ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE UN PROTOTIPO DE SISTEMA DE CONTROL DE ASISTENCIA Y DE HORARIO LABORAL DE UN CENTRO DE DESARROLLO INICIAL

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

CRISTHIAN XAVIER MOROCHO VALLEJO

DIRECTORA: PHD. SORAYA LUCÍA SINCHE MAITA

Quito, octubre 2022

AVAL

Certifico que el presente trabajo fue desarrollado por Cristhian Xavier Morocho Vallejo, bajo mi supervisión.

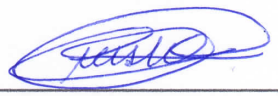
PhD. Soraya Lucía Sinche Maita

DIRECTORA DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Cristhian Xavier Morocho Vallejo, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.



CRISTHIAN XAVIER MOROCHO VALLEJO

DEDICATORIA

Este logro se lo dedico a mis padres, quienes me han acompañado desde el primer día hasta el último. A mi madre, quien siempre se preocupó por mí, me escuchó y me ayudó muchas veces a tomar la mejor decisión. A mi padre, quien ha sido mi gran referente y quien despertó esta vocación en mí desde pequeño. Siempre los llevo en mi mente y mi corazón. A Evelyn, llegó en un momento crucial a mi vida cambiando lo malo por lo bueno.

A mi familia, gracias por el respaldo y por desear que me vaya bien.

Por último, este logro se lo dedico a mi ex profesor del colegio, Ing. Christian Hurtado, quien junto a mi padre me motivaron a seguir esta hermosa carrera. A pesar de que no se encuentre cerca, aun me sigue motivando. Sé que desde el cielo, él estará feliz.

Este logro personal es para ustedes, por ser los pilares fundamentales en todo este proceso.

AGRADECIMIENTO

Extiendo un agradecimiento a mi Padre Celestial que me ha llenado de fuerzas para lograr este objetivo.

A padres, María Beatriz Vallejo y Luis Aníbal Morocho, por ser mi apoyo incondicional en esta etapa de mi vida. Por levantarme, por creer en mí y evitar que renuncie a mis sueños. Muchas gracias por acompañarme en mis buenos y malos momentos y por ser un gran motivo para seguir adelante. Me faltará vida para devolverles todo el cariño y amor que me han brindado y agradecerles lo mucho que han hecho por mí.

A mi compañera de vida, Evelyn Dayana, por motivarme y respaldarme siempre. Muchas gracias por tu compañía en este proceso, por creer en mí y motivarme a ser mejor persona.

A mis compañeros de clase, muchas gracias por tantos momentos buenos y malos que hemos pasado.

A mi directora, muchas gracias por la paciencia y el tiempo invertido en el desarrollo de este trabajo de titulación.

Quedo sumamente agradecido con ustedes por ser parte de este logro tan anhelado por mi persona.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE CÓDIGOS

RESUMEN

El objetivo de este trabajo de titulación es desarrollar un prototipo de sistema de control de asistencia y de horario laboral de un centro de desarrollo inicial, con el fin de solucionar los problemas creados por la falta de control a la entrada y salida de la institución. El prototipo implementado incluye una función capaz de registrar la temperatura corporal del personal adicional a la huella digital, donde los registros recolectados se guardan en una base de datos en la nube. Se implementa una interfaz móvil y una interfaz web para facilitar el acceso y visualización de los datos.

En el capítulo 1 se presentan los objetivos, descripción del alcance del proyecto, y marco teórico.

En el capítulo 2 se muestra el desarrollo del prototipo en base a la metodología. El desarrollo de este prototipo se lo realizó en 3 fases: fase de teórica, fase de diseño y fase de implementación. En la fase teórica se detallan las definiciones y características de las herramientas a utilizar en el prototipo. En la fase de diseño se recolecta información, mediante entrevistas, para obtener los requerimientos del sistema y en base a esto, estructurar el funcionamiento del prototipo. Por último, en la fase de implementación se realiza lo diseñado en la fase anterior.

En el capítulo 3 se presentan los resultados obtenidos por las pruebas de funcionamiento del prototipo. Se detallan las pruebas realizadas a los módulos que dan funcionalidad al sistema.

En el capítulo 4 se muestran las conclusiones con referencia a los resultados obtenidos en las pruebas de funcionamiento. Se adjuntan recomendaciones para futuras modificaciones e implementaciones de este prototipo.

Por último, en los anexos se incluyen las encuestas realizadas para obtener los requerimientos del sistema, códigos de los módulos, descripción de las funciones codificadas, aplicación instalable, manual de usuario, códigos para verificación de funcionamiento y cuestionarios finales de comprobación funcional.

PALABRAS CLAVE: Control de asistencia, Angular, Firebase, Ionic, ESP32.

ABSTRACT

Staff attendance control is an administrative process with the aim of controlling the time of entry, exit and working hours of your employees. It is necessary to carry out this process to avoid tardiness and absenteeism. In this titling work, a prototype capable of recording the attendance and entry/exit time of the staff working in a specific institution is designed and developed. The prototype implemented includes a function to register the body temperature of the staff in addition to the fingerprint, where the collected records are saved in a database in the cloud. A mobile interface and a web interface are implemented to facilitate data access and visualization.

Chapter 1 presents the objectives, description of the project´s scope, and theoretical framework.

Chapter 2 shows the development of the prototype based on the methodology. Its development was carried out in 3 phases: theoretical phase, design phase and implementation phase. In the theoretical phase, the definitions and characteristics of the tools to be used in the prototype are detailed. Then, in the design phase, information is collected, through interviews, to obtain the system requirements and based on this, structure the operation of the prototype. Finally, in the implementation phase, what was designed in the previous phase is carried out.

Chapter 3 presents the results obtained by the prototype performance tests. The tests carried out on the modules that give functionality to the system are detailed.

In the chapter 4, the conclusions with reference to the results obtained in the performance tests are included. Additionally, recommendations for future modifications and implementation of this prototype are attached.

Finally, the annexes include the surveys carried out to obtain the system requirements, module codes, description of the coded functions, installable application, user manual, codes for operation verification, and final functional verification questionnaires.

KEYWORDS: Attendance control, Angular, Firebase, Ionic, ESP32.

1. INTRODUCCIÓN

El control de asistencia de personal, en una determinada empresa, es un proceso administrativo con el objetivo de controlar la hora de entrada, salida y el horario laboral de sus empleados. Este procedimiento es realizado con registros, sea digitales o físicos, de las horas que trabaja todo empleado en la empresa. La falta de control del personal podría generar casos de impuntualidad y ausentismo; lo que ocasionaría problemas con la productividad, reducción en la calidad de servicio y falta de compromiso laboral.

