UNIVERSIDAD TÉCNICA DEL NORTE

FACULTAD DE INGENIERÍA EN CIENCIAS APLICADAS

CARRERA DE INGENIERÍA MECATRÓNICA

INFORME FINAL DEL TRABAJO DE INTEGRACIÓN CURRICULAR

TEMA:

APLICACIÓN PARA ADMINISTRACIÓN DE UN SISTEMA DE IOT DE RIEGO INTELIGENTE

Trabajo de grado previo a la obtención de título de Ingeniero en Mecatrónica.

AUTOR:

Bryan Rolando Dávalos Alcívar

DIRECTOR:

Ing. Carlos Xavier Rosero Chandi, PhD.

Ibarra, 2024

UNIVERSIDAD TÉCNICA DEL NORTE
BIBLIOTECA UNIVERSITARIA

AUTORIZACIÓN DE USO Y PUBLICACIÓN A FAVOR DE LA UNIVERSIDAD TÉCNICA DEL NORTE

1. IDENTIFICACIÓN DE LA OBRA

En cumplimiento del Art. 144 de la Ley de Educación Superior, hago la entrega del presente trabajo a la Universidad Técnica del Norte para que sea publicado en el Repositorio Digital Institucional, para lo cual pongo a disposición la siguiente información:

DATOS DE CONTACTO			
CÉDULA DE IDENTIDAD:	1722988191		
APELLIDOS Y NOMBRES:	Dávalos Alcívar Bryan Rolando		
DIRECCIÓN:	Avenida Abraham Calazón y Aníbal Villacis		
EMAIL:	brdavalosa@gmail.com		
TELÉFONO FIJO:	3752780	**TELÉFONO MÓVIL:**	0962768453

DATOS DE LA OBRA	
TÍTULO:	Aplicación para administración de un sistema de IoT de riego inteligente
AUTOR (ES):	Bryan Rolando Dávalos Alcívar
FECHA: DD/MM/AAAA	08/02/2024
SOLO PARA TRABAJOS DE GRADO	
PROGRAMA:	■ **PREGRADO** □ **POSGRADO**
TITULO POR EL QUE OPTA:	Ingeniero en Mecatrónica
ASESOR /DIRECTOR:	Ing. Carlos Rosero, PhD - Ing. Fernando Valencia, MSc.

2. CONSTANCIAS

El autor (es) manifiesta (n) que la obra objeto de la presente autorización es original y se la desarrolló, sin violar derechos de autor de terceros, por lo tanto, la obra es original y que es (son) el (los) titular (es) de los derechos patrimoniales, por lo que asume (n) la responsabilidad sobre el contenido de la misma y saldrá (n) en defensa de la Universidad en caso de reclamación por parte de terceros.

Ibarra, a los 08 días del mes de febrero de 2024

EL AUTOR:

.....

Nombre: Dávalos Alcívar Bryan Rolando

CERTIFICACIÓN DIRECTOR DEL TRABAJO DE INTEGRACIÓN CURRICULAR

Ibarra, 08 de Febrero de 2024

Ing. Carlos Xavier Rosero Chandi, PhD.

DIRECTORA DEL TRABAJO DE INTEGRACIÓN CURRICULAR

CERTIFICA:

Haber revisado el presente informe final del trabajo de titulación, el mismo que se ajusta a las normas vigentes de la Unidad Académica de la Universidad Técnica del Norte; en consecuencia, autorizo su presentación para los fines legales pertinentes.

(f) .....

Ing. Carlos Xavier Rosero Chandi, PhD.
C.C.:100251582-1

APROBACIÓN DEL COMITÉ CALIFICADOR

El Tribunal Examinador del trabajo de titulación "APLICACIÓN PARA ADMINISTRACIÓN DE UN SISTEMA DE IOT DE RIEGO INTELIGENTE." elaborado por Bryan Rolando Dávalos Alcívar, previo a la obtención del título del Ingeniero en Mecatrónica, aprueba el presente informe de investigación en nombre de la Universidad Técnica del Norte:

(f):.....

Ing. Carlos Xavier Rosero Chandi, PhD.

(f):.....

Ing. Fernando Vinicio Valencia Aguirre, MSc.

Dedicatoria

Quiero expresar mi más profundo agradecimiento a mis padres y mi familia. Su inquebrantable apoyo, confianza han sido la fuerza impulsora en este proceso. A pesar de las dificultades y los desafíos, su fe en mi capacidad y su generosidad desinteresada han sido una luz en el camino.

A mis padres les debo todo. Su sabiduría, paciencia y dedicación han sido mi mayor inspiración. Gracias por brindarme no solo recursos materiales, sino también el regalo invaluable de su tiempo y presencia. Agradezco sinceramente a toda mi familia extendida por su apoyo continuo.

Este logro no solo es mío, sino también de ustedes. Cada victoria es un reflejo del amor, dedicación y valores que me han inculcado. Gracias por creer en mí cuando dudaba, por inspirarme a superar obstáculos y por ser mi fuente constante de fortaleza.

Me disculpo pues mis palabras nunca serán suficientes para expresar mi gratitud.

Con amor y agradecimiento,

Bryan Rolando Dávalos Alcívar

Agradecimiento

Quiero expresar mi profundo agradecimiento a mi familia, quienes han sido mi soporte incondicional en este viaje académico. Su amor y apoyo constante han sido la fuerza impulsora que me ha llevado a alcanzar cada logro.

A todas las personas no importa si solo nos hayamos visto, cruzado palabras, compartido salones o haber sido amigos, todos han significado algo y conforman la experiencia que ha sido estudiar en esta Universidad, quiero agradecerles sinceramente.

En especial, quiero reconocer y agradecer al cuerpo docente de la carrera de mecatrónica. A mi tutor, Xavier Rosero, por compartir su profundo conocimiento y ser un modelo ejemplar de ingeniería. A David Ojeda, quien no solo ha sido un docente increíble, sino también un amigo invaluable en este camino.

A Gabriela Verdezoto, la mejor profesora de matemáticas de mí vida, por ser una inspiración de dedicación y talento. A Brizeida Gamez por sus valores y vocación. A Marco Ciaccia, por exigir lo mejor de nosotros y ser un ejemplo de excelencia académica. A Fernando Valencia, Cosme Mejía, Iván Iglesias y Víctor Erazo, gracias por compartir sus vastas experiencias y conocimientos en el mundo de la ingeniería.

Cada uno de ustedes ha contribuido de manera única a mi formación, y estoy agradecido por haber tenido la oportunidad de aprender con ustedes. Su impacto perdurará en mi carrera profesional y en mi vida.

Con gratitud,

Bryan Rolando Dávalos Alcívar

Índice general

Índice general **VII**

Índice de figuras **X**

Índice de tablas **XII**

Resumen **XIII**

Abstract **XIV**

Introducción **1**

1.1 Problema ... 1
1.2 Objetivos ... 2
1.2.1 Objetivo general ... 2
1.2.2 Objetivo específico ... 2
1.3 Alcance ... 2
1.4 Justificación ... 2

Revisión literaria **4**

2.1 Estado del arte ... 4
2.2 Sistema de riego ... 6
2.2.1 Riego inteligente ... 7
2.2.2 Variables de adquisición ... 8
2.3 Internet de las cosas ... 9
2.3.1 Arquitectura ... 10
2.3.2 Comunicación IoT ... 11
2.4 Cloud Computing ... 12
2.4.1 Clasificación de servicios de computación en la nube ... 13
2.4.2 Plataformas en la nube ... 14
2.5 Firebase ... 15
2.5.1 Realtime Database ... 15
2.5.2 Auth ... 16

2.5.3 Cloud Functions 16
2.5.4 Firebase Cloud Messaging 16
2.5.5 Hosting 17
2.5.6 Firebase Performance Monitoring 17
2.6 Microservicios 18
2.7 Base de datos 19
2.6.1 Relacional y no relacional 19
2.6.2 SQLite 19
2.9 Análisis 20

Marco Metodológico **21**

3.1 Metodología 21
3.2 Dinámica del sistema IoT 22
3.3 Sistema de riego 22
3.3.1 Requisitos del sistema de riego 22
3.3.2 Diseño del sistema de riego 23
3.4 Nube de Firebase 24
3.4.1 Realtime Database 25
3.4.2 Firebase Auth 27
3.4.3 Cloud Functions 28
3.4.4 Firebase Cloud Messaging 28
3.5 Aplicación para la administración del sistema de riego 29
3.5.1 Diagrama de flujo de la aplicación 30
3.6 Base de datos SQlite 31
3.6.1 Entidad y valores. 32

Resultados y análisis **33**

4.1 Implementación 33
4.1.1 Programa del microprocesador con Python 33
4.1.2 Aplicación web de usuario para la administración 37
4.1.3 Almacenamiento en la base de datos SQLite 56
4.2 Pruebas 59
4.2.1 Pruebas de integración del sistema 60
4.2.2 Pruebas de rendimiento del sistema 62
4.3 Discusión 68

Conclusiones, recomendaciones y trabajo a futuro **69**

5.1 Conclusiones 69
5.2 Recomendaciones 70
5.3 Trabajo Futuro 70

Bibliografía **71**

Anexos **77**

Anexo A. Programa del sistema de riego **78**

Anexo B. Programa de la base de datos SQLite **83**

Anexo C. Programa de la aplicación web en Astro **87**

Índice de figuras

2.1. Arquitectura orientada al servicio. 10
2.2. Arquitectura de Firebase Cloud Messaging. 17
2.3. Ejemplo de arquitectura de microservicios para servicio de taxis. 19

3.1. Modelo en cascada 21
3.2. Diagrama del sistema de riego IoT 22
3.3. Diagrama de flujo del programa en Python del sistema IoT. 24
3.4. Flujo del ciclo de riego automático de la aplicación. 24
3.5. Nodo raíz y nodos secundarios del proyecto en Realtime 25
3.6. Nodos finales del nodo data en Realtime. 26
3.7. Nodos finales de los nodos control, settings y users en Realtime. 26
3.8. Nodo con los datos de los usuarios 27
3.9. Procedimiento del desarrollo de funciones en Cloud Functions. 28
3.10. Microservicios de Firebase para la aplicación 30
3.11. Diagrama de flujo de la aplicación usuarios no autenticados 30
3.12. Diagrama de flujo aplicación para usuarios autenticados. 31
3.13. Diagrama de flujo base de datos SQLite 32

4.1. Respuesta de la terminal del sistema de Riego 36
4.2. Escritura del programa en le base de datos en tiempo real 36
4.3. Configuración del SDK en la consola de Firebase 37
4.4. Panel Autenticación en la consola de Firebase 39
4.5. Interfaz para inicio de sesión en la aplicación 40
4.6. Sesión de usuario iniciada en la aplicación 41
4.7. Verificación de correo del usuario de la aplicación 41
4.8. Restablecimiento de contraseña en la aplicación 42
4.9. Interfaz del emulador de Cloud Functions 47
4.10. Llamada a la función para escritura en Realtime Database frente a eventos 47
4.11. Llamada a la función de notificación en base a eventos en Realtime Database 48
4.12. Notificación de la aplicación en el sistema. 48

4.13. Par de claves obtenidas desde la consola de Firebase 49
4.14. Permisos de notificación y generación del token FCM en la aplicación..... 51
4.15. Nodo users en Realtime Database..... 51
4.16. Despliegue de la aplicación web en Firebase Hosting 54
4.17. Sección de Monitoreo de Variables de la aplicación web..... 55
4.18. Sección de acciones de la aplicación web..... 55
4.19. Sección de configuración y acciones de control de la aplicación. 56
4.20. Registro de datos del sistema de riego en la Entidad History en SQLite..... 57
4.21. Estructura de la base de datos de SQLite 58
4.22. Base de datos cargada en la nube de Firebase Storage. 59

Índice de tablas

2.1. Sistemas de Riego, características y aplicación. 6
2.2. Características de las Plataformas de desarrollo en la nube 15
3.1. Descripción de las acciones de Realtime Database. 26
3.2. Descripción de las configuraciones de Realtime Database. 26
4.1. Autenticación y administración de usuarios 60
4.2. Gestión de la información del sistema 61
4.3. Monitoreo y control del sistema de riego 61
4.4. Características del rendimiento y sus respectivas métricas según ISO/IEC 25010. 63
4.5. Ca racterísticas del computador de la prueba de rendimiento 63
4.6. Navegadores para pruebas de rendimiento 64
4.7. Tiempo de respuesta de la aplicación web 64
4.8. Métricas del seguimiento de carga de la página en Firebase Monitoring. 65
4.9. Tiempo de sincronización con la base de datos en tiempo real 66
4.10. Tiempo de funciones de Cloud Functions 66
4.11 Utilización de memoria de la aplicación web 66
4.12 Utilización de CPU de la aplicación web 67
4.13. Solicitudes en red de las bibliotecas en la aplicación 67

Resumen

La agricultura de precisión se refiere al uso de herramientas basadas en tecnologías informáticas, junto con sensores y dispositivos de Internet de las cosas. Su aplicación busca mejorar el rendimiento, la rentabilidad y la calidad de los cultivos, al tiempo que reduce el impacto ambiental. Los sistemas de riego inteligente son esenciales en la agricultura de precisión, ya que permiten la automatización y aumentan la eficiencia del riego. Este trabajo se centra en desarrollar una aplicación para administrar un sistema de riego inteligente basado en IoT, facilitando así el desarrollo e implementación de sistemas y aplicaciones IoT en el ámbito agrícola. La aplicación web creada consta de interfaces diseñadas para el monitoreo de variables ambientales y el control del sistema de riego. Esta aplicación web utiliza los servicios de Google Firebase. La arquitectura de hardware incluye sensores de variables ambientales y una bomba de agua conectados a un microcomputador, que envía datos a una base de datos en tiempo real en la nube. Además, se almacena información en una base de datos local para su análisis posterior, asegurando así la disponibilidad de datos. Se aborda la utilización de herramientas con potencial y gratuitas en el desarrollo de aplicaciones IoT. Los resultados muestran las capacidades y funcionalidades de la propuesta, mejorando la gestión del sistema de riego para los usuarios. La aplicación de nuevas tecnologías, como la computación en la nube, impulsa la integración de sistemas IoT en la agricultura, especialmente en sistemas de riego basados en IoT.

Palabras clave: Agricultura de precisión, Internet de las cosas (IoT), Riego inteligente, Google Firebase, IoT (Internet de las cosas), Aplicación web, Administración de sistema IoT.

Abstract

Precision agriculture refers to the use of computer technology-based tools, along with sensors and Internet of Things devices. Its application aims to improve crop performance, profitability, and quality while reducing environmental impact. Smart irrigation systems are crucial in precision agriculture, as they allow for automation and enhance irrigation efficiency. This work focuses on developing an application to manage a smart irrigation system based on IoT, thus facilitating the development and implementation of IoT systems and applications in agriculture. The created web application consists of interfaces designed for monitoring environmental variables and controlling the irrigation system. This web application uses Google Firebase services. The hardware architecture includes sensors for environmental variables and a water pump connected to a microcomputer, which sends data to a real-time cloud database. Additionally, information is stored in a local database for later analysis, ensuring data availability. The utilization of potential and free tools in IoT application development is addressed. The results showcase the capabilities and functionalities of the proposal, enhancing irrigation system management for users. The application of new technologies, such as cloud computing, drives the integration of IoT systems in agriculture, especially in IoT-based irrigation systems.

Keywords: Precision agriculture, Internet of Things (IoT), Smart irrigation, Google Firebase, IoT (Internet of Things), Web application, IoT system management.

Capítulo I

Introducción

La agricultura de precisión consiste en la aplicación de herramientas basadas en tecnologías informáticas actuales en conjunto con sensores y herramientas de internet de las cosas (IoT) [1]. Su aporte radica en mejorar el rendimiento, la rentabilidad y la calidad de los cultivos, con una menor afectación al medio ambiente. Por la cual el Internet de las cosas está captando atención como un área de investigación y desarrollo para ingenieros y científicos. Su aplicación para la industria respecta a la automatización y la mejora de la eficiencia sin requerir intervención humana, sumado, su flexibilidad para abarcar una amplia variedad de dispositivos lo posiciona como la próxima revolución en la industria.

El Internet de las cosas es una herramienta importante tanto para estudiantes como para docentes, que contribuye a abordar desafíos educativos y situaciones del entorno. Además, promueve la implicación de los estudiantes en la resolución de problemas prácticos de ingeniería.

1.1 Problema

Cada vez más dispositivos se conectan a internet y proporcionan datos de importancia; acceder a grandes cantidades de datos es complicado con las técnicas convencionales. El avance tecnológico y situaciones como la pandemia demandan que los dispositivos se configuren a distancia y faciliten el trabajo [2].

Las nuevas tendencias de ingeniería apoyadas sobre computación distribuida se enfocan en sistemas de riego semiautomáticos, automáticos e inteligentes, tanto para aplicaciones urbanas como para rurales. Todas ellas tienen como factor común el uso de sensores de variables ambientales y actuadores de apertura/cierre, conectados a un sistema microprocesado que a la vez se enlaza con un servidor en la nube [3].

Bajo la arquitectura descrita, el servidor en la nube es parte de los servicios proporcionados por plataformas IoT gratuitas y pagadas. Dentro de este contexto se puede adquirir información a través de bases de datos de tiempo real, por ejemplo, Firebase [4]. Sin embargo, el monitoreo de las variables inmersas en el control de riego, el control de la válvula de riego, la puesta en

marcha/apagado del sistema y la administración de la información desde computadores remotos, integrados en una aplicación se vuelven una cuestión a resolver.

1.2 Objetivos

1.2.1 Objetivo general

Desarrollar una interfaz de programación de aplicaciones para administración de un sistema IoT de riego inteligente.

1.2.2 Objetivo específico

- Proponer un algoritmo para la administración de un sistema IOT de riego inteligente.
- Implementar el algoritmo propuesto en una aplicación.
- Validar el funcionamiento de la aplicación.

1.3 Alcance

La interfaz se conecta con los servicios de Google Firebase, puede realizar funciones como pedir, borrar información, crear información, establecer fecha, límites de la obtención de datos y gestionar otras funciones en la web como encender, apagar y monitorear el tiempo de funcionamiento del sistema.

Los programas instalados en los dispositivos habilitarán la sincronización del hardware con la plataforma en la nube de Firebase. Los controladores facilitarán la transmisión y recepción de parámetros, y cuando se cumplan determinadas condiciones, se ejecutarán diferentes acciones.

1.4 Justificación

El IoT se ha desarrollado rápidamente y se ha propuesto una gran cantidad de tecnologías habilitadoras. Cada cosa disponible se está volviendo inteligente. Existe un amplio margen para la investigación. El futuro de IoT es muy brillante, desde facturas hasta vehículos, todo estaría conectado brindando un mejor estilo de vida [5].

La automatización permite mejorar la eficiencia y calidad en los cultivos agrícolas, además facilita la administración del proceso de producción. En una actividad como el riego, se requiere bastante mano de obra, que no conoce precisamente cuando el cultivo necesita agua esto genera costos adicionales en la producción. En una investigación realizada en Ecuador sobre sistemas de

riego automatizados en jardines se alcanza la viabilidad de la implementación de estos, logrando el ahorro de agua y presupuesto por contratación de personal [6].

Uno de los conceptos que han venido tomando fuerza en los últimos años en el mundo y en el país, es el concepto de cultivos o agricultura urbana, la cual responde a las necesidades alimentarias actuales y a varias problemáticas como la explotación y deterioro de suelos, sequías, contaminación y la salubridad de los alimentos.

En la Universidad Técnica del Norte, se han llevado a cabo numerosos proyectos de IoT por parte de estudiantes y profesores, y muchos de estos proyectos están operativos de manera independiente en diversos lugares. Sin embargo, para lograr una mayor integración y eficiencia, se deben abordar varias problemáticas. Estas incluyen la creación de la infraestructura y la elección de las tecnologías de comunicación en red para hacerlo en los sistemas IoT existentes, pero también adoptar una plataforma de computación en la nube, como la plataforma de Firebase, crear algoritmos destinados a operar con el backend en la nube y emplear los servicios en el desarrollo de la aplicación como es el caso de la base de datos en tiempo real.

Capítulo II

Revisión literaria

2.1 Estado del arte

Se analiza la información actualizada y relevante para comprender y abordar el problema. El estado del arte proporciona una visión general en el campo de la administración de sistemas IoT, destacando proyectos de investigación en relación con la aplicación de interfaces de programación en el manejo de la información de proyectos de IoT y problemas relacionados realizados con los servicios en la nube de la plataforma Firebase.

En el 2017 se expuso el trabajo “Real-Time Communication Network using Firebase Cloud IoT Platform for ECMO Simulation” [7]. Describe la implementación de tres unidades de simulación que trabajan simultáneamente para recrear condiciones ECMO. Estas unidades se comunican en tiempo real y se controlan a través de una aplicación móvil. Firebase se utiliza como plataforma de computación en la nube centralizada, que garantiza fiabilidad y comunicación ininterrumpida. Firebase también ofrece una API para facilitar la compatibilidad y proporciona herramientas para el desarrollo de la aplicación, incluyendo una base de datos en tiempo real que permite una sincronización rápida de datos con baja latencia. La tecnología de comunicación seleccionada es WiFi para lograr robustez y compatibilidad.

En una conferencia del 2018 se presentó el trabajo “JustIoT Internet of Things based on the Firebase Real-time Database” [8] . La aplicación recopila datos de diversos controladores, los usuarios pueden establecer las reglas de control, llevar a cabo monitoreo y control remotos. La página web de administración, se conecta a una base de datos en tiempo real proporcionada por Firebase. Esta base de datos permite el acceso y la modificación directa de datos desde dispositivos front-end, que pueden ser aplicaciones web, móviles o controladores. Además, se utiliza un servidor MQTT para la inteligencia del sistema, que es compatible con controladores de recursos limitados, como el controlador Arduino.

En el año 2018 se presentó el trabajo de investigación “Android based Home Security Systems using Internet of Things and Firebase” [9]. En este artículo se aborda la propuesta de un sistema de seguridad IoT para viviendas, la propuesta se basa en una aplicación móvil para dispositivos

Android y el empleo de un microcontrolador Esp8266. El sistema establece una conexión bidireccional con un microprocesador para el intercambio de datos con Firebase Database. La otra parte del sistema es la aplicación móvil, que se conecta a Firebase y ofrece funcionalidades de alertas y control remoto, entre otras. El microprocesador envía los datos a la base de datos, cuando se ingresa el dato en la tabla de notificaciones, se recupera la información. Firebase se encarga de enviar esta información al dispositivo correspondiente después de una verificación adecuada. En cuanto al hardware, se utilizaron sensores PIR (sensor pasivo infrarrojo) y sensor de detección de fuego.

En el artículo "Implementación of Industrial Automatization System using Raspberry pi by IoT with Firebase" [10]. El artículo resalta el diseño e implementación de Sistemas de Automatización Industrial con la ayuda de Raspberry Pi como una puerta enlace programada en lenguaje Python, además de IoT y Firebase para la computación en la nube. El sistema consta de un módulo de sensor que es la Raspberry Pi, utilizada para monitorear y controlar diversos parámetros de una planta industrial y la gestión de energía. El módulo coordinador se implementa también utilizando la Raspberry Pi y se conecta al módulo de sensor. Ambos módulos de Raspberry Pi son programados en Python y utilizan la base de datos de Firebase para el almacenamiento en la nube.

En el artículo "POLYCRYSTALLINE SOLAR PANEL MONITORING SYSTEM DESIGN USING IoT-BASED FIREBASE WEB TECHNOLOGY" [11] se propone un sistema de monitoreo del voltaje y corriente de paneles solares, se utilza un micontrolador NodeMCU basado en ESP8266, conectado a internet. La información es monitoreada a través de la aplicación web de Firebase. Un controlador de carga solar sirve para mantener el voltaje entrante de los paneles solares de acuerdo con la distancia de funcionamiento de la batería. Los datos de medición mostrados tienen un tiempo de actualización de 1 segundo. Esta información se puede utilizar como referencias adicionales y análisis.

"IoT based Weather Monitoring System Using Firebase Real Time Database with Mobile Application" [12]. Este artículo introduce un sistema de monitoreo meteorológico de bajo costo que utiliza el microcontrolador Arduino MKR WiFi 1010, optimizado para aplicaciones de IoT con un enfoque en la eficiencia energética y la economía. Este sistema emplea varios sensores, incluyendo el DHT11 para medir temperatura y humedad, el BMP180 para la presión barométrica, un módulo de detección de lluvia y el sensor UV ML8511. Los datos recopilados por estos sensores se almacenan en la base de datos de tiempo real de Firebase. Además, se crea una aplicación móvil con MIT App Inventor 2 para facilitar el acceso a los informes climáticos desde dispositivos móviles. Este enfoque ofrece una solución rentable y eficaz para el monitoreo climático.

2.2 Sistema de riego

Los sistemas de riego convencionales dependen de un operador para suministrar agua a los cultivos. Este enfoque presenta desafíos, ya que determinar cuándo el cultivo requiere agua y realizar el riego en el momento adecuado puede ser problemático. Además, implica una considerable cantidad de mano de obra para su operación. En contraste, los sistemas de riego automatizados implementan estrategias que permiten el ahorro de agua y reducen la necesidad de trabajo manual [13].

El sistema de riego engloba una serie de componentes que dirigen el agua hacia el cultivo. En este sistema, se presentan varias opciones, comenzando con la fuente de agua, que puede ser de flujo constante o provenir de un depósito de agua. El desplazamiento del agua puede ser resultado de una bomba o de la gravedad [14]. Por último, el método de distribución del agua puede clasificarse de diversas maneras.

A continuación, una tabla explicativa de diferentes tipos de sistemas de riego, sus características y aplicaciones [14]:

Tabla 2.1

Sistemas de Riego, características y su aplicación.

Tipo de Riego	Descripción	Aplicaciones
Riego Localizado	Entrega agua directamente a las raíces de las plantas.	Jardinería residencial, agricultura de precisión, cultivos perennes.
Riego por Goteo	Sistema de tuberías con pequeños orificios, o tubos porosos para liberar agua directamente sobre el suelo.	Huertos, jardines, cultivos en hileras, invernaderos.
Riego por Aspersión	El agua se distribuye en forma de pequeñas gotas a través de boquillas o aspersores.	Campos de cultivo, parques, campos deportivos.
Riego de superficie	El agua se distribuye sobre la superficie del suelo y se deja fluir a través de gravedad.	Campos de cultivo, pastizales.
Riego Subterráneo	El agua se suministra en la profundidad de la capa freática a bajo caudal.	Cultivos en zonas con limitaciones de agua, reducción de evaporación.
Riego por Inundación	Se inunda el suelo con agua, creando un nivel de agua superficial.	Arrozales, ciertos cultivos acuáticos, campos con pendientes suaves.

La elección del tipo de riego depende de diversos factores, tales como el tipo de cultivo, la disponibilidad de agua, las condiciones climáticas, la naturaleza del terreno y la eficiencia deseada. Aunque los sistemas de riego localizado y por goteo son más eficientes, los métodos de superficie, como el riego por aspersión, son más simples y menos costosos, lo que los hace adecuados para áreas más extensas, siendo comúnmente utilizados en la agricultura [14].

2.2.1 Riego inteligente

En un sistema de riego inteligente, se implementan tecnologías como sensores y algoritmos para optimizar y controlar el riego. Al ser inteligente, se integra la conexión a internet para permitir la comunicación con otros sistemas y facilitar la administración remota y autónoma.

2.2.1.1 Aplicaciones de riego inteligente

- Sistemas de riego inteligente basados en la aplicación de software especializado para analizar el estado de la planta en tiempo real. Un ejemplo de estos sistemas es aquel que, mediante la captura de imágenes, utiliza un software de riego para analizar la condición del cultivo. Este software toma decisiones de manera autónoma y empírica, identificando las zonas donde el riego es más necesario [15].
- Sistemas de riego por aproximación agro-meteorológica utiliza condiciones climáticas obtenidas mediante sensores o datos meteorológicos, como la evaporación de la superficie de la planta. Esto permite determinar cuándo y cuánta agua se debe suministrar. En la práctica, se emplean estaciones meteorológicas que miden variables ambientales y calculan la evaporación potencial utilizando la ecuación de Penman-Monteith [16].
- Sistemas de riego con enfoque en el contenido de agua o en las mediciones de la tensión del agua en el suelo. Se emplea la reflectometría de dominio temporal (Time Domain Reflectometry, TDR) para determinar de manera absoluta el contenido de agua en el suelo. A diferencia de las sondas de suelo, las cuales no miden directamente el contenido de agua y requieren calibración. Es importante tener en cuenta que las sondas de suelo pueden estar influenciadas por la temperatura y salinidad del suelo, y solo miden un área limitada de volumen [16].
- Los sistemas de riego con control remoto posibilitan a los usuarios administrar el sistema de riego a través de la red. Los usuarios tienen la capacidad de supervisar el estado del sistema de riego mediante sensores y ajustar los parámetros de riego. Este tipo de sistema puede funcionar de manera autónoma, pero brinda la flexibilidad de permitir la intervención según el criterio del usuario [17].

2.2.1.2 Dispositivos IoT

Los sistemas de riego inteligente se componen de componentes electrónicos, la cantidad de elementos y sus características dependen de los criterios de monitoreo y riego, tales como las variables a medir, la precisión de los sensores y la programación del riego (cantidad, tipo y tiempo), pueden ajustarse según las necesidades específicas del proyecto o del cultivo.

Algunos componentes clave de estos sistemas incluyen sensores de humedad del suelo, dispositivos meteorológicos para monitorear las condiciones climáticas, actuadores y válvulas de riego

responsables de dirigir el flujo de agua a las áreas correspondientes. Además, se incorporan elementos de conectividad, como el acceso a internet, para obtener datos meteorológicos y gestionar el sistema de riego de manera eficiente.

- **Red IoT:** La red del Internet de las Cosas conecta los sensores, actuadores y otros dispositivos a través de Internet. Los datos recopilados por los sensores se envían a través de esta red para su procesamiento y análisis en centros de datos o plataformas en la nube. Los comandos para los actuadores también se transmiten a través de esta red. Un ejemplo de ello son los sistemas de riego inteligente que están conectados a Internet para acceder a datos meteorológicos en tiempo real y permitir el control remoto.

