void sensor_loop() {
2    // Store current time for sensor readings
3    unsigned long sensor_time_now = millis();
4
5    // Update sensor readings at regular intervals
6    if (sensor_time_now - sensor_time_prev >= SENSOR_PERIOD) {
7      sensor_time_prev = sensor_time_now;  // Update the last sensor read time
8
9      // Read data from JSN sensor (tank level)
10     jsnSerial.write(0x55);
11     delay(50);  // Wait for the sensor to respond
12
13     if (jsnSerial.available() > 0) {
14       tank_level = get_tank_level();  // Get tank water level
15     }
16
17     // Create JSON payload to send sensor data
18     String payload = "{\"deviceId\":\""
19                      + device_id
20                      + "\",␣\"tankLevel\":"
21                      + String(tank_level)
22                      + "}";
23
24     // Convert payload to char[] for MQTT message
25     char payloadBuffer[100];
26     payload.toCharArray(payloadBuffer, 100);
27
28     // Publish sensor data to MQTT
29     client.publish("devices/readings", payloadBuffer);
30   }
31
32   // Set error status based on tank level
33   if (tank_level >= 900) {
34     if (status >= 1) {
35       status = 1;  // Critical error (tank too full)
36     }
37   } else if ((tank_level > 750) && (tank_level < 900)) {
38     if (status >= 2) {
39       status = 2;  // Warning (tank getting full)
40     }
41   } else {
42     status = 0;  // Normal state (tank level OK)
43   }
44 }
45
46 unsigned int get_tank_level() {
47   unsigned int tank_level = 0;
48   unsigned char check_sum;
49   unsigned char buf[4] = {0};
50
51   if (jsnSerial.read() == 0xff) {
52     // Insert header into the array
53     buf[0] = 0xff;
54
55     // Insert the rest of the characters
56     for (int i = 1; i < 4; i++) {
57       buf[i] = jsnSerial.read();
58     }
59
60     // Evaluate the checksum to verify data
61     check_sum = buf[0] + buf[1] + buf[2];
62
63     // If checksum is correct, return the distance
64     if (buf[3] == check_sum) {
```

```
65        tank_level = (buf[1] << 8) + buf[2];
66      } else {
67        Serial.println("TANK LEVEL MEASUREMENT INVALID! CHECK WIRING.");
68        return 0;
69      }
70    }
71    return tank_level;
72  }
```

Cuadro 7: *Loop* de sensores para torre

3. Conexión a wifi

La sección de conexión wifi es bastante breve. Únicamente contiene una función que verifica la disponibilidad de red y otra que utiliza las funciones de la librería wifi para establecer la conexión. Este chequeo de conexión se realiza periódicamente, con un intervalo definido por la constante `CHECK_wifi_INTERVAL`.
Además, si el estado es mayor a 4, este se ajusta a 4 mientras se intenta conectar a la red, lo que activa el parpadeo azul como indicador visual.

```
1
2  void connectToWiFi() {
3    Serial.println("Attempting to connect to WiFi...");
4    WiFi.begin(ssid, password);  // Attempt to connect to WiFi with provided SSID and password
5  }
6
7  void manageWiFiConnection() {
8    unsigned long current_time = millis();
9
10   // Periodically check WiFi connection status
11   if (current_time - wifi_last_check >= WIFI_CHECK_INTERVAL) {
12     wifi_last_check = current_time;
13
14     // If not connected to WiFi, attempt reconnection
15     if (WiFi.status() != WL_CONNECTED) {
16       Serial.println("WiFi connection lost. Reconnecting...");
17       WiFi.reconnect();
18     } else {
19       is_wifi_connected = true;
20     }
21   }
22 }
```

Cuadro 8: Controlador de conexión a wifi para torre

4. Conexión a servidor MQTT

La sección del manejo de MQTT es más extensa. Primero, se cuenta con una función para conectar al servidor MQTT, la cual verifica si el dispositivo está conectado. En caso contrario, realiza un intento de reconexión cada 5 segundos. Durante este proceso, si el estado actual es 6, se cambia a 5 para indicar que se está intentando conectar. Una vez establecida la conexión, el dispositivo se suscribe a los canales MQTT correspondientes.
La función de *callback* es más compleja y se ejecuta cada vez que se recibe un mensaje en los canales MQTT a los que está suscrito el dispositivo. En esta función, se obtiene la información del mensaje, incluyendo el tópico y su contenido. Posteriormente, se convierte esta información en un objeto JSON para facilitar su manejo.
Una vez procesado el mensaje, se verifica si está dirigido al dispositivo actual. En caso afirmativo, se ejecutan diferentes acciones dependiendo del tópico del mensaje y del contenido del atributo `command`. Además, se actualiza el estado a 5 para notificar la recepción de un nuevo comando de la API.

- `towers/register-ok`: indica que se ha recibido una ID para el dispositivo, en caso de que no cuente con una.

- `devices/api-commands`: ejecuta diferentes acciones según el *payload* del mensaje. Los comandos disponibles son los siguientes:
  - `PUMP-ON`: enciende la bomba.
  - `PUMP-OFF`: apaga la bomba.
  - `LED-ON`: enciende las barras LED.
  - `LED-OFF`: apaga las barras LED.
  - `AUTO-ON`: activa el modo automático.
  - `AUTO-OFF`: desactiva el modo automático.
  - `LED-#NUM`: ajusta la intensidad de iluminación de las barras LED.
  - `GET-TIMER`: retorna el valor actual del temporizador.
  - `STOP-TIMER`: detiene el temporizador.
  - `RESUME-TIMER`: eeanuda el temporizador.
  - `UPDATE-TIMER`: actualiza los valores del temporizador.

```
// Generate a connection with the MQTT server
void connect_mqttServer() {
  // While there is no connection to the MQTT server...
  if (!client.connected()) {

    unsigned long mqtt_time_now = millis();

    if (mqtt_time_now - mqtt_time_prev >= 5000) {
      mqtt_time_prev = mqtt_time_now;
      // Try to reconnect to the MQTT server
      Serial.println("Trying␣to␣connect␣to␣the␣MQTT␣server...");

      // If the connection is successful...
      if (client.connect("ESP_tower")) {
        Serial.println("Successfully␣connected!");

        status = 6;

        // Subscribe to topics
        client.subscribe("devices/api-commands");
        client.subscribe("devices/register-ok");
      }
      // If not successful...
      else {
        unsigned long wait_time = 500;  // Wait time of 1 second
        if (status >= 4) {
          status = 4;
        }
        Serial.print("Connection␣failed!␣rc␣=␣");
        Serial.println(client.state());
        Serial.println("Trying␣again␣in␣5␣seconds.");
        Serial.println("␣");
      }
    }
  }
}

void callback(char* topic, byte* message, unsigned int length) {
  Serial.println("========================");
  Serial.println("=␣MQTT␣MESSAGE␣RECEIVED␣");
  Serial.println("========================");
  Serial.println("Message␣Information");
  Serial.println("------------------------");
  Serial.print("-␣Topic:␣");
  Serial.println(topic);
  Serial.print("-␣Message:␣");

  String messageTemp;
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    messageTemp += (char)message[i];
  }

  // Parse the JSON message
  DeserializationError error = deserializeJson(jsonDoc, messageTemp);
  if (error) {
    Serial.print("JSON␣Parsing␣failed:␣");
    Serial.println(error.c_str());
    return;
```

```
60  }
61
62  Serial.println("");
63  Serial.println("------------------------");
64  Serial.println("Extracted Information");
65  Serial.println("------------------------");
66
67  if (String(topic) == "devices/register-ok" && !has_id) {
68    // Notify receipt of command
69    if (status >= 5) {
70      f_to_z_now = millis();
71      status = 5;
72    }
73    // Extract values
74    const char* deviceIdValue = jsonDoc["id"];
75
76    if (deviceIdValue) {
77      Serial.printf("deviceId: %s\n", deviceIdValue);
78
79      // Save the deviceId (assuming device_id is a global variable)
80      if (strcmp(deviceIdValue, "") != 0) {
81        device_id = String(deviceIdValue);
82        has_id = true;
83        Serial.println("------------------------");
84        Serial.println("Action performed");
85        Serial.println("------------------------");
86        Serial.println("Device registered!!!");
87      }
88    } else {
89      Serial.println("No deviceId found in JSON.");
90    }
91  } else if (String(topic) == "devices/api-commands") {
92    // Extract values
93    const char* deviceIdValue = jsonDoc["deviceId"];
94    const char* payloadValue = jsonDoc["payload"];
95
96    if (deviceIdValue && payloadValue) {
97      Serial.printf("deviceId: %s\n", deviceIdValue);
98      Serial.printf("payload: %s\n", payloadValue);
99
100     // Handle commands
101     if (String(deviceIdValue) == device_id) {
102       Serial.println("------------------------");
103       Serial.println("Action performed");
104       Serial.println("------------------------");
105
106       if (strcmp(payloadValue, "PUMP-ON") == 0) {
107         Serial.println("Pump turned on!!!");
108         digitalWrite(LEDPIN, HIGH);
109         pump_on = true;
110       } else if (strcmp(payloadValue, "PUMP-OFF") == 0) {
111         Serial.println("Pump turned off");
112         digitalWrite(LEDPIN, LOW);
113         pump_on = false;
114       } else if (strcmp(payloadValue, "LEDS-ON") == 0) {
115         Serial.println("LED turned on!!!");
116         led_on = true;
117       } else if (strcmp(payloadValue, "LEDS-OFF") == 0) {
118         Serial.println("LED turned off");
119         led_on = false;
120       } else if (strcmp(payloadValue, "AUTO-ON") == 0) {
121         Serial.println("Automatic mode activated!!!");
122         auto_mode = true;
123       } else if (strcmp(payloadValue, "AUTO-OFF") == 0) {
124         Serial.println("Automatic mode deactivated");
125         auto_mode = false;
126       } else if (strncmp(payloadValue, "LED-", 4) == 0) {
127         const char* numberPart = payloadValue + 4; // Skip the "LED-" part
128         // Convert the numeric part to an integer
129         brightness = map(atoi(numberPart), 0, 100, 0, 255);
130         if (brightness > 255) {
131           brightness = 255;
132         }
133         Serial.print("LED intensity updated: ");
134         Serial.println(brightness);
135
136       // --- Timer actions ---
137       } else if (strcmp(payloadValue, "GET-TIMER") == 0) {
138         int remainingMinutes = 0, remainingSeconds = 0;
139         unsigned long remainingMillis = current_timer_duration - (millis() - timerStartTime);
140         remainingMillis = remainingMillis < 0 ? 0 : remainingMillis;
141
142         remainingMinutes = remainingMillis / 60000;
143         remainingSeconds = (remainingMillis % 60000) / 1000;
144
145         // Build JSON string
146         String payload = "{\"deviceId\":\""
147                          + device_id
148                          + "\", \"minutes\":"
149                          + String(remainingMinutes)
150                          + ", \"seconds\":"
151                          + String(remainingSeconds)
```

```
                        + ",␣\"timerRunning\":"
                        + (timerRunning ? "true" : "false")
                        + ",␣\"pumpState\":"
                        + (pump_on ? "true" : "false")
                        + "}";