Se ha propuesto el desarrollo de un prototipo capaz de registrar la asistencia y hora de entrada/salida del personal que trabaja en una determinada institución. Los registros recolectados se guardarán en una base de datos en la nube para poder acceder a ellos mediante conexión a Internet. Se implementará una interfaz móvil y una interfaz web para facilitar el acceso y visualización de los datos. Cada interfaz permitirá mostrar datos e interactuar con los mismos dependiendo del perfil o rol de cada usuario autenticado el mismo que puede ser Administrador o Empleado. Además, se propone implementar en el prototipo, una función capaz de registrar la temperatura corporal del personal y evitar que se disponga del tiempo laboral de un determinado empleado para cumplir dicho procedimiento.

La institución en donde se implementará el prototipo y se realizarán las pruebas de funcionamiento es el centro de desarrollo inicial “Cielo Infantil” (CDI-CI). Este centro es una institución particular que brinda servicios de educación inicial, preescolar y ofrece servicios de tareas dirigidas, tutorías y refuerzo académico.

1.1. OBJETIVOS

1.1.1. OBJETIVO GENERAL

Desarrollar un prototipo de sistema de control de asistencia y de horario laboral de un centro de desarrollo inicial.

1.1.2. OBJETIVOS ESPECÍFICOS

- Analizar los dispositivos, herramientas, lenguajes de programación y tecnologías para el desarrollo del sistema.
- Diseñar los módulos que intervendrán en el sistema de control del personal.
- Implementar los módulos siguiendo el diseño planteado para dar funcionamiento al sistema propuesto.
- Validar el correcto funcionamiento del sistema distribuido desarrollado.

1.2. ALCANCE

La estructura del prototipo se describe mediante la Figura 1.1.

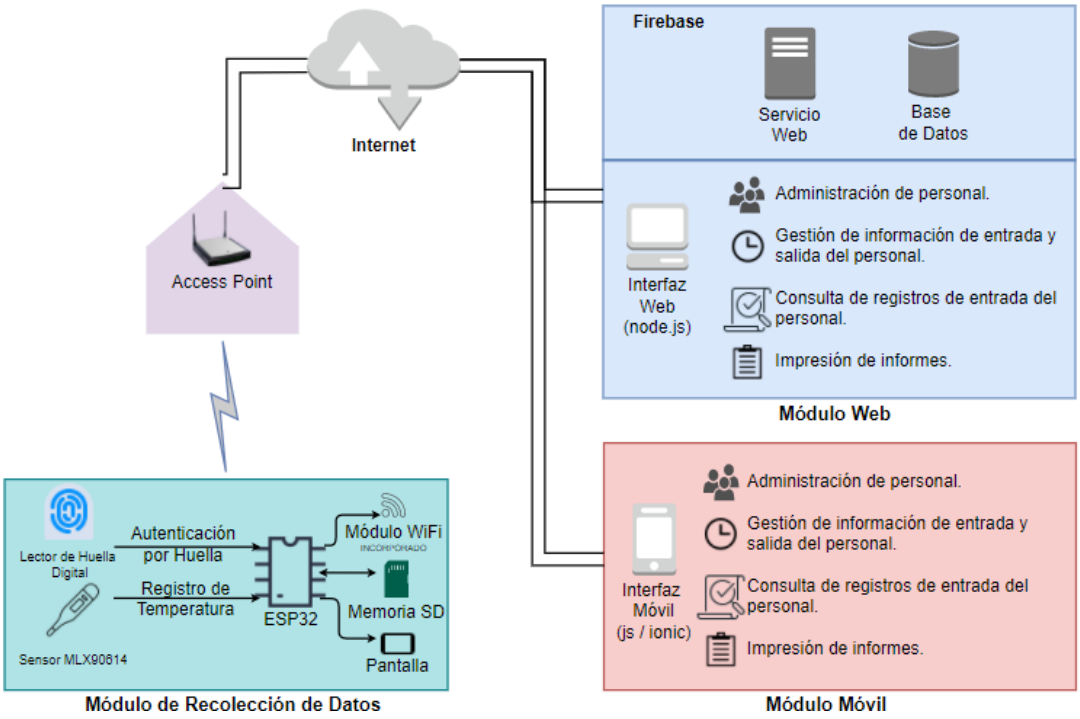


Figura 1.1. Diagrama del prototipo planteado.

El prototipo consta de tres módulos:

- **Módulo de Recolección de Datos**

Este módulo será el encargado de identificar al personal y registrar su entrada o salida así como su temperatura corporal, procesar los datos obtenidos y enviarlos hacia el Internet.

Estará compuesto por una placa desarrollada por la empresa ESPRESSIF modelo ESP32, una pantalla, una memoria microSD.

Se conectarán dos entradas a la placa ESP32 para la adquisición de datos:

- Módulo de lector de huella dactilar (Modelo: universal).
- Sensor para registrar temperatura corporal (Modelo: Mlx90614, Sensor infrarrojo para registrar temperatura sin contacto).

Se conectará una pantalla en donde se mostrará el nombre, hora de ingreso/salida y la temperatura del usuario que se ha identificado mediante la lectura de huella dactilar.

Se realizará una conexión *wireless* entre la placa ESP32 (consta de un módulo Wifi integrado) y la red inalámbrica de la institución para acceder a los servicios que estarán

alojados en el Internet. Los servicios permitirán procesar y almacenar los datos tomados por este módulo.

El módulo SD permitirá almacenar datos localmente en caso de que no se disponga de conexión a Internet; los datos se enviarán cuando se retome la conexión.

- **Módulo Web**

El Módulo Web constará de servicios alojados en la nube diseñados en base a un patrón de diseño MVC (Modelo – Vista - Controlador). Permitirá el acceso remoto a los datos, funcionalidades e interfaz de usuario para la gestión del sistema.

Para implementar este módulo, se utilizarán los servicios de Firebase (*hosting*, base de datos, funciones *backend*) que interactuarán con una interfaz web de usuario desarrollada en Node.JS mediante el IDE Visual Studio Core.

La interfaz de usuario permitirá acceder vía web a las funcionalidades del sistema, las cuales son:

- Administración del personal.
- Gestión de información recolectada.
- Visualización de datos de usuario.

Además, se definirán dos roles para el personal: Administrador y Empleado. A continuación, se detalla la funcionalidad de cada rol:

- **Rol: Administrador**

En el rol *Administrador* se podrá realizar una gestión CRUD (*Create*, *Read*, *Update*, *Delete*) de los usuarios (personal) con la capacidad de configurar horas laborales y el tipo de rol de usuario. Además, se podrá acceder a toda la información recolectada para visualizarla mediante filtros. La información que se podrá visualizar será la siguiente:

  - Información de asistencia del personal.
  - Cantidad de horas cumplidas.
  - Cantidad de horas extras.
  - Justificación de horas extras.
  - Información de hora entrada y salida.
  - Registro de temperatura.

El usuario con rol “Administrador” podrá elegir el tipo de información que desea visualizar en base a cada empleado, para así, crear un reporte personalizado y orientado a las necesidades del usuario.

- **Rol: Empleado**

El usuario con rol *Empleado* podrá únicamente visualizar los datos relacionados con su personal.