- **Sensores:** Los sensores son dispositivos utilizados para captar datos del entorno. En el caso de los sistemas de riego, los sensores miden parámetros como la humedad del suelo, humedad del aire, la temperatura, la radiación solar y otros factores que sean de interés para el proyecto. Sensores de Humedad del Suelo –Buscar especifico-

- **Actuadores:** dispositivos que realizan acciones físicas en respuesta a los datos recopilados por los sensores o a comandos recibidos de un sistema de control. Válvulas de Riego son actuadores que controlan el flujo de agua hacia las áreas de riego. Cuando un sensor indica que el suelo está seco, la válvula se activa para proporcionar agua a las plantas.

- **Procesadores y microprocesadores:** Estos dispositivos centralizan la gestión del sistema de riego. Pueden recibir datos de sensores para su procesamiento y análisis, son los encargados de los controlar el riego y la cantidad de agua aplicada. Conecta los sensores, actuadores y otros dispositivos físicamente dependiendo de la configuración del sistema IoT. Los comandos para los actuadores también se pueden enviar a través de este dispositivo. Rasperry y Arduino son placas con comunes en el desarrollo de proyectos IoT, debido a su bajo costo y tamaño reducido [17].

2.2.2 Variables de adquisición

Para el sistema IoT de riego, se establecieron como variables ambientales fundamentales la temperatura ambiente, la humedad del suelo y la humedad del aire. Estos parámetros fueron seleccionados no solo por su relevancia en el proceso de cultivo, sino también por su factibilidad, ya que existen diversos dispositivos en el mercado que permiten medirlos a un costo relativamente asequible. Además, estos parámetros son fácilmente integrables con un sistema microprocesado. Se excluyeron variables como la velocidad del viento o la radiación solar, ya que estas se

encuentran más allá del enfoque de un cultivo urbano que puede estar resguardado de la exposición directa al sol y al viento.

2.2.2.1 Temperatura del aire

La temperatura del aire tiene un impacto directo en procesos clave para los cultivos, como la germinación, el crecimiento, la floración y la maduración. Regula las reacciones bioquímicas y fisiológicas de las plantas. Además, influye en la evaporación del agua en el suelo, disminuyendo la humedad rápidamente durante condiciones de altas temperaturas. Asimismo, aumenta la transpiración en las plantas, un factor relevante a tener en cuenta para la planificación del riego.

2.2.2.2 Humedad del suelo

La humedad es un fenómeno natural que se manifiesta a nivel molecular y está estrechamente vinculado con la cantidad de moléculas de agua presentes en una sustancia específica, ya sea en estado sólido o gaseoso [18].

El agua es imprescindible para las plantas afecta directamente a su crecimiento y desarrollo. Un nivel adecuado de humedad permite que las plantas realicen la fotosíntesis de manera eficiente y evita que pierdan demasiada agua a través de la transpiración.

2.2.2.3 Humedad relativa

La humedad relativa (HR) se define como la relación entre la cantidad de vapor de agua presente en un metro cúbico de aire, en condiciones específicas de temperatura y presión, y la cantidad que tendría si estuviera saturado a esa misma temperatura y presión. Este parámetro se emplea con regularidad para describir la humedad atmosférica [19].

La humedad relativa también influye en la prevención de enfermedades y en la gestión del riego, lo que es esencial para mantener la calidad y la cantidad de la producción agrícola.

Estrés hídrico en las plantas se refiere a la condición en la que hay una disponibilidad limitada de agua, lo que puede ser causado por sequías, riego insuficiente o condiciones de alta salinidad en el suelo. Este estrés afecta a las funciones vitales de las plantas, como el crecimiento, la floración y la fructificación, y puede llevar a una reducción significativa en la producción y la calidad de las cosechas. En casos extremos, el estrés hídrico puede provocar la muerte de las plantas. Por lo tanto, es importante gestionar adecuadamente la humedad y el riego para asegurar la salud y la productividad de los cultivos [20].

2.3 Internet de las cosas

La red de objetos físicos conectados a través de Internet que logran interactuar a través de sistemas embebidos, redes de comunicación, mecanismos informáticos redundantes y

aplicaciones generales en la nube [21]. Interconexión en redes públicas o privadas de todos los objetos cotidianos que a están equipados con algún tipo de inteligencia, es decir que tienen la capacidad de interactuar con otros, generar datos y realizar acciones con un mínimo de autonomía [22].

El IoT no se puede definir con una estructura única, es un concepto tan vasto del que se pueden seguir diferentes enfoques. Siguiendo esta premisa, el IoT para efectivamente debe incorporar una variedad de sensores, redes (también conocidas como puertas de enlace o Gateway), comunicaciones y tecnologías de computación entre otras [23] [24].

2.3.1 Arquitectura

La arquitectura genérica se categoriza el IoT en tres niveles. El inicial comprende los dispositivos finales, que están inmersos en el entorno físico y se encargan de la recolección de datos. El segundo son las tecnologías que permite la comunicación con la red u otros dispositivos. Por último, se encuentra la nube, las aplicaciones o servicios que hacen uso del sistema IoT. Esta es general que se ajusta a los requerimientos básicos sin embargo existen diferentes estructuras de acuerdo con los requerimientos de cada industria.

En la arquitectura de la figura 2.1 se muestra un ejemplo de arquitectura orientada al servicio. La configuración de un sistema IoT puede variar considerablemente en función de los requerimientos específicos y el propósito del estudio. La elaboración de la arquitectura considera una serie de elementos, como lo son: redes, comunicación, modelos empresariales y procesos, así como aspectos de seguridad. Al estructurar la arquitectura de un sistema IoT, es fundamental asegurar el alcance, la escalabilidad, la capacidad de los dispositivos de interactuar y comunicarse sin errores[5].

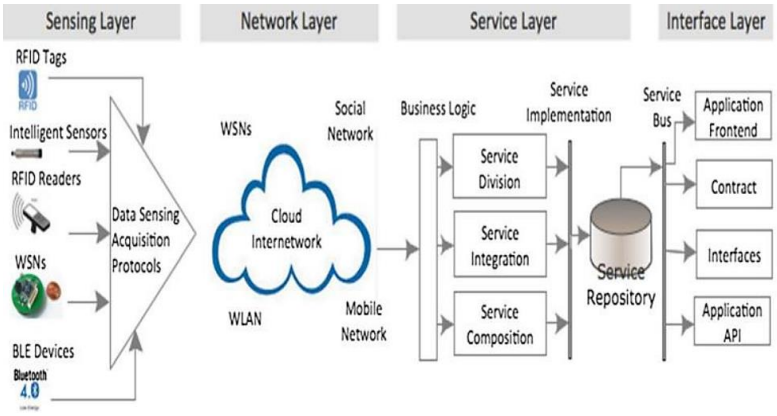


Figura 2.1: Arquitectura orientada al servicio. [5]

2.3.1.1 Middleware

El middleware se asemeja a un traductor oculto, el cual se encuentra entre las aplicaciones y el sistema operativo. Permite la comunicación y administración de datos en aplicaciones o componentes de aplicaciones distribuidas en una red. Existen middleware de diferentes tipos, proporcionan servicio de comunicación a través de marcos de mensajería como JSON, REST, XML, SOAP o servicios web. El middleware también puede permitir la comunicación de diferentes lenguajes de programación como Java, C++, PHP y Python [25].

2.3.2 Comunicación IoT

2.3.2.1 Red de área local inalámbrica (WLAN)

Las redes de área local inalámbricas son las redes definidas por su alcance como su nombre lo indica para la conexión de áreas dentro de un entorno local, como edificios, instituciones gubernamentales, centros educativos, hospitales, entre otros. No llegando a ocupar extensiones más amplias como las redes de áreas metropolitanas (MAN).

Existen varias tecnologías para formar redes WLAN, como Wifi, Bluetooth, ZigBee, Z-Wave, WiMAX y Li-Fi. Estas tecnologías tienen diferentes características, como el alcance, la velocidad, la frecuencia y el consumo de energía. Sin embargo, Wifi es la más común en redes domesticas o de oficina para la conexión de dispositivos a internet.

2.3.2.2 Estándar 802.11 (Wi-Fi)

Wi-Fi es el nombre comercial de una tecnología inalámbrica que utiliza los protocolos 802.11 para crear redes de área local inalámbricas (WLAN).

IEEE 802.11 es un conjunto de protocolos de comunicación que definen los estándares para las redes inalámbricas de área local (WLAN). Fue publicado en 1997 y desde entonces ha tenido varias variantes que mejoran la velocidad, la seguridad y la compatibilidad de los productos inalámbricos. Algunas de las variantes más conocidas son 802.11a, 802.11b, 802.11g y 802.11n. Estos protocolos operan en las bandas de frecuencia ISM de 2,4 GHz y 5 GHz y utilizan diferentes técnicas de modulación como DSSS, FHSS y OFDM1 [26].

2.3.2.3 HTTP

El Protocolo de Transferencia de Hipertexto (HTTP) es un protocolo de la capa de aplicación para sistemas de información hipermedia distribuidos y colaborativos. Es la base de la comunicación de datos para la World Wide Web, donde los documentos de hipertexto incluyen hipervínculos a otros recursos que el usuario puede acceder fácilmente.

HTTP funciona como un protocolo de solicitud-respuesta en el modelo cliente-servidor de computación. Un cliente web, como un navegador, envía una solicitud a un servidor como es el caso de Firebase. Luego, el servidor procesa la solicitud y devuelve una respuesta al cliente. La respuesta contiene un código de estado que indica el resultado de la solicitud, junto con cualquier dato solicitado.

HTTP ha evolucionado a lo largo del tiempo con varias versiones con mejoras en eficiencia y seguridad. HTTP también tiene una variante segura llamada HTTPS que se utiliza para conexiones seguras y cifradas.

2.4 Cloud Computing

En las últimas décadas, el aumento de los procesos de tercerización, deslocalización e internacionalización de empresas, junto con el auge de tecnologías de información y procesamiento de datos, ha generado una creciente demanda de capacidades de cómputo. Además, en el caso del desarrollo de proyectos IoT, como es el caso de este trabajo, gestionar hardware propio sería un costo mayor e innecesario. Para abordar estas necesidades, ha surgido una evolución significativa en las arquitecturas de cálculo, centrada principalmente en la ejecución simultánea de procesos en varios equipos informáticos [27].

Según el Instituto Nacional de Estándares y Tecnología de EE. UU, cloud computing es un modelo que facilita el acceso a la red bajo demanda y a un conjunto compartido de recursos informáticos configurables [28] . Estos recursos, que incluyen redes, servidores, almacenamiento, aplicaciones y servicios, pueden ser provisionados y liberados rápidamente con un esfuerzo mínimo de gestión o interacción con el proveedor de servicios. Esta definición es ampliamente aceptada tanto en la academia como en la industria [29].

La computación en la nube se puede describir de manera simple como la capacidad para acceder a archivos, datos, programas y servicios de terceros a través de Internet, utilizando un navegador web. Estos recursos están alojados por un proveedor externo y se distinguen por el modelo de pago, que implica abonar únicamente por los recursos informáticos y servicios que se emplean [30].

Los servicios informáticos son configurables, es decir, se ofrecen bajo demanda, es decir el servicio y costo dependen del consumo. También son escalable a través de Internet permitiendo a las organizaciones crecer junto con el servicio de manera más rápida y eficiente sin la necesidad de adquirir y mantener sus propios centros de datos físicos y servidores.

2.4.1 Clasificación de servicios de computación en la nube

Existen varios tipos de servicios de computación en la nube, cada uno de los cuales ofrece diferentes niveles de control, flexibilidad y gestión.

2.4.1.1 Infraestructura como Servicio (IaaS)

Los consumidores de esta familia de servicios utilizan directamente las infraestructuras informáticas, como capacidad de cómputo, espacio de disco y bases de datos, como un servicio proporcionado en la nube (IaaS). En lugar de adquirir directamente servidores o recursos de centro de datos, los clientes optan por externalizar estos servicios [27]. Un ejemplo de IaaS es Amazon Web Services (AWS) ofrece instancias de máquinas virtuales (VM) y almacenamiento escalable para IoT [31]. Los usuarios tienen el mayor nivel de flexibilidad y control sobre sus recursos con IaaS. Las facturas se calculan según la cantidad de recursos consumidos por el cliente, siguiendo el modelo de pago por uso.

2.4.1.2 Plataforma como Servicio (PaaS)

PaaS es un servicio que proporciona un entorno directo para desarrollar, probar e implementar aplicaciones de software. A diferencia de SaaS, PaaS ofrece una plataforma de desarrollo que aloja aplicaciones en la nube, ya sean completas o en progreso. Ejemplos de PaaS incluyen Firebase de Google y Azure IoT Hub de Microsoft, que permiten conectar, administrar y escalar miles de millones de dispositivos IoT [31].

2.4.1.3 Software como Servicio (SaaS)

SaaS proporciona a los usuarios aplicaciones como servicio, es decir, un producto completo que es ejecutado y gestionado por el proveedor del servicio. El proveedor ofrece licencias de su aplicación a los clientes como un servicio “bajo demanda”. Los proveedores de SaaS pueden tener instalada la aplicación en sus propios servidores web, y sus clientes no tienen control sobre la infraestructura de la nube. Ejemplos de SaaS incluyen Salesforce.com, Google Mail y Google Docs [31].

2.4.1.4 Almacenamiento de datos como servicio (DaaS)

La entrega de almacenamiento virtualizado bajo demanda se convierte en un servicio de nube separado, podría considerarse como un tipo especial de IaaS [31]. Además, existe un cuarto modelo llamado computación sin servidor que permite a los desarrolladores escribir y desplegar código sin preocuparse por la infraestructura subyacente. En este modelo, la infraestructura se ajusta automáticamente para satisfacer las necesidades de tráfico y capacidad.

2.4.2 Plataformas en la nube

Como ya se ha mencionado, ofrecen una plataforma completa para desarrollar y ejecutar aplicaciones, incluyendo la infraestructura necesaria. Las plataformas en la nube más conocidas, como Google Firebase, AWS IoT Core y Microsoft Azure IoT, ofrecen una variedad de servicios para almacenar y gestionar los datos recogidos por los sensores en un sistema IoT. Además, permiten alojar aplicaciones para la gestión y facilitan la conexión, administración y escalabilidad.

2.4.2.1 Google Firebase

Firebase, desarrollado por Google, es una plataforma integral para el desarrollo de aplicaciones móviles y web. Ofrece una amplia gama de herramientas y servicios diseñados para ayudar a los desarrolladores a construir, mejorar y hacer crecer sus aplicaciones de manera eficiente. En el contexto del Internet de las cosas (IoT), Firebase se utiliza para almacenar y gestionar datos recopilados por sensores. Sus capacidades incluyen el almacenamiento en tiempo real, lo que permite que los datos se actualicen instantáneamente en la base de datos de Firebase, proporcionando actualizaciones en tiempo real a todos los clientes conectados. Además, Firebase ofrece diversas características de seguridad para proteger los datos.

2.4.2.2 AWS IoT Core

AWS IoT Core, de Amazon Web Services (AWS), es un servicio que facilita la interacción entre dispositivos conectados y aplicaciones en la nube, así como con otros dispositivos. Diseñado para soportar cargas de trabajo a gran escala, AWS IoT Core puede manejar miles de millones de dispositivos y mensajes, procesando y enrutando esos mensajes a otros dispositivos y servicios de AWS. También proporciona servicios para garantizar una interacción segura con dispositivos conectados, gestionar el estado del dispositivo y leer y escribir datos del dispositivo.

2.4.2.3 Microsoft Azure IoT

Azure IoT, gestionados por Microsoft, es una colección de servicios en la nube junto con componentes de borde y SDKs, permite conectar, monitorear y controlar activos de IoT a gran escala. En resumen, una solución de IoT consiste en dispositivos IoT que se comunican con servicios en la nube. Azure IoT Hub es un servicio central que facilita la comunicación bidireccional entre dispositivos IoT y el servicio en la nube.

Cada plataforma ofrece sus propias ventajas y puede ser más adecuada para determinados casos de uso según las necesidades específicas del sistema.

Tabla 2.2

Características de las Plataformas de desarrollo en la nube

Características	Google Firebase	AWS IoT Core	Microsoft Azure IoT
Desarrollo de Aplicaciones	Servicios completos para desarrollo móvil y web, con integración completa de servicios de Google.	Servicios y flexibilidad para el desarrollo de aplicaciones IoT en el ecosistema de AWS.	SDKs enfocados en la conexión, monitoreo y control de elementos IoT en el ecosistema de Microsoft Azure.
Almacenamiento de Datos	Almacenamiento en tiempo real para actualizaciones instantáneas.	Capacidad para manejar grandes volúmenes de datos y mensajes.	Centro de operaciones para comunicación entre dispositivos IoT y el servicio en la nube
Facilidad de Uso	Documentación completa, adecuada para a diversos niveles de experiencia de desarrollo.	Documentación amplia y soporte técnico, con herramientas que facilitan el desarrollo.	Interfaz intuitiva con servicios gestionados, facilitando la implementación y gestión en el ecosistema de Microsoft Azure.
Costos	Permite uso gratuito bajo ciertos límites y pago por uso.	Modelo de precios basado en el consumo y la escala.	Diversas opciones de precios según el uso y las necesidades específicas en el ecosistema de Microsoft Azure.

2.5 Firebase

Firebase es una plataforma de desarrollo de aplicaciones móviles y web de Google que proporciona una variedad de herramientas y servicios para ayudar a los desarrolladores a construir, mejorar y hacer crecer sus aplicaciones.

2.5.1 Realtime Database

Firebase Realtime Database es una base de datos NoSQL en la nube que posibilita el almacenamiento y sincronización de datos en tiempo real entre usuarios. Su estructura de almacenamiento se basa en árboles JSON y es compatible con múltiples plataformas. Facilita la comunicación entre dispositivos de manera menos compleja, especialmente a través de la comunicación inalámbrica con WiFi mediante Firebase. Esta opción ofrece una mayor simplicidad en el desarrollo de código, permitiendo el uso de funciones del SDK de Firebase admin de Python o del SDK de Javascript [32].

Además de simplificar la comunicación entre dispositivos, Realtime Database permite establecer reglas personalizadas de seguridad para la lectura y escritura en la base de datos. Al combinarlo con Firebase Authentication, se obtiene un conjunto integral de herramientas para gestionar la seguridad de la aplicación, posibilitando la identificación de usuarios, la concesión de permisos personalizados y la validación de las entradas de información.

En Realtime Database, los datos se almacenan en un árbol JSON, donde cada nodo del árbol puede tener uno o varios nodos hijos. Los datos almacenados en los nodos de la base de datos pueden ser de cualquier tipo de datos JSON, como un objeto, una matriz, un número, una cadena o un booleano [33].

Estructura de almacenamiento basada en árboles JSON:

1. **Nodo raíz:** Representa la base de datos completa y se accede mediante el URL de la base de datos en tiempo real.
2. **Nodos secundarios:** Hijos del nodo raíz o de otros nodos secundarios. Representan subdivisiones o categorías dentro de la base de datos.
3. **Nodos finales:** No tienen hijos y contienen los datos reales de la base de datos. Se dividen en dos partes: la "Key" (nombre o clave del nodo) y su valor o propiedad "Value".

2.5.2 Auth

Firebase Authentication proporciona servicios backend, SDKs fáciles de usar y bibliotecas de interfaz de usuario listas para usar para autenticar a los usuarios en su aplicación. Soporta autenticación utilizando contraseñas, números de teléfono, proveedores de identidad federados populares como Google, Facebook y Twitter, y más.

2.5.3 Cloud Functions

Cloud Functions para Firebase es un marco sin servidor que permite ejecutar automáticamente el código del backend en respuesta a eventos activados por características de Firebase y solicitudes HTTPS. El código JavaScript o TypeScript se almacena en la infraestructura de Google Cloud y se ejecuta en un entorno gestionado.

2.5.4 Firebase Cloud Messaging

FCM, o Firebase Cloud Messaging, representa el servicio gratuito de mensajería de Firebase. Permite el envío de mensajes o notificaciones a través de varias plataformas, ya sea a dispositivos individuales, grupos de dispositivos o usuarios suscritos a temas específicos que pueden crearse. Este servicio se caracteriza por su seguridad, practicidad y la ausencia de la necesidad de desarrollar lógica de mensajería para diferentes tipos de dispositivos o configurar múltiples llamadas para dispositivos distintos.

Para enviar mensajes a través de Firebase Cloud Messaging (FCM), se requiere un remitente, que puede ser el propio servidor o un entorno confiable en la nube, como Cloud Functions, y una aplicación cliente a la cual se dirigirá el mensaje.

Cuando un dispositivo se registra por primera vez, recibe un token de identificación con el cual puede solicitar suscribirse a temas. Además, se asigna una ID única al usuario registrado, la cual Analytics, el servicio de Firebase que se inicia automáticamente al utilizar FCM, puede utilizar para enviar mensajes según ciertos datos del usuario en Analytics.

Una vez asignado un token de FCM, el usuario puede recibir mensajes. Estos mensajes pueden crearse y enviarse desde la consola del proyecto, desde un servidor y aplicación propios, o a usuarios según los datos de comportamiento definidos en Analytics. Cuando el dispositivo está en línea, el mensaje se envía a través de la capa de transporte específica de la plataforma.

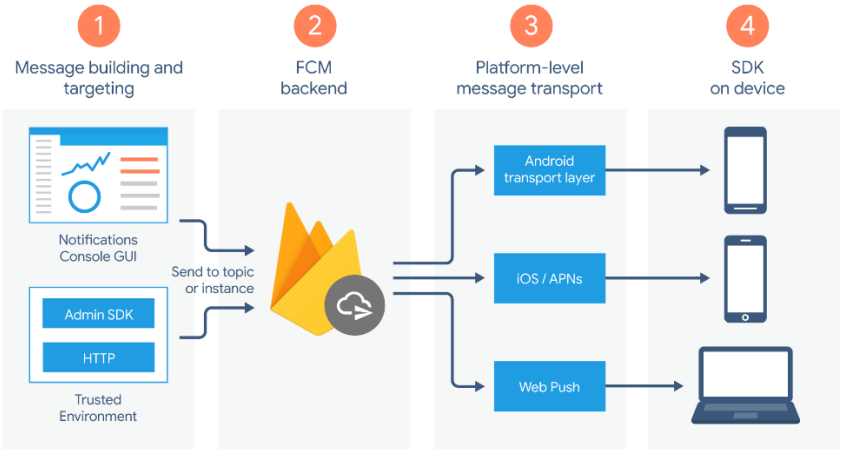


Figura 2.2: Arquitectura de Firebase Cloud Messaging [34].

2.5.5 Hosting

Firebase Hosting proporciona alojamiento rápido y seguro para su aplicación web, contenido estático y dinámico, y microservicios. Con un solo comando, puedes desplegar rápidamente aplicaciones web y servir tanto contenido estático como dinámico a una red global de distribución de contenido (CDN).

2.5.6 Firebase Performance Monitoring

Firebase Performance Monitoring se presenta como una herramienta integral que ofrece información detallada acerca del rendimiento de aplicaciones web, abarcando tanto plataformas Apple como Android. Mediante la implementación sencilla del SDK de Performance Monitoring, se pueden recopilar datos de rendimiento de la aplicación para, posteriormente, analizar y examinar estos datos en la consola de Firebase. Esta herramienta permite la identificación en tiempo real

de áreas de la aplicación que podrían beneficiarse de mejoras en el rendimiento. De esta manera, se facilita la resolución de cualquier problema de rendimiento detectado [35].

Firebase Performance Monitoring emplea informes basados en datos recopilados entre dos puntos en el tiempo dentro de la aplicación, denominados "seguimientos". Las métricas, que constituyen los datos de rendimiento extraídos de cada seguimiento, varían según el tipo de seguimiento realizado. Por ejemplo, si la aplicación ejecuta una solicitud de red, el seguimiento recopilará métricas cruciales para supervisar la solicitud, como el tiempo de respuesta y el tamaño de la carga útil [36].

2.6 Microservicios

La arquitectura de microservicios se basa en la construcción de una aplicación como un conjunto de servicios independientes entre sí. Cada servicio puede ser desarrollado en lenguajes distintos y administrado por equipos separados [37]. Hasta ahora, las aplicaciones se desarrollaban como unidades únicas, siguiendo la arquitectura monolítica. Era un monolito, un único ejecutable lógico. Cualquier cambio requería construir y desplegar una nueva versión de la aplicación del lado del servidor [38]. La arquitectura de microservicios presenta ventajas significativas en términos de escalamiento y mantenimiento. Proporciona a los desarrolladores una estructura más eficiente para el desarrollo de aplicaciones, permitiéndoles administrar recursos de manera más específica al centrarse únicamente en los servicios necesarios. Además, posibilita realizar modificaciones o cambios sin la necesidad de alterar la totalidad de la aplicación, facilitando así el proceso de desarrollo.

Ventajas del uso de microservicios para el desarrollo de aplicaciones IoT:

- **Modularidad:** Dividen la lógica de la aplicación en componentes más pequeños y especializados, lo que facilita su comprensión, mantenimiento y evolución independiente.
- **Flexibilidad:** La arquitectura basada en microservicios permite cambiar, escalar, desplegar o eliminar cada componente de forma independiente.
- **Tolerancia a errores:** Se facilita la detección, resolución de los errores y los impactos de los errores se limitan a un servicio específico y evita la propagación de fallos en toda la aplicación.
- **Entrega Continua (CI/CD):** Permiten implementaciones frecuentes y cambios incrementales sin interrupciones en los otros servicios.

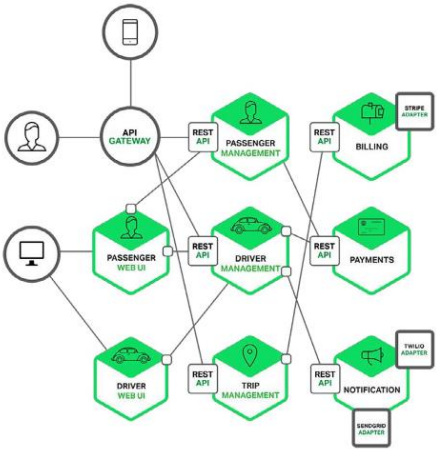


Figura 2.3: Ejemplo de arquitectura de microservicios para servicio de taxis [37].

2.7 Base de datos

Una base de datos es una colección organizada de datos almacenados y accesibles electrónicamente. Las bases de datos permiten a los usuarios crear, leer, actualizar y eliminar datos de manera eficiente.

2.6.1 Relacional y no relacional

Las bases de datos se pueden clasificar como bases de datos relacional y no relacional.

2.6.1.1 Bases de datos relacionales

La base de datos relacionales se representa como un conjunto de tablas, donde cada fila representa un registro, cada columna representa una propiedad del registro y relaciones entre un conjunto de valores. Las tablas están interconectadas mediante claves primarias y foráneas, facilitando la realización de consultas complejas a través de múltiples tablas [39].

2.6.1.2 Bases de datos no relacionales

Las bases de datos no relacionales, también conocidas como bases de datos NoSQL, no emplean tablas para almacenar datos. En su lugar, pueden almacenar datos en diversos formatos como documentos JSON, pares clave-valor, columnas anchas o grafos. Las bases de datos de clave-valor son altamente divisibles, lo que posibilita la adaptación de las aplicaciones al rendimiento a medida que aumenta el número de usuarios. Estas bases de datos resultan especialmente útiles para gestionar grandes volúmenes de datos, ya sean estructurados o no estructurados [40].

2.6.2 SQLite

SQLite es una biblioteca de software que proporciona un sistema de gestión de bases de datos relacionales. Este sistema es autónomo, no requiere configuración ni servidor, y es transaccional. Además, es gratuito tanto para fines privados como comerciales. A diferencia de la mayoría de

las bases de datos SQL, SQLite es un motor de base de datos SQL integrado que no requiere un proceso de servidor o software adicional, sino que está incrustado directamente en el programa final. Debido a su ligereza, con menos de 500 KB, es fácil de utilizar como software incrustado en dispositivos como televisores, teléfonos móviles, cámaras y dispositivos electrónicos domésticos [41].

En el contexto del Internet de las cosas (IoT), SQLite se destaca como una opción popular para el motor de base de datos en teléfonos móviles, PDAs, reproductores MP3, decodificadores y otros dispositivos electrónicos. La huella pequeña en el código, el uso eficiente de la memoria y el espacio en disco, junto con su alta confiabilidad y la falta de necesidad de mantenimiento por parte del administrador de la base de datos, hacen que SQLite sea una elección popular como formato de archivo para aplicaciones en el ámbito de IoT [41].

2.9 Análisis

En la literatura existente, se han presentado diversas soluciones para desarrollar aplicaciones de monitoreo y control de sistemas IoT. Estos trabajos consideran varios parámetros que son esenciales para el proceso de desarrollo, tales como el hardware del sistema, el método de comunicación entre los componentes, la accesibilidad del sistema IoT, sea local o alojado en la red, el almacenamiento y la cantidad de información generada, así como las necesidades en cuanto al acceso y gestión de la información.

Un desafío inherente al Internet de las cosas (IoT) que se destaca es la centralización, gestión y acceso a la información generada, especialmente cuando hay múltiples sistemas IoT en funcionamiento. En el ámbito de un sistema de cultivo inteligente, existe hardware disponible en el mercado capaz de enviar datos a la red. Sin embargo, la premisa de este trabajo es desarrollar un software personalizado para administrar el sistema IoT, aprovechando los servicios de Firebase de Google. Esta elección se sustenta en las ventajas y facilidades ofrecidas por Firebase, así como en la novedad de su aplicación a un sistema de riego inteligente, presentando oportunidades aún no exploradas en proyectos anteriores.

La propuesta consiste en la creación de una aplicación web destinada a recopilar y visualizar información de la base de datos en tiempo real. Además, se busca implementar un control sobre los datos de variables ambientales, rutinas de riego y activación de un sistema de riego inteligente. Esta interfaz tiene como objetivo reducir el tiempo de integración de aplicaciones en el sistema, facilitar la administración y permitir el escalamiento. Se aprovecha la capacidad actual de los navegadores web para acceder y controlar dispositivos IoT directamente desde el navegador, lo que brinda a los usuarios una experiencia integrada sin necesidad de utilizar programas adicionales [42].

Capítulo III

Marco Metodológico

En este capítulo se establecen los requisitos de la aplicación en relación con las variables adquiridas, el almacenamiento, el acceso a la nube y la gestión. Se diseña el sistema para monitorear y operar el riego de manera eficiente y segura, utilizando la información adquirida.