          // Allocate buffer dynamically based on the payload size
          char payloadBuffer[payload.length() + 1];
          payload.toCharArray(payloadBuffer, sizeof(payloadBuffer));

          // Publish to the MQTT topic
          client.publish("devices/timers", payloadBuffer);
        } else if (strcmp(payloadValue, "STOP-TIMER") == 0) {
          Serial.println("Timer␣stopped.");
          stopTimer();
        } else if (strcmp(payloadValue, "RESUME-TIMER") == 0) {
          Serial.println("Timer␣resumed.");
          resumeTimer();
        } else if (strcmp(payloadValue, "UPDATE-TIMER") == 0) {
          unsigned long new_off_time = jsonDoc["off_time"];
          unsigned long new_on_time = jsonDoc["on_time"];

          if (new_off_time > 0 && new_on_time > 0) {
            updateTimerValues(new_off_time, new_on_time);
          } else {
            Serial.println("Invalid␣timer␣values␣received.");
          }
        } else {
          Serial.println("Unknown␣command");
          return;
        }
        // Notify receipt of command
        if (status >= 5) {
          f_to_z_now = millis();
          status = 5;
        }
      }
    } else {
      Serial.println("Error:␣deviceId␣or␣payload␣is␣missing␣in␣JSON.");
    }
  }
  Serial.println("");
}
```

Cuadro 9: Controlador de conexión a servidor MQTT para torre

5. Controlador de temporizadores

La sección del temporizador incluye un conjunto de funciones diseñadas para gestionar los tiempos de encendido y apagado, tanto en modos automáticos como manuales. Estas funciones permiten un manejo preciso del tiempo y aseguran que el dispositivo opere de manera confiable según los parámetros establecidos. A continuación, se describen las principales funcionalidades incluidas:

- **`startTimer`**: esta función inicializa el temporizador con una duración específica, almacenada en `timerDuration`. También registra el tiempo de inicio con `millis()` y establece el estado del temporizador como activo (`timerRunning = true`). Además, reinicia la variable `elapsedMillis` para asegurar que no haya valores acumulados previamente.

- **`stopTimer`**: permite detener el temporizador en cualquier momento. Si el temporizador está activo, se acumula el tiempo transcurrido desde el último inicio en *elapsedMillis*. Luego, se marca el temporizador como inactivo (*timerRunning* = *false*). Si el temporizador ya estaba detenido, se notifica mediante un mensaje en la consola.

- **`resumeTimer`**: reanuda un temporizador previamente detenido. Si el temporizador no está activo y aún no ha alcanzado la duración total (`elapsedMillis < timerDuration`), se reinicia el conteo a partir del tiempo actual (`millis()`), manteniendo el tiempo restante por ejecutar. En caso contrario, se genera un mensaje indicando que no es posible reanudar el temporizador.

- **`updateTimerValues`**: permite actualizar los valores de duración para los estados de apagado y encendido del temporizador (`off_time` y `on_time`). Después de realizar la actualización, se reinicia el temporizador utilizando la nueva duración de apagado por defecto (`off_time`). Este método asegura que los cambios surtan efecto de inmediato.

- **`timerHandler`**: es la función principal para el manejo del temporizador. Comprueba de forma continua si el temporizador activo ha alcanzado su duración. Cuando esto ocurre:

  - Se alterna el estado de la bomba (`pump_on`) mediante una inversión lógica.
  - Se actualiza el estado físico del pin conectado a la bomba, encendiéndolo o apagándolo según corresponda.
  - Se reinicia el temporizador con la duración correspondiente al nuevo estado de la bomba: `on_time` si está encendida o `off_time` si está apagada.

  Esta función permite la operación cíclica automática de la bomba, asegurando que se respeten los intervalos configurados.

Estas funciones trabajan en conjunto para proporcionar un sistema de temporización flexible y robusto, adecuado para aplicaciones que requieren control preciso de dispositivos como bombas o actuadores. Además, los mensajes seriales incluidos en cada función facilitan la depuración y monitoreo del comportamiento del temporizador durante la ejecución.

```
void startTimer(unsigned long durationMs) {
  timerDuration = durationMs;
  timerStartTime = millis();
  timerRunning = true;
  elapsedMillis = 0;
}

void stopTimer() {
  if (timerRunning) {
    elapsedMillis += millis() - timerStartTime; // Accumulate elapsed time
    timerRunning = false;
    Serial.println("Timer stopped");
  } else {
    Serial.println("Timer is not running, cannot stop");
  }
}

void resumeTimer() {
  if (!timerRunning && elapsedMillis < timerDuration) {
    timerStartTime = millis();
    timerRunning = true;
    Serial.println("Timer resumed");
  } else {
    Serial.println("Cannot resume. Either timer is already running or time has expired.");
  }
}

void updateTimerValues(unsigned long new_off_time, unsigned long new_on_time) {
  off_time = new_off_time;
  on_time = new_on_time;

  startTimer(off_time);
  Serial.println("Timer values updated and restarted");
}


void timerHandler() {
  if (timerRunning && millis() - timerStartTime >= timerDuration) {
    pump_on = !pump_on; // Toggle pump state
    digitalWrite(PUMPPIN, pump_on ? HIGH : LOW); // Update physical pin state
    Serial.println(pump_on ? "Pump turned ON" : "Pump turned OFF");

    // Restart timer with the appropriate duration
    startTimer(pump_on ? on_time : off_time);
  }
}
```

Cuadro 10: Controlador de termporizadores para torre

6. Funciones auxiliares

La función `blink` controla el parpadeo del LED `WS2812B` sin bloquear el flujo de ejecución del microcontrolador. Permite ajustar el color, la velocidad y la cantidad de pasos de brillo del parpadeo. Los parámetros de entrada incluyen:

- `color`: el color del parpadeo en formato RGB.
- `wait_time`: el tiempo de pausa entre ciclos de parpadeo.
- `alarm_speed`: la velocidad del parpadeo.
- `steps`: el número de pasos de brillo durante el ciclo.

La función ajusta la intensidad del LED en cada paso de brillo, alternando entre aumentar y disminuir la luminosidad. Cuando se alcanza el número de pasos definidos, se realiza una pausa antes de reiniciar el ciclo. Todo esto se gestiona de forma no bloqueante, permitiendo que el microcontrolador siga realizando otras tareas.

```
// Save connection info
void saveNetworkConfig(const char* ssid, const char* password, const char* ip) {
  preferences.begin("network", false);

  preferences.putString("ssid", ssid);
  preferences.putString("password", password);
  preferences.putString("ip", ip);

  Serial.println("Configuracion guardada con Preferences");

  preferences.end();
}

// Blink function
void blink(uint32_t color,
           unsigned long wait_time,
           int alarm_speed,
           int steps) {

  static unsigned long last_blink_time = 0;
  static unsigned long last_pause_time = 0;
  static int brightness = 0;
  static bool increasing = true;
  static int blink_count = 0;
  static bool paused = false;

  unsigned long current_time = millis();

  // Handle pause between blinking cycles
  if (paused) {
    if (current_time - last_pause_time >= wait_time) {
      paused = false; // End the pause
      blink_count = 0; // Reset blink count
      brightness = 0;  // Ensure brightness starts from 0
    } else {
      pixels.setPixelColor(0, 0, 0, 0); // Turn off during pause
      pixels.show();
      return;
    }
  }

  // Handle blinking
  if (current_time - last_blink_time >= alarm_speed) {
    last_blink_time = current_time;

    // Adjust brightness
    if (increasing) {
      brightness++;
      if (brightness >= steps) {
        increasing = false;
        blink_count++; // Count a completed blink
      }
    } else {
      brightness--;
      if (brightness <= 0) {
        increasing = true;
      }
    }

    // Update pixel color with adjusted brightness
    pixels.setPixelColor(0,
```

```
      (brightness * ((color >> 16) & 0xFF)) / steps, // Red
      (brightness * ((color >> 8) & 0xFF)) / steps,  // Green
      (brightness * (color & 0xFF)) / steps          // Blue
    );
    pixels.show();

    // Check if blink count reached steps, then pause
    if (blink_count >= steps) {
      paused = true;
      last_pause_time = current_time;
    }
  }
}
```

Cuadro 11: Funciones auxiliares para torre

B. Código de estación de variables ambientales

El código de este dispositivo es similar al de las torres en cuanto al manejo del la conexión a wifi y la configuración de MQTT, auque los tópicos y la función *callback* difieren. Además, el loop de sensores es distinta, ya que no se utiliza únicamente el módulo de sensor DHT22. A continuación se presentan las secciones de dicho código.

1. Librerías, constantes, variables, *setup* y *loop*

Esta sección guarda una gran similitud con el código de la torre, ya que utiliza librerías parecidas, con la principal diferencia en los módulos de los sensores. En este caso, se incorpora la librería correspondiente al sensor DHT22. La única instancia de objeto adicional es la asociada a este sensor. Las variables se simplificaron, quedando únicamente las necesarias, y las variables de los sensores (de tipo *float*) se definieron a nivel global. En cuanto a la configuración, solo se inicializa el DHT22, y se han eliminado las configuraciones de los sensores que no están presentes en este sistema. El ciclo *loop* se ajusta al eliminar el control del temporizador y al publicar la solicitud de ID en el canal *stations/register-need*, a diferencia del canal *towers/register-need* utilizado en el código de las torres. A continuación, se presenta el código correspondiente.

```
#include <ArduinoJson.h>
#include <Adafruit_NeoPixel.h>
#include <DHT.h>
#include <PubSubClient.h>
#include <WiFi.h>
#include <Wire.h>

// ================================================
// DEFINITIONS
// ================================================
#define LEDPIN 2

#define PIXEL_PIN 25
#define PIXEL_QTY 1

// #define SSID "Galaxy S21 FE 5G 5a70"
// #define PASS "ifqb4038"
#define SSID "ARRIS-CF84"
#define PASS "70DFF7A1CF84"

// #define MQTT_SERVER_IP "192.168.70.110"
#define MQTT_SERVER_IP "192.168.1.23"

#define WIFI_CHECK_INTERVAL 5000 // 5 seconds

#define SENSOR_PERIOD 5000 // Sensor measurement period (5 seconds)

// ================================================
// OBJECT INSTANCING AND VARIABLES
// ================================================
Adafruit_NeoPixel pixels(PIXEL_QTY,
                         PIXEL_PIN,
                         NEO_GRB + NEO_KHZ800); // Initialize NeoPixel