- **Módulo Móvil**

El Módulo Móvil será una aplicación que permitirá realizar las mismas funciones que la interfaz en el Módulo Web. Dicha aplicación se instalará en el teléfono móvil de cada empleado de la institución. Será desarrollada en *JavaScript* usando el *framework* Ionic y Angular. Se codificará en el IDE Visual Studio Code.

Funcionamiento

En primer lugar, se realizará el registro del personal en la base de datos mediante el uso de la interfaz de usuario. Cuando se crea un usuario en el sistema, se deberá registrar su respectiva huella dactilar desde el Módulo de Recolección de Datos.

El Módulo de Recolección de Datos estará instalado en un lugar estratégico de la institución previamente determinado. Será capaz de identificar al personal mediante la lectura de huella dactilar, mostrar su nombre en pantalla y solicitar que se acerque al sensor de temperatura corporal para registrar su lectura; el módulo estará conectado a la red por conexión *wireless* y procederá a registrar la hora de entrada en la memoria SD y en la base de datos del Módulo Web, siempre y cuando haya registrado una lectura de temperatura corporal válida.

En caso de que, en el momento que el módulo deba registrar la información en la base de datos no exista conexión a Internet, el Módulo de Recolección de Datos esperará retomar conexión y así subir los datos pendientes que estarán almacenados en memoria. Cabe recalcar que, el Módulo de Recolección de Datos debe funcionar independientemente del Módulo Web, es decir, será capaz de identificar al personal (únicamente sí se encuentra registrado) sin necesidad de tener conexión a la red.

Los usuarios deberán autenticarse, mediante el uso de una de las interfaces visuales (web o móvil), para acceder a las funciones y los datos que el Módulo Web ofrece. Si el usuario autenticado es *Administrador*, el módulo permitirá la función de administrar y gestionar los usuarios en el sistema y podrá visualizar toda la información registrada en la base de datos. Por otra parte, si el usuario autenticado es *Empleado*, el módulo permitirá visualizar únicamente la información registrada por el usuario identificado.

En el Módulo Móvil se tendrán las mismas funciones del Módulo Web en una aplicación que el usuario podrá instalar en un dispositivo móvil.

En caso que el personal haya trabajado horas extras, deberá haber una función que permita justificar el motivo de la extensión de su jornada laboral.

Para el desarrollo del prototipo, se usará la metodología Kanban para definir flujos de tareas en cada fase de desarrollo del proyecto. Se estudiará la estructura de la institución CDI-CI. Se determinará mediante entrevistas las actividades y funcionalidades que debe realizar el prototipo.

Una vez desarrollado el prototipo, se realizarán pruebas de funcionamiento del sistema completo para verificar el cumplimiento de los requerimientos establecidos.

El presente trabajo de titulación brinda un producto final demostrable.

1.3. MARCO TEÓRICO

1.3.1. SITUACIÓN ACTUAL DEL CONTROL DE ASISTENCIA EN CDI-CI

Analizando el caso de esta institución, la cual no dispone de algún control en específico del personal, el retraso en la llegada del personal implica, por momentos, que las actividades se distribuyan entre el personal docente en la institución, creando un problema de organización entre ellos; por ejemplo: si un determinado docente está ausente al inicio de la jornada, otra persona debe cubrir su lugar para organizar a los niños, un problema que se refleja reiteradamente en las jornadas laborales y limita la organización del personal en la institución.

Además, al no disponer de un control de registro de salida del personal, existe un inconveniente en el cálculo del pago de horas extras. Cuando un empleado haya extendido su horario laboral, deberá notificar al personal administrativo la causa de este hecho para la contabilidad de horas extra. Actualmente, al momento del pago, se realiza el cálculo en base a un conteo aproximado de horas extra justificadas. El problema radica en que el valor que debe pagar la institución al empleado, en ocasiones puede ser menor o mayor, desfavoreciendo económicamente a una de las partes.

Por otra parte, debido a la pandemia del COVID-19, al momento que el personal ingresa a la institución debe cumplir con el protocolo de bioseguridad establecido; para esto, se toma la temperatura corporal de la persona y se procede a realizar la desinfección adecuada. Ya que no se dispone de guardianía o portería para realizar dicho proceso, un docente delegado por turno debe encargarse de verificar la temperatura corporal a cada empleado que ingrese; la desinfección por otro lado, es personal. Por lo tanto, es un proceso que toma tiempo en cada empleado, pues se añade a sus obligaciones laborales e influye en la organización al inicio de la jornada.

1.3.2. MODELO MVC

Model-View-Controller (MVC) es un patrón arquitectónico de diseño de software. Divide las responsabilidades en tres roles principales (Figura 1.2), lo que permite una colaboración más eficiente. Estos roles son: el desarrollo descrito por el Modelo, el diseño descrito por la Vista y la integración descrito por el controlador [1]. Además, el comportamiento específico de MVC describe al ciclo natural de una aplicación: el usuario realiza una acción y, en respuesta, la aplicación cambia su modelo de datos y ofrece una vista actualizada al usuario, el ciclo se repite. Ésta es una opción muy conveniente para las aplicaciones web entregadas como una serie de solicitudes y respuestas [2].

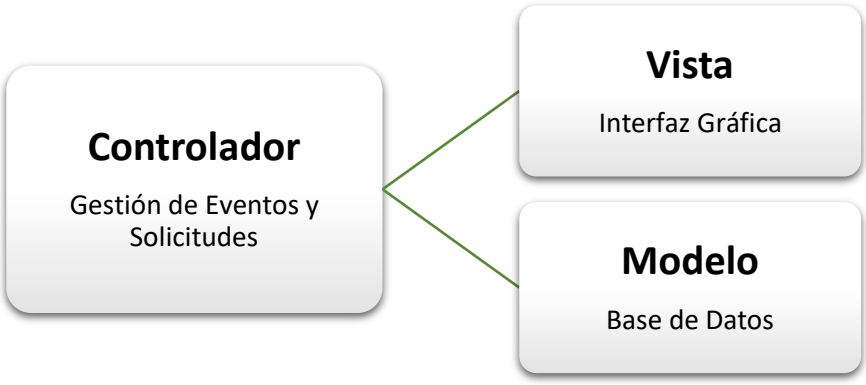


Figura 1.2. Patrón de diseño MVC [2].

1.3.2.1. Modelo

Es la parte del sistema que gestiona todas las tareas relacionadas con los datos: validación, estructura de almacenamiento de datos, gestión de base de datos [2]. El modelo simplifica la complejidad del código, ya que encapsulará métodos o funciones para acceder a los datos, y pondrá a disposición una biblioteca de clases reutilizable [1]. El Modelo maneja principalmente la abstracción y validación del acceso a datos.