3.1 Metodología

El modelo de cascada, una metodología tradicional utilizada en este trabajo de grado se fundamenta en un proceso lineal y secuencial. Esta metodología describe el desarrollo del proyecto en fases consecutivas y escalonadas, dividiendo el cumplimiento de los objetivos en etapas bien definidas. Proporciona un camino claro de actividades y objetivos por fase, permitiendo avanzar a la siguiente una vez cumplida cada etapa.

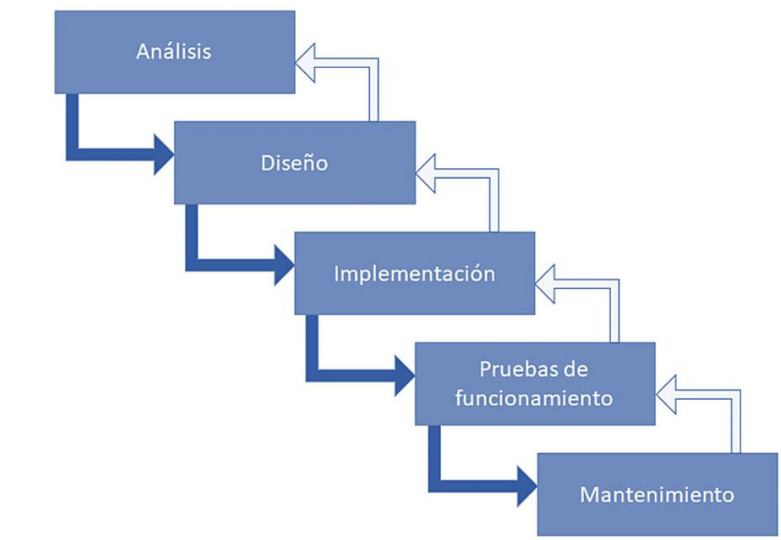


Figura 3.1: Modelo en cascada [33].

1. **Análisis de Requisitos:** En el análisis de requisitos se acotan las soluciones, se exponen las actividades a realizar y los mínimos aceptables que responden al cumplimiento de los objetivos.
2. **Diseño del sistema:** Definidos los requisitos a cumplir se diseña la forma de llevarlo a cabo. En esta fase, se diseña el software.

3. **Implementación:** En las fases de desarrollo e implementación se abordan de forma independiente el algoritmo del sistema de riego y la aplicación para la administración. El programa del sistema de riego se prueba y demuestra su funcionamiento en un computador local, mientras que la aplicación se desarrolla utilizando la plataforma de desarrollo de aplicaciones de Firebase para operar en la nube.

3.2 Dinámica del sistema IoT

Se estableció un sistema con la dinámica que se observa en la figura 3.2. La dinámica del sistema permite estructurar el comportamiento del sistema, por medio de la interacción de las partes, comunicación entre las partes, los bucles de realimentación y la evolución en el tiempo.

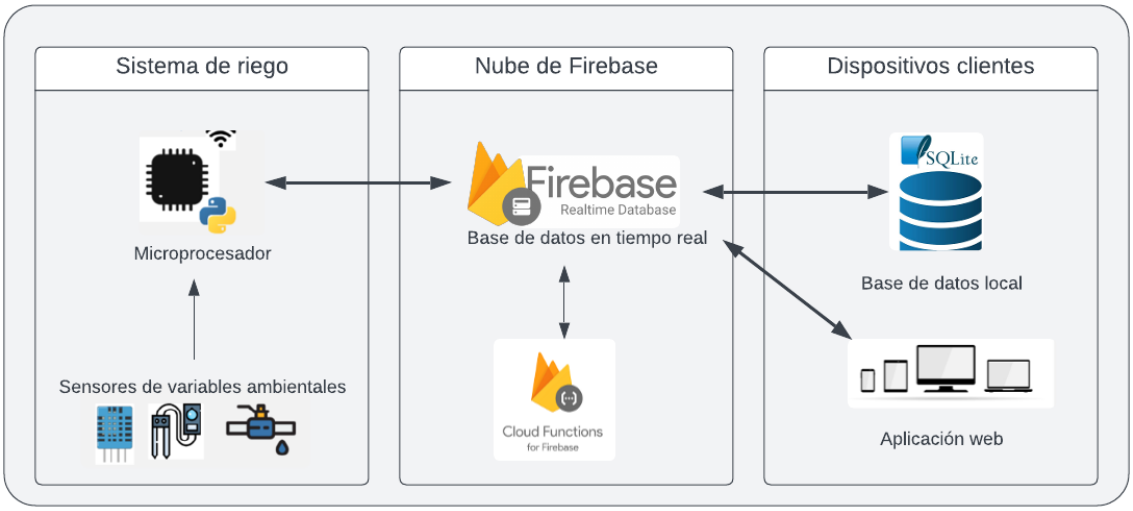


Figura 3.2: Diagrama del sistema de riego IoT

3.3 Sistema de riego

3.3.1 Requisitos del sistema de riego

El sistema de riego para la prueba de concepto se compone de sensores de variables ambientales (temperatura, humedad del aire, humedad del suelo) y una bomba de agua como actuadores, todos conectados a un microcomputador. La integración de los servicios de Firebase con la lógica del microcomputador se realizará mediante la inclusión de las dependencias adecuadas y el uso de las herramientas de desarrollo de Firebase Admin para Python. La base de datos en tiempo real servirá como medio para el envío de datos y la recepción de instrucciones, asegurando una comunicación adecuada con los permisos correspondientes. El objetivo es administrar remotamente el sistema de riego, permitiendo el monitoreo de información y funciones como la modificación de parámetros de funcionamiento, así como la ejecución y detención de acciones en el sistema.

3.3.2 Diseño del sistema de riego

El algoritmo de control y monitoreo del sistema de riego es ejecutado por el microcomputador, el cual interactúa con la nube de Firebase enviando información y recibiendo instrucciones.

El microcomputador correrá un programa en Python con las instrucciones y configuración para la comunicación con Firebase y la realización de las operaciones del sistema de riego.

3.3.2.1 Diagrama de flujo del control del riego en el microcomputador

La funcionalidad del algoritmo de control del sistema de riego es gestionar los componentes de hardware, monitorear las variables ambientales y el estado de riego (bomba) y ejecutar acciones. Acciones como iniciar los ciclos de riego manual o automático, ejecutar comandos de terminal, publicar la información, detener o iniciar el programa.

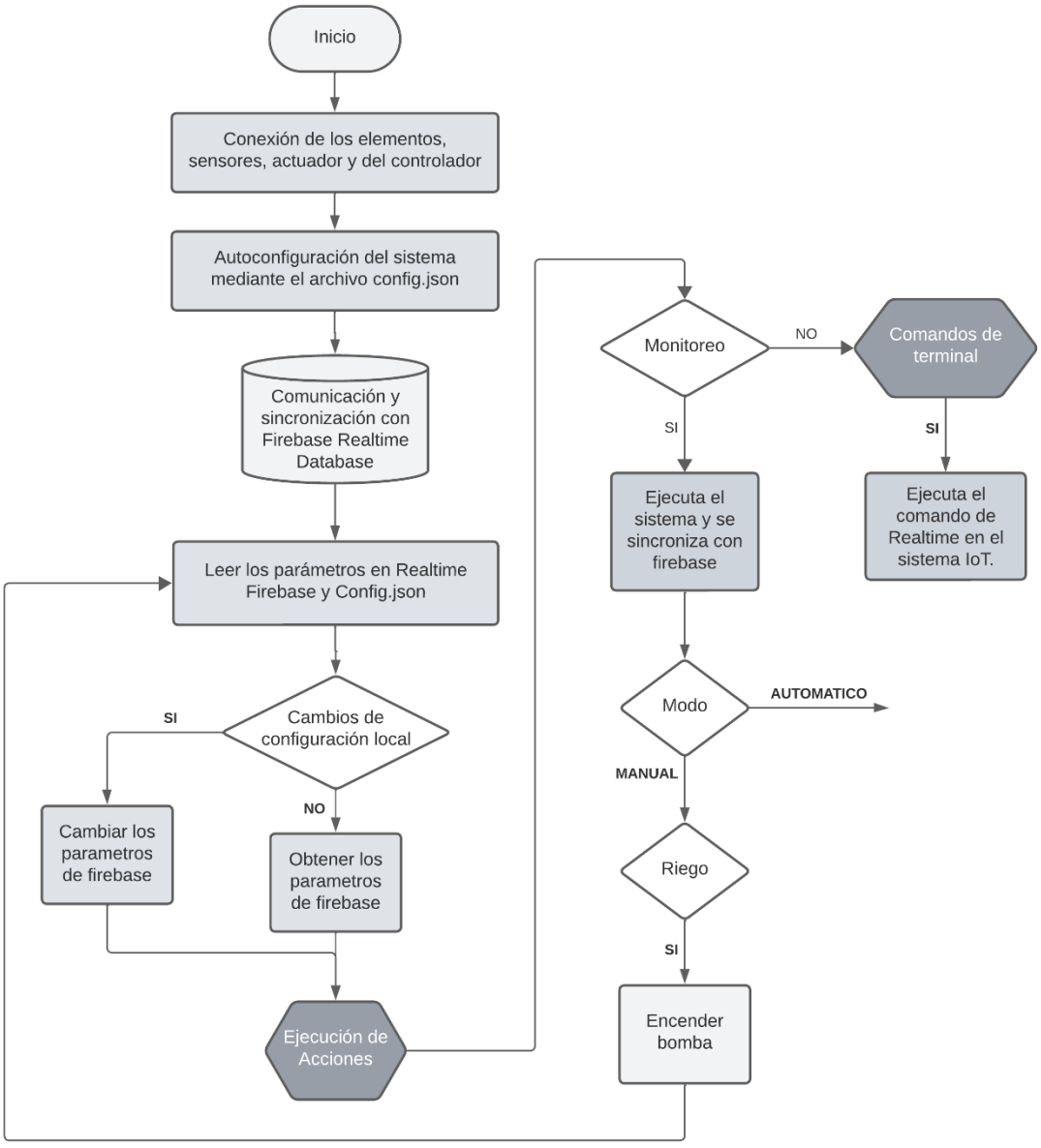


Figura 3.3: Diagrama de flujo del programa en Python del sistema IoT.

3.3.2.2 Ciclo de riego automático

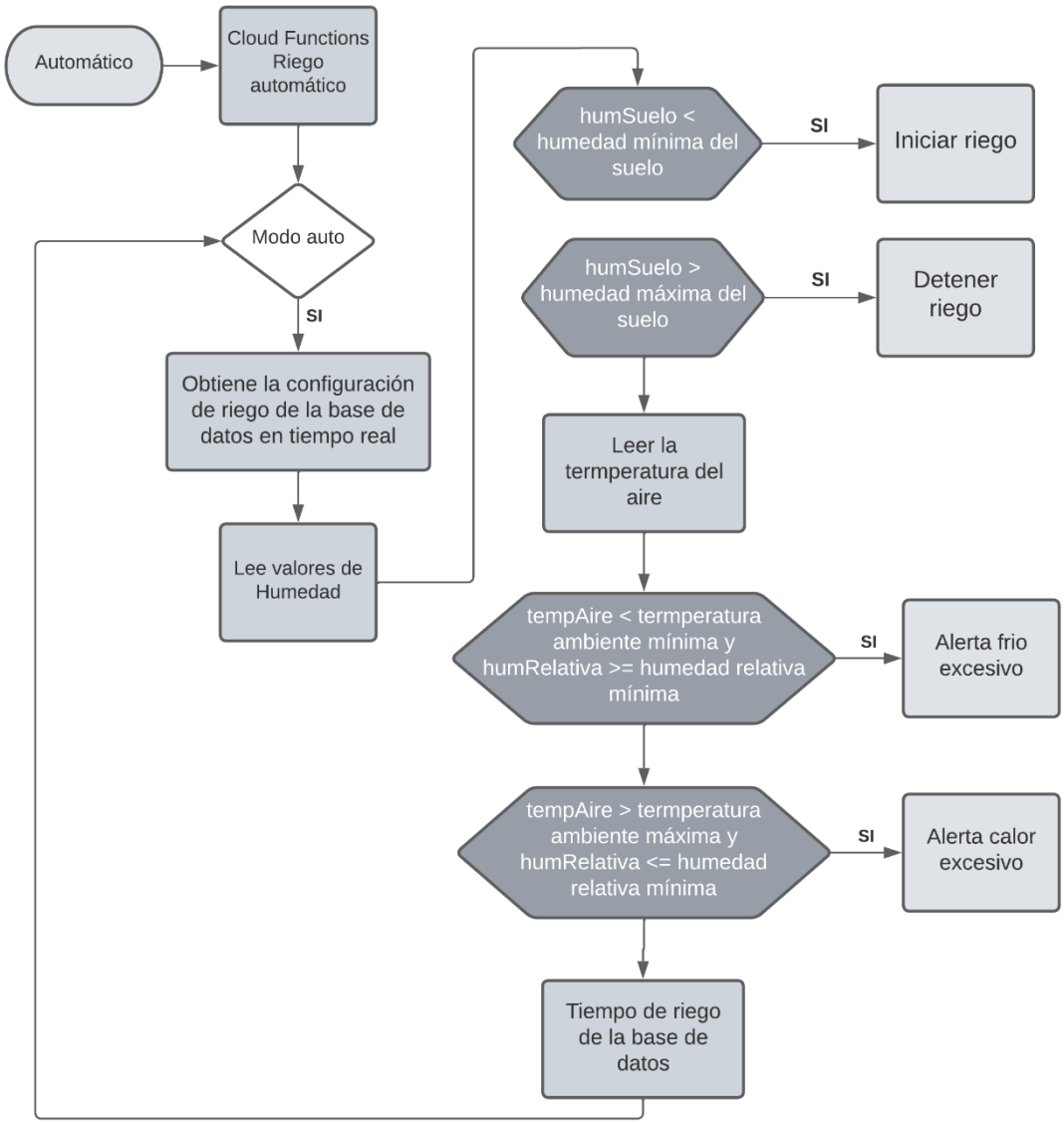


Figura 3.4: Flujo del ciclo de riego automático de la aplicación.

3.4 Nube de Firebase

La plataforma Firebase opera como el punto central del proyecto, conectando de manera integral todas las partes del sistema de riego y proporcionando funcionalidades a través de microservicios. Estos servicios en la nube desempeñan un papel esencial en la gestión inteligente del sistema de riego, facilitando la adquisición, almacenamiento y procesamiento de datos, así como las

actualizaciones y ejecuciones remotas. A continuación, se detallan los requisitos y características específicas de cada servicio.

3.4.1 Realtime Database

El servicio centralizado de base de datos elimina la necesidad de desarrollar o utilizar sistemas de comunicación distintos entre las diversas plataformas de hardware, permitiendo la comunicación y sincronización eficientes de los elementos del sistema. Facilita la consulta, creación y actualización de información de manera remota, asegurando su compatibilidad con todas las plataformas del sistema. Firebase Realtime Database se establecerá como el nodo central del sistema, posibilitando la sincronización entre todas las partes. Esta base de datos en tiempo real albergará información en tiempo real de los sensores, así como la configuración del sistema y las instrucciones para las acciones definidas. El sistema de riego, a su vez, obtendrá los parámetros necesarios y ejecutará acciones según lo establecido en Firebase Realtime.

3.4.1.1 Base de datos en tiempo real

Árbol JSON de la base de datos en tiempo real de Firebase.

Figura 3.5: Nodo raíz y nodos secundarios del proyecto en Realtime

En el nodo data, se encuentran los nodos finales con los datos de las variables ambientales y el estado de riego.

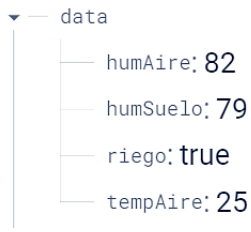


Figura 3.6: Nodos finales del nodo data en Realtime.

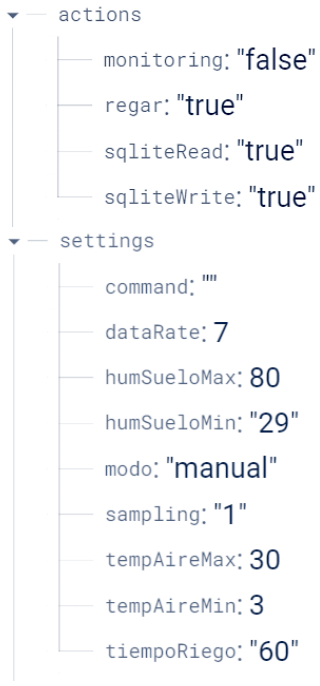


Figura 3.7: Nodos finales de los nodos control, settings y users en Realtime.

En el nodo de actions se encuentran las acciones y en el nodo settings alberga los parámetros para ejecución en el sistema de riego.

Tabla 3.1

Descripción de las acciones de Realtime Database.

Clave	Valor
regar	True/ False, ejecuta el riego.
modo	Modo de riego manual o auto.
monitoring	True/False, publica los valores en Firebase.
sqLiteRead	True/False, descarga datos de la base de datos histórica.
sqLiteWrite	True/False, guarda datos en la base de datos histórica.

Tabla 3.2

Descripción de las configuraciones de Realtime Database.

Clave	Valor
command	Comandos de terminal
dataRate	True/ False, ejecuta el riego.
humSueloMax	Humedad del suelo para detener el riego
humSueloMin	Humedad del suelo para iniciar el riego
tempAireMax	Temperatura ambiente máxima, envío alerta de calor
tempAireMin	Temperatura ambiente mínima, envío alerta de frío
tiempoRiego	Duración del riego automático.

El nodo user crea con la autenticación de los usuarios, siendo un registro de los usuarios autenticados y albergando los datos anexados a su perfil, como el token para recibir mensajes desde Firebase Cloud Messaging y sus configuraciones de visualización de la página. En la imagen se observa el uid del usuario donde se guarda el token, junto con la configuración del widget de visualización para la variable humAire.

Figura 3.8: Nodo con los datos de los usuarios

3.4.2 Firebase Auth

El método de seguridad por defecto en el proyecto de Firebase para la identificación y acceso a los usuarios es el sistema de autenticación. Este sistema permite obtener las credenciales de los usuarios, lo que es fundamental para realizar funciones personalizadas como el envío de notificaciones por correo, el almacenamiento de configuraciones e información del perfil.

3.4.3 Cloud Functions

En Cloud Functions, se aprovechan las capacidades de procesamiento en la nube para realizar funciones de control del sistema, superando las limitaciones de procesamiento en microcontroladores y microprocesadores. Se definen funciones predefinidas que se activan en respuesta a eventos específicos. Las condiciones que desencadenan estas acciones incluyen alcanzar valores máximos, mínimo o igual en la base de datos en tiempo real.

Las acciones de respuesta asociadas a estas condiciones pueden ser notificaciones o modificaciones de valores en la base de datos en tiempo real. Los usuarios configuran los parámetros para estas condiciones o acciones según sus necesidades.

Cuando se cumplen las condiciones, Cloud Functions recibe llamadas para ejecutar las funciones con parámetros previamente definidos. Una vez activada, realiza acciones pertinentes, como modificaciones en la base de datos en tiempo real de Firebase. Es esencial destacar que las modificaciones se distribuyen automáticamente a todos los dispositivos finales conectados, asegurando una ejecución consistente y sincronizada.

En el desarrollo de Cloud Functions, se implementarán y desplegarán las funciones creadas, escritas en JavaScript o Python, utilizando el SDK correspondiente. Cloud Functions no es un servicio unificado, sino un conjunto de funciones que se ejecutan en la nube y responden a distintas solicitudes de la aplicación. El procedimiento para desarrollar estas funciones implica un enfoque específico y detallado.

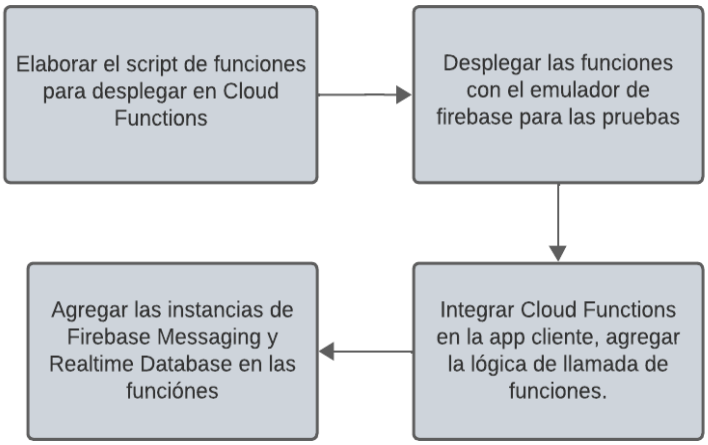


Figura 3.9: Procedimiento del desarrollo de funciones en Cloud Functions.

3.4.4 Firebase Cloud Messaging

Para utilizar el sistema de mensajes de Firebase, el primer paso consiste en inicializar la instancia de FCM en nuestra aplicación cliente. Para esto, se requieren las credenciales web “VAPID” keys, que se utilizan para suscribir nuestra aplicación al servicio de notificaciones push.

Este proceso incluye la inicialización del servicio de mensajería, así como funciones para solicitar permisos a los usuarios, obtener el token de FCM para cada usuario y almacenarlo en el nodo correspondiente en la base de datos en tiempo real.

3.5 Aplicación para la administración del sistema de riego

La aplicación no solo permite visualizar y recopilar datos, sino que también habilita un control ligero del sistema a través de acciones en la interfaz del usuario o condiciones que generen acciones específicas. Utiliza la base de datos en tiempo real de Firebase para realizar operaciones CRUD, lo que significa que se pueden crear, leer, actualizar y borrar datos mediante métodos HTTP. Facilita la gestión a través de una interfaz de usuario con elementos visuales para la monitorización.

La aplicación funciona de manera serverless y está alojada en Firebase Hosting, una plataforma para páginas web estáticas sin necesidad de gestionar hardware de servidor físico. Aunque Firebase Hosting se limita inicialmente al alojamiento de contenido estático, se puede implementar código backend utilizando un servidor propio o un servicio adicional de computación en la nube, como Cloud Run. En este caso, se elige diseñar una aplicación estática y agregar las funcionalidades de servidor necesarias como microservicios. La aplicación simula el funcionamiento de una SPA (Single Page Application), donde todo el contenido se ajusta dentro de una sola página. La eliminación de la necesidad de navegar entre varias páginas simplifica el desarrollo y aprovecha la actualización de contenido según la interacción del usuario.

La arquitectura de microservicios divide las funcionalidades en servicios más pequeños y desacoplados que se comunican entre sí. La aplicación se desarrolla con HTML, CSS y JavaScript, utilizando el framework web Astro. Astro permite el uso de plantillas y la creación de componentes reutilizables, así como la integración con herramientas web como Tailwind y otros frameworks como React.

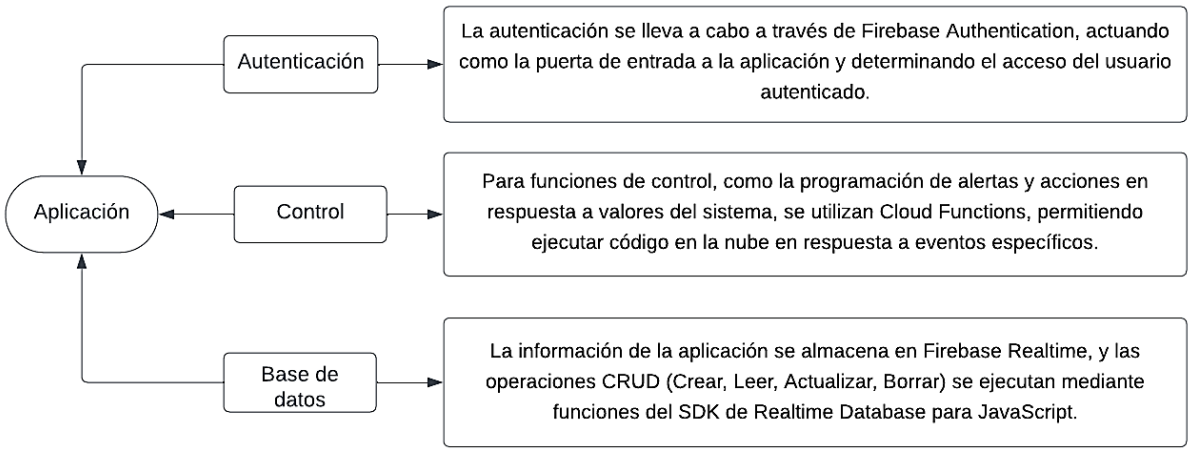


Figura 3.10: Microservicios de Firebase para la aplicación

3.5.1 Diagrama de flujo de la aplicación

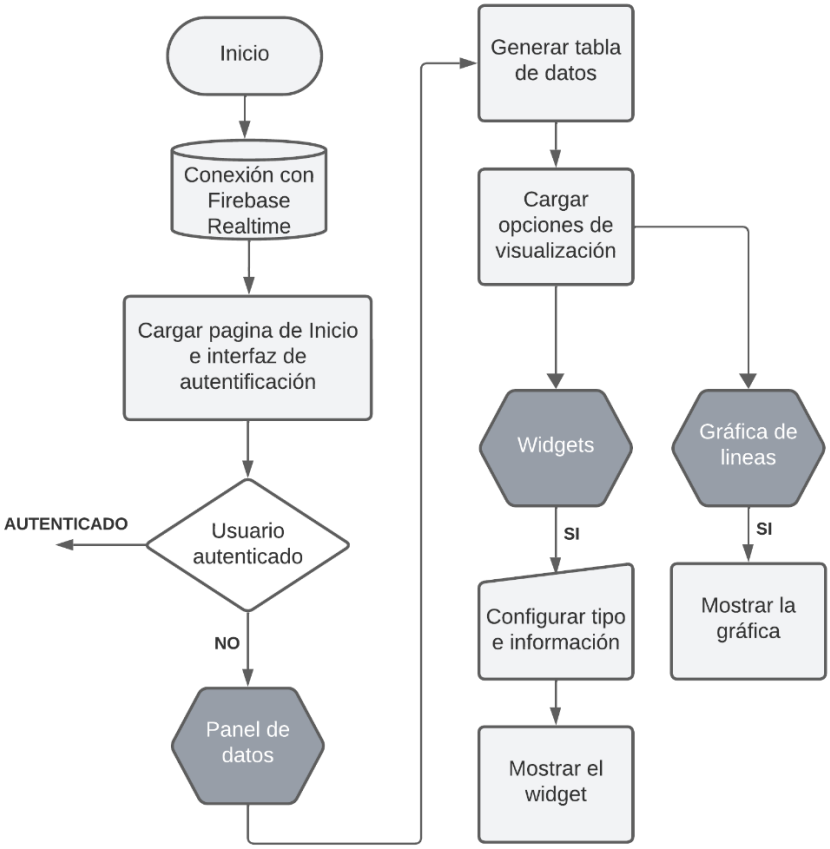


Figura 3.11: Diagrama de flujo de la aplicación usuarios no autenticados

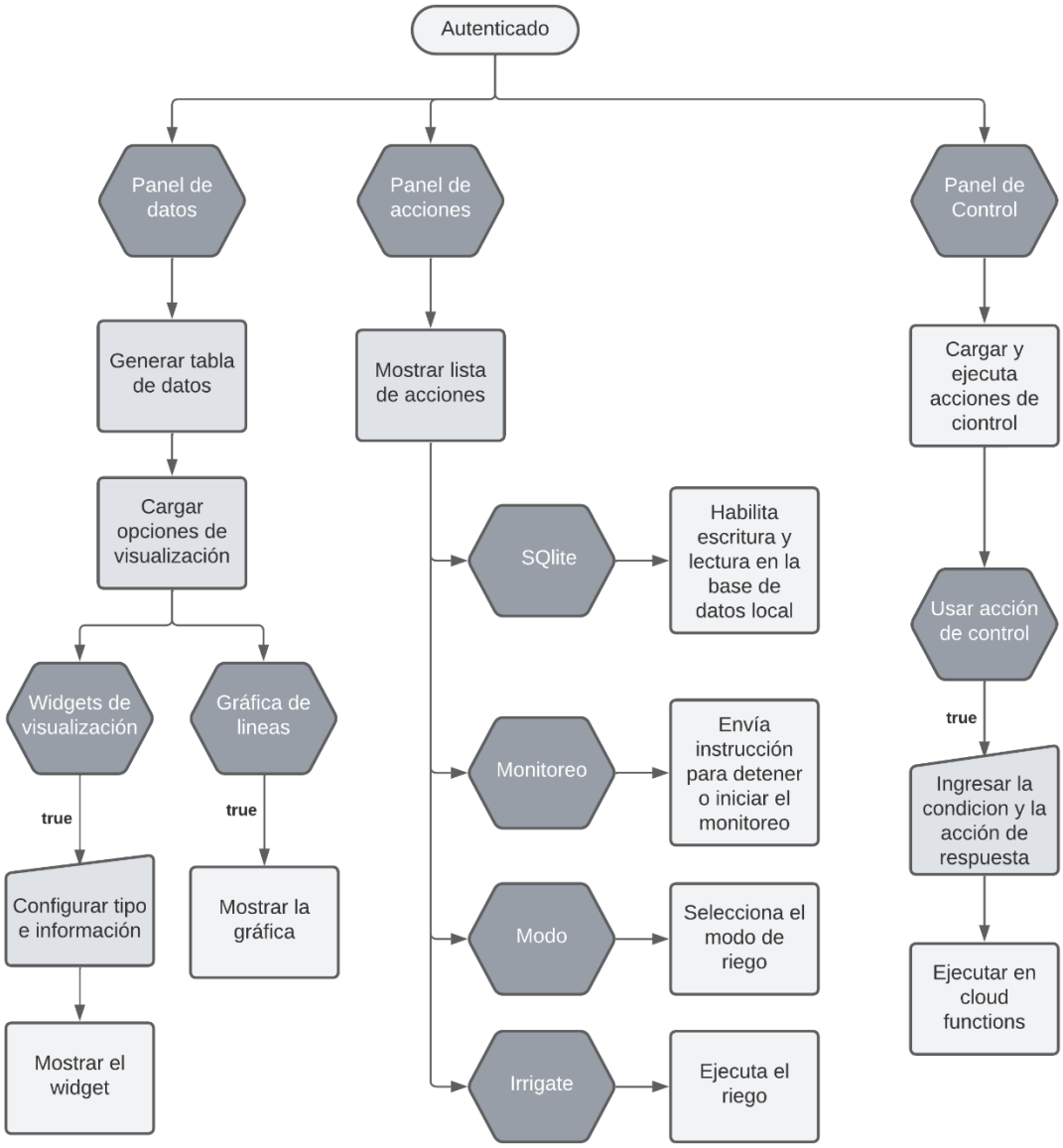


Figura 3.12: Diagrama de flujo aplicación para usuarios autenticados.