WiFiClient wifiClient;
```

```
36 PubSubClient client(wifiClient); // MQTT client
37
38 DHT dht(26, DHT22);
39
40 StaticJsonDocument<256> jsonDoc; // JSON document to parse incoming MQTT messages
41
42 // Device variables
43 String device_id = "";  // Device ID for the device
44 bool has_id = false;    // Flag to indicate if device ID is present
45 unsigned long id_req_prev = 0;  // Timer for ID request interval
46
47 // WiFi variables
48 unsigned long wifi_last_check = 0;  // Last WiFi check time
49 bool is_wifi_connected = false;    // WiFi connection status
50
51 // MQTT variables
52 bool mqtt_connecting = false;  // Flag to track if connecting to MQTT server
53 unsigned long mqtt_time_prev = 0; // Time of last attempt for MQTT connection
54 unsigned long f_to_z_now = 0; // Timer for notification blink
55
56 // Sensor variables
57 unsigned long sensor_time_prev = 0; // Last sensor reading time
58 float temp = 0;
59 float humi = 0;
60
61 // Sistema
62 uint8_t status = 6;    // Error severity status
63 // 6 - normal
64 // 5 - message from API
65 // 4 . no connection to MQTT
66 // 3 - no connection to WiFi
67 // 2 - warning
68 // 1 - error
69
70 // ================================================
71 // INITIAL SETUP
72 // ================================================
73 void setup() {
74   Serial.begin(115200);  // Start serial communication for debugging
75
76   pinMode(LEDPIN, OUTPUT);  // Set LED pin as output
77
78   pixels.begin();  // Initialize NeoPixel
79   pixels.clear();
80   pixels.show();   // Clear and update NeoPixel display
81
82   connectToWiFi();  // Connect to WiFi
83
84   client.setServer(MQTT_SERVER_IP, 1883); // Set MQTT server IP and port
85   client.setCallback(callback);  // Set the callback function for incoming MQTT messages
86
87   dht.begin();
88 }
89
90 // ================================================
91 // MAIN LOOP
92 // ================================================
93 void loop() {
94   // Check WiFi connection status
95   manageWiFiConnection();
96
97   // If MQTT server is disconnected, try to reconnect
98   if (!client.connected()) {
99     connect_mqttServer();
100  }
101
102  // If device ID has not been assigned yet, request it from the server
103  if (!has_id && client.connected()) {
104    unsigned long id_req_now = millis();
105
106    // Send an ID request every 5 seconds
107    if (id_req_now - id_req_prev >= 5000) {
108      id_req_prev = id_req_now;
109      client.publish("stations/register-need", "id-request");
110    }
111  }
112
113  // Handle different status codes and control NeoPixel LED
114  switch (status) {
115    case 1:
116      blink(pixels.Color(255, 0, 0), 0, 200, 1);  // Red blink for error severity 1
117      break;
118    case 2:
119      blink(pixels.Color(255, 255, 0), 10000, 40, 3);  // Yellow blink for error severity 2
120      break;
121    case 3:
122      blink(pixels.Color(0, 0, 255), 10000, 40, 2);  // Blue blink for error severity 3
123      break;
124    case 4:
125      blink(pixels.Color(255, 0, 255), 10000, 40, 2);  // Purple blink for error severity 4
126      break;
127    case 5:
```

```
128       if (f_to_z_now + 5000 > millis()) {
129         blink(pixels.Color(0, 255, 255), 10000, 40, 1);  // Cyan blink for normal operation
130       } else {
131         status = 6;
132       }
133       break;
134     default:
135       break;
136   }
137
138   // If device is connected and in normal state, perform sensor readings
139   if ((status == 5 || status == 0) && device_id) {
140     sensor_loop();  // Handle sensor data collection and MQTT publishing
141   }
142
143   // Continuously check for incoming MQTT messages
144   client.loop();
145 }
```

Cuadro 12: Librerías, constantes, variables, setup y loop, código de estación de variables ambientales

2. *Loop* de sensor

La función `sensor_loop()` se encarga de realizar lecturas periódicas de los sensores y publicar los datos en un servidor MQTT. En intervalos determinados por la constante `SENSOR_PERIOD`, la función realiza las siguientes acciones:

- Obtiene los valores de temperatura y humedad a través del sensor `DHT`.
- Genera un mensaje en formato JSON que incluye el *ID* del dispositivo, la temperatura y la humedad.
- Convierte el mensaje a un formato adecuado (`char[]`) para su publicación.
- Envía los datos al servidor MQTT en el canal `"devices/readings"`.

```
1  void sensor_loop() {
2    // Store current time for sensor readings
3    unsigned long sensor_time_now = millis();
4
5    // Update sensor readings at regular intervals
6    if (sensor_time_now - sensor_time_prev >= SENSOR_PERIOD) {
7      sensor_time_prev = sensor_time_now;  // Update the last sensor read time
8
9      // get sensor values
10     temp = dht.readTemperature();
11     humi = dht.readHumidity();
12
13     // Create JSON payload to send sensor data
14     String payload = "{\"deviceId\":\""
15                      + device_id
16                      + "\",␣\"temperature\":"
17                      + String(temp)
18                      + ",␣\"humidity\":"
19                      + String(humi)
20                      + "}";
21
22     // Convert payload to char[] for MQTT message
23     char payloadBuffer[100];
24     payload.toCharArray(payloadBuffer, 100);
25
26     // Publish sensor data to MQTT
27     client.publish("devices/readings", payloadBuffer);
28   }
29 }
```

Cuadro 13: Loop de sensor, para estación de variables ambientales

3. Conexión a wifi

Para la conexión a wifi se cuenta con las mismas funciones, para conectar y manejar la disponibilidad de conexión.

```
void connectToWiFi() {
  Serial.println("Attempting to connect to WiFi...");
  WiFi.begin(SSID, PASS);  // Attempt to connect to WiFi with provided SSID and password
}

void manageWiFiConnection() {
  unsigned long current_time = millis();

  // Periodically check WiFi connection status
  if (current_time - wifi_last_check >= WIFI_CHECK_INTERVAL) {
    wifi_last_check = current_time;

    // If not connected to WiFi, attempt reconnection
    if (WiFi.status() != WL_CONNECTED) {
      Serial.println("WiFi connection lost. Reconnecting...");
      WiFi.reconnect();
    } else {
      is_wifi_connected = true;
    }
  }
}
```

Cuadro 14: Conexión a wifi para estación de variables ambientales

4. Conexión a servidor MQTT

El controlador MQTT presenta funcionalidades similares a las del controlador MQTT utilizado en las torres. En la función *connect_ mqttServer*, se mantienen exactamente las mismas operaciones, con la única diferencia de que se elimina la suscripción al canal de *timers*.
Por otro lado, en la función *callback*, se suprime la sección correspondiente al canal *api_ commands* y se ajusta el tópico de suscripción, cambiando de *towers/register-ok* a *stations/register-ok*. El resto de las funciones permanecen sin modificaciones en este código.

```
// Generate a connection with the MQTT server
void connect_mqttServer() {
  // While there is no connection to the MQTT server...
  if (!client.connected()) {

    unsigned long mqtt_time_now = millis();

    if (mqtt_time_now - mqtt_time_prev >= 5000) {
      mqtt_time_prev = mqtt_time_now;
      // Try to reconnect to the MQTT server
      Serial.println("Trying to connect to the MQTT server...");

      // If the connection is successful...
      if (client.connect("ESP_station")) {
        Serial.println("Successfully connected!");

        status = 6;

        // Subscribe to topics
        client.subscribe("stations/register-ok");
      }
      // If not successful...
      else {
        unsigned long wait_time = 500;  // Wait time of 1 second
        if (status >= 4) {
          status = 4;
        }
        Serial.print("Connection failed! rc = ");
        Serial.println(client.state());
        Serial.println("Trying again in 5 seconds.");
        Serial.println(" ");
      }
    }
  }
}

void callback(char* topic, byte* message, unsigned int length) {
  Serial.println("========================");
  Serial.println("= MQTT MESSAGE RECEIVED ");
  Serial.println("========================");
  Serial.println("Message Information");
  Serial.println("------------------------");
  Serial.print("- Topic: ");
  Serial.println(topic);
  Serial.print("- Message: ");

  String messageTemp;
  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
```

```
    messageTemp += (char)message[i];
  }

  // Parse the JSON message
  DeserializationError error = deserializeJson(jsonDoc, messageTemp);
  if (error) {
    Serial.print("JSON Parsing failed: ");
    Serial.println(error.c_str());
    return;
  }

  Serial.println("");
  Serial.println("------------------------");
  Serial.println("Extracted Information");
  Serial.println("------------------------");

  if (String(topic) == "stations/register-ok" && !has_id) {
    // Notify receipt of command
    if (status >= 5) {
      f_to_z_now = millis();
      status = 5;
    }
    // Extract values
    const char* deviceIdValue = jsonDoc["id"];

    if (deviceIdValue) {
      Serial.printf("deviceId: %s\n", deviceIdValue);

      // Save the deviceId (assuming device_id is a global variable)
      if (strcmp(deviceIdValue, "") != 0) {
        device_id = String(deviceIdValue);
        has_id = true;
        Serial.println("------------------------");
        Serial.println("Action performed");
        Serial.println("------------------------");
        Serial.println("Device registered!!!");
      }
    } else {
      Serial.println("No deviceId found in JSON.");
    }
  }
  Serial.println("");
}
```

Cuadro 15: Conexión a servidor MQTT para estación de variables ambientales

5. Funciones auxiliares

Por último, también se utilizó en la estación el titileo del módulo WS2812B para indicar problemas de desconexión u otros eventos que requiren de notificación al usuario.

```
void blink(uint32_t color,
           unsigned long wait_time,
           int alarm_speed,
           int steps) {

  static unsigned long last_blink_time = 0;
  static unsigned long last_pause_time = 0;
  static int brightness = 0;
  static bool increasing = true;
  static int blink_count = 0;
  static bool paused = false;

  unsigned long current_time = millis();

  // Handle pause between blinking cycles
  if (paused) {
    if (current_time - last_pause_time >= wait_time) {
      paused = false; // End the pause
      blink_count = 0; // Reset blink count
      brightness = 0;  // Ensure brightness starts from 0
    } else {
      pixels.setPixelColor(0, 0, 0, 0); // Turn off during pause
      pixels.show();
      return;
    }
  }

  // Handle blinking
  if (current_time - last_blink_time >= alarm_speed) {
    last_blink_time = current_time;

    // Adjust brightness
    if (increasing) {
      brightness++;
      if (brightness >= steps) {
```

```
        increasing = false;
        blink_count++; // Count a completed blink
      }
    } else {
      brightness--;
      if (brightness <= 0) {
        increasing = true;
      }
    }

    // Update pixel color with adjusted brightness
    pixels.setPixelColor(0,
      (brightness * ((color >> 16) & 0xFF)) / steps, // Red
      (brightness * ((color >> 8) & 0xFF)) / steps,  // Green
      (brightness * (color & 0xFF)) / steps          // Blue
    );
    pixels.show();