1.3.2.2. Vista

Es responsable de la gestión de la interfaz gráfica de usuario, es decir, los formularios, botones, elementos gráficos y todos los elementos presentes en la interfaz con la que el usuario puede interactuar. Al separar el diseño de la lógica de la aplicación, se reduce en gran medida el riesgo de que aparezcan errores cuando se requiere hacer cambios en una interfaz [1]. Controla la forma en que se muestran los datos y cómo el usuario interactúa con ellos, así como también proporciona formas de recopilar datos de los usuarios. Las tecnologías que se utilizan principalmente en las vistas son HTML (*HyperText Markup Language*), CSS (*Cascading Style Sheets*) y JavaScript (o TypeScript). La vista es parte del *FrontEnd*.

1.3.2.3. Controlador

Un controlador gestiona la relación entre una vista y un modelo. El controlador es responsable del manejo de eventos, estos pueden ser desencadenados desde la vista por un usuario que interactúa con la aplicación [1]. Acepta solicitudes desde la vista, accede a las funciones del Modelo, interpreta datos recibidos por el Modelo y prepara el formato y los datos para una respuesta hacia la vista. En la Figura 1.3 se muestra el ciclo de ejecución del patrón MVC, en donde empieza con un evento en la vista, el controlador recepta el evento y solicita actualizaciones desde el modelo. La respuesta del modelo vuelve a ser interpretada por el controlador y enviada hacia la vista.

Figura 1.3. Ciclo de ejecución MVC [2].

1.3.3. SISTEMAS DE CONTROL DE ASISTENCIA

Actualmente, se usan diferentes tipos de sistemas electrónicos para la recolección de datos de asistencia y hora de entrada/salida del personal. Se tienen varios métodos para realizar este proceso, algunos de los más comunes son [3]:

- Sistemas Declarativos: Los empleados suelen acceder a un sistema para registrar cuando inicia y termina su jornada.
- Sistemas de Fichaje: Los empleados poseen una tarjeta (ficha) única. La usará para registrar su hora de entrada/salida por un lector.
- Sistemas Biométricos: Recolecta información en base a la biometría de cada persona (rostro, huellas digitales, entre otros).

En este trabajo de titulación se usarán sistemas biométricos basados en las huellas digitales. Para el usuario final, este sistema será fácil de usar ya que no requiere un dispositivo extra (como en los sistemas de fichajes) y genera un registro automáticamente, sin necesidad de ingresar información extra, al identificar un usuario mediante su huella dactilar.

2. METODOLOGÍA

2.1. FASE TEÓRICA

En esta sección se detallará el escenario donde se va a realizar el proyecto. Se analizarán los materiales y herramientas que se necesitarán para llevar a cabo el desarrollo de cada uno de los Módulos propuestos en el alcance.

2.1.1. INSTITUCIÓN CENTRO DE DESARROLLO INFANTIL "CIELO INFANTIL"

El centro de desarrollo inicial "Cielo Infantil" (CDI-CI) es una institución particular, que brinda servicios de educación inicial y preescolar a niños y niñas desde los 3 meses de nacidos hasta los 5 años. Además, ofrece servicios de tareas dirigidas, tutorías y refuerzo académico. El CDI-CI garantiza sus servicios en base a profesionales calificados con experiencia en la educación inicial y el cuidado maternal. Se usarán los datos de esta institución para el desarrollo del presente trabajo de titulación [4].

La estructura organizacional del CDI-CI describe un modelo jerárquico. Está conformado por: personal profesional administrativo, personal de aseo y limpieza, personal de cocina, personal docente y personal para transporte. En la Figura 2.1 se describe su distribución organizacional en base a la información brindada por el personal administrativo (gerencia) de la institución. La nómina de sus empleados se describe en la Tabla **2.1**.

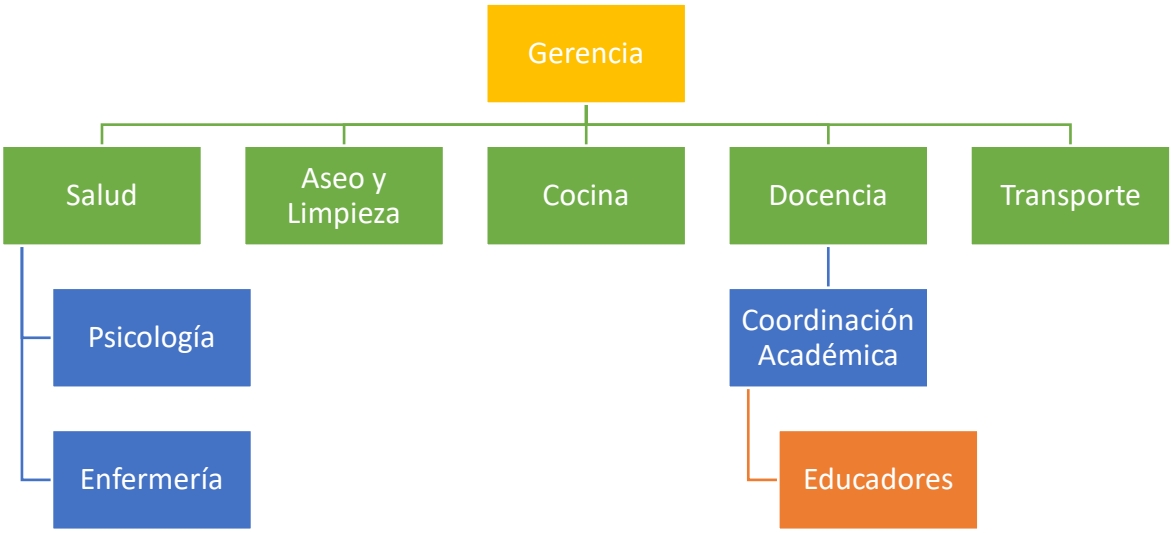


Figura 2.1. Estructura Organizacional de la Institución CDI-CI.

El horario de atención de la institución es de 7:00 AM a 6:00 PM. La mayoría de los niños ingresan a la institución entre las 7:00 y 7:30 AM. La hora de salida del personal docente depende de la hora en que los padres retiren a sus niños. Se tiene un horario flexible con el riesgo de aumentar horas laborales.

Tabla 2.1. Distribución del personal de CDI-CI

Departamento	Personal	Horario
Gerencia	1 gerente	Tiempo completo
Salud	1 psicólogo	Por horas
	1 enfermera	Por horas
Aseo y Limpieza	1 empleado	Medio tiempo
Cocina	1 cocinera	Tiempo Completo
Docencia	6 parvularios/as	Tiempo Completo (4)
		Medio Tiempo (2)
Transporte	1 conductor	Trabajo por horas

2.1.2. HERRAMIENTAS A UTILIZARSE EN EL DISEÑO DE LOS MÓDULOS

En esta sección se mencionarán las herramientas que intervienen en el diseño e implementación de los módulos. Estos módulos se encuentran definidos en el alcance en la Figura 1.1.