3.6 Base de datos SQlite

En un servidor remoto independiente se encuentra la base de datos local SQLite. El programa en Python realiza la recopilación de datos del proyecto, extrayendo la información del sistema de riego para el almacenamiento local. Los datos se almacenan en una base de datos SQLite y pueden obtenerse desde la aplicación. La gestión de esta base de datos se lleva a cabo mediante un programa en Python que utiliza SQLite3 como el módulo de integración correspondiente.

A diferencia de la base de datos en tiempo real, la base de datos local SQLite funciona como un registro histórico del proyecto. Al ser una base de datos relacional, toda la información del proyecto se organiza en una tabla, donde cada nuevo registro se convierte en una fila. La

información se puede consultar de diversas maneras utilizando consultas en Python y se almacena en un solo archivo, optimizando el espacio en el servidor.

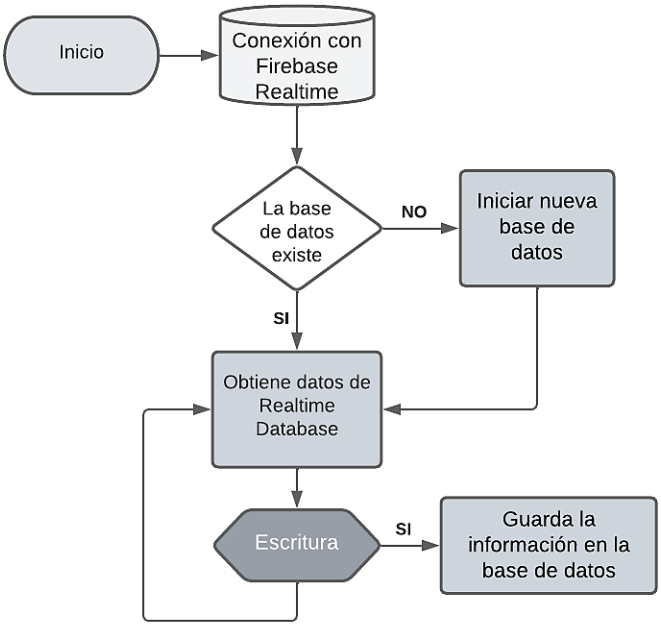


Figura 3.13: Diagrama de flujo base de datos SQLite

La colección de datos en SQLite posibilita realizar verificaciones y análisis complejos de los resultados con Python. Además, permite el entrenamiento de modelos inteligentes con la información procesada.

3.6.1 Entidad y valores.

El modelo de datos para el almacenamiento en la base de datos SQLite sigue el mismo formato y precisión heredados de la base de datos en tiempo real, manteniendo la consistencia en la estructura de los datos. La cantidad y frecuencia de los datos dependerán de los resultados y mediciones obtenidos del sistema de riego.

Los datos se guardan en la entidad llamada "History", organizada con las siguientes columnas de atributos:

- EstampaTiempo: Momento de la toma de datos en milisegundos.
- HumedadSuelo: Valor del porcentaje de humedad en el suelo.
- HumedadAire: Valor del porcentaje de humedad del aire.
- TemperaturaAire: Valor de la temperatura ambiente.
- Riego: Estado del riego.

Esta colección de datos constituye el historial de los valores del nodo data en la base de datos en tiempo real, donde cada fila representa nuevos valores obtenidos por el sistema de riego.

Capítulo IV

Resultados y análisis

En este capítulo se presenta la implementación, los resultados y las pruebas. En la implementación se muestra los requerimientos funcionales, los procesos de desarrollo del programa del sistema IoT y de la aplicación, comenzando con el desarrollo de los servicios de Firebase y terminando con el despliegue del sitio web en el hosting. Se demostrará el correcto funcionamiento de las partes de la aplicación, de los programas de la base de datos interna SQLite y del programa del microcomputador rpiScript.py.

4.1 Implementación

4.1.1 Programa del microprocesador con Python

Para visualizar la implementación del programa se exponen los diferentes procesos junto a breves fragmentos del código de programa correspondiente. En la comunicación con Firebase Realtime Database, se utiliza la biblioteca de Python Firebase Admin SDK, que constituye un conjunto de herramientas de código, como se explicó en el marco teórico.

4.1.1.1 Obtención de datos de Realtime Database

Desde Firebase Admin SDK, se importan "credentials" para la autenticación mediante una cuenta de servicio y "db" para la inicialización de la base de datos desde la URL especificada en el programa. Al otorgar las credenciales como cuenta de servicio, esta instancia tendrá acceso total de escritura y lectura sin importar las reglas de seguridad configuradas.

```
import firebase_admin
from firebase_admin import credentials, db

#Credenciales de cuenta de servicio de firebase
cred = credentials.Certificate(
    'iotriego-17bad-firebase-adminsdk-pcsm5-38d08da6a5.json')

firebaseConfig = {
    "databaseURL": "https://iotriego-17bad-default-rtdb.firebaseio.com"}

# Inicializa la aplicación con las credenciales y la configuración.
firebase_admin.initialize_app(cred, firebaseConfig)

# db reference representa el nodo en esa dirección (databaseURL).
```

```
realtimedb = db.reference()
```

El contenido de datos del sistema, los valores de los componentes de sistema de riego como sensores y actuadores, acciones de control y configuraciones de funcionamiento se encuentran reflejadas un archivo config.json en concordancia con las funciones del programa.

Importación los valores de este registro para la configuración de parámetros por defecto e inicialización de variables.

```
with open('config.json', 'r') as archive:
        config = json.load(archive)
        proyect = str(config["proyect"])
        settings = config["settings"]
        actions = config["actions"]
        data = config["data"]
        devices = config["devices"]
```

Se envía este registro Firebase, esta es la misma estructura de datos que se utilizará en Realtime Database, de esta forma nos aseguramos de que la base de datos en tiempo real esté sincronizada.

```
        fb = realtimedb.child(proyect)
        fb.child('settings').set(settings)
        fb.child('actions').set(actions)
        fb.child('data').set(data)
        fb.child('devices').set(devices)
```

el registro de datos `config.json` es de donde Firebase obtiene los datos y realiza los cambios, así mismo estos datos de configuración y acciones serán tomados desde este registro ser aplicados en el sistema de riego por el programa en el microcomputador.

El programa lee los datos del registro

```
        with open('config.json', 'r') as archive:
            config = json.load(archive)
            n_settings = config["settings"]
            n_actions = config["actions"]
```

El usuario puede escribir directamente en este registro de datos y de igual manera los datos serán tomados y ejecutados en el sistema de riego, aunque estos cambios no hayan sido realizados desde la base de datos en tiempo real. Esto es útil para permitir a programas u operarios administrar el sistema. Cuando esto no suceda, desde Firebase se escribiría los cambios en el registro

Los datos de las variables ambientales se pueden adquirir de los sensores mediante implementación de las bibliotecas necesarios, en este proyecto no se cuenta con elementos de hardware en la realidad como sensores, por ende, para prueba de funcionamiento se utilizan valores de muestra de valores reales de humedad y temperatura del área de Ibarra.

4.1.1.2 Acciones en el sistema de riego

Este módulo se encarga de tomar decisiones sobre el sistema de riego.

La publicación de datos en Firebase se realiza mientras la opción “monitoring” sea True. La frecuencia en el envió de datos se hace en base a un tiempo “data_rate”, ambos parámetros ubicados en Realtime Database controlados por el usuario.

```
    # Envía datos hacia Firebase
    while actionsRef.child("monitoring").get():

        processTime = startTime-endTime
        try:
            time.sleep(settings.get("data_rate")-(processTime))
        except:
            pass

        try:
            data = outputs()
            result = sendData(data)

def sendData(value):
    # Json to send to firebase
    result = fb.child('data').update(value)
    return result
```

Ejecución de commandos por terminal, se ejecutarán los comandos que provengan de Realtime Database, una solución más directa y con menos latencia, la lectura directa desde la Raspberry Pi podría ser más adecuada2mientras estos se encuentren dentro de una lista permitida, esto solo funcionará mientras “monitoring” en Realtime Database sea False, es decir mientras el programa no este enviando datos a Firebase.

```
    # Lista de comandos permitidos sin necesidad
    allowed_commands = ['pgrep', 'top', 'htop','df', 'ping', 'uname', 'ls',
'cat']
```

Si el comando es permitido entonces:

```
            result = subprocess.run(command, shell=True, check=True,
stdout=subprocess.PIPE, stderr=subprocess.PIPE)
```

Este es la función principal del programa, desde donde se evalúa en base a los datos los parámetros y acciones a ejecutar.

```
def main():
    while True:

# Sincronización de Realtime Database y config.json
        firebaseSinchronize()

        print(readTime(0)+" -- waiting for instructions from Firebase or ter-
minal")

# Monitoreo y publicación de valores en Realtime Database
```

```python
if actions.get("monitoring"):
    monitoring()

fbCommand = actions.get("command")

# Ejecución de comandos de Realtime Database
if fbCommand != "":
    if fbCommand == "salir":
        break
    else:
        # Ejecutar el comando en el sistema
        actionsRef.child("command").set("")
        exCommands(fbCommand)
```

```
PS C:\Users\Dávalos Bryan\OneDrive - Universidad Tecnica del Norte\Tesis App\Tesis> & "C:/Users/Dáv
ython311/python.exe" "c:/Users/Dávalos Bryan/OneDrive - Universidad Tecnica del Norte/Tesis App/Tes
23 Jan 21:12:16 -- waiting for instructions from Firebase or terminal
Datos enviados a Firebase, data rate: 7s
{'humAire': 65, 'humSuelo': 42, 'tempAire': 22, 'riego': False, 'irrigate': None}
Datos enviados a Firebase, data rate: 7s
{'humAire': 64, 'humSuelo': 42, 'tempAire': 22, 'riego': False, 'irrigate': None}
Datos enviados a Firebase, data rate: 7s
{'humAire': 64, 'humSuelo': 42, 'tempAire': 22, 'riego': False, 'irrigate': None}
Datos enviados a Firebase, data rate: 7s
{'humAire': 65, 'humSuelo': 42, 'tempAire': 22, 'riego': False, 'irrigate': None}
Datos enviados a Firebase, data rate: 7s
```

Figura 4.1: Respuesta de la terminal del sistema de Riego

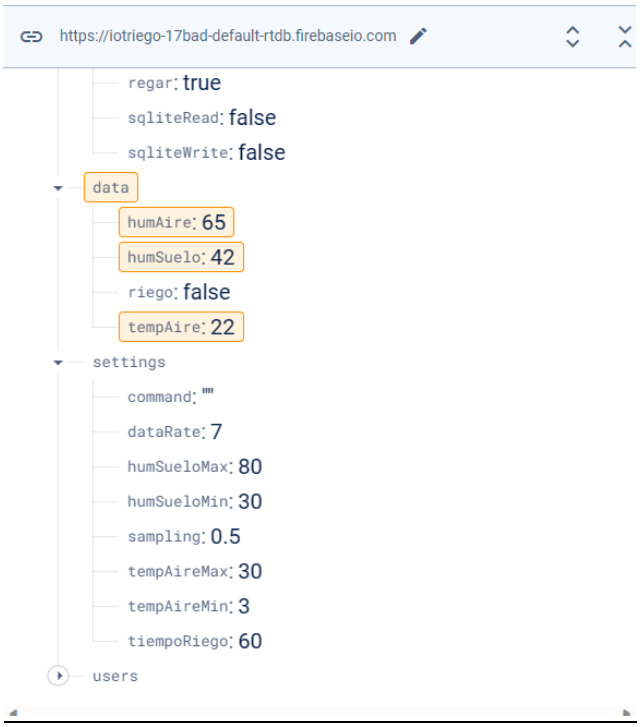


Figura 4.2: Escritura del programa en le base de datos en tiempo real

4.1.2 Aplicación web de usuario para la administración

El proceso de comunicación e integración de los microservicios desarrollados en la nube se describe mediante las acciones realizadas para cada parte de la aplicación y el funcionamiento propuesto. Se proporcionan detalles sobre las operaciones llevadas a cabo para asegurar la cohesión y colaboración eficiente entre los diversos microservicios en la nube.

4.1.2.1 Configuración de Firebase

Para usar los servicios y conectar la aplicación al proyecto de Firebase se debe inicializar la aplicación con la configuración del SDK del proyecto obtenida desde la consola.

Figura 4.3: Configuración del SDK en la consola de Firebase

Esto se realiza dentro del archivo de configuración de Firebase denominado firebase.js dentro de la carpeta de JavaScript.

firebase.js

```
//Dependencias de Firebase requeridas para este proyecto
import { initializeApp } from 'firebase/app';
import { getDatabase } from 'firebase/database';
import { getAuth } from 'firebase/auth';
import { getFunctions } from 'firebase/functions';
import { getMessaging, isSupported } from "firebase/messaging";
import { getPerformance } from "firebase/performance";

//Configuración del proyecto de firebase de la aplicación
const firebaseConfig = {
    apiKey: "AIzaSyBcLGOVLIr-R64vY7dT1mWGdYhd4M26bp8",
```

```
    authDomain: "iotriego-17bad.firebaseapp.com",
    databaseURL: "https://iotriego-17bad-default-rtdb.firebaseio.com",
    projectId: "iotriego-17bad",
    storageBucket: "iotriego-17bad.appspot.com",
    messagingSenderId: "842817138478",
    appId: "1:842817138478:web:3cc5b91a6b3a0196bb94bf"
};

// Inicializa Firebase, después de inicializar con la configuración de nues-
tro proyecto podemos inicializar y llamar a otras instancias del SDK de Fire-
base para usar servicios como auth, messaging, database y functions.

export const app = initializeApp(firebaseConfig);
export const database = getDatabase(app);
export const auth = getAuth(app);
export const functions = getFunctions(app);
export const performance = getPerformance(app);
```

Al inicializar la instancia de Cloud Messaging getMessaging() arroja un error, de que el servicio no está disponible. Para solucionar se debe utilizar una función en el SDK de Javascript de FCM, isSupported() , que devuelve verdadero si la instancia de FCM se puede inicializar.

```
export const messaging = async() => await isSupported() && getMessaging(app);
```

Muchos de estos servicios requieren primero haber sido activados desde la consola del proyecto en la página de Firebase antes de su implementación en código. Se utilizan todas las versiones Web del SDK para los diferentes servicios en la aplicación puesto que Firebase en cada servicio soporta muchas otras plataformas.

4.1.2.2 Autenticación

Es el método de seguridad por defecto en el proyecto de Firebase para la identificación y acceso a los usuarios. Permite obtener las credenciales de los usuarios para funciones personalizadas como él envió de notificaciones por correo, almacenamiento de configuraciones e información del perfil. La autenticación de usuarios es el proceso de identificación, permite el acceso total solo a usuarios registrados para el proyecto.

Se puede registrar usuarios fuera de la consola del proyecto implementado las funciones de registro del SDK de autenticación sin embargo en la aplicación desarrollada no se implementa esta funcionalidad. El registro de usuarios se restringe a la consola del proyecto en Firebase con el fin de que solo el administrador del proyecto pueda como se aprecia en la figura 4.4, agregar usuarios, inhabilitar o borrar cuentas. Esto simplifica el trabajo y da robustez a la seguridad de la aplicación.

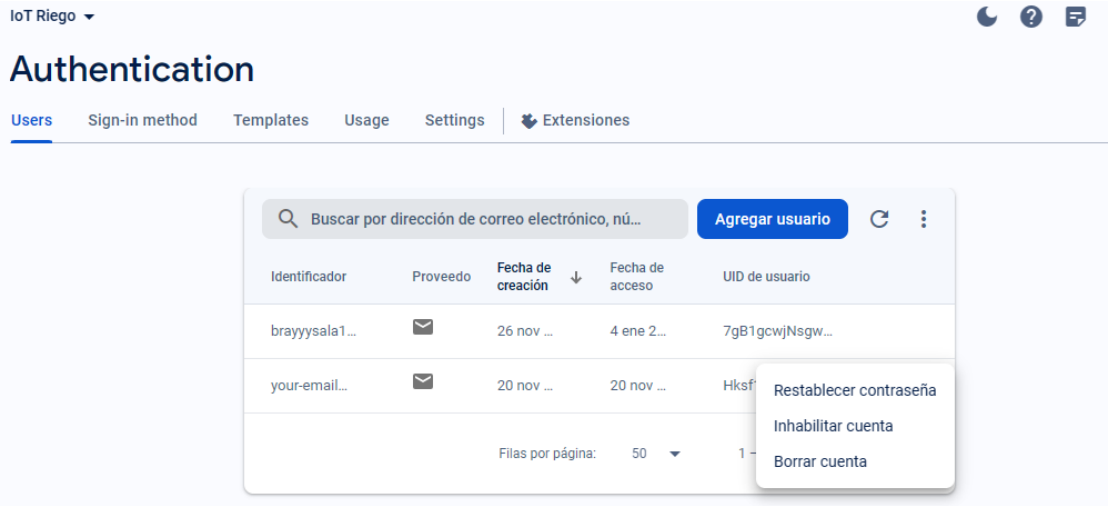


Figura 4.4: Panel Autenticación en la consola de Firebase

auth.js

Para las funciones de autenticación se requieren las dependencias

```
import {
    createUserWithEmailAndPassword,
    onAuthStateChanged,
    sendEmailVerification,
    sendPasswordResetEmail,
    signInWithEmailAndPassword,
    signOut,
} from 'firebase/auth';
```

Primero se importa la instancia de Firebase auth desde el archivo de configuración de Firebase

```
import {auth} from './firebase.js';
```

Con auth obtenemos el objeto de referencia al sistema de autenticación del proyecto. Con el cual podemos llamar a muchas funciones de autentificación de firebase.

El sistema utiliza autenticación de correo electrónico/contraseña, mediante la función signInWithEmailAndPassword del SDK de autenticación y un formulario, donde con los parámetros email y password se ejecuta la función.

```
signInWithEmailAndPassword(auth, email, password).
```

Se aplica un observador al estado de autenticación, este permite en todas las páginas de la aplicación obtener información del usuario.

```
onAuthStateChanged(auth, function (user) {
    if (user) {
        // User is signed in.
        const uid = user.uid;
        signInStatus.textContent = 'Signed in';
        userUid.textContent = uid;
```

```
        accountDetails.textContent = JSON.stringify(user, null, '  ');
        authUser.style.display = "block";
        authSign.style.display = "none";

    } else {
        // User is signed out.
        signInStatus.textContent = 'Signed out';
        accountDetails.textContent = 'null';
        authUser.style.display = "none";
        authSign.style.display = "block";
    }
});
```

Cada vez que cambia el estado de autenticación se llama al observador. De esta manera podemos crear contenido condicional y llamar a funciones con los datos de cada usuario.

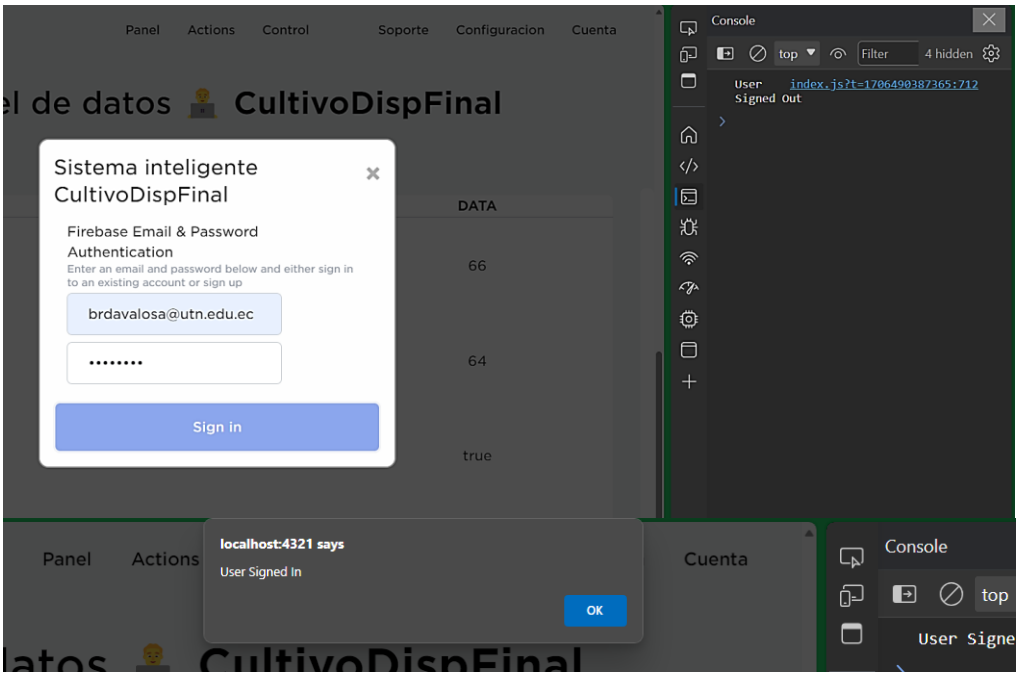


Figura 4.5: Interfaz para inicio de sesión en la aplicación

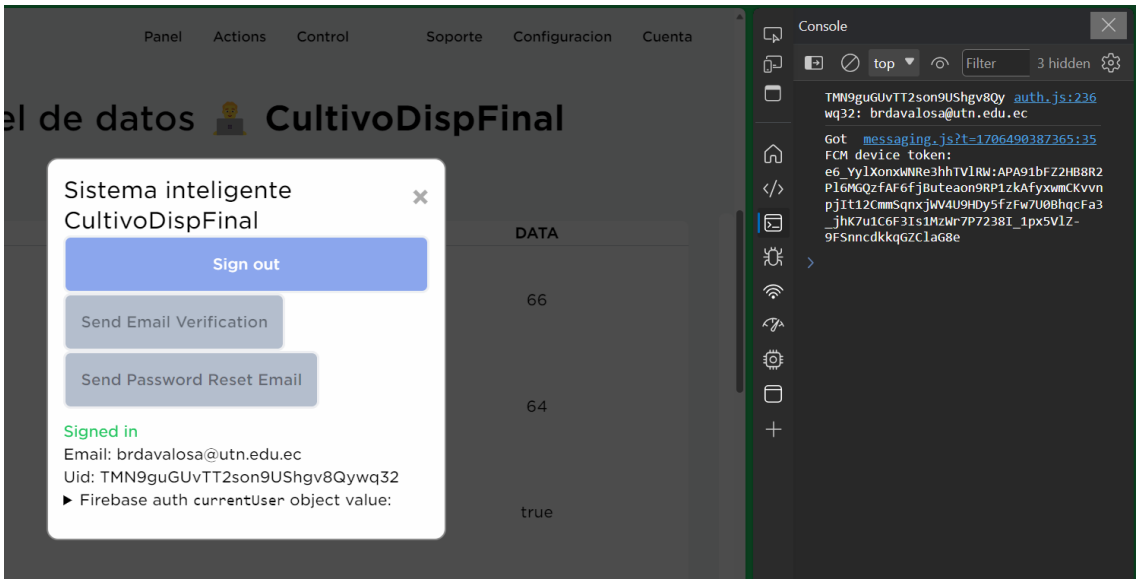


Figura 4.6: Sesión de usuario iniciada en la aplicación

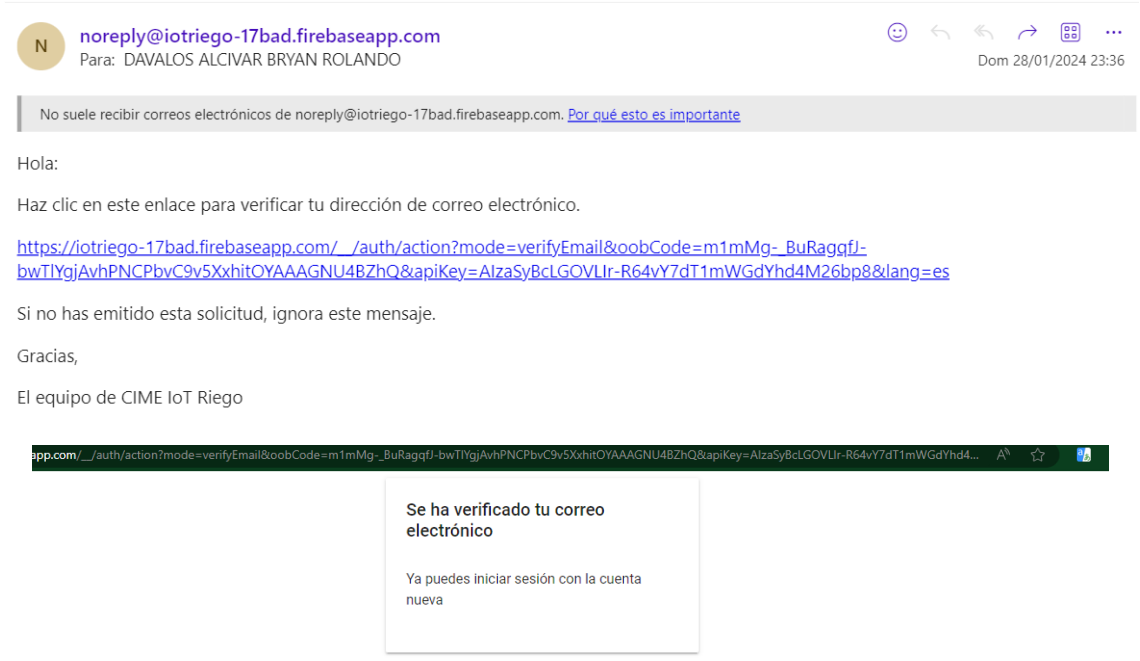


Figura 4.7: Verificación de correo del usuario de la aplicación

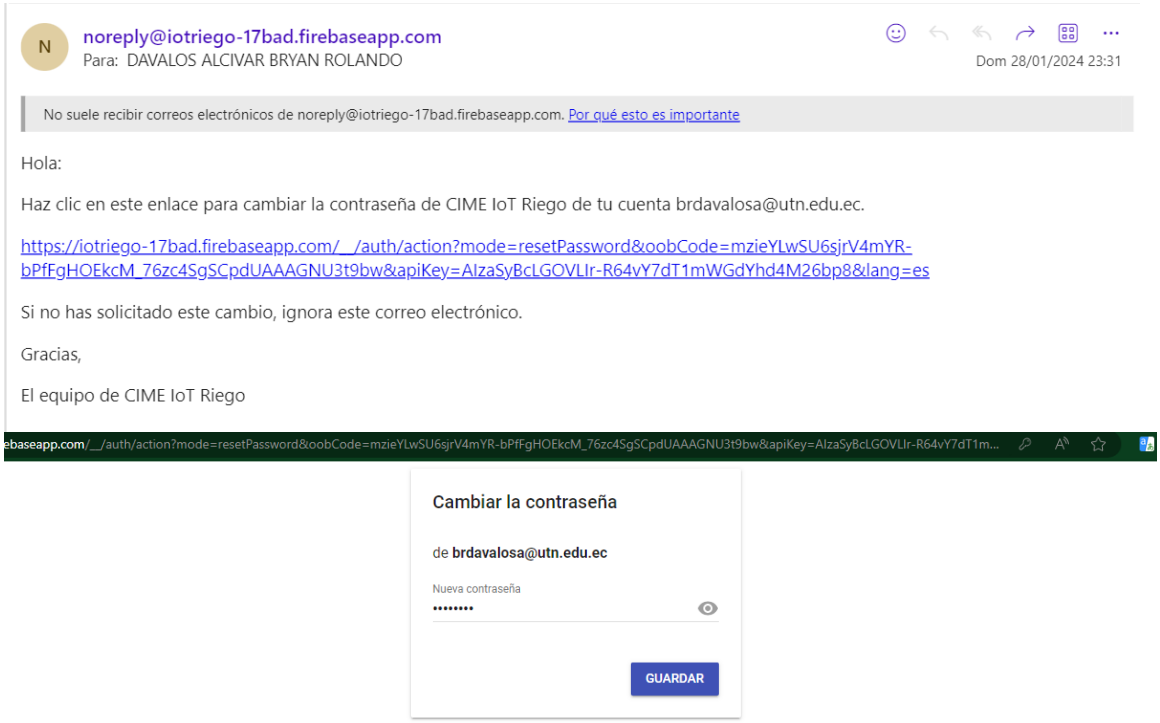


Figura 4.8: Restablecimiento de contraseña en la aplicación

4.1.2.3 Realtime Database

El funcionamiento de Realtime Database sigue el mismo principio y estructura que el servicio anterior. El SDK proporciona funciones definidas en la documentación del servicio, permitiendo que Firebase se encargue de ejecutar el código que, de otra manera, tendríamos que manejar en el lado del servidor. Estas funciones se encuentran desarrolladas en el archivo realtime.js en la carpeta javascript y son invocadas y utilizadas en toda la aplicación.

realtime.js

Las funciones devuelven una promesa con los datos de la referencia especificada, asegurando que no se devuelva un valor hasta que la función se complete y manejando errores en caso de que la ejecución no sea exitosa.

La obtención de valores inmediatos se realiza mediante la función get().

```
export function realtimeGet(reference = '/') {
    return new Promise((resolve, reject) => {

// La referencia (ref) o ubicación del nodo en la base de datos
        let Elements_ref = ref(database, reference);

//Función get()
        get(Elements_ref).then((snapshot) => {
            if (snapshot.exists()) {
                const data = snapshot.val();
                resolve(data);
```

```
            } else {
                console.log(`No data available realtimeGet('${reference}')`);
            }

        }).catch((error) => {
            console.error(`The read failed, method realtimeGet('${refer-
ence}')`, error.name);
            reject(error);
        });
    });
}
```

La función onValue() del SDK se utiliza para ejecutarse cada vez que se producen cambios en la ruta especificada, devolviendo el evento con la información correspondiente.