    // Check if blink count reached steps, then pause
    if (blink_count >= steps) {
      paused = true;
      last_pause_time = current_time;
    }
  }
}
```

Cuadro 16: Funciones auxiliares para estación de variables ambientales

CAPÍTULO XI

Conexión entre API y dispositivos

A. Apertura de ingress NGINX

Al finalizar la programación de la API, se configuró el proxy de ingress NGINX para aceptar tráfico sin necesidad de un nombre de *host*. Este proxy redirige las solicitudes a los diferentes microservicios dependiendo de la ruta especificada en la URL. Gracias a esta configuración, es posible acceder a la API desde cualquier dispositivo conectado a la misma red.
Sin embargo, este enfoque no es recomendable desde una perspectiva de seguridad, ya que permite que cualquier dispositivo en la red tenga acceso a la API, lo que podría facilitar la inyección de datos maliciosos. Por ejemplo, más adelante se detalla cómo el *gateway* realiza peticiones HTTP a la API utilizando la dirección IP de la máquina anfitriona que ejecuta los contenedores Docker con los microservicios. Una URL tan simple como `http://192.168.1.1/api/users/signup/` podría ser utilizada para crear usuarios ilimitados mediante un *script*, saturando rápidamente la base de datos de Mongoose.
Para mitigar estos riesgos, se recomienda exponer únicamente un servicio a la red local (y en otra iteración, a internet), preferiblemente un *gateway API* que gestione todas las solicitudes hacia los microservicios. Este enfoque permite implementar controles de acceso más estrictos, como autenticación, autorización, validación de datos y límites de tasa (*rate limiting*), reduciendo significativamente las vulnerabilidades.

B. Configuración de Raspberry Pi

Para configurar la Raspberry Pi como extitgateway, se instaló el sistema operativo Raspberry Pi OS de 32 bits. La versión de 64 bits fue descartada debido a problemas con la inicialización de Mosquitto. El acceso inicial a la Raspberry Pi se realiza mediante una aplicación de SSH, utilizando el nombre extitpi.local o el extithostname definido durante la instalación del sistema operativo. Una vez dentro, es necesario conectar el dispositivo a la red local (en caso de no haberlo hecho durante la instalación) utilizando el comando extttsudo raspi-config. Este comando abre el menú de configuración, donde se debe navegar a extitSystem Options > Wireless LAN e ingresar las credenciales de la red. Luego, se puede ejecutar el comando extttifconfig para obtener la dirección IP del dispositivo y acceder a él mediante

extitRealVNC.
Para habilitar el acceso mediante VNC, primero es necesario activarlo desde las configuraciones de la Raspberry Pi. Además, se debe configurar el dispositivo para que inicie con la interfaz gráfica en lugar de la consola, lo que facilita su manipulación. Una vez completado esto, se procede a instalar el servidor MQTT Mosquitto utilizando el siguiente comando:

```
sudo apt install -y mosquitto mosquitto-clients
```

Tras la instalación, es necesario reiniciar la Raspberry Pi antes de comenzar a utilizar Mosquitto. Para verificar su funcionamiento, se pueden ejecutar los siguientes comandos:

```
mosquitto_sub -h "localhost" -p 1883 -t "topic"
mosquitto_pub -h "localhost" -p 1883 -t "topic" -m "mensaje"
```

Si las pruebas son exitosas, se debe configurar Mosquitto editando su archivo de configuración. Esto se realiza con el siguiente comando:

```
sudo nano /etc/mosquitto/conf.d/mosquitto.conf
```

En este archivo, se deben agregar las líneas:

```
listener 1883
allow_anonymous true
```

Finalmente, se reinicia Mosquitto para aplicar los cambios:

```
sudo systemctl restart mosquitto
```

Con esto, la configuración de la Raspberry Pi como extitgateway y servidor MQTT queda completada.

C. Código *bridge* en la Raspberry Pi

El código ejecutado en la Raspberry Pi, denominado puente o *bridge*, implementa un procedimiento para procesar solicitudes HTTP enviadas al gateway, gestionarlas y transmitirlas a los dispositivos conectados al servidor MQTT, también alojado en la Raspberry Pi. De manera similar, si un dispositivo desea enviar lecturas, este debe enviar un mensaje a través de MQTT, el cual será procesado por el puente mediante sus suscripciones MQTT y reenviado al sistema principal.
El código define varias variables y constantes importantes, entre ellas el diccionario *listings*, que almacena los promedios de las lecturas enviadas por los dispositivos. Estos datos son eliminados del diccionario una vez que han sido enviados. Además, se incluye la función *get_machine_ip*, diseñada para obtener automáticamente la dirección IP de la Raspberry Pi. También se desarrolló la función *send_req*, que permite enviar solicitudes a la API con una estructura predefinida.
La función de *callback on_connect* gestiona diferentes procedimientos según el tópico desde el cual se origina un mensaje MQTT. Por ejemplo, para el registro de dispositivos, el código envía una solicitud al servicio de dispositivos, permitiendo asignar datos al nuevo dispositivo. En el caso del tópico *devices/readings*, el puente recopila información sobre las lecturas de los

dispositivos y, cada cierto intervalo de tiempo (en este caso, 5 minutos), envía el promedio de todas las lecturas. Existe una excepción: si el dispositivo que envió la lectura no había enviado ninguna anteriormente, esta primera lectura se envía de inmediato para evitar problemas en la API.

Por otra parte, el código incluye la clase *RequestHandler*, que se encarga de recibir y gestionar las solicitudes según su tipo y contenido. En el caso de solicitudes POST, los datos del *body* se convierten a un objeto JSON para facilitar su manipulación. Según el contenido, el puente envía mensajes MQTT a los dispositivos correspondientes. El caso más común es cuando se envía un comando desde la API: el puente recibe la solicitud y envía un mensaje MQTT para que el dispositivo ejecute el comando.

Finalmente, el código implementa funciones del tipo *run* para inicializar tanto el servidor HTTP como el cliente MQTT. Estas funciones tienen como propósito estructurar mejor el código y simplificar su ejecución. A continuación, se presenta el código completo del puente en el cuadro 17.

```
import sys
import json
import logging
import threading
import requests
import socket
import paho.mqtt.client as paho
from http.server import BaseHTTPRequestHandler, HTTPServer

# MQTT Broker settings
MQTT_BROKER = "localhost"
MQTT_PORT = 1883

# API Endpoint
# API_URL = "http://192.168.70.70"
API_URL = "http://192.168.1.9"

# Logging configuration
logging.basicConfig(level=logging.INFO, format="%(asctime)s - %(message)s")

# Global MQTT client
mqtt_client = paho.Client()

# Received readings dictionary
readings = {}

### Gets the IP of the machine in the WLAN interface
def get_machine_ip():
    try:
        # Create a temporary socket to determine the machine's IP
        with socket.socket(socket.AF_INET, socket.SOCK_DGRAM) as s:
            s.connect(("8.8.8.8", 80))  # Google's public DNS server
            ip = s.getsockname()[0]  # Get the local IP from the socket
        return ip
    except Exception as e:
        return f"Error: {e}"

def send_req(req, route):
    response = ""
    print(req)
    try:
        # Parse the JSON message
        data = json.loads(json.dumps(req))

        # Make an HTTP request to the API with the received data
        response = requests.post(API_URL + route, json=data)
        logging.info(f"API Response: {response.status_code} - {response.text}")
    except json.JSONDecodeError as e:
        logging.error(f"Failed to parse JSON: {e}")
    except requests.RequestException as e:
        logging.error(f"HTTP Request failed: {e}")
    return response

# MQTT Callback to handle messages
def on_message(client, userdata, msg):
    logging.info(f"Message received on topic '{msg.topic}': {msg.payload.decode('utf-8')}")

    # If one device publishes it needs an ID...
    if msg.topic == "devices/register-need" and msg.payload.decode('utf-8') == "id-request":
        logging.info(f"Registering new device")
        # Build HTTP request
        register_req = {
            "type": "tower",
            "name": "testTower",
            "status": "online",
```

```
            "gatewayIp": get_machine_ip(),
            "userId": "673bf1a3a401ce7ffca7a030"
        }
        # Make request to API
        try:
            response = send_req(register_req, "/api/devices/via-gateway")
            mqtt_client.publish("devices/register-ok", response.text, qos=0)
            logging.info(f"Data published to MQTT topic 'devices/register'")
        except Excepection as e:
            print(e)

    # If a device publishes a reading
    elif msg.topic == "devices/readings":
        # Decode the reading's payload
        reading = json.loads(msg.payload.decode('utf-8'))

        # Depending on the type of device, fill with the vars in the payload
        vars = []
        for var in reading.keys():
            vars.append(var)

        # If the deviceId is already in the readings dictionary...
        if reading["deviceId"] in readings:
            readings[reading["deviceId"]][vars[1]].append(int(reading[vars[1]]))
            readings[reading["deviceId"]][vars[2]].append(int(reading[vars[2]]))
            if len(readings[reading["deviceId"]][vars[1]]) > 60:
                reading_req = {
                    "deviceId": reading["deviceId"],
                    "payload": {
                        vars[1]: round(sum(readings[reading["deviceId"]][vars[1]]) / len(readings[reading["deviceId"]][
                            vars[1]]), 1),
                        vars[2]: round(sum(readings[reading["deviceId"]][vars[2]]) / len(readings[reading["deviceId"]][
                            vars[2]]), 1)
                    }
                }
                response = send_req(reading_req, "/api/readings")
                readings[reading["deviceId"]][vars[1]] = []
                readings[reading["deviceId"]][vars[2]] = []

        # If the deviceId is NOT in the readings dictionary...
        else:
            # add the device to the readings dictionary
            readings[reading["deviceId"]] = {
                vars[1]: [int(reading[vars[1]])],
                vars[2]: [int(reading[vars[2]])]
            }
            # build and send a new reading POST request to
            # readings microservice
            reading_req = {
                "deviceId": reading["deviceId"],
                "payload": {
                    vars[1]: int(reading[vars[1]]),
                    vars[2]: int(reading[vars[2]])
                }
            }
            response = send_req(reading_req, "/api/readings")

    elif msg.topic == "devices/timers":
        pass


# HTTP Handler class
class RequestHandler(BaseHTTPRequestHandler):
    # setup the response to the API
    def _set_response(self, body = None):
        self.send_response(200)
        self.send_header("Content-Type", "application/json")
        self.end_headers()
        if body:
            self.wfile.write(json.dumps(body).encode("utf-8"))

    # handler for POST requests
    def do_POST(self):
        # extract information from the request
        content_length = int(self.headers["Content-Length"])
        post_data = self.rfile.read(content_length).decode("utf-8")
        logging.info(f"POST request received:\n{post_data}")

        try:
            # Publish the POST data to the MQTT topic
            mqtt_client.publish("devices/api-commands", post_data, qos=0)
            logging.info(f"Data published to MQTT topic 'devices/api-commands'")

            # Create reponse
            response = {"message": "success"}
            self._set_response(body=response)
        except Exception as e:
            logging.error(f"Failed to publish data to MQTT: {e}")

            # Create reponse
            response = {"message": "error", "details": str(e)}
            self._set_response(body=json.dumps(response))
```

```
        self._set_response()

# Run HTTP server
def run_http_server(port=3000):
    server_address = ("", port)
    httpd = HTTPServer(server_address, RequestHandler)
    logging.info(f"Starting HTTP server on port {port}...")
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass
    httpd.server_close()
    logging.info("Stopping HTTP server...")