2.1.2.1. Hardware para recolección de datos

En esta sección se mencionarán los elementos que intervienen en el proceso de recolección de datos, así como sus características y la forma en la que se comunican con la placa de desarrollo planteada en el alcance. En la Figura 2.2 se muestra la estructura de este Módulo.

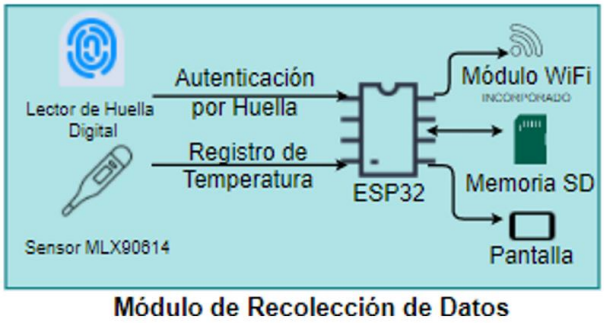


Figura 2.2. Estructura del Módulo de Recolección de Datos.

2.1.2.1.1. Placas ESP32

ESP32 es una familia de placas de desarrollo producidas por Espressif Systems. Están compuestas por un módulo microcontrolador basado en un chip SoC (*System on a chip*) y dispositivos complementarios tales como reguladores, sensores y programador. Estos son embebidos en una placa PBC (*Printed Circuit Board*) denominada ESP32-DevKit [5]. La placa de desarrollo se muestra en Figura 2.3.

Las placas ESP-32 son de bajo consumo, diseñadas para aplicaciones móviles, aplicaciones portátiles y aplicaciones IoT (*Internet of Things*).

La placa de desarrollo brinda una alta compatibilidad gracias a sus módulos integrados en su circuito impreso (PCB). Contiene sensores, entradas y salidas de propósito general, reguladores, amplificadores de potencia, CAD (convertidor analógico - digital), CDA (convertidor digital - analógico), filtros y antenas embebidos en su placa. Entre sus características destaca su sistema de comunicación inalámbrica integrado, capaz de conectarse a redes Wifi y a dispositivos *Bluetooth*.

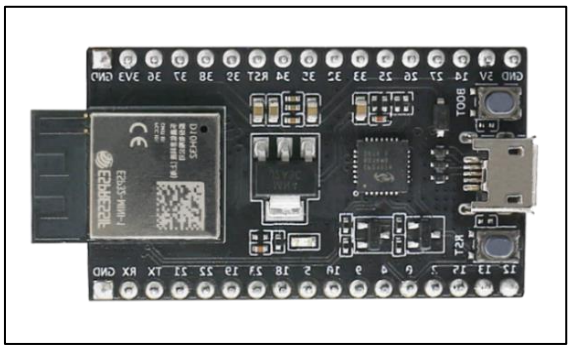


Figura 2.3. ESP-32 DeviKitC [6].

Además, se han desarrollado librerías y bibliotecas que permiten la compatibilidad entre módulos externos diseñados para placas Arduino. Con esto, brinda integración con las placas de desarrollo Arduino y es posible programarlo en el IDE Arduino con C++. En la Tabla 2.2 se observan algunas características de los SoCs usados como módulo microcontrolador en las placas de desarrollo ESP32.

Tabla 2.2. Características de chips SoCs [7].

SoC	Core	Dimensiones (mm)	Pines	Memoria (KB)	Flash (MB)
ESP32-D0WD-V3	Dual	Encapsulado QFN (*Quad Flat No-Leads*) 5x5	48	SRAM: 520	No
ESP32-D0WD	Dual				No
ESP32-D0WDQ6-V3	Dual			ROM: 448	No
ESP32-D2WD	Dual				2
ESP32-U4WDH	Single			RTC SRAM: 16	4
ESP32-S0WD	Single				No

En la Tabla 2.3 se observan algunas características de los diferentes módulos microcontroladores que están presentes en las placas de desarrollo ESP32. Mientras que en la Tabla 2.4 se observan algunas características de las placas de desarrollo ESP32 más comerciales.

Tabla 2.3. Características de módulos ESP32 [8].

	ESP32 WROOM-32E (UE)	ESP32 WROVER-E (IE)	ESP32 MINI-1	ESP32 SOLO-1	ESP32 WROOM-32SE
CPU y Memoria					
Core	ESP32-D0WD-V3	ESP32-D0WD-V3	ESP32-U4WDH	ESP32-S0WD	ESP32-D0WD
Frecuencia	240 MHz	240 MHz	160 MHz	160 MHz	240 MHz
Flash (MB)	4/8/16	4/8/16	4 (embebido)	4	4/8/16
Periféricos					
Interfaces	Tarjeta SD, UART, SPI, SDIO, I^2C, PWM, Motor PWM, I^2S, IR, GPIO, contador de pulsos, sensor *touch* capacitivo, sensor hall, ADC, DAC				
General					
Temperatura de Operación	Min: - 40 °C Max: + 85 °C				
Antena	PCB / IPEX	PCB / IPEX	PCB	PCB	PCB
Dimensiones (mm)	18×25.5×3.1 18×19.2×3.2	18×31.4×3.3	13.2×19×2.4	18×25.5×3.1	18×25.5×3.1
Pines	38	38	55	38	38
Protocolo Wifi	IEEE 802.11 b/g/n (2.4 GHz)				
Protocolo Bluetooth	Bluetooth V4.2, Bluetooth LE (Low Energy)				

Tabla 2.4. Características de ESP32-Devkits [9].

	ESP32-DevKitC	ESP32-DevKitM-1	ESP-WROVER-KIT
Descripción	ESP32-DevKitC es una placa de desarrollo de nivel de entrada. Tiene todos los pines ESP32 expuestos y es fácil de conectar y usar.	ESP32-DevKitM-1 es una placa de desarrollo basada en el módulo ESP32-MINI-1.	La placa incluye un interfaz USB que permite a los desarrolladores usar JTAG directamente para depurar el módulo ESP32 a través de la interfaz USB. Es usado para proyectos de alto rendimiento
Flash (MB)	4/8	4	4 MB Flash + 8 MB PSRAM
Interfaces físicas	I/O, USB	I/O, USB	I/O, JTAG, USB, Cámara, UART, SPI, MicroSD
Interfaces de interacción	Botones y LEDs	Botones y LEDs	Pantalla LCD, Botones, LEDs RGB
Módulos	ESP32-WROOM-32E, ESP32-WROOM-32UE	ESP32-MINI-1	ESP32-WROVER-E

2.1.2.1.2. ESP-DeviKitC

Para la implementación de este prototipo, se usará la placa de desarrollo ESP-DevKitC V4. En la Figura 2.4, se muestran las dimensiones del módulo ESP32-DevKitC v4.