En esta función, se proporciona una función de retorno callback que onValue() ejecutará cada vez que haya un cambio en la ruta especificada, tomando el valor del cambio como parámetro para la función callback. Esto se utiliza para actualizar valores en indicadores, tablas y widgets en la aplicación.

```
export function realtimeVal(reference = '/', callback) {

    let Elements_ref = ref(database, reference);

    onValue(Elements_ref, (snapshot) => {

        if (snapshot.exists()) {
            const changedPost = snapshot.val();
            callback(changedPost)
        } else {
            console.log(`No data available realtimeVal('${reference}')`);
        }

    }, (error) => {
        console.error(`The read failed realtimeVal('${reference}')` + er-
ror.name);
        reject(error);
    });
}
```

Se ha creado una función que modifica el valor del nodo en la ruta especificada, junto con los nodos secundarios de esa rama. Esta función posibilita la actualización del contenido en la base de datos en tiempo real, la creación de nueva información o la inserción de cambios por parte de los usuarios.

```
export function write(reference = '/', content) {
    let Elements_ref = ref(database, reference)

    set(Elements_ref, content)
        .catch((error) => {
            console.error(`The write failed in ${Elements_ref}`, error);
        });
}
```

4.1.2.4 Cloud Functions

En Cloud Functions, hay cambios en el sentido de que el programa se desplegará y ejecutará directamente en la nube, con las consideraciones y limitaciones que esto implica. En primer lugar, es necesario importar todos los módulos requeridos de Cloud Functions y del SDK Firebase Admin. La aplicación se inicializa mediante el SDK Admin, de manera similar a lo que se hace en la aplicación cliente, para obtener compatibilidad, y utilizar todos los servicios del proyecto, como Authentication, FCM y Realtime Database.

Cloud Functions permite cargar funciones escritas en Python, JavaScript o TypeScript. Al optar por JavaScript, se utiliza Node.js para la importación de los módulos, lo que facilita el desarrollo de funciones. Las funciones se cargan desde un archivo index.js en la carpeta functions.

index.js

```
//Dependencias de Firebase requeridas
const { onCall, onRequest, HttpsError } = require("firebase-func-
tions/v2/https");
const { onValueUpdated, onValueWritten } = require("firebase-functions/v2/da-
tabase");
const { initializeApp, applicationDefault } = require('firebase-admin/app');
const { getDatabase } = require("firebase-admin/database");
const { getMessaging } = require("firebase-admin/messaging");
const { getFunctions } = require("firebase-admin/functions");

// Inicializa Firebase, después de inicializar y llamar a otras instancias
del SDK de Firebase
const firebaseConfig = {
    apiKey: "AIzaSyBcLGOVLIr-R64vY7dT1mWGdYhd4M26bp8",
    authDomain: "iotriego-17bad.firebaseapp.com",
    databaseURL: "https://iotriego-17bad-default-rtdb.firebaseio.com",
    projectId: "iotriego-17bad",
    storageBucket: "iotriego-17bad.appspot.com",
    messagingSenderId: "842817138478",
};
const app = initializeApp(firebaseConfig);
const functions = getFunctions(app);
const database = getDatabase(app);
const messaging = getMessaging(app);
```

Las funciones implementadas en las acciones definidas en respuesta a condiciones establecidas por el usuario en la aplicación son del tipo onCall(). Functions creará una instancia y ejecutará la función cada vez que esta sea llamada. La función se ejecuta con los parámetros data y context, que obtienen los datos ingresados con la llamada a la función y el contexto.

Los servicios de la aplicación han sido inicializados con Admin, por lo tanto, se pueden utilizar las herramientas del SDK. En las funciones que siguen los cambios en la base de datos y comparan los resultados para realizar acciones, emplean una lógica similar a la expuesta en el microservicio de Realtime Database, pero al utilizar Admin, la sintaxis es un poco diferente.

Función para enviar notificaciones en base a condiciones en Realtime Database:

```
exports.realtimeNotification = onCall((data, context) => {
    // Obtinene los parámetros de la llamada a la función
    const uid = data.auth.uid;
    const msg = data.data.msg;

        // Lee los datos en la ruta especificada
        const listener = ref.on('value', function (snapshot) {

            let actVal = snapshot.val()
            let conditionMet = false;

            // Si se cumple la condición, envía la notificación
            if (conditionMet) {
                ref.off('value', listener);

                        const message = {
                            token,
                            notification: {
                                title: 'IoT riego functions'
                                 body: msg.toString();
                            }
                         };

                         // Función para enviar mensajes con FCM
                         messaging.send(message).then((response) => {
                                console.log('Successfully', response);
                            }).catch((error) => {
                                console.log('error: ', error);
                            })
                // Envía la notificación a todos los dispositivos
                return `realtimeNotification: finished,user: ${uid}`;
            }
        });
});
```

La función se comporta como una función regular de JavaScript y se ejecuta de manera asíncrona, por lo que se llama a través de promesas, asegurando la ejecución hasta que se devuelva un valor de la función.

En Cloud Functions, se escriben funciones que se ejecutan frente a eventos y no solo en respuesta a llamadas HTTP. Se desarrollaron funciones con activadores de Realtime Database, como onValueUpdated(), que se activa cuando se actualizan valores en la dirección de la base de datos definida. Estas son las funciones del riego automático que reacciona frente a los cambios en el nodo “modo”, que se refiere al modo de riego, y frente a los valores de las configuraciones para el riego en “settings”.

Se implementó una función que se ejecuta cada vez que se agrega un nuevo usuario para enviar un correo de bienvenida y subscribirlos al “topic” general de FCM de la aplicación para el envío de notificaciones a todos los usuarios.

```
exports.subscribeTopicFCM = onValueUpdated('/CultivoDispFinal/users', (event)
=> {

    const afterUsers = event.data.after.val()
    // Tokens de registro de clientes del SDK de FCM
    const registrationTokens = [];

    // Itera sobre cada hijo de afterUser
    for (let userId in afterUsers) {
        const user = afterUsers[userId];
        const token = user.fcmtoken; // Obtiene el valor de fcmtoken
        registrationTokens.push(token); // Agrega el token al arreglo
    }

// Subscribe los dispositivos de los tokens correspondientes al topic de FCM.
    messaging.subscribeToTopic(registrationTokens, 'general')
        .then((response) => {
            console.log('Successfully subscribed:', response);
        })
        .catch((error) => {
            console.log('Error subscribing to topic:', error);
        });
});
```

Para llamar funciones desde la aplicación cliente en el script de `functions.js` se importan los módulos correspondientes.

```
import { httpsCallable, connectFunctionsEmulator } from "firebase/functions";
```

Las funciones son llamadas desde cualquier parte de la aplicación de acuerdo con las necesidades del proyecto, en este caso se llama seleccionando desde la sección de Control en la aplicación, eso dispara la función la cual toma parámetros ingresados por el usuario y la función devuelve un mensaje o resultado de la ejecución.

```
functionNotification.addEventListener('click', () => {
    console.log('holamessaginF');
    const realtimeNotification = httpsCallable(functions, 'realtimeNotifica-
tion');

    realtimeNotification({
        compareRef: `${nameProyect}/data/tempAire`,
        target: 22,
        msg: 'Este es un mensaje de prueba',
        operation: '=='
    })
        .then((result) => {
            // Read result of the Cloud Function
            const data = result.data
        })
        .catch((error) => {
            const code = error.code;
        });
});
```

Configuración para la ejecución de las funciones en el emulador de Functions para las pruebas y proceso de desarrollo previo al deploy en Firebase Cloud Funcitions.

```
if (location.hostname === 'localhost') {
    connectFunctionsEmulator(functions, 'localhost', 5001);
}
```

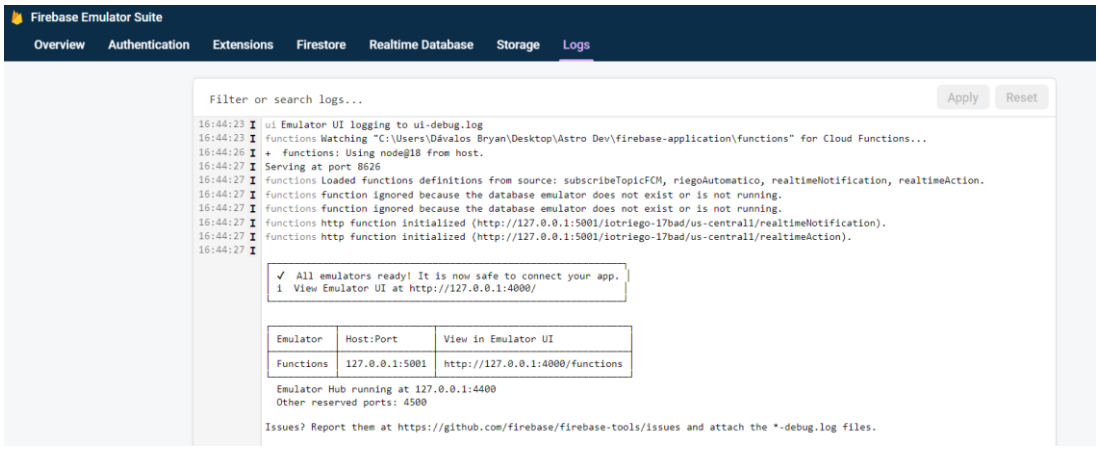


Figura 4.9: Interfaz del emulador de Cloud Functions

Ejecución de las funciones en el emulador de Cloud Functions.

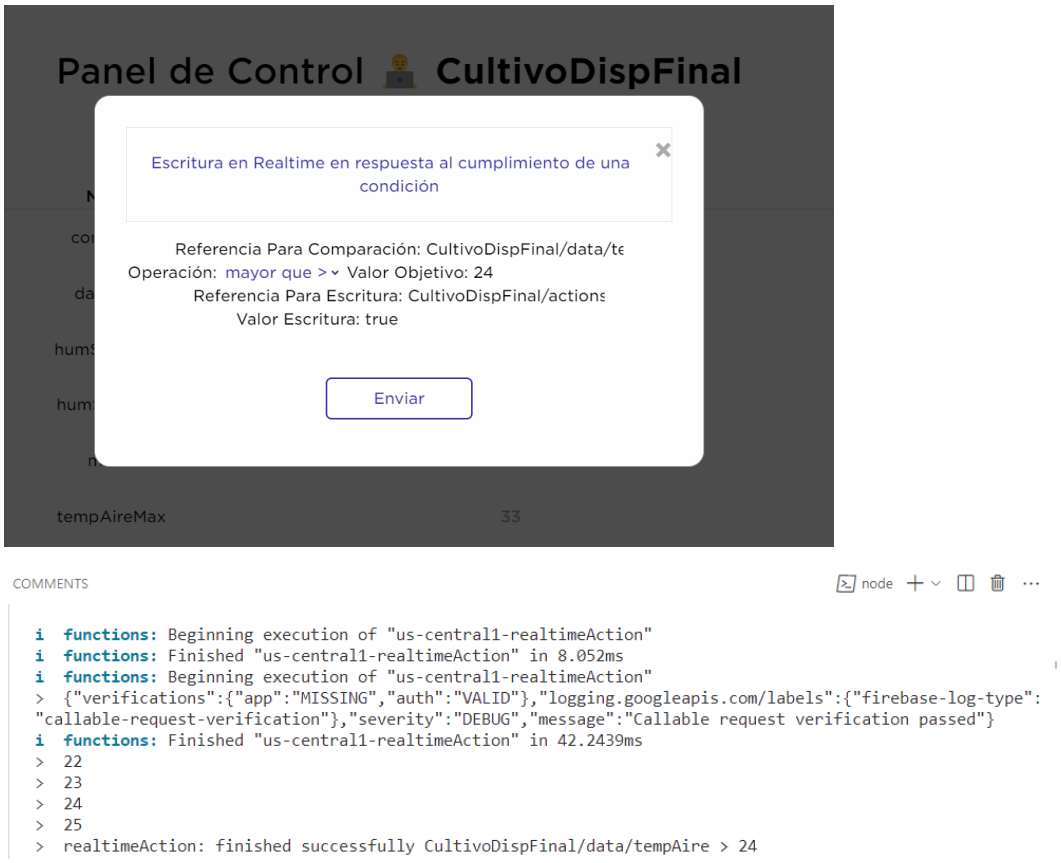
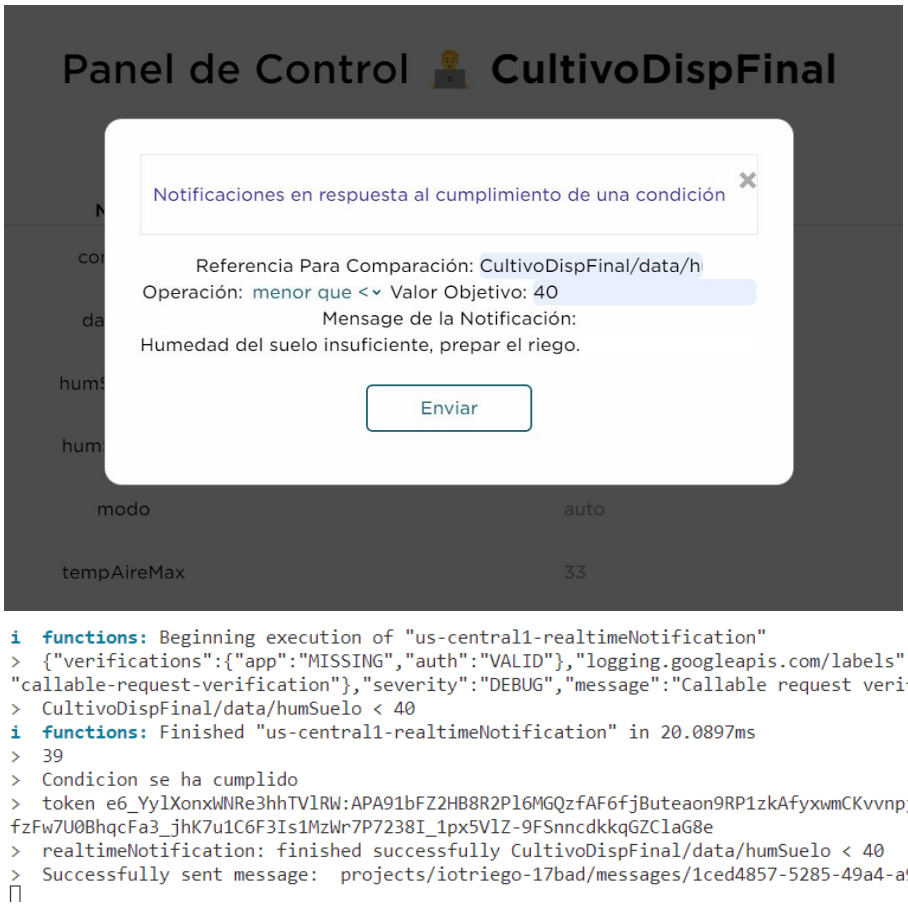


Figura 4.10: Llamada a la función para escritura en Realtime Database frente a eventos




```
i  functions: Beginning execution of "us-central1-realtimeNotification"
>  {"verifications":{"app":"MISSING","auth":"VALID"},"logging.googleapis.com/labels":{"firebase-log-type":
"callable-request-verification"},"severity":"DEBUG","message":"Callable request verification passed"}
>  CultivoDispFinal/data/humSuelo < 40
i  functions: Finished "us-central1-realtimeNotification" in 20.0897ms
>  39
>  Condicion se ha cumplido
>  token e6_YylXonxWNRe3hhTVlRW:APA91bFZ2HB8R2Pl6MGQzfAF6fjButeaon9RP1zkAfyxwmCKvvnpjIt12CmmSqnxjWV4U9HDy
fzFw7U0BhqcFa3_jhK7u1C6F3Is1MzWr7P7238I_1px5VlZ-9FSnncdkkqGZClaG8e
>  realtimeNotification: finished successfully CultivoDispFinal/data/humSuelo < 40
>  Successfully sent message:  projects/iotriego-17bad/messages/1ced4857-5285-49a4-a96b-59ccc9019282
```

Figura 4.11: Llamada a la función de notificación en base a eventos en Realtime Database

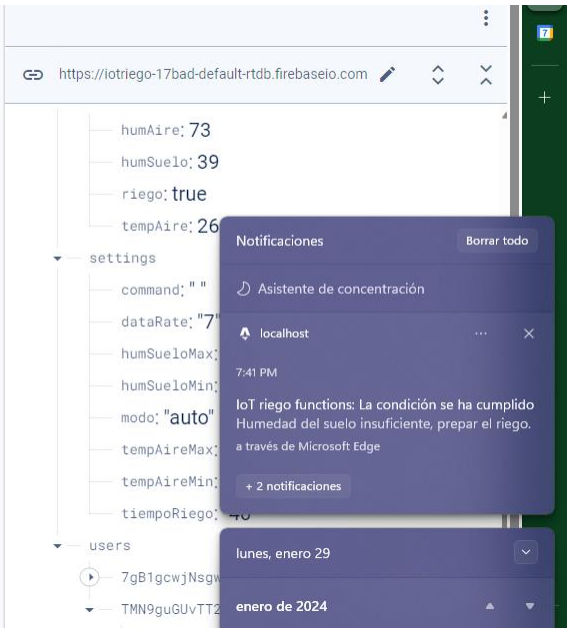


Figura 4.12: Notificación de la aplicación en el sistema.

4.1.2.5 Firebase Cloud Messaging

Como en todos los servicios de la aplicación cliente se inicializa la instancia específica para Cloud Messaging en la aplicación llamando a getMessaging(). Como se había mencionado el servicio de mensagería de Firebase requiere que se registre la aplicación o servidor de aplicaciones con las credenciales “VAPID”. Para suscribir nuestra aplicación a este servicio y recibir notificaciones automáticas, obtenemos este par de claves desde nuestra consola de proyecto de Firebase.

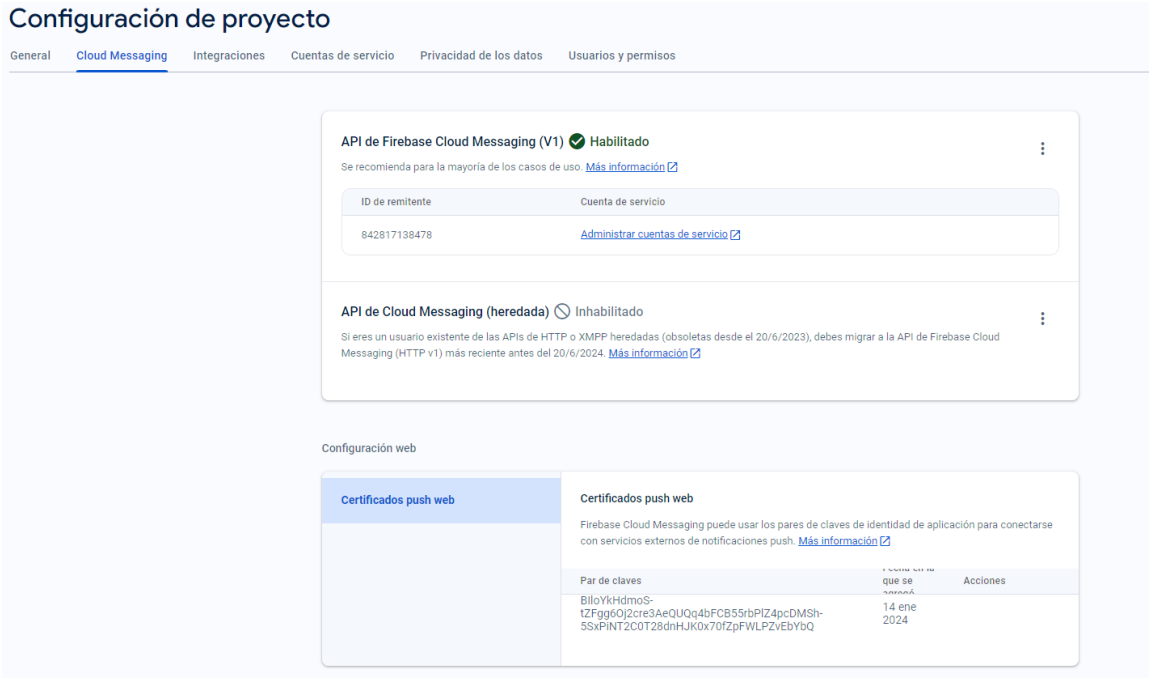


Figura 4.13: Par de claves obtenidas desde la consola de Firebase

El par de claves se usará en el archivo messaging.js donde se escriben las funciones que se ejecutaran de lado del cliente de la aplicación. Se importan además los siguientes módulos

messaging.js

```
import { getToken, onMessage } from 'firebase/messaging';
import { messaging } from "./firebase";
import { write } from "./realtime"

const VAPID_KEY = 'BIloYkHdmoS-tZFgg6Oj2cre3AeQUQq4bFCB55rbPlZ4pcDMSh-
5SxPiNT2C0T28dnHJK0x70fZpFWLPZvEbYbQ'
```

Función para habilitar las notificaciones de la aplicación en el navegador del usuario.

```
async function requestNotificationsPermissions(uid) {
    console.log('Requesting notifications permission...');
    const permission = await Notification.requestPermission();

    if (permission === 'granted') {
        console.log('Notification permission granted.');
```

```
        // Permiso concedido.
        await saveMessagingDeviceToken(uid);
    } else {
        console.log('Unable to get permission to notify.');
    }
}
```

Cuando se haya habilitado las notificaciones en el navegador, se obtendrá el token de dispositivo el cual se utiliza para identificar y enviar notificaciones a ese navegador en particular. Para guardar los token de dispositivo se utiliza la función saveMessagingDeviceToken() en la que se obtiene el token y se lo guarda en Realtime Database bajo el "uid" de usuario respectivo.

```
// Guarda el token de FCM del dispositivo en Realtime Database.
export async function saveMessagingDeviceToken(uid) {

    try {
        const msg = await messaging();
        const fcmToken = await getToken(msg, { vapidKey: VAPID_KEY });

        if (fcmToken) {
            // Guarda el token debajo del uid de usuario correspondiente
            write(`CultivoDispFinal/users/${uid}/fcmToken`, fcmToken);

           // Esto se activará cuando se reciba un mensaje mientras la apli-
cación está en primer plano.

            onMessage(msg, (message) => {
    console.log('Received foreground message ', message.notification);

                new Notification(message.notification.title, { body: mes-
sage.notification.body });
            });
        } else {
            // Si el token no existe llama a la función:
            requestNotificationsPermissions(uid);
        }
    } catch (error) {
        console.error('Unable to get messaging token.', error);
    };
}
```

La función saveMessagingDeviceToken( ) será llamada desde donde sea conveniente para en la aplicación. En este caso las notificaciones deben estar disponibles solo para usuarios autenticados y no responden a una sola acción en específico de la aplicación más bien siempre es necesario para los propósitos del sistema que los usuarios puedan recibir notificaciones y el token se mantenga actualizado por ello se llama a la función desde el observador del estado de autenticación onAuthStateChanged(), así se ejecutará para cada usuario cada vez que inicie sesión y se guardará el token debajo de su uid respectivo.

Llamada de prueba

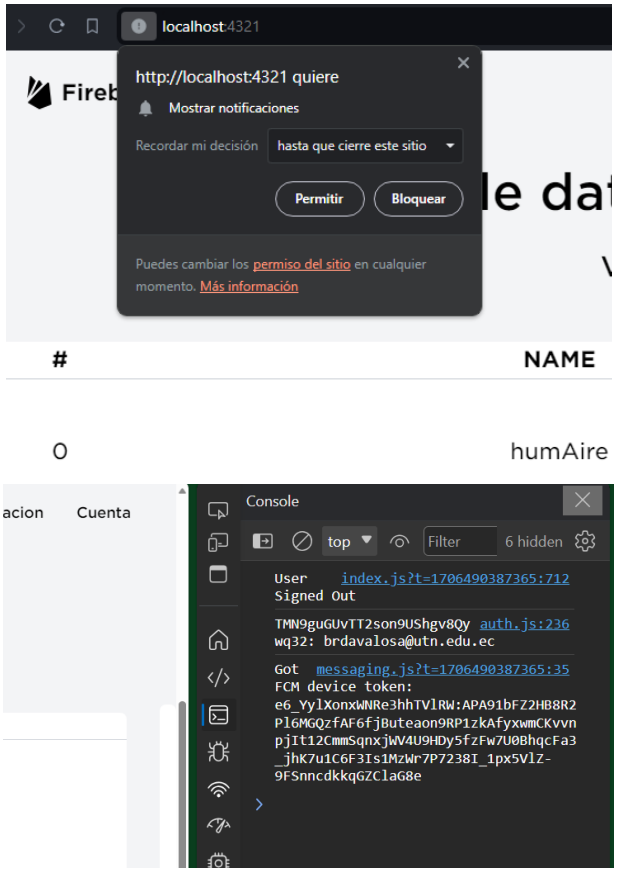


Figura 4.14: Permisos de notificación y generación del token FCM en la aplicación

Registro del token de FCM bajo el uid de usuario respectivo en Realtime Database

Figura 4.15: Nodo users en Realtime Database.

Firebase-messaging-sw.js

Este archivo en la carpeta pública es el trabajador del servicio (service worker), cuando la aplicación este en segundo plano o no está abierta firebase-messaging-sw.js recibirá los mensajes. Se importan los módulos necesarios, se inicializa la aplicación y la instancia de Cloud Messaging como se ha realizado en la aplicación anteriormente.

```
//Dependencias requeridas para este programa
importScripts('https://www.gstatic.com/firebasejs/8.2.0/firebase-app.js');
importScripts('https://www.gstatic.com/firebasejs/8.2.0/firebase-mes-
saging.js');

const firebaseConfig = {
    apiKey: "AIzaSyBcLGOVLIr-R64vY7dT1mWGdYhd4M26bp8",
    authDomain: "iotriego-17bad.firebaseapp.com",
    databaseURL: "https://iotriego-17bad-default-rtdb.firebaseio.com",
    projectId: "iotriego-17bad",
    storageBucket: "iotriego-17bad.appspot.com",
    messagingSenderId: "842817138478",
    appId: "1:842817138478:web:3cc5b91a6b3a0196bb94bf"
};

firebase.initializeApp(firebaseConfig);

// Inicializa Firebase Cloud Messaging
const messaging = firebase.messaging();


messaging.onBackgroundMessage(payload => {
    console.log('Received background message ', payload);

    const notificationTitle = payload.notification.title;
    const notificationOptions = {
        body: payload.notification.body,
    };

    self.registration.showNotification(notificationTitle, notification-
Options);
});
```

4.1.2.6 Aplicación web estática

Este es un fragmento del index.astro con todo el contenido de la página que se importan como componentes .astro que son bloques que pueden contener documentos o colección de etiquetas HTML como <scripts> y <style> propio para la lógica y estilización del componente.

El contenido entre las (---) es el frontmatter, el cual es el contenido que se ejecuta en el servidor para la construcción de la página, se escriben funciones como realtimeGet para obtener el título del proyecto, el cual se ingresa como parámetro en los componentes de las secciones abajo.

Index.astro

```
---
import Layout from "../layouts/Layout.astro";
import LandingHeader from "../components/LandingHeader.astro";
import HomeSection from "../components/HomeSection.astro";
import PanelSection from "../components/PanelSection.astro";
import ActionsSection from "../components/ActionsSection.astro";
import ControlSection from "../components/ControlSection.astro";
import { realtimeGet } from "../javascript/realtime";
let proyect = await realtimeGet();
const nameProyect = Object.keys(proyect)[0];
---
```

Este es el contenido del index.astro, de estructura y sintaxis similar a html, el cual contiene todo el contenido de la aplicación web dividido por secciones que conformar aplicación. La sección de Panel de Datos (PanelSecition) es donde el usuario tiene acceso al monitoreo de la información, puede activar acciones de visualización configurables, las cuales se guardan y son exclusivas para la sesión de cada usuario.

```
<Layout title={`Apliacación del sistema de riego inteligente ${namePro-
yect}`}>
    <LandingHeader nameProyect={nameProyect} />
    <main
        class="snap-y snap-mandatory relative w-full h-screen overflow-auto
scroll-m-0"
    >
        <div class="snap-center">
            <HomeSection />
        </div>
        <div class="snap-center">
            <PanelSection nameProyect={nameProyect} />
        </div>
        <div id="actions" class="snap-center" style="display:none">
            <ActionsSection nameProyect={nameProyect} />
        </div>
        <div id="control" class="snap-center" style="display:none">
            <ControlSection nameProyect={nameProyect} />
        </div>

       </main>
</Layout>
<script src="../javascript/index.js"></script>
```

Para los elementos de visualización se utiliza la biblioteca de Canvas Gauges HTML con la cual se generan los widgets de medidor para representar las variables de la tabla de datos.

```
const data = {
        renderTo: renderTo,
        title: name,
        minValue: min,
        maxValue: max  };

// Asignamos las propiedades del objeto data, con el cual se crea un nuevo
elemento medidor (gauge).
    var newGauge = new RadialGauge(data).draw();

// Función para actualizar el valor representado en el medidor
function updateObject(changedValue) {
    try {
         newGauge.value = changedValue;
    } catch (error) {
          console.error(`failed updateObject('${newGauge.title}')`, error);
      }
}
// Función de realtime.js para obtener valores en respuesta a cambios.
realtimeVal(`${nameProyect}/data/${element}`, updateObject)
```

```
// Si se crea un elemento indicador en la sesión de un usuario, las configu-
raciones se mantienen guardadas bajo el uid de usuario en Realtime Database

    if (auth.currentUser) {
        const userUid = document.getElementById("user-uid").innerText
        write(`${nameProyect}/users/${userUid}/${source}`, data)
    }
```

4.1.2.7 Hosting

Para el lanzamiento de proyectos utilizando Firebase Hosting, se siguen los siguientes pasos según la consola de Firebase:

1. **Iniciar sesión:** Es necesario iniciar sesión con la cuenta con la que se creó el proyecto en Firebase.
2. **Inicializar el proyecto:** Mediante el comando firebase init en la terminal, se inicializa el proyecto de Firebase.
3. **Despliegue en Hosting:** Si ya se cuenta con el proyecto iniciado, se procede a desplegar la página en Hosting utilizando el comando firebase deploy --only hosting.

Este proceso garantiza que la aplicación o sitio web se publique y esté disponible para su acceso a través de Firebase Hosting.

```
PS C:\Users\Dávalos Bryan\Desktop\Astro Dev\firebase-application>  firebase login
Already logged in as brdavalosa@gmail.com

PS C:\Users\Dávalos Bryan\Desktop\Astro Dev\firebase-application> firebase deploy --only hosting

=== Deploying to 'iotriego-17bad'...

i  deploying hosting
i  hosting[iotriego-17bad]: beginning deploy...
i  hosting[iotriego-17bad]: found 15 files in dist
+  hosting[iotriego-17bad]: file upload complete
i  hosting[iotriego-17bad]: finalizing version...
+  hosting[iotriego-17bad]: version finalized
i  hosting[iotriego-17bad]: releasing new version...
+  hosting[iotriego-17bad]: release complete

+  Deploy complete!
```

Figura 4.16: Despliegue de la aplicación web en Firebase Hosting

Aplicación desplegada en Hosting.