# Run MQTT client
def run_mqtt_client():
    logging.info("Connecting to MQTT broker...")
    mqtt_client.on_message = on_message
    mqtt_client.connect(MQTT_BROKER, MQTT_PORT, 60)

    # subscribe to different MQTT channels
    mqtt_client.subscribe("devices/register-need")
    mqtt_client.subscribe("devices/readings")
    mqtt_client.subscribe("devices/notify")
    logging.info(f"Subscribed to topic 'devices/register-need'")
    logging.info(f"Subscribed to topic 'devices/readings'")
    logging.info(f"Subscribed to topic 'devices/notify'")
    mqtt_client.loop_forever()

# Main function
if __name__ == "__main__":
    try:
        # Start HTTP server in a separate thread
        http_thread = threading.Thread(target=run_http_server, args=(3000,))
        http_thread.daemon = True
        http_thread.start()

        # Run MQTT client in the main thread
        run_mqtt_client()
    except Exception as e:
        logging.error(f"Unexpected error: {e}")
    finally:
        mqtt_client.disconnect()
```

Cuadro 17: Código del puente

CAPÍTULO XII

Implementación de *machine learning*

Para la aplicación de *machine learning*, se recolectaron resultados relacionados con las variables del sistema y del entorno. Además, se varió el intervalo de encendido y apagado en incrementos de 5 minutos. Este procedimiento se llevó a cabo durante 45 días, recolectando datos diariamente en la Raspberry Pi. Cabe destacar que la nota, con un rango de 0 a 100, se ingresó manualmente.
Una vez recopiladas todas las lecturas necesarias, se desarrolló un código en Python para evaluar cuatro modelos de *machine learning*: regresión lineal, regresión polinómica de grado 2, árbol de decisión y bosque aleatorio. A continuación, se presenta el código utilizado para generar los resultados que se discutirán más adelante.

A. Código de *machine learning*

En el código se empleó la librería *scikit-learn* para realizar las pruebas de *machine learning*. Utilizando *pandas*, se cargaron los datos recopilados desde el archivo `readings.xlsx`. Las variables independientes se almacenaron como un *dataframe* individual denominado X, mientras que la columna `nota` se guardó en Y, correspondiente a la variable dependiente.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error, r2_score

# Load the data from the generated file
df = pd.read_excel("readings.xlsx")

# Separate independent and dependent variables
X = df[["Temperature", "Humidity", "Ambient light", "Off interval",
        "On interval"]]
y = df["Grade"]

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Models to test
models = {
    "Linear Regression": LinearRegression(),
    "Polynomial Regression (degree 2)": PolynomialFeatures(degree=2),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
```

```
    "Random Forest": RandomForestRegressor(random_state=42),
}

# Train and evaluate the models
results = []

# Linear regression
linear_model = models["Linear Regression"]
linear_model.fit(X_train, y_train)
linear_predictions = linear_model.predict(X_test)
linear_mse = mean_squared_error(y_test, linear_predictions)
linear_r2 = r2_score(y_test, linear_predictions)
results.append(("Linear Regression", linear_mse, linear_r2))

# Polynomial regression
poly = models["Polynomial Regression (degree 2)"]
X_poly_train = poly.fit_transform(X_train)
X_poly_test = poly.transform(X_test)
polynomial_model = LinearRegression()
polynomial_model.fit(X_poly_train, y_train)
polynomial_predictions = polynomial_model.predict(X_poly_test)
polynomial_mse = mean_squared_error(y_test, polynomial_predictions)
polynomial_r2 = r2_score(y_test, polynomial_predictions)
results.append(("Polynomial Regression (degree 2)", polynomial_mse, polynomial_r2))

# Decision tree
tree_model = models["Decision Tree"]
tree_model.fit(X_train, y_train)
tree_predictions = tree_model.predict(X_test)
tree_mse = mean_squared_error(y_test, tree_predictions)
tree_r2 = r2_score(y_test, tree_predictions)
results.append(("Decision Tree", tree_mse, tree_r2))

# Random forest
forest_model = models["Random Forest"]
forest_model.fit(X_train, y_train)
forest_predictions = forest_model.predict(X_test)
forest_mse = mean_squared_error(y_test, forest_predictions)
forest_r2 = r2_score(y_test, forest_predictions)
results.append(("Random Forest", forest_mse, forest_r2))

# Display results
results_df = pd.DataFrame(results, columns=["Model", "MSE", "R^2"])
results_df.sort_values(by="R^2", ascending=False, inplace=True)
print(results_df)

# Plot predictions for each model
def plot_predictions(y_test, predictions, title):
    plt.figure(figsize=(8, 6))
    plt.scatter(range(len(y_test)), y_test, color="blue", label="Actual Data")
    plt.scatter(range(len(predictions)), predictions, color="red", label="Predictions")
    plt.title(title)
    plt.xlabel("Sample Index")
    plt.ylabel("Score")
    plt.legend()
    plt.show()

# Plot results
model_predictions = {
    "Linear Regression": linear_predictions,
    "Polynomial Regression (degree 2)": polynomial_predictions,
    "Decision Tree": tree_predictions,
    "Random Forest": forest_predictions,
}

for model_name, predictions in model_predictions.items():
    plot_predictions(y_test, predictions, f"{model_name} - Actual vs Predictions")

# Analyze the impact of independent variables
for column in X.columns:
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=df[column], y=df["Grade"], hue=df["Off interval"], palette="viridis")
    plt.title(f"Impact of {column} on Score")
    plt.xlabel(column)
    plt.ylabel("Score")
    plt.legend(title="Shutdown Interval in minutes", loc="best")
    plt.show()

# Analize the best on and off intervals
average_scores = df.groupby(["On interval", "Off interval"])["Grade"].mean().reset_index()

plt.figure(figsize=(12, 8))
heatmap_data = average_scores.pivot(
    index="On interval",
    columns="Off interval",
    values="Grade"
)

sns.heatmap(
    heatmap_data,
    annot=True,
    fmt=".1f",
```

```
    cmap="coolwarm",
    cbar_kws={'label': 'Average Score'},
    annot_kws={"size": 14}
)

# Personalize titles and labels
plt.title("On and Off Intervals vs Average Score", fontsize=18)
plt.xlabel("Shutdown Interval in minutes", fontsize=16)
plt.ylabel("Startup Interval in minutes", fontsize=16)

# Make labels bigger
plt.xticks(fontsize=14)
plt.yticks(fontsize=14)

# Change font size
cbar = plt.gca().collections[0].colorbar
cbar.ax.tick_params(labelsize=14)

plt.show()
```

Cuadro 18: Modelos de machine learning

Para generar un modelo de *machine learning*, es necesario dividir el conjunto de datos en dos subconjuntos: uno para el entrenamiento del modelo y otro para realizar pruebas. Los *dataframes* de entrenamiento son $X\_\ train$ y $y\_\ train$, mientras que los de prueba son $X\_\ test$ y $y\_\ test$. Además, se definió una lista de resultados vacía, la cual se fue llenando a medida que se evaluaban los modelos.

En la siguiente sección, se aplicaron las funciones disponibles en la librería *scikit-learn*. La función `fit` se encargó de entrenar el modelo utilizando los datos de entrenamiento. Posteriormente, se empleó la función `predict` para generar predicciones basadas en los datos de prueba. Finalmente, se calcularon las métricas de evaluación: `mean_squared_error` para medir el error cuadrático medio de las predicciones, y `r2_score` para determinar la proporción de la variabilidad de los datos explicada por el modelo. En el caso de `mean_squared_error` o MSE, los valores van de 0 a ∞, siendo los valores cercanos a 0 los mejores. Por otro lado, en `r2_score` o R2, los valores van de $-\infty$ a 1, siendo los valores cercanos a 1 los mejores.

El procedimiento descrito se repitió para los cuatro modelos. En el caso de la regresión polinómica, se realizaron transformaciones adicionales. Primero, se utilizó la función `fit_transform` para convertir las características de $X\_\ train$ a una versión polinómica de grado 2. Posteriormente, se aplicó `transform` a $X\_\ test$ para asegurar que las mismas transformaciones se aplicaran de manera consistente. Es importante notar que `fit_transform` reajusta completamente el modelo, mientras que `transform` aplica las transformaciones ya calculadas. Tras este paso, el procedimiento continuó de forma similar al de los otros modelos.

B. Resultados de los modelos

En la sección anterior, se describió el código utilizado para generar diversas gráficas, las cuales incluyen los datos reales y las predicciones de los modelos, así como gráficas de impacto para cada variable independiente y una representación del intervalo que obtuvo los mejores resultados.

1. Modelos y predicciones

La primera figura a analizar es la Figura 32, que muestra los resultados de las predicciones del modelo de regresión lineal. Se observa que las predicciones, representadas en rojo, se acercan ligeramente a los valores reales, mostrados en azul. Este modelo presenta un MSE de 10.54 y un R^2 de 0.6359. Estos resultados indican un error significativo en las predicciones, con un

rango de operación limitado. Por lo tanto, queda claro que la regresión lineal multivariable no es el modelo más adecuado para esta aplicación.

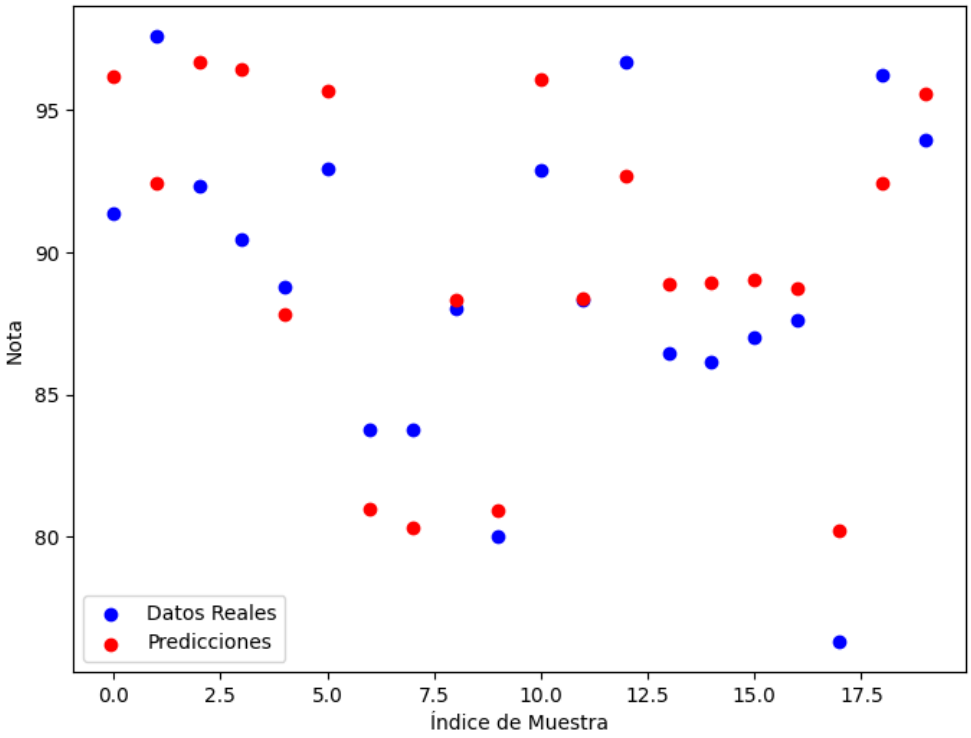

Figura 32: Modelo de regresión lineal

La Figura 33 muestra los resultados de las predicciones del modelo de regresión polinómica de grado 2. En este caso, las predicciones son ligeramente menos precisas en comparación con el modelo lineal, lo que se refleja en un MSE de 13.16 y un R^2 de 0.5442. Una característica de este modelo es que, a medida que se incrementa el grado del polinomio, el R^2 puede disminuir. Sin embargo, este aumento también puede llevar a una reducción en el rango de operación, ya que el modelo tiende a ajustarse específicamente a los datos de entrenamiento.