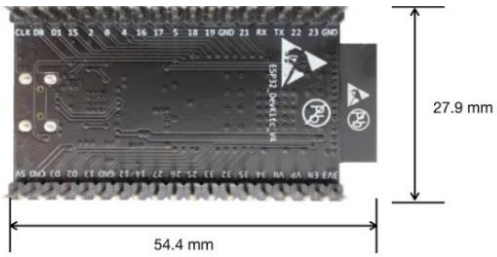


Figura 2.4. Dimensiones de ESP32-DevKitC V4 [6].

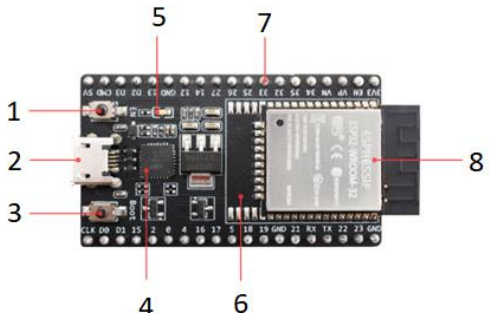


Figura 2.5. Pines de ESP32-DevKitC V4 [6].

En la Figura 2.5, se observa la placa de desarrollo. A continuación, se detallan sus partes:

1 → Botón *Enable*: para reinicio.

2 → Puerto micro USB: sirve como fuente de alimentación de la placa, así como interfaz de comunicación entre la placa y la computadora para programación.

3 → Botón *Boot*: para carga de datos.

4 → Chip puente de USB a UART.

5 → Led de encendido.

6 → Espacio opcional para ESP32-WROVER.

7 → Pines: están divididos en los bordes de la placa.

8 → Módulo microprocesador ESP-WROOM-32E.

2.1.2.1.3. ESP-WROOM-32E

ESP32-WROOM-32E es un módulo microcontrolador que incluye conectividad Wifi, *Bluetooth* y *Bluetooth Low Energy*. Puede trabajar con aplicaciones que van desde redes de sensores de baja potencia hasta las aplicaciones como codificación de voz, transmisión de música y decodificación de MP3. Integra una antena de PCB para conectividad inalámbrica.

Tabla 2.5. Características del módulo microprocesador ESP32-WROOM-32E [10].

CHIP	
Modelo	SoC ESP32-D0WD-V3
Core	Dual Core
Frecuencia	240 MHz
MEMORIA	
SRAM	520 KB
ROM	440 KB
RTC SRAM	16 KB
WIFI	
Estándar	IEEE 802.11b/g/n
Intervalo de Guarda	0.4 µs
Frecuencia de Canal	2412 ~ 2484 MHz
BLUETOOTH	
Bluetooth V4.2	
Bluetooth Low Energy	
HARDWARE	
Interfaces	18 convertidores *Analog-to-Digital* (ADC)
	3 interfaces *Serial Peripheral Interface* (SPI)
	3 interfaces *Universal Asynchronous Receiver-Transmitter* (UART)
	2 interfaces *Inter-Integrated Circuit* (I2C)
	16 salidas *Pulse Width Modulation* (PWM)
	2 convertidores *Digital-to-Analog* (DAC)
	2 interfaces *Inter-IC Sound* (I2S)
	10 GPIO (*General Purpose Input/Output*)
Cristal Oscilador	40 MHz
Flash	4 KB
Regulador de Voltaje	3.3 V
Voltaje de Operación	3.0 ~ 3.6 V
Temperatura de Operación	-40 a 85 °C

El módulo se basa en el chip ESP32-D0WD-V3, tiene dos núcleos de procesador que pueden funcionar individualmente, se puede ajustar la velocidad del procesador de 80 MHz a 240 MHz. Este chip también tiene un procesador de baja potencia que puede reemplazar al CPU en aplicaciones que requieren ahorro de energía. El sistema operativo que brinda funcionamiento al módulo es *freeRTOS* [10]. En la Tabla 2.5 se detallan las características principales del módulo microprocesador.

2.1.2.1.4. Protocolos de comunicación en esp32

En esta sección se mencionarán los protocolos que se usarán para la comunicación entre los módulos y la placa de desarrollo que se implementará en el prototipo.

UART (Universal Asynchronous Receiver-Transmitter)

UART (*Universal Asynchronous Receiver-Transmitter*) es un protocolo de comunicación simple que permite la comunicación con dispositivos en serie de forma asincrónica. Usa los pines digitales RX y TX para realizar la transferencia de datos [11]. En la Figura 2.6, se muestra la interconexión entre dispositivos mediante el protocolo UART.

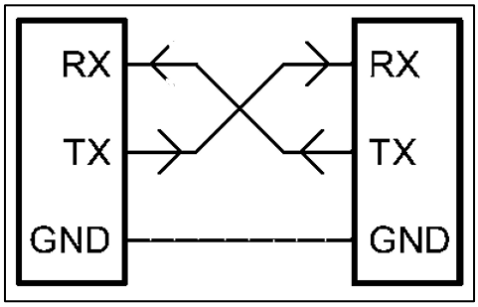


Figura 2.6. Comunicación UART [11].

I2C (Inter Integrated Circuits)

I2C (*Inter Integrated Circuits*) es un protocolo de comunicaciones basado en un esquema maestro-esclavo, consta de un bus de dos líneas que permite conectar dispositivos de baja velocidad. En la Figura 2.7, se muestra el bus de conexión I2C con sus respectivos dispositivos. Se utiliza principalmente para la comunicación entre microcontroladores, módulos y sensores, permite conectar hasta 112 dispositivos [12].

El bus de comunicaciones del protocolo tiene dos líneas, las cuales son:

- SDA (*Serial Data Line*): para el intercambio de datos en serie.
- SCL (*Serial Clock Line*): señal de reloj (100 KHz).

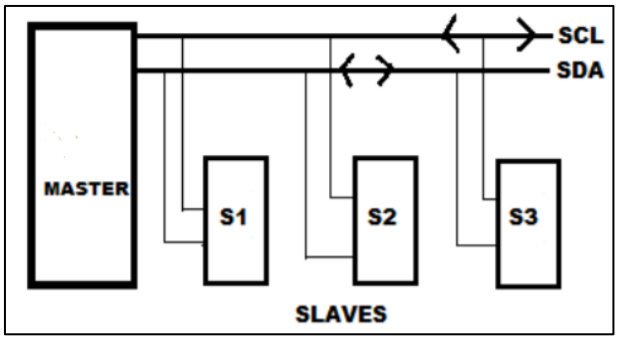


Figura 2.7. Comunicación I2C [12].

El protocolo permite múltiples maestros y esclavos, donde el primer dispositivo que desea transmitir se convierte en maestro. La comunicación entre ellos se basa en direcciones. Cada dispositivo esclavo que trabaja con este protocolo tiene preconfigurada una dirección de 7 bits, única para cada tipo de módulo I2C. Se utiliza un sistema de direcciones y un bus compartido, lo que permite que muchos dispositivos pueden conectarse al mismo bus.