La aplicación web posibilita la visualización y recopilación de datos, los cuales son recuperados en función de los cambios en el sistema y almacenados en la base de datos en tiempo real. La gestión de la aplicación se simplifica mediante una interfaz de usuario que incorpora elementos visuales para facilitar la monitorización.

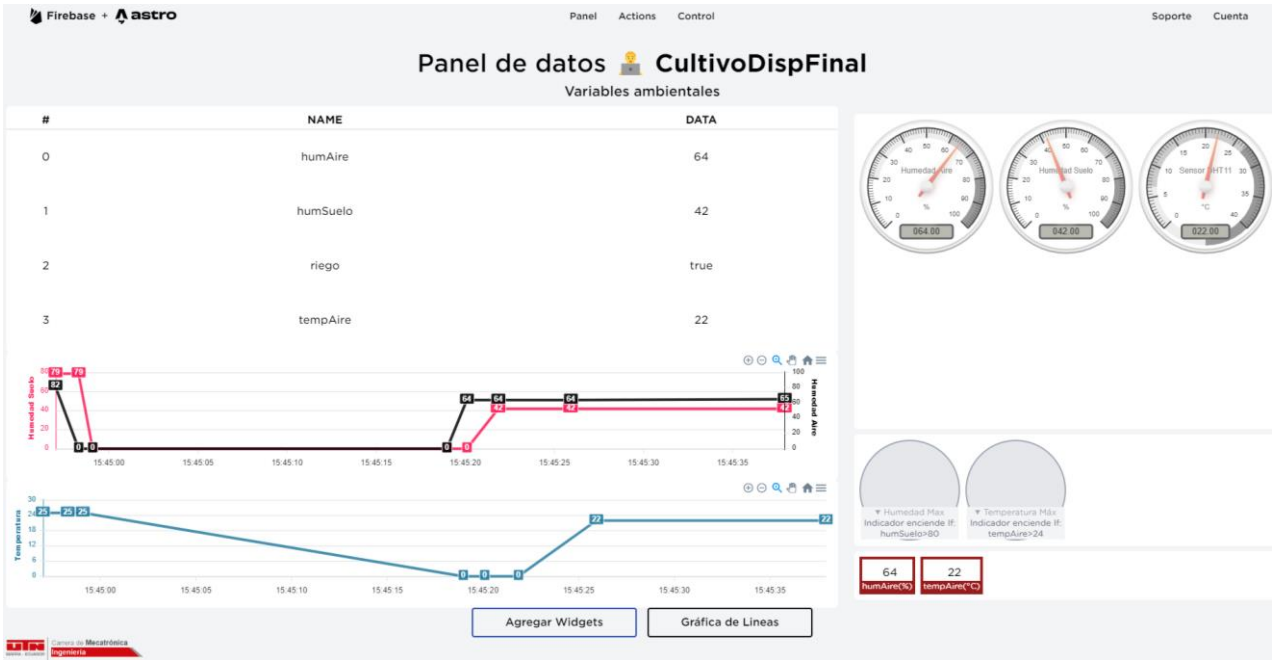


Figura 4.17: Sección de Monitoreo de Variables de la aplicación web.

En la sección de acciones de la aplicación, se ofrecen opciones para iniciar o detener el monitoreo de datos, activar el riego de manera manual, y realizar operaciones de lectura y escritura en la base de datos SQLite.

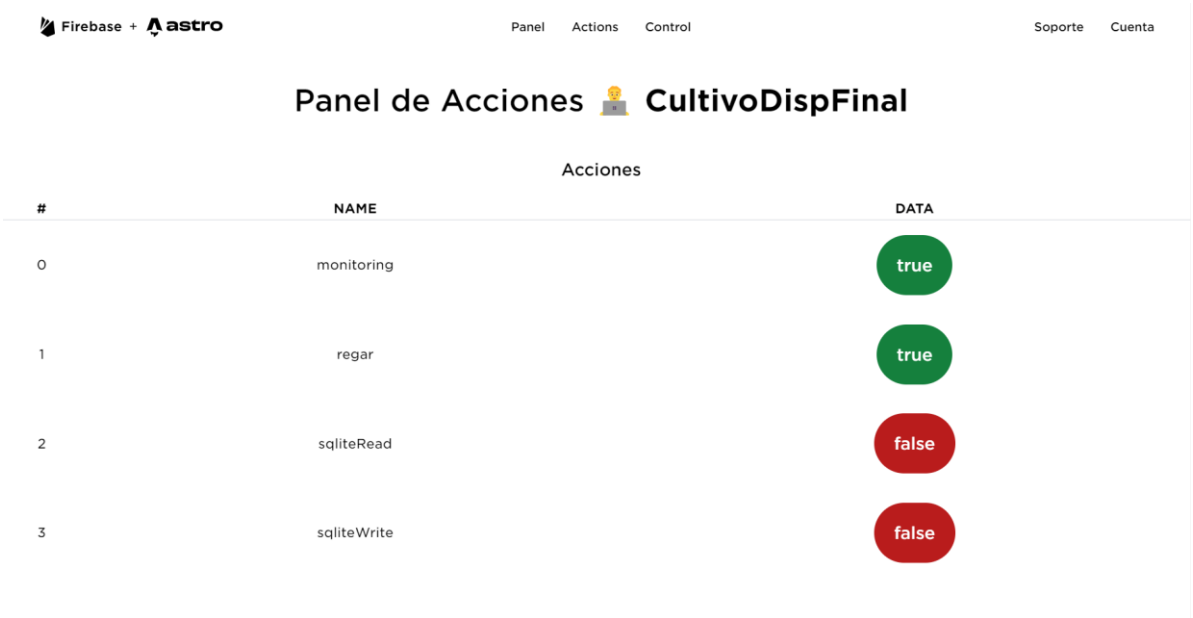


Figura 4.18: Sección de acciones de la aplicación web.

La sección de control permite un manejo ligero del sistema, ya sea a través de acciones directas en la interfaz de usuario o mediante condiciones que generan acciones específicas.

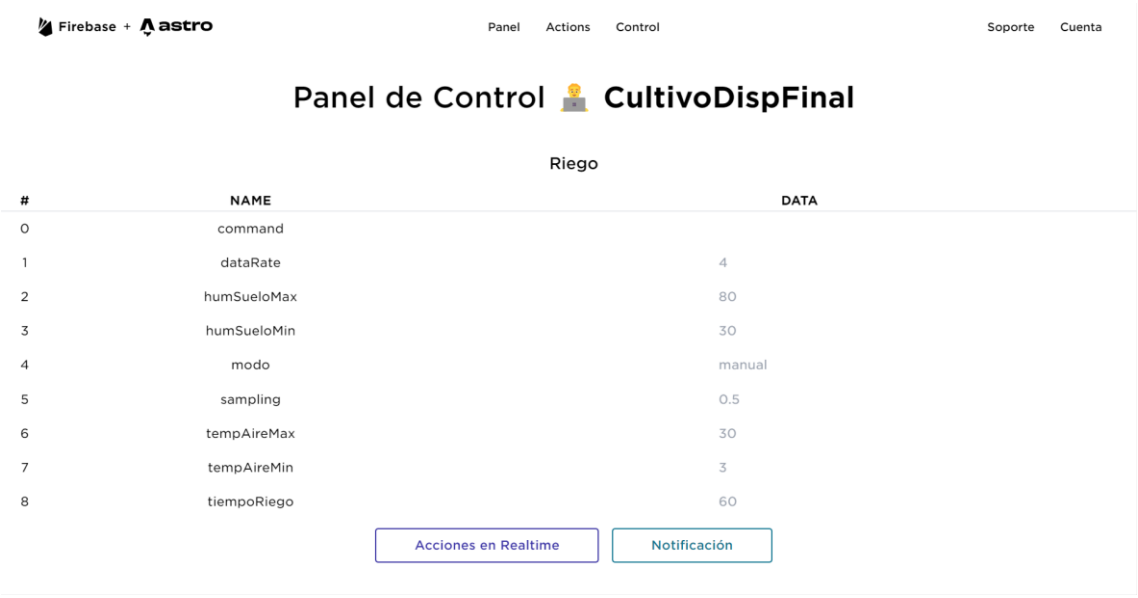


Figura 4.19: Sección de configuración y acciones de control de la aplicación.

4.1.3 Almacenamiento en la base de datos SQLite

Se realiza la conexión con la base de datos en tiempo real de la misma manera que se realizó en el programa del microcontralor. Se realiza la comunicación con Firebase Realtime Database mediante la biblioteca Admin SDK con las credenciales como cuenta de servicio.

Después de realizar se comprueba la existencia de la base de datos de no existir se inicia una nueva base de datos

```
if os.path.isfile(dataBaseName) == False
        initDB()  # si no existe, se la crea
        print("Un nuevo archivo de base de datos ha sido creado")
    else:
        print("Un archivo de base de datos ha sido encontrado")
```

Función para crear la base de datos con el nombre determinado.

```
def initDB():
    conn = sqlite3.connect(dataBaseName)
    c = conn.cursor()
    c.execute('''CREATE TABLE History (ID integer primary key, EstampaTiempo
real, \HumedadSuelo real, HumedadAire real, TemperaturaAire real, Riego
real)''')
    conn.commit()  # save changes
    conn.close()
```

El servidor de la base de datos SQLite se mantendrá siempre en ejecución, sin embargo, en el programa se obtiene la lectura y almacenamiento de los datos siempre y cuando la instrucción

sqliteWrite sea igual a True en la base de datos en tiempo real. Cuando se cumple esta condición la base de datos escucha y guarda los cambios en el nodo data de la base de datos en tiempo real.

```
 while bool(fb.child('actions/sqliteWrite').get()):
        listener = fb.child('data').listen(dataChanges)
    listener.close()
```

La función listen del módulo db llama a la función callback dataChanges() con un objeto db.Event con los valores del cambio en la base de datos, esos valores se ingresan a la función addDataDB().

```
def dataChanges(event):
    global humedadSuelo, humedadAire, temperaturaAire, riego

    try:
        humedadSuelo = fb.child('data/humSuelo').get()
        humedadAire = fb.child('data/humAire').get()
        temperaturaAire = fb.child('data/tempAire').get()
        riego = fb.child('data/riego').get()
        print(f"{readTime(0)}cambios agregados a la base de datos:",
event.data)
        addDataDB()

    except Exception as e:
        print(e)
        print("Error al cargar datos del listen() la base de datos")
```

La función addDataDB() inserta una nueva fila en la tabla History con la hora actual, la humedad del suelo, la humedad del aire, la temperatura del aire y el estado del riego.

```
def addDataDB():
    global humedadSuelo, humedadAire, temperaturaAire, riego
    conn = sqlite3.connect(dataBaseName)
    c = conn.cursor()
    c.execute('''INSERT INTO History
        (EstampaTiempo, HumedadSuelo, HumedadAire, TemperaturaAire, Riego)
        VALUES (?, ?, ?, ?, ?)''', (readTime(0), humedadSuelo, humedadAire,
temperaturaAire, riego))
    conn.commit()
    conn.close()
```

```
"C:/Users/Dávalos Bryan/AppData/Local/Programs/Python/Python311/python.exe" "c:/Users/
Dávalos Bryan/OneDrive - Universidad Tecnica del Norte/Tesis App/Tesis/DataBaseML.py"
Un archivo de base de datos ha sido encontrado
Listen Realtime Database Proyect Data
23 Jan 22:08 - cambios agregados a la base de datos: {'humAire': 64, 'humSuelo': 42, '
riego': True, 'tempAire': 22}
23 Jan 22:08 - cambios agregados a la base de datos: {'humAire': 65}
23 Jan 22:08 - cambios agregados a la base de datos: {'humAire': 64}
23 Jan 22:08 - cambios agregados a la base de datos: {'humAire': 65}
Finish listening Realtime Database Proyect Data
Esperando instrucciones de firebase
```

Figura 4.20: Registro de datos del sistema de riego en la Entidad History en SQLite.

	ID	EstampaTiempo	HumedadSuelo	HumedadAire	TemperaturaAire	Riego
	Filter	Filter	Filter	Filter	Filter	Filter
1	1	1706128475.91735	42.0	64.0	22.0	1.0
2	2	1706128496.26941	42.0	65.0	22.0	1.0
3	3	1706128510.6472	42.0	64.0	22.0	1.0
4	4	1706128531.5218	42.0	65.0	22.0	1.0
5	5	1706128538.25445	42.0	64.0	22.0	1.0
6	6	1706128545.30974	42.0	65.0	22.0	1.0
7	7	1706128559.31456	42.0	64.0	22.0	1.0
8	8	1706128573.31821	42.0	65.0	22.0	1.0
9	9	1706128580.28221	42.0	64.0	22.0	1.0
10	10	1706128588.25929	42.0	65.0	22.0	1.0

Figura 4.21: Estructura de la base de datos de SQLite

El programa también permite la descarga del total de los registros de la base de datos mediante la instrucción sqliteRead de la base de datos en tiempo real.

```
if bool(fb.child('actions/sqliteRead').get()):
        fb.child('actions/sqliteRead').update(False)
        url = uploadDB(dataBaseName, f'./{dataBaseName}')
        print(url)
        print("se ha exportado la base de datos SQlite en firebase")
```

La base de datos se carga por completo en Firebase Storage del proyecto, la función para la carga de la base de datos devuelve un url desde el cual podemos descargar de la nube el archivo subido, en este caso la base de datos del sistema.

```
def uploadDB(blobName, filePath):
    try:
        bucket = storage.bucket(f'{bucketURL}')
        Objeto = bucket.blob(blobName)
        Objeto.upload_from_filename(filePath)
        print(f"File {filePath} uploaded to {blobName}.")

    except Exception as e:
        # muestra del error en la consola
        print("Error al enviar datos a Firebase:", e)

    # Devuelve una URL de descarga pública
    return Objeto.public_url
```

```
PS C:\Users\Dávalos Bryan\OneDrive - Universidad Tecnica del Norte\Tesis App\Tesis>
ython311/python.exe" "c:/Users/Dávalos Bryan/OneDrive - Universidad Tecnica del Nort
Un archivo de base de datos ha sido encontrado
24 Jan 2024 15:20:01File ./baseDatosCultivo.db uploaded to urlbaseDatosCultivo.db.
https://storage.googleapis.com/iotriego-17bad.appspot.com/urlbaseDatosCultivo.db
se ha exportado la base de datos SQlite en firebase
Esperando instrucciones de firebase
```

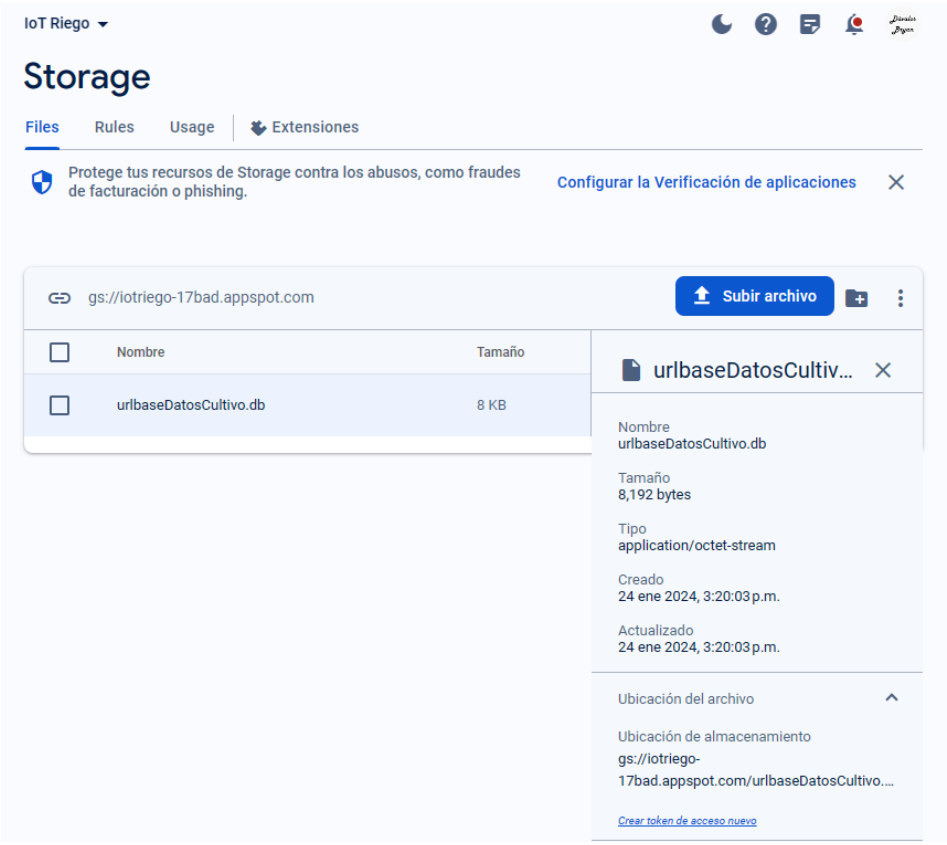


Figura 4.22: Base de datos cargada en la nube de Firebase Storage.

4.2 Pruebas

Después de implementar el sistema, se llevan a cabo pruebas para evaluar su rendimiento y validar su funcionamiento. Las pruebas son fundamentales para evaluar la calidad, en el contexto del software, las pruebas desempeñan un papel crucial en el control y aseguramiento de la calidad, teniendo un impacto directo en la satisfacción del cliente [43].

En el contexto de las pruebas de funcionamiento del software, se distinguen principalmente dos niveles. A nivel bajo, las pruebas de componentes individuales se dividen en pruebas de unidad, que evalúan el funcionamiento de módulos o componentes específicos, y pruebas de integración, que examinan la estructura del programa al combinar los componentes previamente probados [43]. En el nivel alto de las pruebas, se encuentran las pruebas orientadas al producto completo. Dentro de este nivel, destacan las pruebas de sistema, que incluyen pruebas de rendimiento.

Estas pruebas evalúan el comportamiento del sistema en su conjunto, midiendo su rendimiento bajo diversas condiciones y cargas de trabajo [44].

En la sección de implementación del proyecto, se ha demostrado el correcto funcionamiento individual de los módulos. Una vez confirmado que cada componente del sistema opera adecuadamente, se procede con la validación de la comunicación entre módulos para garantizar la cohesión y la interoperabilidad del sistema. La comunicación entre módulos constituye un requisito habilitante para llevar a cabo las pruebas de integración del sistema en su conjunto.

4.2.1 Pruebas de integración del sistema

Se lleva a cabo una prueba de funcionalidad de la aplicación para validar el cumplimiento de los requisitos del sistema. Tras exponer el funcionamiento de cada módulo en la implementación, se procede a evaluar la integración y el correcto funcionamiento de los módulos en conjunto [45].

Estas pruebas se sustentan principalmente en la verificación de la comunicación evaluando la interacción entre ellos, con el propósito de verificar de manera adecuada si el sistema cumple con las expectativas previstas en los requisitos establecidos por el usuario y el sistema.

Prueba 1: Funcionamiento del sistema de autenticación y administración de usuarios en la aplicación, se prueba la integración de Auth con Realtime Database y Cloud Messaging.

Tabla 4.1

Autenticación y administración de usuarios

No.	Objetivo	Procedimiento de la prueba	Resultado esperado	Resultado
1	Autenticación de usuarios	Ingreso de credenciales Usuario y Contraseña	Datos del usuario autenticado. Acceso a acciones y control. En caso de error debería mostrar una alerta.	Aprobado
2	Gestión de Usuarios por el Administrador	Desde la consola de Autenticación: • Agregar usuario. • Restablecer contraseña. • Eliminar Cuenta	Envío de correo de bienvenida. Envío de correo para restablecimiento de contraseña. Eliminar información de cuentas eliminadas	Aprobado
3	Registro de datos por Usuario	Selección y configuración de widgets. Habilitación de notificaciones en el sitio	Almacenamiento de las configuraciones y widgets del usuario. Registro del usuario y el token FCM correspondiente en Realtime Database	Aprobado
4	Interfaz de Usuario para autenticación	Solicitar credenciales de usuario correspondientes.	Devuelve la información de usuario. Habilita las acciones de verificación de correo y restablecimiento de contraseña	Aprobado

Prueba 2: Almacenamiento y gestión de datos de la aplicación entre los programas del sistema de riego, la base de datos SQLite y Realtime Database.

Tabla 4.2

Gestión de la información del sistema

No.	Objetivo	Proceso de la prueba	Resultado esperado	Resultado
1	Adquisición de datos	Ejecución del programa del sistema de riego en python.	Registro de los valores en Realtime Database en la frecuencia determinada en la configuración.	Aprobado
2	Sincronización entre Realtime Database y los dispositivos finales.	Actualizaciones en la base de datos en tiempo real dese los diferentes dispositivos.	Sincronización en todo el sistema de los cambios actualizados en ambas direcciones. Comunicación con baja latencia.	Aprobado
3	Recolección de la información	Ejecución del programa de la base de datos SQLite del servidor. Instrucción para recolectar datos desde Realtime Database	Almacenamiento de las variables ambientales y el estado de riego en el registro historico del proyecto. La información se almacena en respuesta a los cambios en Realtime Database.	Aprobado
4	Recuperación de la información de la base de datos SQLite	Ejecución del programa de la base de datos SQLite del servidor. Instrucción de Realtime Database para leer la base de datos	La base de datos se carga al servicio de Cloud Storage. Obtención de un enlace para la descarga.	Aprobado

Prueba 3: Monitoreo de los datos en la aplicación, ejecución de acciones y control en la nube.

Tabla 4.3

Monitoreo y control del sistema de riego

No.	Objetivo	Proceso de la prueba	Resultado esperado	Resultado
1	Monitoreo del sistema de riego.	Cargar la aplicación web. Navegar en la sección Panel de Datos. Seleccionar y configurar opciones de visualización	Se muestra una interfaz para el monitoreo de los datos de variables ambientales y riego en la Aplicación a través de tablas gráficos líneas e indicadores.	Aprobado

2	Sincronización entre Realtime Database y los dispositivos finales.	Actualizaciones en la base de datos en tiempo real dese los diferentes dispositivos.	Se Realizan acciones hacia la base de datos de tiempo real ingresando valores desde la aplicación, los cambios se distribuyen mediante la base de datos en tiempo real y son ejecutados en el sistema IoT	Aprobado
4	Envío de Notificaciones y Mensajes	Ingresar a la consola del proyecto en Firebase en la sección de Autenticación. Crear una campaña, seleccionar y redactar la notificación. Enviar a usuarios específicos, posterior mente a todos.	Los usuarios reciben la notificación correspondiente en el navegador cuando la aplicación está en primer o en segundo plano. El usuario sin conexión no perderá la notificación el mensaje será enviado en cuanto este vuelva a estar conectado.	Aprobado
5	Ejecución de funciones en la nube	Cargar la aplicación web. En la sección de Control, configurar y llamar a las funciones de Cloud Functions varias veces y de forma simultánea.	Se cumplen todas las llamadas de función sin errores, se respetan los parámetros ingresados. El tiempo de la llamada y del procesamiento de la respuesta va en el orden de los milisegundos (ms).	Aprobado
3	Control automático	Inicio del riego automático y configuración desde la sección Control de la aplicación.	Permite el control del riego automático. Un control tipo On-Off de acuerdo con el cumplimiento de condiciones de humedad y temperatura definidos por el usuario.	Aprobado

4.2.2 Pruebas de rendimiento del sistema

Para cuantificar la calidad del software de manera satisfactoria, nos basamos en el modelo de evaluación ISO/IEC 25010. Este modelo tiene como objetivo proporcionar un sistema para la evaluación de la calidad de productos de software, considerando que un producto de software de calidad resulta de la calidad de sus elementos constituyentes. En este contexto, se establecen características de calidad fundamentales, entre las cuales se incluyen: adecuación funcional, fiabilidad, rendimiento, operabilidad, seguridad, compatibilidad mantenibilidad y transferibilidad [46]. las aplicaciones web pueden ser diversas y tienen una amplia gama de atributos y criterios para evaluar su calidad. Esto puede dificultar la cobertura completa de todos los aspectos relevantes y la normalización de la evaluación de calidad del sistema.

Para este trabajo se evaluará la característica de calidad rendimiento. El rendimiento se destaca como uno de los atributos de calidad más cruciales, según se menciona en [47], basándose en una encuesta realizada por Capgemini, Sogeti y Hewlett Packard (HP). Este aspecto debe ser

prioritario en una evaluación de software, ya que posibilita la medición de su eficiencia al interactuar con los usuarios [48]. En la Tabla 8 se pueden observar las subcaracterísticas de rendimiento y sus respectivas métricas [49].

Tabla 4.4

Características del rendimiento y sus respectivas métricas según ISO/IEC 25010.

Subcaracterísticas de rendimiento	Métricas
Comportamiento en el tiempo	Tiempo de respuesta
	Tiempo de espera
	Rendimiento
Utilización de recursos	Utilización de CPU
	Utilización de memoria
	Utilización de dispositivos de E/S
Capacidad	Número de peticiones en línea
	Número de accesos simultáneos
	Sistemas de transmisión de ancho de banda

Este tipo de prueba se realiza, desde una perspectiva, para determinar lo rápido que realiza una tarea un sistema en condiciones particulares de trabajo. También puede ser utilizada para validar y verificar algunos atributos de calidad, tales como la escalabilidad, fiabilidad y el uso de los recursos [44].

Tiempo de respuesta y tiempo de espera: estos aspectos se consideraron ya que permiten conocer el comportamiento en el tiempo de la aplicación web para realizar una actividad. El tiempo de respuesta se refiere al tiempo que tarda el sistema en responder a una solicitud del usuario y el tiempo de espera es la duración entre iniciar y completar un trabajo [48]. Cuanto menor sea el tiempo de respuesta, mejor será la percepción del usuario en términos de velocidad y eficiencia del software.

4.2.2.1 Condiciones de la evaluación

Tabla 4.5

Características del computador de la prueba de rendimiento

Recursos de computador	
Sistema Operativo	Microsoft Windows 11 Pro
CPU	Intel(R) Core(TM) i5-1035G1 2.19 GHz
GPU	Gráficos Intel UHD
RAM	8 GB 25600 MB/s

DISCO	NVMe - Escritura 1,720 MB/s - Lectura 716 MB/s

Para la métrica de utilización de recursos es importante conocer las características del computador donde se realizan las pruebas. Esto permite bridar un contexto para interpretar los resultados y un conocimiento de las condiciones iniciales antes de las pruebas.

Para las pruebas de comportamiento en el tiempo, se evaluó el tiempo de respuesta en 3 navegadores. Los navegadores y sus versiones se muestran en la tabla a continuación.

Tabla 4.6

Navegadores para pruebas de rendimiento

Navegadores	
Google Chrome	Versión 121.0.6167.140
Microsoft Edge	Versión 121.0.2277.98
Mozilla Firefox	Versión 122.0

4.2.2.2 Comportamiento en el tiempo

En la ejecución de estas pruebas, se emplean las utilidades proporcionadas por Firebase Performance. Performance Monitoring utiliza "seguimientos" para recopilar información sobre los procesos monitorizados en la aplicación. Un seguimiento consiste en un informe que abarca datos capturados entre dos puntos temporales en la aplicación[50].

Tiempo de respuesta

Según los resultados del tiempo de carga de Firebase Performance se observa un tiempo de carga reducido para todos los casos, Estos resultados indican que, en general, los tres navegadores ofrecen tiempos de respuesta que se considerarían satisfactorios para proporcionar una experiencia de usuario fluida.

Tabla 4.7

Tiempo de respuesta de la aplicación web

Navegador	**Tiempo (ms)**
Google Chrome	519
Microsoft Edge	430
Mozilla Firefox	396

Es crucial destacar que el tiempo de respuesta puede verse afectado por diversos factores, como la velocidad de conexión a Internet, las condiciones o el contenido nuevo en la página, así como la capacidad del hardware del dispositivo. Por ejemplo, cuando se carga la página por primera vez en un dispositivo nuevo con recursos limitados, el tiempo de respuesta podría prolongarse a varios segundos.

Firebase Performance Monitoring para el caso de aplicaciones web, genera automáticamente un seguimiento de carga de página para la aplicación. En la tabla 9 se muestra las métricas del seguimiento obtenidos en el panel Rendimiento de la consola del proyecto.

Tabla 4.8

Métricas del seguimiento de carga de la página en Firebase Monitoring.

Seguimientos personalizados	**Tiempo (ms)**
Primer procesamiento de imagen	146
Primer procesamiento con contenido	743
domContentLoadedEventEnd	764
domInterative	678
loadEventend	769

Tiempo de espera

La medición del tiempo de ejecución de actividades se realiza mediante los seguimientos personalizados de Firebase Performance. Estos seguimientos personalizados constituyen una manera de cuantificar el tiempo que la aplicación emplea en completar una tarea específica o un conjunto de tareas, y permiten añadir métricas y atributos personalizados para obtener información adicional. Los datos de estos seguimientos pueden visualizarse en la consola de Firebase, específicamente en la pestaña de Seguimientos personalizados [36].

Para implementar los seguimientos personalizados, se integra el SDK de Firebase Performance en la aplicación, y se agrega código en los puntos clave para definir el inicio y el final de cada seguimiento [36].

Se configuró el seguimiento para las interacciones con la base de datos en tiempo real, abarcando tanto las operaciones de lectura como de escritura, así como el tiempo que lleva cargar datos en las tablas para la visualización en la aplicación.

Tabla 4.9

Tiempo de sincronización con la base de datos en tiempo real

Seguimientos personalizados	Tiempo (ms)
Lectura Realtime Database	0.4
Escritura Realtime Database	1
Carga de datos	3

El tiempo de procesamiento en la realización de las funciones en Cloud Functions se obtuvo del terminal del emulador.

Tabla 4.10

Tiempo de funciones de Cloud Functions

Seguimientos personalizados	Tiempo (ms)
Función Notificación	17-86
Función en Realtime Database	8-64
Función Riego Automático	17-86

4.2.2.3 Utilización de recursos

Utilización de memoria y CPU

A continuación, se presentan los resultados de la utilización de recursos, específicamente la memoria y la CPU, para diferentes navegadores:

Tabla 3.11

Utilización de memoria de la aplicación web

Utilización de memoria (MB)	
Google Chrome	71.53
Microsoft Edge	93.98
Mozilla Firefox	63

La eficiencia en la utilización de memoria y CPU es fundamental para garantizar un rendimiento óptimo en la aplicación de administración de IoT, sobre todo al considerar la expansión e implementación en dispositivos con recursos limitados.