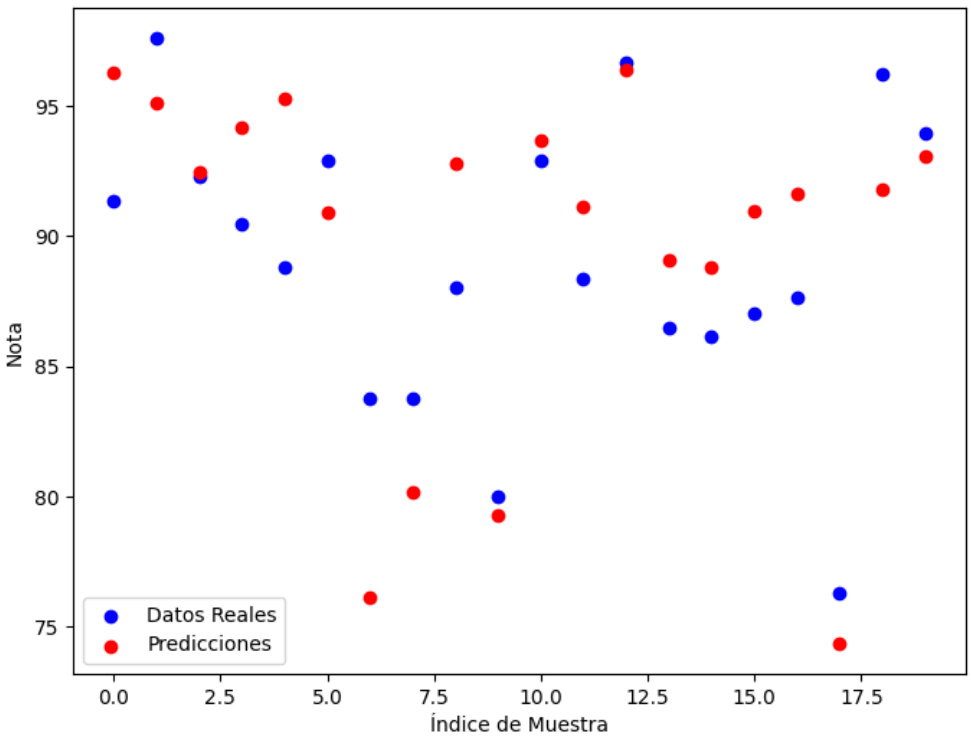

Figura 33: Modelo de regresión polinómica de grado 2

En el caso del modelo de árbol de decisión, las predicciones se acercan aún más a los datos reales. Este modelo obtuvo un MSE de 8.876 y un R^2 de 0.6926, lo que demuestra un mejor desempeño en términos de error y dispersión en comparación con el modelo lineal.

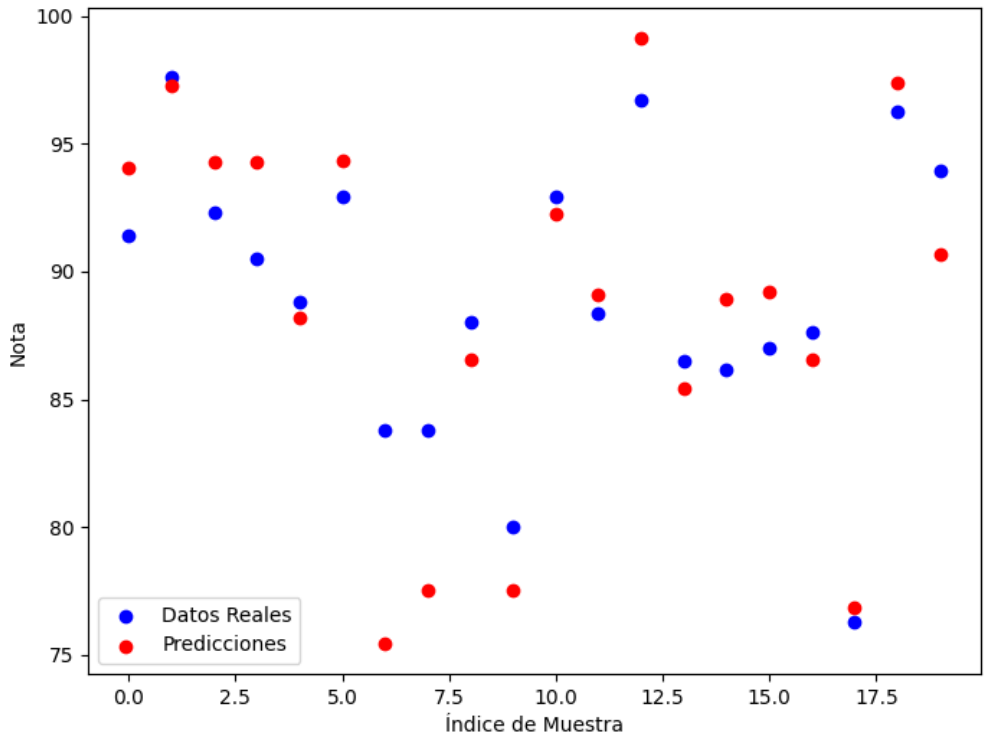

Figura 34: Modelo de árbol de decisión

Por último, el modelo de bosque aleatorio demostró ser el más eficaz de todos. Este modelo obtuvo un MSE de 5.655 y un R^2 de 0.8042, lo que indica un ajuste superior a los datos. Aunque algunas predicciones se desviaron de los valores reales, estos casos corresponden a las observaciones con calificaciones más bajas. En general, el modelo de bosque aleatorio ofrece los mejores resultados para esta aplicación.

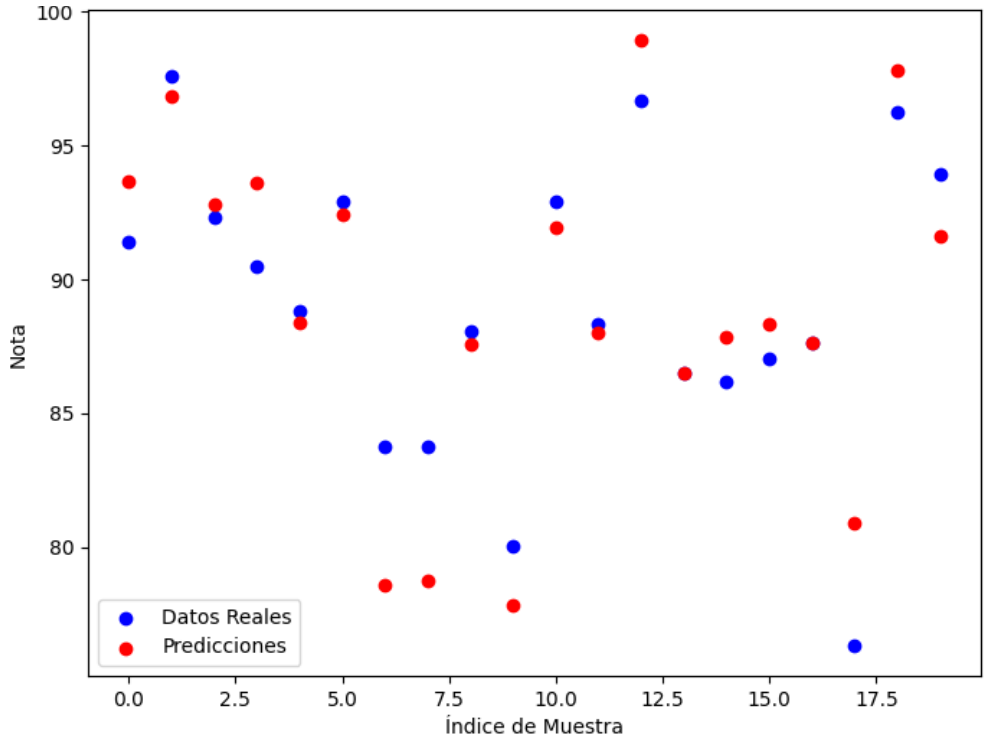

Figura 35: Modelo de *bosque aleatorio*

2. Análisis de variables independientes

Se realizó un análisis de las variables independientes para evaluar su impacto en los resultados. A continuación, se presentan gráficas que comparan los valores de las variables independientes con las calificaciones obtenidas en cada caso.

En la Figura 36, que analiza el impacto de la temperatura, se observa que esta variable tiene poca influencia en la calificación final. Esto se concluye debido a la dispersión de los puntos a lo largo del eje vertical, sin una tendencia clara. Además, los puntos aparecen agrupados verticalmente debido a que los valores de temperatura se redondearon al número entero más cercano. Las gráficas también incluyen una codificación de colores basada en el intervalo de tiempo apagado, que se analizará más adelante por ser la variable más influyente.

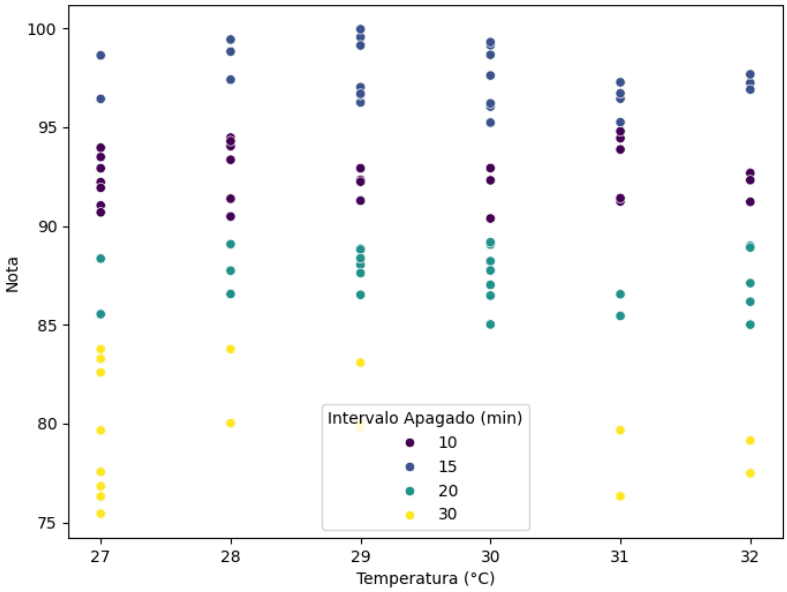

Figura 36: Impacto de la temperatura

La Figura 37 muestra el impacto de la humedad. En este caso, los datos presentan una mayor dispersión en comparación con los de la temperatura, ya que se conservaron más puntos decimales. Sin embargo, al igual que la temperatura, la humedad tiene poca influencia en las calificaciones. Además, como el sistema se encuentra en un entorno cerrado, las variaciones de humedad son limitadas en comparación con un entorno abierto.