Una vez que se establece el módulo maestro del bus, todos los demás dispositivos se consideran esclavos. El maestro empezará la comunicación notificando que hará uso del canal, los dispositivos escucharán en busca de solicitudes entrantes. Luego, el maestro envía la dirección del dispositivo al que quiere acceder. Una vez recibida la dirección, todos dispositivos conectados al bus, la compararán con su propia dirección. Si no coincide, esperan hasta que el bus se libere. Si la dirección coincide, el dispositivo producirá una respuesta hacia el maestro para empezar a transmitir y recibir datos [13].

SPI (Serial Peripheral Interface)

SPI (*Serial Peripheral Interface*) es un protocolo de comunicaciones serial, tipo maestro-esclavo, usado para la comunicación entre microcontroladores y módulos que requieren velocidades altas. SPI permite un solo dispositivo maestro, como se muestra en la Figura 2.8, esto significa que un dispositivo central inicia todas las comunicaciones con los esclavos. Es aplicado en circuitos que requieren alta velocidad o cuando la información se actualiza y cambia rápidamente.

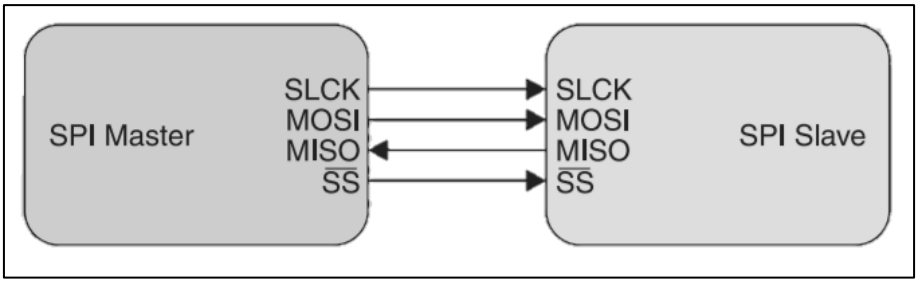


Figura 2.8. Comunicación SPI [13].

- SCLK (*Serial Clock*): señal de reloj a 20 MHz enviada desde el bus maestro a todos los esclavos, la comunicación SPI es sincrónica mediante esta señal de reloj.
- SS (*Slave Select*): señal para seleccionar el esclavo con el que se comunicará el maestro.
- MOSI (*Master Out – Slave In*): línea de datos desde el maestro a los esclavos.
- MISO (*Master In – Slave Out*): línea de datos de los esclavos al maestro.

Cuando el maestro SPI desea enviar datos a un esclavo o solicitarle información, selecciona un esclavo mediante la correspondiente línea SS. El maestro genera información en la línea MOSI mientras toma muestras de la línea MISO [13].

2.1.2.1.5. Lector de huellas digitales

Los lectores de huellas digitales se aplican en sistemas de control de acceso, dispositivos móviles, laptops y aplicaciones que requieren autenticación para su acceso. Estos usan sensores para identificar la biometría de la huella, entre los que se tiene:

Sensor Capacitivo

Su funcionamiento es similar a una pantalla táctil, ya que identifica la diferencia de carga entre la pantalla y la piel de los dedos. Consta de paneles capacitivos que pueden identificar las crestas papilares de los dedos que forman la huella digital. Al colocar el dedo sobre el sensor, la diferencia de carga entre los paneles permite saber qué parte del dedo toca el sensor; recreando una imagen digital de la huella dactilar [14].

Sensor Óptico

Es el más utilizado en aplicaciones de escritorio y aplicaciones electrónicas, gracias a su alta compatibilidad. El sensor emite luz que choca sobre la superficie de la huella digital, un lector captura y reconstruye imágenes de huellas digitales [14].

Sensor Ultrasónicos

Son sensores que cuando se coloca el dedo en el sensor, se emiten pulsos ultrasónicos que rebotan en toda la superficie del dedo; la imagen de la huella se digitaliza con varios valores tomados por el sensor [14]. En la Tabla 2.6 se observan algunas especificaciones de un sensor óptico y un capacitivo.

Tabla 2.6. Especificaciones de sensores de huella dactilar [15].

Sensor	***Optical Sensor***	***UPEK's/TouchChip***
Tipo de sensor	Óptico	Capacitivo
Resolución	500 dpi	508 dpi
Área de Sensor (mm)	16 × 19	12.8 × 18
Tamaño imagen (pixel)	280 × 320	256 × 360

Para este prototipo se usará un lector de huellas digitales basado en sensor óptico, ya que es más común en aplicaciones de electrónica, existe compatibilidad con los módulos a usar y está disponible en el mercado. Este tipo de lector almacena en su memoria hasta 162 huellas dactilares a las cuales se le asigna un identificador numérico [16].

2.1.2.1.6. Sensor de temperatura corporal

El sensor que se usa para medir la temperatura corporal en este prototipo es el MLX90614. Es un termómetro infrarrojo para la medición de temperatura sin contacto con el objeto. En la **Figura 2.9** se muestra la distribución de pines del sensor.

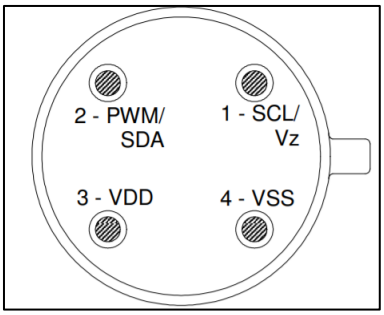


Figura 2.9. Diagrama de pines de MLX90614 [17].

En la Tabla 2.7 se muestra la descripción de los pines y algunas características del módulo basado en el sensor MLX90614 para Arduino.

Tabla 2.7. Pines y características de MLX90614 [17].

MLX90614		
PINES	1 → Pin SCL para la comunicación I2C.	
	2 → Pin SDA para la comunicación I2C, también funciona como salida PWM del valor de lectura del termómetro	
	3 → Pin Voltaje de alimentación 5v.	
	4 → Pin para GND	
CARACTERÍSTICAS	Tamaño pequeño, fácil de integrar y de bajo costo.	
	Interfaz de comunicación: I2C.	
	Rango de temperatura	-40 a 125 ˚C para temperatura ambiente.
		-70 a 380 ˚C para temperatura por infrarrojo.
	Resolución de medición de 0,02 ° C	
	Salida PWM para lectura continua	
	Modo de ahorro de energía	
	Integración con placas de desarrollo	

2.1.2.1.7. Memoria SD

Para el almacenamiento de los datos recolectados, se usará un módulo de memoria microSD para Arduino, compatible con el ESP32. Utiliza comunicación SPI para leer y escribir datos en la memoria [18]. En la Figura 2.10 se observa el módulo con sus respectivos pines.