Tabla 4.12

Utilización de CPU de la aplicación web

Utilización de CPU (%)	
Google Chrome	15.6
Microsoft Edge	22.1
Mozilla Firefox	13.7

4.2.2.4 Capacidad

Peticiones en línea

Esta métrica es la medición de peticiones realizadas al servidor para cumplir funcionalidades de la aplicación, estás peticiones son procesadas en un cierto intervalo de tiempo.

Tabla 4.13

Solicitudes en red de las bibliotecas en la aplicación

URL	Respuesta (ms)
*.firebaseio.com/**	377
firebase.googleapis.com/**	660
firebaseremoteconfig.goo-gleapis.com/**	709
firebaseinstallations.goo-gleapis.com/**	786
fcmregistrations.goo-gleapis.com/**	750
identitytoolkit.googleapis.com/**	500
securetoken.googleapis.com/**	603
*.cloudfunctions.net/**	316
apis.google.com/**	345

Esta métrica permite medir cuántas solicitudes en línea pueden procesarse por unidad de tiempo. Se calcula dividiendo el número máximo de peticiones entre el tiempo de la operación lo que da un valor de 443 milisegundos por solicitud lo que es igual a 2.25 solicitudes por segundo, el cual se puede considerar como un valor medio en páginas web [46].

4.3 Discusión

Se ha desarrollado un sistema centralizado y gratuito para la administración de un sistema IoT de riego inteligente, mediante una aplicación de carácter gratuito y de código abierto. La implementación se destaca por utilizar Firebase en todo el proceso. El desarrollo de la aplicación se basa en el uso de herramientas y dependencias de Firebase, centrándose en la creación de microservicios que son escalables y configurables. Esta aproximación en Firebase es adecuada para diversos niveles de experiencia en desarrollo, lo que facilita la replicación, integración y escalabilidad del sistema para proyectos futuros de administración de sistemas IoT.

El componente Realtime Database de Firebase ha demostrado ser una herramienta versátil y potente. Ofrece un conjunto completo de herramientas de desarrollo para realizar operaciones CRUD, con opciones de seguridad altamente configurables. Su integración perfecta con otros servicios de Firebase facilita considerablemente el desarrollo y la gestión de usuarios, fuentes y acceso a los datos.

La aplicación web utiliza diversos servicios de Firebase para monitorear en tiempo real la información del sistema de riego, almacenar datos, supervisar la utilización y enviar notificaciones a los usuarios de manera fiable. Firebase Authentication, Cloud Messaging y Cloud Functions se emplean para garantizar la autenticación, la entrega de mensajes y la ejecución de funciones en la nube. Cloud Functions, mediante la ejecución de funciones en la nube, contribuye a reducir la carga de procesamiento del sistema, en el control del riego automático y otras funciones.

Las pruebas de funcionamiento a nivel bajo, que incluyen la implementación y evaluación de componentes individuales, confirman su correcto funcionamiento. En las pruebas de integración, se asegura el cumplimiento de los objetivos establecidos para el sistema. Posteriormente, en las pruebas de nivel alto del producto final, se evalúa la característica de calidad de rendimiento según el modelo de la norma ISO/IEC 25010. Los resultados indican que las métricas de rendimiento de la aplicación web responden eficientemente, con tiempos de respuesta en el orden de los milisegundos y así mismo con bajos tiempos de espera en el cumplimiento de las funciones. El consumo de memoria y procesamiento se mantiene en niveles aceptables para su ejecución en dispositivos diversos y con recursos limitados. Estos datos permiten concluir que la aplicación web es viable para la administración del sistema de riego inteligente. Además, se destaca que la calidad y rendimiento de la aplicación dependerán del tipo de dispositivo en el que se ejecute, y el uso de diferentes navegadores no representa cambios significativos en el rendimiento.

Capítulo V

Conclusiones, recomendaciones y trabajo a futuro

5.1 Conclusiones

- La utilización de Firebase como plataforma central, El desarrollo de una aplicación web para la gestión de un sistema de riego inteligente, ha sido un proceso fundamental en esta investigación. La integración de diversos servicios de Firebase ha permitido crear una solución completa y escalable para la monitorización y control del sistema. La propuesta consigue mejorar la gestión y accesibilidad de la información generada por sistemas IoT, proporcionando una solución eficiente y amigable para el monitoreo y control de dispositivos conectados.
- La aplicación permite conectar un ordenador para obtener y operar la información de las bases de datos en la nube, cumple con los requisitos funcionales esperados, sino que también ofrece una interfaz de usuario intuitiva y amigable. La capacidad de visualizar y controlar el sistema de riego de forma remota contribuye significativamente a la comodidad y eficacia en la gestión de recursos hídricos.
- Firebase opción adecuada como plataforma central de IoT, ya que ofrece herramientas potentes y versátiles, permite la sincronización, comunicación bidireccional de muchos dispositivos a la vez, la gestión de datos en tiempo real, la autenticación de usuarios, el envío de notificaciones, entre otros. Esto ha facilitado enormemente el desarrollo de la aplicación y su capacidad de escalamiento para futuros proyectos de administración de sistemas IoT.
- Las pruebas de rendimiento realizadas según el modelo ISO/IEC 25010 han arrojado resultados positivos en cuanto a tiempos de respuesta menores a 800 milisegundos, tiempo de espera en el procesamiento de tareas menores a 100 milisegundos y consumo de recursos moderados. Esto demuestra que la aplicación es capaz de ofrecer una experiencia de usuario fluida y eficiente.

5.2 Recomendaciones

- Explorar opciones para optimizar aún más el rendimiento de la aplicación en dispositivos móviles o con conexiones a Internet más lentas. Además, se podría considerar la implementación de nuevas funcionalidades o la expansión a otros tipos de sistemas IoT.
- Explora nuevas funcionalidades y características que enriquezcan la experiencia del usuario de la aplicación. Se podría realizar la integración de sensores adicionales, opciones de personalización avanzada para los usuarios o la implementación de algoritmos más sofisticados para el control del sistema de riego automático.
- Realizar un análisis con colaboración activa con los usuarios finales del rendimiento y usabilidad de la aplicación en el tiempo, recopilar retroalimentación de manera continua puede ser beneficioso para identificar áreas de mejora y nuevas funcionalidades que realmente agreguen valor.

5.3 Trabajo Futuro

- Con el aumento de la conectividad y el desarrollo de la tecnología, es interesante abordar la integración de modelos inteligentes en la nube. Explorar e integrar al sistema las herramientas para aprendizaje de máquina de Firebase para dispositivos móviles.
- Realizar un manual o documentación detallada con guías de usuario para facilitar la adopción y el uso de la aplicación. Esto puede incluir tutoriales, demostraciones en video y recursos escritos que ayuden a los usuarios a comprender plenamente las capacidades y funciones de la aplicación.
- Implementar el sistema en un cultivo donde se realice un análisis completo de las eficiencias, determinando el impacto real ambiental en el ahorro del agua y el ahorro económico.

Bibliografía

[1] L. M. Fernández-Ahumada, J. Ramírez-Faz, M. Torres-Romero, and R. López-Luque, "Proposal for the Design of Monitoring and Operating Irrigation Networks Based on IoT, Cloud Computing and Free Hardware Technologies," *Sensors*, vol. 19, no. 10, p. 2318, May 2019, doi: 10.3390/s19102318.

[2] G. Culcay and V. S. Manzano, "API REST para la transmisión de información y control de redes de sensores IOT," Carrera de Ingeniería en Electrónica y Comunicaciones, Universidad Técnica de Ambato, Ambato, 2022. Accessed: Nov. 18, 2022. [Online]. Available: https://repositorio.uta.edu.ec/jspui/handle/123456789/35022

[3] O. Elijah, T. A. Rahman, I. Orikumhi, C. Y. Leow, and M. H. D. N. Hindia, "An Overview of Internet of Things (IoT) and Data Analytics in Agriculture: Benefits and Challenges," *IEEE Internet Things J*, vol. 5, no. 5, pp. 3758–3773, Oct. 2018, doi: 10.1109/JIOT.2018.2844296.

[4] L. Moroney, *The Definitive Guide to Firebase*. Berkeley, CA: Apress, 2017. doi: 10.1007/978-1-4842-2943-9.

[5] O. Bhat, P. Gokhale, and S. Bhat, "Introduction to IOT," *International Advanced Research Journal in Science, Engineering and Technology ISO*, vol. 3297, no. 1, 2007, doi: 10.17148/IARJSET.2018.517.

[6] M. Edmundo José Jalón Arias, M. Luis Orlando Albarracín Zambrano, M. Luis Javier Molina Chalacan, and L. Jeannette Alexandra Laverde Mena, "Insertion of an automated sprinkler irrigation system for parks in the Quevedo canton," Quevedo, 2019.

[7] A. Alsalemi *et al.*, "Real-Time Communication Network Using Firebase Cloud IoT Platform for ECMO Simulation," in *2017 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, IEEE, Jun. 2017, pp. 178–182. doi: 10.1109/iThings-GreenCom-CPSCom-SmartData.2017.31.

[8] W.-J. Li, C. Yen, Y.-S. Lin, S.-C. Tung, and S. Huang, "JustIoT Internet of Things based on the Firebase real-time database," in *2018 IEEE International Conference on Smart Manufacturing, Industrial & Logistics Engineering (SMILE)*, IEEE, Feb. 2018, pp. 43–47. doi: 10.1109/SMILE.2018.8353979.

[9] S. Sarkar, S. Gayen, and S. Bilgaiyan, “Android Based Home Security Systems Using Internet of Things(IoT) and Firebase,” in *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*, IEEE, Jul. 2018, pp. 102–105. doi: 10.1109/ICIRCA.2018.8597197.

[10] A. P. Divyesh Zanzmeriya, “Implementation of Industrial Automation Systems using Raspberry pi by IOT with FIREBASE,” *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, no. 5, May 2018.

[11] Moranain Mungkin, Habib Satria, Jeddah Yanti, Grace Boni A Turnip, and Suwarno, “PERANCANGAN SISTEM PEMANTAUAN PANEL SURYA POLYCRYSTALLINE MENGGUNAKAN TEKNOLOGI WEB FIREBASE BERBASIS IoT,” *Journal of Information Technology and Computer Science.*, vol. 3, no. 2, Dec. 2020.

[12] Zaw Lin Oo, Theint Win Lai, Su Mon Ko, and Aung Moe, “IoT based Weather Monitoring System Using Firebase Real Time Database with Mobile Application,” *International Symposium on Environmental-Life Science and Nanoscales Technology 2019*, Dec. 2019.

[13] S. Rajendrakumar, V. K. Parvati, and Rajashekarappa, “An Efficient Irrigation System for Agriculture,” in *2018 International Conference on Communication, Computing and Internet of Things (IC3IoT)*, IEEE, Feb. 2018, pp. 132–136. doi: 10.1109/IC3IoT.2018.8668206.

[14] V. González Breña, “Riego inteligente,” Dec. 2017, Accessed: Jan. 17, 2024. [Online]. Available: http://dehesa.unex.es/handle/10662/6698

[15] G. Rendón, J. Cortes, D. Juárez, and M. Ortegea, “Sistema de riego inteligente utilizando electroválvulas a partir de sensores de visión,” *Pistas Educativas No.113. Instituto Tecnoloógico de Celaya*, Oct. 2015. Accessed: Feb. 01, 2024. [Online]. Available: https://pistaseducativas.celaya.tecnm.mx/index.php/pistas/article/view/337/326

[16] M. López *et al.*, “Sistema de riego por goteo automático utilizando una red de sensores inalámbricos. ,” *Revista de Investigación de Arequipa*, 2015. Accessed: Feb. 01, 2024. [Online]. Available: https://web.archive.org/web/20180421074513id_/http://ucsp.edu.pe/investigacion/wp-content/uploads/2017/01/4.-Sistema-de-riego-por-goteo-autom%C3%A1tico.pdf

[17] E. Cangás, “Aplicación web de administración de base de datos y de aplicación de aprendizaje de máquina en línea para un sistema de IoT de riego Inteligente.,” Universidad Técnica del Norte, Ibarra, 2023. Accessed: Feb. 01, 2024. [Online]. Available:

http://repositorio.utn.edu.ec/bit-stream/123456789/15339/2/04%20MEC%20524%20TRA-BAJO%20DE%20GRADO.pdf

[18] Glaría Jaime and Samir Kouro, "Sensores de humedad," *Universidad Técnica Federíco Santa María*, 2001.

[19] Sheyla Jimenez, Luciana Scarioni, and Kelim Vano, "Nota Técnica: Sensores de humedad de tipo capacitivo y resistivo," *Revista Ingeniería Vol. 20, No. 1*, Valencia, pp. 83–86, 2013.

[20] Liz Moreno, "Respuesta de las plantas al estrés por déficit hídrico. Una revisión.," SciELO Colombia. Agronomía Colombiana 27.

[21] J. Pizarro, *Internet de las cosas (IoT) con Arduino. Manual práctico*. 2019. Accessed: Feb. 12, 2023. [Online]. Available: https://books.google.es/books?id=73yJDwAAQBAJ&lpg=PP1&hl=es&pg=PP1#v=onepage&q&f=false

[22] D. Evans, "Internet de las Cosas. Cómo la próxima evolución de Internet lo cambia todo.," 2011.

[23] M. Gigli and S. Koo, "Internet of Things: Services and Applications Categorization," *Advances in Internet of Things*, vol. 01, no. 02, pp. 27–31, 2011, doi: 10.4236/ait.2011.12004.

[24] S. Madakam, R. Ramaswamy, and S. Tripathi, "Internet of Things (IoT): A Literature Review," *Journal of Computer and Communications*, vol. 03, no. 05, pp. 164–173, 2015, doi: 10.4236/jcc.2015.35021.

[25] Microsoft, "¿Qué es middleware?," cloud computing dictionary.

[26] G. de N. A. Profesionales, "La situación de las Tecnologías WLAN basadas en el estándar IEEE 802.11 y sus variantes ('Wi-Fi')".

[27] O. Campos, J. Correa, and G. Zeballos, "IMPLEMENTAR UN SISTEMA DE INFRAESTRUCTURA COMO SERVICIO (IAAS) EN CLOUD COMPUTING QUE SIRVA DE ALOJAMIENTO AL ERP EN UNA EMPRESA COMERCIAL," Lima, 2012.

[28] P. M. Mell and T. Grance, "The NIST definition of cloud computing," Gaithersburg, MD, 2011. doi: 10.6028/NIST.SP.800-145.

[29] T. Dillon, C. Wu, and E. Chang, "Cloud Computing: Issues and Challenges," in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, 2010, pp. 27–33. doi: 10.1109/AINA.2010.187.

[30] W. Kim, “Cloud computing: Today and tomorrow. J. Object Technol., 2009, vol. 8, no 1, p. 65-72.,” *Journal of Object Technology*, vol. 8, no. 1, pp. 65–72, 2009.

[31] T. Dillon, C. Wu, and E. Chang, “Cloud Computing: Issues and Challenges,” in *2010 24th IEEE International Conference on Advanced Information Networking and Applications*, IEEE, 2010, pp. 27–33. doi: 10.1109/AINA.2010.187.

[32] Google for developeres, “Realtime Database,” Documentación de Firebase. Accessed: Jan. 18, 2024. [Online]. Available: https://firebase.google.com/docs/database?hl=es-419&authuser=1

[33] L. I. Mesa, “SISTEMA DE ADQUISICIÓN DE DATOS EN CULTIVOS DE HUERTOS URBANOS,” Universidad Técnica del Norte, Ibarra, 2023. Accessed: Feb. 03, 2024. [Online]. Available: http://repositorio.utn.edu.ec/bitstream/123456789/14365/2/04%20MEC%20478%20TRABAJO%20DE%20GRADO.pdf

[34] Google for Developers, “Cloud Messaging,” Documentación de Firebase. Accessed: Jan. 18, 2024. [Online]. Available: https://firebase.google.com/docs/cloud-messaging?hl=es-419&authuser=1

[35] Google for Developers, “Performance Monitoring,” Documentación de Firebase. Accessed: Jan. 29, 2024. [Online]. Available: https://firebase.google.com/docs/perf-mon?authuser=1&_gl=1*d0ssw9*_ga*MTE4MDI3NTY-wMS4xNjgxMTg2NTM1*_ga_CW55HF8NVT*MTcwNjY2NDkyNS4yODguMS4xNzA2NjY1MDAxLjU2LjAuMA..&hl=es-419#implementation_paths

[36] “Firebase Performance Monitoring.” Accessed: Feb. 04, 2024. [Online]. Available: https://firebase.google.com/docs/perf-mon?hl=es-419

[37] D. Barrios Contreras, “Arquitectura de Microservicios,” *Publicación de la Facultad de Ingeniería Universidad Distrital Francisco Jose de Caldas*, 7, Jan. 2018.

[38] J. Lewis and M. Fowler, “martinFowler,” MIcroservices, a definition of this new architectural term. Accessed: Jan. 18, 2024. [Online]. Available: https://martinfowler.com/articles/microservices.html

[39] F. Rivera, *Base de datos relacionales*. 2008. Accessed: Jan. 21, 2024. [Online]. Available: https://books.google.es/books?id=buM5rlZME-cC&lpg=PA9&ots=6N1InuHFLO&dq=Base%20de%20datos&lr&hl=es&pg=PA9#v=onepage&q=Base%20de%20datos&f=false

[40] V. Valverde, N. Portalanza, and P. Mora, “Análisis descriptivo de base de datos relacional y no relacional,” *Revista Atlante; Cuadernos de Educación y Desarrollo.*, Jun. 2019. Accessed: Jan. 21, 2024. [Online]. Available: www.eumed.net/rev/atlante/2019/06/base-datos-relacional.html

[41] S. Bhosale, T. Patil, and P. Patil, “International Journal of Computer Science and Mobile Computing,” in *SQLite: Light Database System*, Researchgate, 2015. Accessed: Jan. 21, 2024. [Online]. Available: http://www.ijcsmc.com/

[42] N. Sahni, J. Bose, and K. Das, “Web APIs for Internet of Things,” in *2018 International Conference on Advances in Computing, Communications and Informatics, ICACCI 2018*, Institute of Electrical and Electronics Engineers Inc., Nov. 2018, pp. 2175–2181. doi: 10.1109/ICACCI.2018.8554612.

[43] S. Verona-Marcos, Y. Pérez-Díaz, L. Torres-Pérez, M. D. Delgado-Dapena, and C. Yáñez-Márquez, “Pruebas de rendimiento a componentes de software utilizando programación orientada a aspectos,” *Ingeniería Industrial*, vol. 37, no. 3, pp. 278–285, 2016, Accessed: Feb. 04, 2024. [Online]. Available: http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1815-59362016000300006&lng=es&nrm=iso&tlng=es

[44] R. Pressman, *Ingeniería del Software: Un enfoque práctico*. Ciudad de México: MacGraw Hill, 2010.

[45] D. Farinango, “Aplicación móvil para rastreo de autobuses,” Universidad Técnica del Norte, Ibarra, 2020.

[46] E. Maila, “EVALUACIÓN DE HERRAMIENTAS OPEN SOURCE PARA PRUEBAS DE FIABILIDAD Y RENDIMIENTO DE APLICACIONES WEB,” Escuela Politécnica Nacional, Quito, 2017. Accessed: Feb. 04, 2024. [Online]. Available: https://bibdigital.epn.edu.ec/bitstream/15000/19947/1/CD-9405.pdf

[47] Á. Javier, D. Sanjuán, J. Luis, P. Rodríguez, and A. A. Rosado, “COMPARACIÓN DE DOS TECNOLOGÍAS DE DESARROLLO DE APLICACIONES MÓVILES DESDE LA PERSPECTIVA DE LOS ATRIBUTOS DE CALIDAD,” *Scientia et Technica*, vol. 20, no. 1, pp. 81–87, Mar. 2015, doi: 10.22517/23447214.9278.

[48] J. Llamuca-Quinaloa, Y. Vera-Vincent, V. Tapia-Cerda, J. Llamuca-Quinaloa, Y. Vera-Vincent, and V. Tapia-Cerda, “Análisis comparativo para medir la eficiencia de desempeño entre una aplicación web tradicional y una aplicación web progresiva,” *TecnoLógicas*, vol. 24, no. 51, pp. 164–185, Jul. 2021, doi: 10.22430/22565337.1892.

[49] International Organization for Standardization, *ISO/IEC 25010:2011*. Ginebra: International Organization for Standardization, 2011. Accessed: Feb. 04, 2024. [Online]. Available: https://www.iso.org/standard/35733.html

[50] "Información sobre los datos de rendimiento de carga de las páginas (apps web) | Firebase Performance Monitoring." Accessed: Feb. 04, 2024. [Online]. Available: https://firebase.google.com/docs/perf-mon/page-load-traces?hl=es-419&authuser=1&platform=ios

Anexos

Anexo A. Programa del sistema de riego

El programa del sistema de riego se encarga de gestionar los componentes de hardware, monitorear las variables ambientales y el estado de riego y ejecutar acciones. Acciones como iniciar riego manual o automático, ejecutar comandos de terminal, publicar la información, detener o iniciar el programa.

```
from __future__ import division
import threading
from datetime import datetime
from firebase import firebase
from google.cloud import functions
from datetime import datetime
import firebase_admin
from firebase_admin import credentials, db
import time
import random
import json
import subprocess
import numpy as np
from aioconsole import ainput
import numpy
import serial
import struct
import sys
import os
import csv
import asyncio

# Time variables
current_second = datetime.now()  # time in format: 2023-08-09 17:22:20.248461
# time in format: 2023-08-09 17:22:20.248461
current_date = datetime.date(current_second)
# time in format: 2023-08-09 17:22:20
Start_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

# Credential of service account from firebase proyect console
cred = credentials.Certificate(
    'iotriego-17bad-firebase-adminsdk-pcsm5-38d08da6a5.json')

firebaseConfig = {
    "apiKey": "AIzaSyBcLGOVLIr-R64vY7dT1mWGdYhd4M26bp8",
    "authDomain": "iotriego-17bad.firebaseapp.com",
    "databaseURL": "https://iotriego-17bad-default-rtdb.firebaseio.com",
    "projectId": "iotriego-17bad",
    "storageBucket": "iotriego-17bad.appspot.com",
    "messagingSenderId": "842817138478",
    "appId": "1:842817138478:web:3cc5b91a6b3a0196bb94bf"
}
# Initialize the application with the firebase SDK configuration
firebase_admin.initialize_app(cred, firebaseConfig)

# database Reference representing the node at the specified path (databa-
seURL).
realtimedb = db.reference()
```

```
try:
    # Open the config file in the app directory, filename = config.json
    with open('config.json', 'r') as archive:
        config = json.load(archive)
        proyect = str(config["proyect"])
        settings = config["settings"]
        actions = config["actions"]
        data = config["data"]
        # devices = config["devices"]

        # Firebase
        fb = realtimedb.child(proyect)
        fb.child('settings').set(settings)
        fb.child('actions').set(actions)
        fb.child('data').set(data)
        # fb.child('devices').set(devices)

except Exception as e:
    print('Configuration file error (config.json)')
    print('error: ', e)
    try:
        fb = realtimedb.child('CultivoDispFinal')
        config = fb.get()
    except Exception as e:
        print('Firebase error: fail getting settings from ', e)

dataRef = fb.child('data')
actionsRef = fb.child('actions')
settingsRef = fb.child('settings')



# ######################################################
# # Read and  return the current time
def readTime(tipo):  # (tipo):
    # Obtener la marca de tiempo actual en formato 'día mes año hora:mi-
nuto:segundo'
    timeStamp = time.strftime("%d %b %Y %H:%M:%S", time.localtime())

    if tipo == 0:
        # Time Day Month Hour Minute
        timeStamp = time.strftime("%d %b %H:%M:%S", time.localtime())
        return timeStamp
    else:
        return timeStamp

######################################################
# sincronice config.json with Firebase
def firebaseSinchronize():
    global settings, actions
    try:
        # Open the config file in the app directory, filename = config.json
        with open('config.json', 'r') as archive:
            config = json.load(archive)
            n_settings = config["settings"]
            n_actions = config["actions"]
        # if exist changes on config.json that do not come from firebase
        if settings != n_settings or actions != n_actions:
```

```
            # Firebase update content
            realtimedb.child(
                config["proyect"]+'/settings').set(n_settings)
            realtimedb.child(config["proyect"]+'/actions').set(n_actions)

            # Get content from config.json
            actions = n_actions
            settings = n_settings
        else:
            n_settings = settingsRef.get()
            n_actions = actionsRef.get()

            if settings != n_settings or actions != n_actions:

                # Get content from firebase
                actions = n_actions
                settings = n_settings
                # Open the config file in the app directory, filename = con-
fig.json and write firebase content
                with open('config.json', 'w') as archivo_json:
                    config["settings"] = settings
                    config["actions"] = actions
                    json.dump(config, archivo_json, indent=4)
    except Exception as e:
        print('error: ', e)
        print('Configuration file error (config.json)')

#####################################################
# Get data from Sensors or others elements
def outputs():
    try:
        # Sensors:
        data["humAire"] = int(random.uniform(60, 75))
        data["humSuelo"] = int(random.uniform(55, 72))
        data["tempAire"] = int(random.uniform(22, 22))
        # Actuators:
        data["riego"] = actions.get("regar")  # water pump status
    except Exception as e:
        print('Error in outpus:', e)
    return data
#####################################################
# Send data from the proyect to firebase program
def sendData(value):
    # Json to send to firebase
    fb.child('data').update(value)
def monitoring():
    # Envía datos hacia Firebase
    while actionsRef.child("monitoring").get():
        endTime = time.time()
        try:
            processTime = endTime - startTime
            time.sleep(settings.get("dataRate")-(processTime))
        except:
            pass

        startTime = time.time()
        firebaseSinchronize()
        try:
            data = outputs()
```

```
            sendData(data)
            dataRate = settings["dataRate"]
            print(f'Datos enviados a Firebase, data rate: {dataRate}s')
            print(data)
        except Exception as e:
            print("Sending data to firebase error:", e)
# Esta es la función que se ejecutará en un hilo separado
def leer_input():
    while True:
        comando = input()
        print(f"Comando: {comando}")
        if comando != '':
            if comando == "salir":
                break
            else:
                # Ejecutar el comando en el sistema
                exCommands(comando)
def exCommands(command):
    # Lista de comandos permitidos sin necesidad de permisos de superusuario
    allowed_commands = ['pgrep', 'top', 'htop',
                        'df', 'ping', 'uname', 'ls', 'cat']
    # Dividir el comando en palabras y verificar si el comando es permitido
    if command in allowed_commands:
        try:
            # Ejecutar el comando y obtener la salida
            result = subprocess.run(
                command, shell=True, check=True, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
            # Decodificar la salida a texto
            output = result.stdout.decode('utf-8')
            error = result.stderr.decode('utf-8')

            if output:
                print(f'Salida del comando: {output}')

            if error:
                print(f'Error al ejecutar el comando: {error}')

        except subprocess.CalledProcessError as e:
            # Manejar errores en la ejecución del comando
            print(f'Error al ejecutar el comando: {e}')
######## Programa principal#############
def main():
    while True:
        # Sincronización de Realtime Database y config.json
        firebaseSinchronize()
        print(readTime(0)+" -- waiting for instructions from Firebase or ter-
minal")
        # Monitoreo y publicación de valores en Realtime Database
        if actions.get("monitoring"):
            monitoring()
        fbCommand = settings.get("command")
        # Ejecución de comandos de Realtime Database
        if fbCommand != "":
            if fbCommand == "salir":
                break
            else:
                # Ejecutar el comando en el sistema
                fb.child('settings/command').set("")
```

```
                exCommands(fbCommand)
        time.sleep(10)
if __name__ == "__main__":
    main()
```