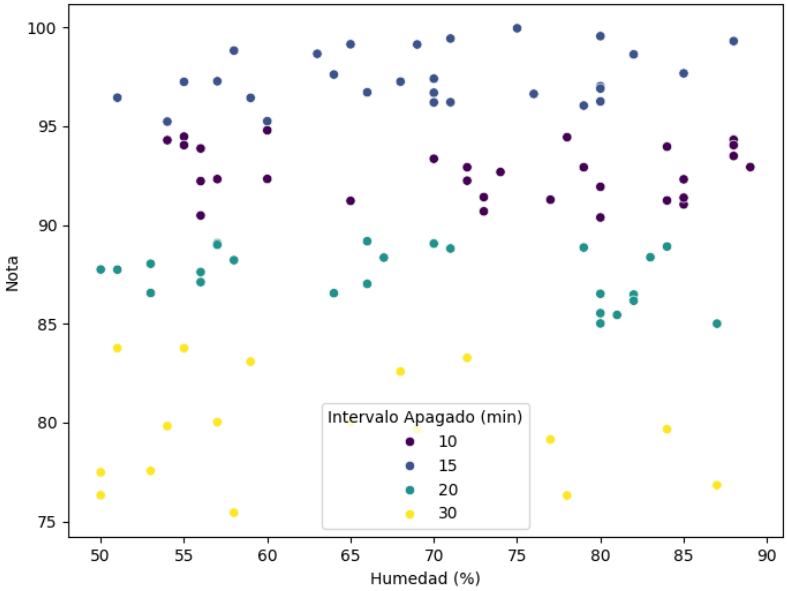

Figura 37: Impacto de la humedad

En la Figura 38, se analiza el impacto de la luz sobre las calificaciones. Al igual que las variables anteriores, la luz muestra una influencia limitada en el estado del cultivo.

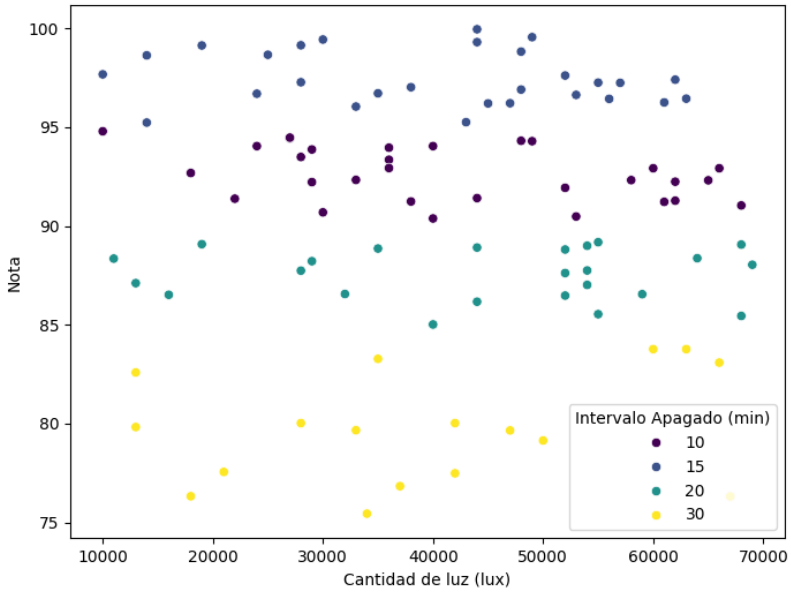

Figura 38: Impacto de la luz

La Figura 39 analiza el impacto del intervalo de encendido de la bomba. Los puntos aparecen alineados verticalmente debido a los intervalos de tiempo establecidos, con un paso de 5 minutos. Por limitaciones de tiempo, no se evaluó el intervalo de 25 minutos, aunque se infiere que los resultados habrían sido similares.

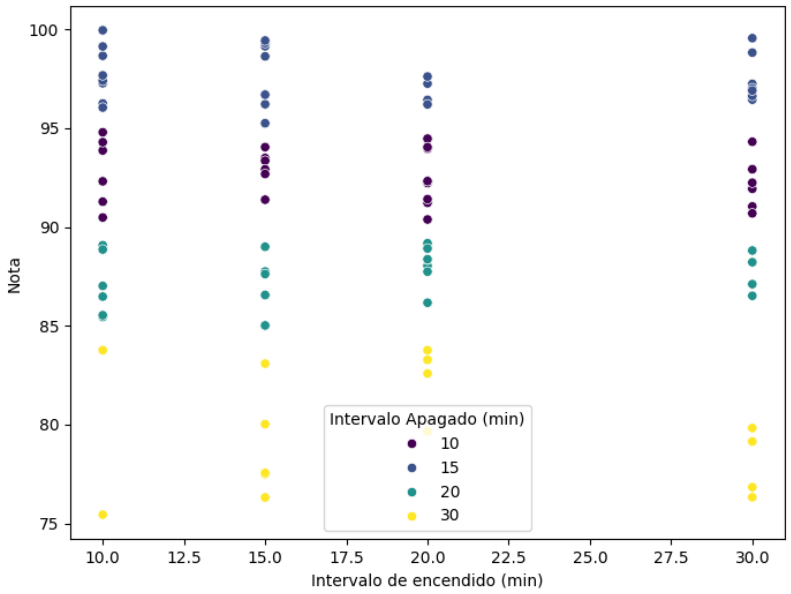

Figura 39: Impacto del intervalo de encendido

Por último, la Figura 40 analiza el intervalo de apagado, que resultó ser la variable con mayor impacto en las calificaciones. Los puntos se concentran en diferentes regiones dependiendo del intervalo utilizado. Se concluye que el intervalo de 15 minutos es el más recomendable, ya que obtuvo las mejores calificaciones. Durante las observaciones, se identificó que reducir

el intervalo de apagado a 10 minutos provocó manchas cafés y negras en las hojas, lo que redujo las calificaciones. Por otro lado, intervalos de 20 y 30 minutos causaron pérdida de pigmentación en las hojas, lo que también afectó negativamente. El intervalo de 15 minutos demostró ser el más equilibrado, logrando un cultivo en óptimas condiciones.

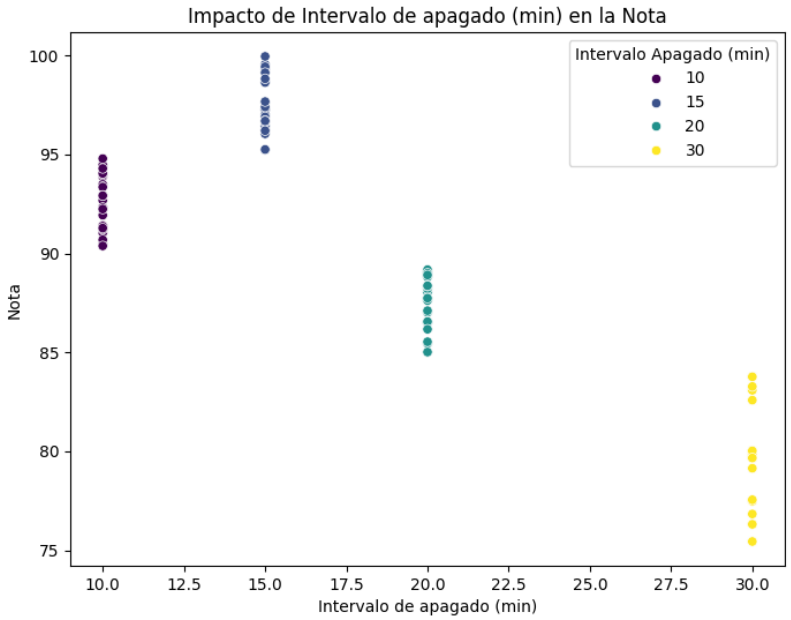

Figura 40: Impacto del intervalo de apagado

CAPÍTULO XIII

Conclusiones

En este trabajo de graduación, se desarrolló e implementó un sistema de torres hidropónicas automatizadas basadas en internet de las cosas (IoT) para la agricultura urbana, que combina sensores y actuadores conectados a través de microcontroladores ESP32 y una Raspberry Pi centralizada que comunica con una API en la red local mediante una arquitectura de microservicios. Este diseño permite el monitoreo y control remoto en tiempo real, facilitando la automatización en la toma de decisiones para optimizar variables de cultivo.

El uso de IoT en este sistema de torres hidropónicas ha demostrado diversos beneficios significativos. En cuanto a la eficiencia en el uso de recursos, la automatización de variables como el riego y la iluminación, junto con el análisis de datos en tiempo real, ha permitido reducir el consumo de agua y energía, haciéndolo un sistema más sostenible y respetuoso con el medio ambiente. El impacto ambiental y la sostenibilidad son aspectos clave, ya que este enfoque optimiza el uso de agua y elimina la necesidad de grandes superficies de tierra y fertilizantes, lo cual representa una alternativa a la agricultura convencional. Esto responde a la creciente demanda de soluciones agrícolas responsables y sostenibles, enfocadas en el uso racional de recursos.

En términos de control remoto y escalabilidad, el sistema apunta a que, en un una futura iteración, se pueda monitorear y gestionar las torres desde cualquier lugar con acceso a internet, lo que facilita la supervisión y administración de múltiples torres en simultáneo. Esta funcionalidad abre la posibilidad de escalar el modelo a proyectos de mayor envergadura, como granjas urbanas, permitiendo que este tipo de cultivo se adapte a las necesidades de producción en áreas metropolitanas. Además, el sistema destaca por su adaptabilidad y flexibilidad, ya que la arquitectura modular, tanto en la estructura como en la API basada en microservicios, permite la integración de nuevos sensores y actuadores, así como la personalización de funciones según las necesidades de cada usuario o tipo de planta, garantizando que el sistema sea fácilmente adaptable a distintos escenarios de cultivo.

La innovación tecnológica en la agricultura urbana es otro aspecto destacado de este proyecto, ya que integra tecnología avanzada en espacios reducidos, característicos de las áreas urbanas, para producir alimentos frescos y de alta calidad. Este sistema hidropónico

basado en IoT no solo promueve la autosuficiencia alimentaria, sino que también ofrece una solución adaptable a las limitaciones espaciales de las ciudades, lo cual es fundamental para responder a la creciente demanda de alimentos en zonas urbanas densamente pobladas. En cuanto al *machine learning*, se incorporaron modelos que optimizan los intervalos de riego y ajustan otras variables del sistema en función de los datos históricos y las condiciones actuales del entorno. Esto permite una autogestión más avanzada, disminuyendo el esfuerzo humano y mejorando el rendimiento de los cultivos al adaptarse de forma automática a cambios en las condiciones ambientales.

Durante el desarrollo de este proyecto, se lograron cada uno de los objetivos específicos propuestos. Se diseñó e implementó un sistema hidropónico vertical automatizado mediante el método de riego por goteo, optimizando el suministro de agua a las plantas. Además, se desarrolló un control central basado en una Raspberry Pi, que recopila y envía datos del sistema a la API en la nube y permite la ejecución de modelos de *machine learning* para mejorar así la eficiencia del riego y otros factores. Asimismo, se crearon sistemas periféricos con ESP32 para medir variables ambientales, asegurando un monitoreo constante de las condiciones de cultivo. La comunicación entre los sistemas periféricos y el control central se estableció de manera inalámbrica mediante el protocolo MQTT, logrando una transmisión de datos fiable y eficiente. Finalmente, se implementaron funcionalidades de IoT para el monitoreo y control remoto del sistema, permitiendo la supervisión en tiempo real y el ajuste de condiciones desde cualquier ubicación (con conexión a la red local), a través de una interfaz web basada en microservicios, que ofrece una visualización amigable y accesible para el usuario final.

Este proyecto no solo cumple con los objetivos iniciales planteados, sino que además sienta las bases para un modelo de agricultura urbana sostenible, tecnológicamente avanzado, y adaptable, contribuyendo al desarrollo de soluciones agrícolas innovadoras que responden a los desafíos actuales de sostenibilidad y demanda de alimentos en las ciudades.

CAPÍTULO XIV

Recomendaciones

Para el desarrollo y expansión del proyecto, existen varias áreas clave que podrían optimizar la implementación actual y abrir nuevas posibilidades de mejora. Las siguientes recomendaciones están dirigidas a aquellos interesados en continuar con la evolución de este sistema, desde la mejora de su arquitectura hasta la incorporación de nuevas funcionalidades que amplíen su alcance. A continuación, se detallan varias sugerencias importantes que podrían ser de utilidad para futuras fases del proyecto, tanto desde el punto de vista técnico como operativo.

A. Conocimientos previos

Es fundamental que las personas interesadas en desarrollar o colaborar en este proyecto cuenten con una base sólida en programación, especialmente en lenguajes como Python, C++, JavaScript, HTML y CSS, ya que son necesarios para interactuar con los diversos componentes del sistema. También se recomienda haber trabajado con el ESP32, particularmente en proyectos que impliquen clientes WiFi y, preferentemente, clientes MQTT. Además, es importante tener conocimientos en el manejo de sistemas basados en Linux, específicamente en la Raspberry Pi, ya que este último es requerido si se decide implementar el *bridge* en la Raspberry Pi para conectar los protocolos HTTP y MQTT. Si no se cuentan con estos conocimientos, existen numerosas herramientas y recursos en línea, como cursos y tutoriales, que pueden proporcionar la ayuda necesaria para superar cualquier dificultad técnica.

B. Eclipse Mosquitto

La Raspberry Pi se utilizó principalmente para levantar el servidor MQTT de Mosquitto y para escuchar peticiones HTTP de la API. Sin embargo, durante la investigación de

imágenes de Docker para implementar dentro de la API (como *Ingress NGINX* y *nats-streaming-server*), se encontró una imagen llamada *Eclipse Mosquitto*, que, de acuerdo a la documentación y ejemplos en internet, ofrece las mismas funcionalidades que el servidor de Mosquitto instalado localmente en la Raspberry Pi. Además, permite la fácil conexión de dispositivos IoT como el ESP32 a la API sin la necesidad de *hardware* adicional. Esta imagen se recomienda para futuras implementaciones, ya que ahorra trabajo y costos, especialmente en caso de no disponer de una Raspberry Pi.

C. Microservicio de *machine learning*

Para optimizar el rendimiento del sistema y permitir una mayor escalabilidad, se recomienda trasladar la implementación de *machine learning* de un código embebido en la Raspberry Pi a un microservicio en la nube. Este enfoque permite que los modelos de *machine learning* recalculen constantemente las variables del sistema, optimizando el cultivo en tiempo real y brindando flexibilidad al usuario para decidir si desea activar esta función. Además, al delegar estos cálculos en la nube, se reduce la carga de procesamiento en dispositivos locales y se proporciona una solución más escalable para sistemas de mayor envergadura.

D. Ampliación de alcance del proyecto

Existen varias formas de ampliar y diversificar los dispositivos involucrados en el proyecto. Se recomienda añadir nuevas clases de dispositivos, como dosificadores, medidores de pigmentación, válvulas inteligentes, entre otros. Por ejemplo, el medidor de pigmentación permitiría obtener datos cuantitativos, lo que favorecería la precisión en el entrenamiento constante del modelo de *machine learning* y facilitaría la emisión de notificaciones cuando se detecten niveles bajos de pigmentación en el cultivo. Las válvulas inteligentes, por su parte, permitirían controlar y distribuir la solución nutritiva entre las torres hidropónicas, mejorando la eficiencia del sistema. Además, los dosificadores contribuirían a una mezcla precisa de los componentes de la solución nutritiva y centralizarían la producción de la misma, reduciendo la necesidad de medidores adicionales de pH y partes por millón. En definitiva, el alcance de este proyecto va más allá de lo desarrollado en esta implementación, pero ciertamente abre las puertas a una nueva era de cultivos inteligentes, especialmente en espacios urbanos.

Conclusión

En resumen, las recomendaciones aquí presentadas abren diversas posibilidades para mejorar y expandir el alcance del proyecto. Incorporando estos elementos, se podrían optimizar los procesos actuales, aumentar la escalabilidad del sistema y reducir costos operativos. La integración de nuevos dispositivos, el uso de tecnologías en la nube y la adopción de imágenes de Docker pueden ser clave para llevar este proyecto a nuevas alturas, consolidando su aplicación en el campo de los cultivos inteligentes y la agricultura de precisión.

CAPÍTULO XV

Bibliografía

[1] D. M. Ponce, «Desarrollo de una plataforma basada en microservicios en la nube pública para la orquestación de dispositivos IoT y un gateway en la red local para el acceso de los dispositivos IoT a la plataforma,» Tesis de Licenciatura, Universidad del Valle de Guatemala, 18 Avenida 11-95 Guatemala, Interior UVG, Cdad. de Guatemala 01015, ene. de 2024.

[2] C. R. G. Monterroso, «Desarrollo de herramientas y aplicaciones para Internet de las cosas y aprendizaje automático utilizando la computadora Raspberry Pi,» Tesis de Licenciatura, Universidad del Valle de Guatemala, 18 Avenida 11-95 Guatemala, Interior UVG, Cdad. de Guatemala 01015, ene. de 2024.

[3] Cropin, *Internet of Things in Agriculture: What is IoT and how is it implemented in agriculture?* Accessed: February 17, 2024. dirección: `https://www.cropin.com/iot-in-agriculture`.

[4] C. A. R. Gómez, «Aplicación del Machine Learning en Agricultura de Precisión,» *Revista Cintex*, vol. 25, 2020.

[5] C. Loh, *DIY Hydroponics Vegetable Growing Tower [Part 3]*, Accessed: 24-05-2024, 2023. dirección: `https://www.youtube.com/watch?v=a7LJsXw-Q2g&t=916s`.

[6] S. Kumar, P. Tiwari y M. Zymbler, «Internet of Things is a revolutionary approach for future technology enhancement: a review,» *Journal of Big Data*, vol. 6, n.º 111, págs. 1-21, 2019.

[7] A. Banks y R. Gupta, *MQTT Version 3.1.1 Plus Errata 01*, OASIS Standard Incorporating Approved Errata 01, Chairs: Raphael J Cohn and Richard J Coppen. Technical Committee: OASIS Message Queuing Telemetry Transport (MQTT) TC, dic. de 2015. dirección: `http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/errata01/os/mqtt-v3.1.1-errata01-os-complete.pdf`.

[8] H. Team. «Introducing the MQTT Protocol – MQTT Essentials: Part 1.» 28 min read. (feb. de 2024), dirección: `https://www.hivemq.com/blog/mqtt-essentials-part-1-introducing-mqtt/`.

[9] IoTicos, *Desentrañando MQTT - ¿Cómo funciona?* Accessed: 24-05-2024, 2019. dirección: https://www.youtube.com/watch?v=Tb1t6GKJ0r0&ab_channel=IoTicos.

[10] A. Jr, *The Essential Guide to HTTP*, 9 min read, sep. de 2021. dirección: https://hackernoon.com/the-essential-guide-to-http.

[11] M. Explainers, «What is machine learning?,» mayo de 2024, McKinsey Experts: Michael Chui is a partner at the McKinsey Global Institute and is based in McKinsey's Bay Area office; Tamim Saleh is a senior partner in the London office, where Alex Sukharevsky is a senior partner; and Alex Singla is a senior partner in the Chicago office.

[12] IBM, *What is logistic regression?* ibm.com. dirección: https://www.ibm.com/topics/logistic-regression#:~:text=Logistic%20regression%20estimates%20the%20probability,data%20set%20of%20independent%20variables..

[13] A. C. Team, «Regression vs Classification in Machine Learning Explained!,» mayo de 2024. dirección: https://www.analyticsvidhya.com/blog/2023/05/regression-vs-classification/.

[14] G. for Geeks, *Decision Tree*, geekforgeeks.com, mayo de 2024. dirección: https://www.geeksforgeeks.org/decision-tree/.

[15] AWS, *Microservicios*, aws.amazon.com, 2023. dirección: https://aws.amazon.com/es/microservices/.

[16] Flatirons, *What is Next.js? An Overview in 2024*, flatirons.com, feb. de 2023. dirección: https://flatirons.com/blog/what-is-nextjs/.

[17] React, *React: The library for web and native user interfaces*, react.dev, 2024. dirección: https://react.dev/.

[18] Docker, *Overview of the Docker workshop*, docs.docker.com, 2024. dirección: https://docs.docker.com/get-started/workshop/.

[19] nabto, *ESP32 for IoT: A Complete Guide*, Accessed: February 22, 2024, 2024. dirección: https://www.nabto.com/guide-to-iot-esp-32/.

[20] E. Systems, *ESP32 Series Datasheet*, Accessed: July 10, 2024, 2024. dirección: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

[21] Pimoroni, *Raspberry Pi 4 Model B – 4GB RAM*, 2024. dirección: https://shop.pimoroni.com/products/raspberry-pi-4?variant=29157087445075.

[22] N. Inc., «NGINX Ingress Controller,» 2024, Accessed: October 28, 2024. dirección: https://hub.docker.com/r/nginx/nginx-ingress/.

CAPÍTULO XVI

Anexos

A. Código en la Raspberry Pi

```
import sys
import logging
import paho.mqtt.client as paho
from http.server import BaseHTTPRequestHandler, HTTPServer

class S(BaseHTTPRequestHandler):
    def _set_response(self):
        self.send_response(200)
        self.send_header('Content-type', 'application/json')
        self.end_headers()

    def do_POST(self):
        # extract information from the request
        content_length = int(self.headers['Content-Length'])
        post_data = self.rfile.read(content_length)
        logging.info("POST request,\nBody:\n%s",
                     post_data.decode('utf-8'))

        # publish information of the request to devices
        client.publish("devices/api-commands", post_data, 0)

        self._set_response()

def run(server_class=HTTPServer, handler_class=S, port=3000):
    # mqtt client initialization
    global client
    client = paho.Client()
    print("Trying to connect to MQTT broker")
    if client.connect("localhost", 1883, 60) != 0:
        print("Could not connect to MQTT Broker")
        sys.exit(-1)

    # http server initialization
    logging.basicConfig(level=logging.INFO)
    server_address = ('', port)
    httpd = server_class(server_address, handler_class)
    logging.info('Starting httpd...\n')

    # loop
    try:
        httpd.serve_forever()
    except KeyboardInterrupt:
        pass

    # ending execution
    client.disconnect()
    httpd.server_close()
    logging.info('Stopping httpd...\n')

if __name__ == '__main__':
    from sys import argv

    if len(argv) == 2:
```

```
    run(port=int(argv[1]))
else:
    run()
```

Cuadro 19: Código *bridge* de HTTP a MQTT