Anexo B. Programa de la base de datos SQLite

```
import time
import sqlite3
import os
import json
import firebase_admin
from firebase_admin import credentials, db, storage
from datetime import datetime
# from google.cloud import storage

proyect = str('CultivoDispFinal')
dataBaseName = 'baseDatosCultivo.db'
tiempoMuestra = 1  # minutos
cred = credentials.Certificate(
    'iotriego-17bad-firebase-adminsdk-pcsm5-38d08da6a5.json')
firebaseConfig = {
    "apiKey": "AIzaSyBcLGOVLIr-R64vY7dT1mWGdYhd4M26bp8",
    "authDomain": "iotriego-17bad.firebaseapp.com",
    "databaseURL": "https://iotriego-17bad-default-rtdb.firebaseio.com",
    "projectId": "iotriego-17bad",
    "storageBucket": "iotriego-17bad.appspot.com",
    "messagingSenderId": "842817138478",
    "appId": "1:842817138478:web:3cc5b91a6b3a0196bb94bf"
}
# Initialize the application with the firebase SDK configuration
firebase_admin.initialize_app(cred, firebaseConfig)

# database Reference representing the node at the specified path (databa-
seURL).
realtimedb = db.reference()

# fb = firebase.FirebaseApplication(urlDB, None)
fb = realtimedb.child(proyect)

# dirección del bucket de Firebase Storage
bucketURL = 'iotriego-17bad.appspot.com'




def downloadDB(archive_name):
    try:
        conn = sqlite3.connect(dataBaseName)
        c = conn.cursor()
        # Ejecutar una consulta SELECT para obtener los datos de la tabla
        c.execute(f"SELECT * FROM {'History'}")
        columns = c.fetchall()
        # Convertir los datos a una lista de diccionarios
        global datos
        datos = []
        for column in columns:
            diccionario_columna = {
```

```
                "id": column[0],
                "tiempo": column[1],
                "humedad_suelo": column[2],
                "humedad_aire": column[3],
                "temperatura": column[4],
            }
            datos.append(diccionario_columna)
        # Escribir los datos en un archivo JSON
        with open(archive_name+".json", 'w') as archivo_json:
            json.dump(datos, archivo_json, indent=4)
        print(
            f"La entidad History ha sido exportada a {archive_name}.json.")
    except sqlite3.Error as e:
        print(f"Error al exportar la tabla '{archive_name}': {e}")
    finally:
        conn.close()

def uploadDB(blobName, filePath):
    # Enviar los datos de la base de datos a Firebase
    try:
        # Referencia al bucket de Firebase Cloud Storage.
        bucket = storage.bucket(f'{bucketURL}')
        # Crea la dirección del nuevo blob (objeto) en tu bucket.
        Objeto = bucket.blob(blobName)
        # sube un archivo a tu blob desde tu sistema de archivos local.
        Objeto.upload_from_filename(filePath)
        print(f"{readTime(1)}File {filePath} uploaded to {blobName}.")
    except Exception as e:
        # muestra del error en la consola
        print("Error al enviar datos a Firebase:", e)
    # Devuelve una URL de descarga pública
    return Objeto.public_url
#####################################################################
def getData():
    # Obtención de datos en tiempo real desde Firebase
    try:
        result = fb.child('data').get()
        humSuelo = fb.child('data/humSuelo').get()
        humAire = fb.child('data/humAire').get()
        tempAire = fb.child('data/tempAire').get()
        irrigate = fb.child('data/riego').get()
    except Exception as e:
        print(e)
    return result, humSuelo, humAire, tempAire, irrigate

#####################################################################
# Creación de la base de datos SQlite
def initDB():
    conn = sqlite3.connect(dataBaseName)
    c = conn.cursor()
    c.execute('''CREATE TABLE History (ID integer primary key, EstampaTiempo
real, \
           HumedadSuelo real, HumedadAire real, TemperaturaAire real, Riego
real)''')
    conn.commit()  # save changes
    conn.close()

#####################################################################
# Añadir datos a la base de datos SQlite
```

```
def addDataDB():
    global humedadSuelo, humedadAire, temperaturaAire, riego
    conn = sqlite3.connect(dataBaseName)
    c = conn.cursor()
    c.execute('''INSERT INTO History
        (EstampaTiempo, HumedadSuelo, HumedadAire, TemperaturaAire, Riego)
        VALUES (?, ?, ?, ?, ?)''', (time.time(), humedadSuelo, humedadAire,
temperaturaAire, riego))
    conn.commit()
    conn.close()
def dataChanges(event):
    global humedadSuelo, humedadAire, temperaturaAire, riego
    try:

        humedadSuelo = fb.child('data/humSuelo').get()
        humedadAire = fb.child('data/humAire').get()
        temperaturaAire = fb.child('data/tempAire').get()
        riego = fb.child('data/riego').get()
        addDataDB()
        print(f"{readTime(0)} - cambios agregados a la base de datos:",
event.data)
    except Exception as e:
        print(e)
        print("Error al cargar datos del listen() la base de datos")

#################################################################
# Obtener la marca de tiempo actual en el formato requerido
def readTime(tipo):
    # Obtener la marca de tiempo actual en formato 'día mes hora:minuto:se-
gundo'
    timeStamp = time.strftime("%d %b %Y %H:%M:%S", time.localtime())
    if tipo == 0:
        # Time Day Month Hour Minute
        timeStamp = time.strftime("%d %b %H:%M", time.localtime())
        return timeStamp
    else:
        return timeStamp

######## Programa principal###############
def main():
    if bool(fb.child('actions/sqliteRead').get()):
        fb.child('actions/sqliteRead').set(False)
        url = uploadDB(f'{dataBaseName}', f'./{dataBaseName}')
        print(url)
        print("se ha exportado la base de datos SQlite en firebase")
    listener = None
    while bool(fb.child('actions/sqliteWrite').get()):
        if listener is None:
            try:
                print("Listen Realtime Database Proyect Data")
                listener = fb.child('data').listen(dataChanges)
            except Exception:
                listener.close()
                print("Error listening", Exception)

    if listener is not None:
        print("Finish listening Realtime Database Proyect Data")
        listener.close()
if __name__ == "__main__":
```

```
    if os.path.isfile(dataBaseName) == False:  # verifica si existe la base
de datos
        initDB()  # si no existe, se la crea
        print("Un nuevo archivo de base de datos ha sido creado")
    else:
        print("Un archivo de base de datos ha sido encontrado")

    while True:
        # programa de la base de datos
        main()
        time.sleep(tiempoMuestra*10)
        print("Esperando instrucciones de firebase")
```

Anexo C. Programa de la aplicación web en Astro

```
// Index de la aplicación, archivo index.astro
---
// este es el frontmatter
import Layout from "../layouts/Layout.astro";
import LandingHeader from "../components/LandingHeader.astro";
import HomeSection from "../components/HomeSection.astro";
import PanelSection from "../components/PanelSection.astro";
import ActionsSection from "../components/ActionsSection.astro";
import ControlSection from "../components/ControlSection.astro";

const nameProyect = 'CultivoDispFinal'
---

<Layout title={`Aplicación del sistema de riego inteligente ${nameProyect}`}>
    <LandingHeader nameProyect={nameProyect} />
    <main
        class="snap-y snap-mandatory relative w-full h-screen overflow-auto
scroll-m-0"
    >
        <div class="snap-center">
            <HomeSection />
        </div>
        <div class="snap-center">
            <PanelSection nameProyect={nameProyect} />
        </div>
        <div id="actions" class="snap-center" style="display:none">
            <ActionsSection nameProyect={nameProyect} />
        </div>
        <div id="control" class="snap-center" style="display:none">
            <ControlSection nameProyect={nameProyect} />
        </div>

    </main>
</Layout>
</script>
<script src="../javascript/index.js"></script>
<script src="../javascript/functions.js"></script>
<script src="../javascript/messaging.js"></script>
```

```
// Sección de Acciones de la aplicación, archivo ActionsSection.astro
---
interface Props {
    nameProyect: string;
}

const { nameProyect } = Astro.props;
---

<section
    id="ActionsSection"
    class="landing-section h-screen w-screen text-center overflow-hidden rel-
ative"
    data-header-color="black"
>
    <div class="z-30 relative h-full flex flex-col">
        <header>
            <h2 class="text-black pt-28 pb-10 text-4xl font-medium">
                Panel de Acciones 👨‍💻 <strong>{nameProyect}</strong>
            </h2>
        </header>
        <div class="flex flex-col flex-grow text-center">
            <h3 class="card-header text-xl font-medium text-center px-10 py-
2">
                Acciones
            </h3>
            <div class="flex flex-grow bg-white py-3" id="div-actions"></div>
        </div>
        <div class="gap-x-4 flex justify-center"></div>
        <footer class="flex justify-end flex-col pb-10">
        </footer>
    </div>
</section>
```

```
// Sección de Control de la aplicación, archivo ControlSection.astro
---
interface Props {
    nameProyect: string;
}

const { nameProyect } = Astro.props;
---

<section
    id="ControlSection"
    class="landing-section h-screen w-screen text-center overflow-hidden rel-
ative"
    data-header-color="black"
>
    <div class="z-30 relative h-full flex flex-col">
        <header>
            <h2 class="text-black pt-28 pb-10 text-4xl font-medium">
                Panel de Control 👷 <strong>{nameProyect}</strong>
            </h2>
        </header>
        <div class="flex flex-col flex-grow text-center">
            <h3 class="card-header text-xl font-medium text-center px-10 py-
2">
                Riego Automatico | <a id="button-riego-img" href="#">
Detalles</a>
            </h3>
            <div class="flex flex-grow bg-white py-3" id="div-control"></div>
        </div>
        <article id="image-modal" class="modal">
            <div class="modal-contenido">
                <span class="cerrar" id="cerrar-imagen-modal">&times;</span>
                <img class="w-full" src="/assets/Riego-auto-functions.png"
alt="imagen del algoritmo de riego automatico">
            </div>
        </article>
        <article id="control-modal" class="modal">
            <div class="modal-contenido" >
                <span class="cerrar" id="cerrar-control-modal">&times;</span>
                <div id="realtime-action" style="display:none">

                    <h3 class="text-indigo-800 text-base py-6 border" >Escri-
tura en Realtime en respuesta al cumplimiento de una condición</h3>
                <form id="realtime-action-form" autocomplete="on" class="py-4
[&>input]:w-auto [&>input]:justify-center [&>input]:text-base [&>input]:font-
normal [&>input]:rounded-md [&>input]:transition-colors">
                    <label for="compareRef">Referencia Para Comparación:</la-
bel>
                    <input type="text" id="compareRef" name="compareRef"
placeholder="CultivoDispFinal/nodo/database" value="CultivoDispFinal/" re-
quired><br>
                    <label for="operation">Operación:</label>
                    <select id="operation" name="operation" class="text-in-
digo-800" required>
                        <option value=">"> mayor que &gt; </option>
                        <option value="<"> menor que &lt; </option>
                        <option value="=="> igual == </option>
                        <option value="!="> diferente != </option>
```

```
                    </select>
                    <label for="target">Valor Objetivo:</label>
                    <input type="text" id="target" name="target"place-
holder="Valor de la condición" required><br>
                    <label for="resolveRef">Referencia Para Escritura:</la-
bel>
                    <input type="text" id="resolveRef" name="resolveRef"
placeholder="CultivoDispFinal/nodo/database"  value="CultivoDispFinal/" re-
quired><br>
                    <label for="resolveVal">Valor Escritura:</label>
                    <input type="text" id="resolveVal" name="resolveVal"
placeholder="" required><br>
                    <br><br>
                    <input type="submit" class="text-indigo-800 border-in-
digo-800 border-[2px] rounded font-medium text-base px-12 py-2 bg-white/5
backdrop-blur-2xl hover:bg-indigo-900 hover:text-white transition-colors"
value="Enviar">
                </form>

                </div>
                <div id="realtime-notification" style="display:none">
                    <h3 class="text-indigo-800 text-base py-6 border">Notifi-
caciones en respuesta al cumplimiento de una condición</h3>
                    <form id="realtime-notification-form" autocomplete="on"
class="py-4 [&>input]:justify-center [&>input]:text-base [&>input]:font-nor-
mal [&>input]:rounded-md [&>input]:transition-colors">
                        <label for="compareRefN">Referencia Para Compara-
ción:</label>
                        <input type="text" id="compareRefN" name="compareRef"
placeholder="CultivoDispFinal/nodo/database" value="CultivoDispFinal/" re-
quired><br>
                        <label for="operationN">Operación:</label>
                        <select id="operationN" name="operationN"
class="text-cyan-800" required>
                            <option value=">"> mayor que &gt; </option>
                            <option value="<"> menor que &lt; </option>
                            <option value="=="> igual == </option>
                            <option value="!="> diferente != </option>
                        </select>
                        <label for="targetN">Valor Objetivo:</label>
                        <input type="text" id="targetN" name="target" place-
holder="Valor de la condición" required><br>
                        <label for="msgN">Mensage de la Notificación:</la-
bel><br>
                        <input type="text" class="w-full" id="msgN"
name="msg" required><br>
                        <br>
                        <input type="submit" class="text-cyan-800 border-
cyan-800 border-[2px] rounded font-medium text-base px-12 py-2 bg-white/5
backdrop-blur-2xl hover:bg-indigo-900 hover:text-white transition-colors"
value="Enviar">
                    </form>
                </div>
            </div>
        </article>
        <div class="gap-x-4 flex justify-center">

            <a
            id="button-show-actions"
```

```
            class="text-indigo-800 border-indigo-800 border-[2px] rounded
font-medium text-base px-12 py-2 bg-white/5 backdrop-blur-2xl hover:bg-in-
digo-900 hover:text-white transition-colors"
            href="#"
            >
            Acciones en Realtime</a
            >
            <a
            id="button-show-messaging"
            class="text-cyan-700 border-cyan-700 border-[2px] rounded font-
medium text-base px-12 py-2 bg-white  backdrop-blur-2xl hover:bg-cyan-900
hover:text-white transition-colors"
            href="#"
            >
            Notificación</a
            >
        </div>
        <footer class="flex justify-center flex-row pb-10">

        </footer>
    </div>
</section>

<script>
    const modal = document.getElementById("control-modal");
    const showRealtimeActions = document.getElementById("button-show-ac-
tions");
    const showRealtimeMessaging = document.getElementById("button-show-mes-
saging");
    const cerrarModal = document.getElementById("cerrar-control-modal");

    const realtimeAction = document.getElementById("realtime-action")
    const realtimeNotification = document.getElementById("realtime-notifica-
tion")

    const modalImg = document.getElementById("image-modal");
    const showRiegoImg = document.getElementById("button-riego-img")
    const cerrarImg = document.getElementById("cerrar-imagen-modal")

     showRealtimeActions.addEventListener("click", function () {
        modal.style.display = "block";
        realtimeAction.style.display = "block";
        realtimeNotification.style.display = "none";
    });

    showRealtimeMessaging.addEventListener("click", function () {
        modal.style.display = "block";
        realtimeNotification.style.display = "block";
        realtimeAction.style.display = "none"
    });

    showRiegoImg.addEventListener("click", function () {
        modalImg.style.display = "block";
        modal.style.display = "none";
    });

    cerrarModal.addEventListener("click", function () {
        modal.style.display = "none";
    });
```

```
    cerrarImg.addEventListener("click", function () {
        modalImg.style.display = "none";
    });

    window.addEventListener("click", function (event) {
        if (event.target === modal) {
            modal.style.display = "none";
        }
        if (event.target ===modalImg) {
            modalImg.style.display = "none";
        }
    });

</script>


<style>
    .modal {
        display: none;
        position: fixed;
        z-index: 1;
        left: 0;
        top: 0;
        width: 100%;
        height: 100%;
        overflow: auto;
        background-color: rgba(0, 0, 0, 0.7);
    }
    .modal-contenido {
        background-color: #fefefe;
        margin: 15% auto;
        padding: 40px;
        border: 1px solid #888;
        width: 60%;
        border-radius: 1rem;
    }
    .cerrar {
        color: #aaa;
        float: right;
        font-size: 40px;
        font-weight: bold;
    }
    .cerrar:hover,
    .cerrar:focus {
        color: black;
        text-decoration: none;
        cursor: pointer;
    }
    .form-control {
        display: none;
        width: 100%;
        font-size: 1rem;
        font-weight: 400;
        line-height: 1.5;
        background-clip: padding-box;
        border: 1px solid #ced4da;
        -webkit-appearance: none;
        -moz-appearance: none;
```

```
        appearance: none;
        border-radius: 0.375rem;
        transition:
            border-color 0.15s ease-in-out,
            box-shadow 0.15s ease-in-out;
    }
</style>
<div
    id="gaugeOptions"
    style="display:none;"
    class="flex flex-col flex-grow flex-nowrap"
>
<p class="text-lg font-semibold text-ellipsis py-4" >Click en Crear para
agregar el Medidor</p>
    <table
        id="widget_params_config_"
        style="padding-left: 10%;"
        class="mx-auto"
    >
        <tbody>
            <tr>
                <td class="widget-table-cells"> Name</td>
                <td colspan="1" class="widget-table-cells">
                    <input
                        required
                        class="form-control"
                        maxlength="20"
                        placeholder="Enter Name for the widget "
                        type="text"
                        id="name_"
                        name="name"
                        value=""
                    />
                </td>
            </tr>
            <tr>
                <td class="widget-table-cells"> Source</td>
                <td colspan="1" class="widget-table-cells">
                    <label
                        >Choose a source from this list: <input
                            class="form-control"
                            list="sources"
                            name="mySource"
                            id="source_"
                            required
                        /></label
                    >
                    <datalist id="sources">
                        <option value="humAire"></option>
                        <option value="humSuelo"></option>
                        <option value="tempAire"></option>
                    </datalist>
                </td>
            </tr>
            <tr>
                <td class="widget-table-cells"> Min</td>
                <td colspan="1" class="widget-table-cells">
                    <input
                        class="form-control"
```

```
            placeholder=" "
            type="number"
            step="any"
            id="min_"
            name="min"
            value="0"
        />
    </td>
</tr>
<tr>
    <td class="widget-table-cells"> Max</td>
    <td colspan="1" class="widget-table-cells">
        <input
            class="form-control"
            maxlength="80"
            placeholder="Should be greater than Min"
            type="number"
            id="max_"
            name="max"
            value="100"
        />
    </td>
</tr>
<tr>
    <td class="widget-table-cells"> Units</td>
    <td colspan="1" class="widget-table-cells">
        <input
            class="form-control"
            maxlength="80"
            placeholder="Enter Measurement Units "
            type="text"
            id="units_"
            name="units"
            value=""
        />
    </td>
</tr>
<tr>
    <td class="widget-table-cells"> Tick Interval</td>
    <td colspan="1" class="widget-table-cells">
        <input
            class="form-control"
            placeholder=" "
            type="number"
            step="any"
            oninput="validity.valid||(value=0)"
            min="0"
            id="tick_interval_"
            name="tick_interval"
            value="10"
        />
    </td>
</tr>
<tr class="my-3">
    <td>
        <p id="p_source" class="text-red-600"></p>
    </td>
    <td>
        <button
```

```
                        type="button"
                        id="gauge_widget_create"
                        class="bg-green-500 text-gray-100 hover:bg-green-400
hover:text-white w-2/3 rounded-md py-3"
                        >Crear</button
                    >
                </td>
            </tr>
        </tbody>
    </table>
</div>
<style>
    .form-control {
        display: block;
        width: 100%;
        padding: 0.375rem 0.75rem;
        font-size: 1rem;
        font-weight: 400;
        line-height: 1.5;
        color: #212529;
        background-color: #fff;
        background-clip: padding-box;
        border: 1px solid #ced4da;
        -webkit-appearance: none;
        -moz-appearance: none;
        appearance: none;
        border-radius: 0.375rem;
        transition:
            border-color 0.15s ease-in-out,
            box-shadow 0.15s ease-in-out;
    }
    .widget-table-cells {
        padding: 5px;
    }
</style>
```

```
// Sección de Inicio de la aplicación, archivo HomeSection.astro
---
import Card from "./Card.astro"
---
<section
    class="landing-section h-screen w-screen text-center overflow-hidden rel-
ative"
    data-header-color="white"
>

<div class="z-30 relative h-full flex flex-col">
    <header class="pt-40">
        <h1 class="text-white text-3xl">Bienvenidos a</h1>
        <h2 class="text-white text-2xl font-medium pb-4">Riego IoT</h2>
        <h3 class="text-white text-xs font-medium"> Aplicación para la admi-
nistración de un sistema de riego inteligente</h3>
            <p class="text-white text-xs">
                Para una introducción revisa los siguientes recursos en el<a
href="https://utneduec-my.sharepoint.com/:f:/g/personal/brdava-
losa_utn_edu_ec/EjrcVu8U3e9Kkv823-
MWF78BOkghHHn4_vtOji7aB1egAg?e=gO2f8B"><code>enlace</code></a> del projecto.
            </p>
    </header>

    <footer class="flex flex-grow flex-nowrap justify-end flex-col pb-10">
        <div>
            <a
                class="text-white border-[3px] rounded font-medium text-xs
px-12 py-2 border-white bg-white/5 backdrop-blur-3xl hover:bg-white
hover:text-black transition-colors"
                href="#PanelSection"
            >
                Continuar al proyecto</a
            >
        </div>
    </footer>
</div>

    <div class="absolute top-0 bottom-0 h-full w-full z-10">
        <video
            class="object-center object-cover h-full w-full"
            autoplay
            loop
            muted
            src="/assets/Homepage-Model-Y-Desktop-NA.mp4"></video>
    </div>

</section>
```

```
// Sección de Monitoreo de la aplicación, archivo PanelSection.astro
---
import GaugueOptions from "./GaugueOptions.astro";
import WidgetContent from "./WidgetContent.astro";
import LampOptions from "./LampOptions.astro";
import Indicator from "./Indicator.astro";

interface Props {
    nameProyect: string;
}

const { nameProyect } = Astro.props;
---

<section
    id="PanelSection"
    class="landing-section h-screen w-screen text-center overflow-hidden rel-
ative bg-gray-100"
    data-header-color="black"
>
    <div class="z-30 relative h-full flex flex-col">
        <header>
            <h2 class="text-black pt-20 text-4xl font-medium">
                Panel de datos 👨‍💻 <strong id="proyect">{namePro-
yect}</strong>
            </h2>
        </header>
        <h3 class="card-header text-xl font-medium text-center py-2">
            Variables ambientales
        </h3>
        <div class="flex flex-col flex-grow pb-2 text-center">
            <div class="flex flex-grow flex-row">
                <div class="flex flex-grow flex-col mx-3">
                    <div class="flex flex-grow bg-white rounded-md py-2"
id="div-data"></div>
                        <div class="relative flex bg-white border-0 rounded-
md px-2 " id="canvas-chart"></div>
                        <div class="relative flex bg-white border-0 rounded-
md pb-2 px-2 " id="canvas-chart-1"></div>
                </div>
                <div class="flex flex-col m-3 gap-y-2">
                    <div class="relative flex flex-grow flex-row flex-wrap
gap-x-2 min-w-0 break-words bg-white border-0 rounded-md py-2 px-2" id="can-
vas-gauge"></div>
                    <div class="relative flex flex-row gap-x-2 min-w-0 break-
words bg-white border-0 rounded-md py-2 px-2 items-center" id="canvas-
lamp"></div>
                    <div class="relative flex flex-row gap-x-2 min-w-0 break-
words bg-white border-0 rounded-md py-2 px-2 items-center" id="canvas-indica-
tor"></div>
                </div>
            </div>
        </div>

        <article id="widget-modal" class="modal">
            <div class="modal-contenido">
                <span class="cerrar" id="cerrarModal">&times;</span>
                <WidgetContent />
                <GaugueOptions />
```

```
                <LampOptions/>
                <Indicator/>

            </div>
        </article>
        <div class="gap-x-4 flex justify-center">
            <a
                id="show-widget"
                class="text-black border-[2px] rounded font-medium text-base
px-12 py-2 border-blue-800 bg-white/5 backdrop-blur-2xl hover:bg-blue-800
hover:text-white transition-colors"
                href="#"
            >
                Agregar Widgets</a
            >
            <a
                id="show-chart"
                class="text-black border-black border-[2px] rounded font-me-
dium text-base px-12 py-2 bg-white/5 backdrop-blur-2xl hover:bg-black
hover:text-white transition-colors"
                href="#"
            >
                Gráfica de Lineas</a
            >
        </div>
        <footer class="pb-3 px-3 flex flex-row justify-start">
            <a href="https://mecatronica.utn.edu.ec/" rel="home" item-
prop="url">
                <img width="204" height="36" src="https://mecatron-
ica.utn.edu.ec/wp-content/uploads/2019/01/cropped-mecatronica.png"
class="custom-logo" alt="" itemprop="logo"
                srcset="https://mecatronica.utn.edu.ec/wp-content/up-
loads/2019/01/cropped-mecatronica.png 511w, https://mecatron-
ica.utn.edu.ec/wp-content/uploads/2019/01/cropped-mecatronica-300x58.png
300w, https://mecatronica.utn.edu.ec/wp-content/uploads/2019/01/cropped-
mecatronica-360x70.png 360w"
                sizes="(max-width: 511px) 100vw, 511px">
            </a>


        </footer>
    </div>

</section>

<script is:inline src="/canvas-gauges/gauge.min.js"></script>
<script>
    const modal = document.getElementById("widget-modal");
    const widgetContent = document.getElementById("widgetContent");
    const gaugeOptions = document.getElementById("gaugeOptions");
    const lampOptions = document.getElementById("lamp-options");
    const indicatorOptions = document.getElementById("indicator-options");
    const showWidgetButton = document.getElementById("show-widget");
    const gaugeSelectButton = document.getElementById("window_pre-
view_gauge");
    const lampSelectButton = document.getElementById("window_preview_lamp");
    const IndicatorSelectButton = document.getElementById("window_preview_nu-
meric");
    const botonCerrarModal = document.getElementById("cerrarModal");
```

```
    showWidgetButton.addEventListener("click", function () {
        modal.style.display = "block";
        widgetContent.style.display = "block";
        gaugeOptions.style.display = "none";
        lampOptions.style.display = "none";
    });

    botonCerrarModal.addEventListener("click", function () {
        modal.style.display = "none";
    });

    botonCerrarModal.addEventListener("click", function () {
        modal.style.display = "none";
    });

    window.addEventListener("click", function (event) {
        if (event.target === modal) {
            modal.style.display = "none";
        }
    });

    gaugeSelectButton.addEventListener("click", function () {
        gaugeOptions.style.display = "block";
        lampOptions.style.display = "none";
        indicatorOptions.style.display = "none";
        widgetContent.style.display = "none";
    });

    lampSelectButton.addEventListener("click", function () {
        lampOptions.style.display = "block";
        gaugeOptions.style.display = "none"
        indicatorOptions.style.display = "none";
        widgetContent.style.display = "none"
    });

    IndicatorSelectButton.addEventListener("click", function (){
        lampOptions.style.display = "none";
        gaugeOptions.style.display = "none";
        indicatorOptions.style.display = "block";
        widgetContent.style.display = "none";
    });

</script>

<style>
    .modal {
        display: none;
        position: fixed;
        z-index: 1;
        left: 0;
        top: 0;
        width: 100%;
        height: 100%;
        overflow: auto;
        background-color: rgba(0, 0, 0, 0.7);
    }
    .modal-contenido {
        background-color: #fefefe;
```

```
    margin: 15% auto;
    padding: 40px;
    border: 1px solid #888;
    width: 60%;
    border-radius: 1rem;
}
.cerrar {
    color: #aaa;
    float: right;
    font-size: 40px;
    font-weight: bold;
}
.cerrar:hover,
.cerrar:focus {
    color: black;
    text-decoration: none;
    cursor: pointer;
}</style>
```

```
// Sección de Registro de la aplicación, archivo Register.astro
---
import Card from "./Card.astro";
interface Props {
    nameProyect: string;
}

const { nameProyect } = Astro.props;
---

<article id="accountModal" class="modal">
    <div class="modal-contenido text-black">
        <header class="justify-center mx-auto">
            <span class="cerrar" id="closeModal">&times;</span>
            <h3 class="text-2xl">{`Sistema inteligente ${nameProyect}`}</h3>
        </header>

        <main>
            <div
            id="auth-sign"
                class="flex flex-col px-4 py-4 [&>button]:py-4 [&>but-
ton]:border-[3px] [&>button]:rounded-lg [&>button]:font-medium [&>button]:px-
4 [&>button]:backdrop-blur-3xl [&>button]:transition-colors"
            >
                <h2>Firebase Email &amp;amp; Password Authentication</h2>
                <p class="text-xs text-gray-400">
                    Enter an email and password below and either sign in to
an
                    existing account or sign up
                </p>

                <input
                    class="form-control my-1 py-3 px-6"
                    style="display: inline; width: auto"
                    type="text"
                    id="email"
                    name="email"
                    placeholder="Email"
                />

                <input
                    class="form-control my-1 py-3 px-6"
                    style="display: inline; width: auto"
                    type="password"
                    id="password"
                    name="password"
                    placeholder="Password"
                />


                <button
                style=" display: none"
                disabled
                id="quickstart-sign-in-google"
                class="text-white opacity-60 bg-red-900 hover:opacity-100"
                >
                Sign in with Google
                </button>
                <button
```

```
                style=" display: none"
                id="quickstart-sign-up"
                name="signup"
                class="text-gray-800 opacity-70 bg-slate-400 hover:opacity-
100"
                >
                Sign Up
                </button>

            </div>
            <button
            disabled
            id="quickstart-sign-in"
            name="signin"
            class="text-white opacity-60 bg-[#3e6ae1] hover:opacity-100 w-
full border-[3px] rounded-lg font-medium px-4 py-4 backdrop-blur-3xl transi-
tion-colors"
            >
            Sign In
            </button>
            <div
                id="auth-user"
                class="flex flex-row justify-center flex-nowrap [&>but-
ton]:py-4 [&>button]:border-[3px] [&>button]:rounded-lg [&>button]:font-me-
dium [&>button]:px-4 [&>button]:backdrop-blur-3xl [&>button]:transition-col-
ors"
            >
                <button
                    disabled
                    id="quickstart-verify-email"
                    class="text-gray-800 opacity-70 bg-slate-400 hover:bg-
black hover:text-white hover:opacity-100"
                    name="verify-email"
                >
                    Send Email Verification
                </button>
                <button
                    id="quickstart-password-reset"
                    name="verify-email"
                    class="text-gray-800 opacity-70 bg-slate-400 hover:bg-
black hover:text-white hover:opacity-100"
                >
                    Send Password Reset Email
                </button>
                <div class="quickstart-user-details-container flex flex-col
flex-grow flex-nowrap justify-start py-3">
                    <span
                        class="text-green-500"
                        id="quickstart-sign-in-status">Unknown</span
                    >
                    <div>
                        Email:
                        <span id="user-email"></span>
                    </div>
                    <div>
                        Uid:
                        <span id="user-uid"></span>
                    </div>
                    <span id="user-photo"></span>
```

```
                    <details>
                        <summary>
                            Firebase auth <code>currentUser</code> object
value:
                        </summary>
                        <pre><code class="text-xs" id="quickstart-account-de-
tails">null</code></pre>
                    </details>
                </div>

            </div>
        </main>
    </div>

</article>
<script>

    const accountModal = document.getElementById("accountModal");
    const accountButton = document.getElementById("account");
    const closeModalButton = document.getElementById("closeModal");

    accountButton.addEventListener("click", function () {
        accountModal.style.display = "block";
    });

    closeModalButton.addEventListener("click", function () {
        accountModal.style.display = "none";
    });

    window.addEventListener("click", function (event) {
        if (event.target === accountModal) {
            accountModal.style.display = "none";
        }
    });
</script>
<script src="../javascript/auth"></script>
<style>
    .form-control {
        display: block;
        width: 100%;
        font-size: 1rem;
        font-weight: 400;
        line-height: 1.5;
        color: #212529;
        background-color: #fff;
        background-clip: padding-box;
        border: 1px solid #ced4da;
        -webkit-appearance: none;
        -moz-appearance: none;
        appearance: none;
        border-radius: 0.375rem;
        transition:
            border-color 0.15s ease-in-out,
            box-shadow 0.15s ease-in-out;
    }

    .modal {
        display: none;
        position: fixed;
```

```
        z-index: 1;
        left: 0;
        top: 0;
        width: 100%;
        height: 100%;
        overflow: auto;
        background-color: rgba(0, 0, 0, 0.7);
    }

    .modal-contenido {
        background-color: #fefefe;
        margin: 15% auto;
        padding: 20px;
        border: 2px solid #888;
        width: 40%;
        border-radius: 0.75rem;
    }

    .cerrar {
        color: #aaa;
        float: right;
        font-size: 40px;
        font-weight: bold;
    }

    .cerrar:hover,
    .cerrar:focus {
        color: black;
        text-decoration: none;
        cursor: pointer;
    }
</style>
```