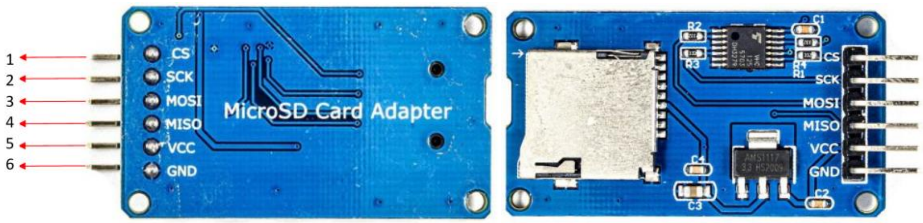


Figura 2.10. Módulo para tarjeta microSD [19].

La asignación de pines se muestra en la Tabla 2.8.

Tabla 2.8. Distribución de pines del módulo SD.

#	Pin	Descripción
1	CS	Pin de señal de selección de chip
2	SCK	para interfaz *Serial Peripheral Interface*
3	MOSI	para interfaz *Serial Peripheral Interface*
4	MISO	para interfaz *Serial Peripheral Interface*
5	VCC	Alimentación 5 V
6	GND	Tierra

Las características del módulo son [19]:

- Admite tarjetas Micro SD (hasta 2 GB)
- Admite tarjetas Micro SDHC (hasta 32 GB)
- Voltaje de alimentación: 5 V
- Regulador de voltaje a 3.3 V
- Corriente de entrada: de 0.2 mA a 200 mA
- Interfaz: *Serial Peripheral Interface* (SPI)
- Número de pines: 6 pines
- Dimensiones (mm): 42 x 24 x 12

2.1.2.1.8. Pantalla

Es un dispositivo orientado a la interacción con el usuario, ya que permite mostrar información relacionada con el sistema. Para proyectos electrónicos, existen módulos (con pantalla embebida) que se integran con placas de desarrollo más comunes [20]. Algunos de estos módulos se mencionan a continuación.

Pantalla LCD Hitachi HD44780

Es un módulo creado por Hitachi. Consta de una pantalla LCD (*Liquid Crystal Display*) monocromática con su respectivo controlador (Figura 2.11). Se tiene en versiones de 2 líneas con 16 caracteres y 4 líneas con 20 caracteres [21].

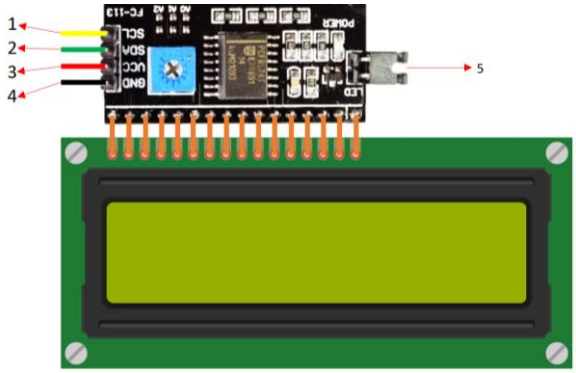


Figura 2.11. Pantalla LCD con módulo I2C [22].

En la Tabla 2.9 se muestra la distribución de pines y algunas características importantes de este módulo.

Tabla 2.9. Pines y características del módulo LCD [22].

LCD Hitachi HD44780	
Pines	1 → Pin SCL para comunicación I2C.
	2 → Pin SDA para comunicación I2C
	3 → Alimentación VCC para 5 V
	4 → GND
	5 → Acople para Luz de fondo
Características	Integración con IDE de Arduino
	Puntos por carácter: 5 × 10
	Rango de voltaje de alimentación: 3 - 11 V
	Interfaz I2C para comunicación

Pantalla LCD OLED I2C

Figura 2.12. Pantalla OLED I2C [23].

La pantalla LCD OLED tiene pixeles para formar caracteres, cada pixel puede tomar un color en particular dependiendo como se lo configura. Se conecta a la placa de desarrollo mediante el protocolo I2C. Existen algunas librerías que facilitan el uso de este módulo; son compatibles con el IDE de Arduino. En la Figura 2.12 se observa la pantalla con sus respectivos pines.

En la Tabla 2.10 se muestra la distribución de pines y algunas características importantes de este módulo.

Tabla 2.10. Pines y características del módulo LCD OLED I2C [23]**.**

LCD OLED I2C	
Pines	1 → GND
	2 → Alimentación VCC para 5 V
	3 → Pin SCL para comunicación I2C.
	4 → Pin SDA para comunicación I2C
Características	Integración con IDE de Arduino
	Tamaño (pixeles): 128 x 64
	Controlador incorporado para comunicación I2C
	Fuente de alimentación de: 3 V

Para este prototipo, se usará un módulo basado en pantalla LCD, debido a que es la pantalla más usada en proyectos electrónicos por su fácil integración y disponibilidad en el mercado. Es compatible con la mayoría de las placas de desarrollo del mercado.

2.1.2.2. Herramientas para la interfaz web

En esta sección se mencionarán las herramientas necesarias para la implementación de la interfaz Web. En la Figura 2.13 se muestra la estructura del Módulo, el cual, usará los servicios de Firebase y será desarrollado en el *framework* Angular de Node.JS.

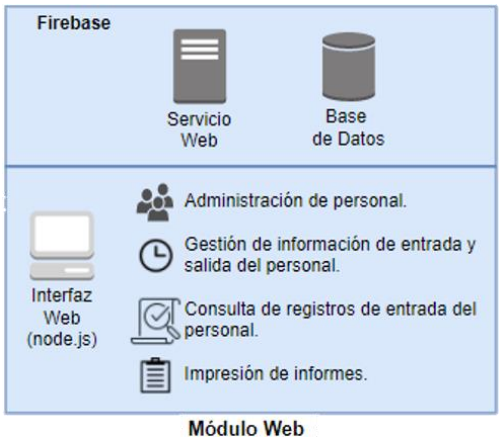


Figura 2.13. Estructura del Módulo Web.

2.1.2.2.1. Node JS

Node.JS es una plataforma basada en el entorno de ejecución de *JavaScript* para crear aplicaciones web. Node.JS utiliza un modelo basado en eventos de entrada y salida no bloqueantes. Es ideal para aplicaciones que requieren mucha intensidad de datos en tiempo real en dispositivos distribuidos [24].

Node.JS usa el motor V8 de Google, es una máquina virtual que ejecuta Google Chrome, para la programación del lado del servidor. El motor V8 le da al Node.JS una gran mejora en el rendimiento, ya que compila el código fuente directamente desde su intérprete [24].

Además, brinda soporte y facilita de uso de bibliotecas de terceros para el desarrollo de aplicaciones. Node tiene su propio sistema de administración de paquetes llamado NPM (*Node Package Manager*), que contiene una colección pública de paquetes que brinda facilidad en la codificación de aplicaciones [25].

2.1.2.2.2. Angular

Es un *framework* de código abierto que ayuda al desarrollo de aplicaciones web. Es una herramienta que corre sobre Node.JS con un lenguaje de programación *TypeScript*. Actualmente, ha sido una alternativa considerable a la hora de desarrollar aplicaciones web [26].

Angular provee una colección de bibliotecas que cubren una variedad de características para el desarrollo, tales como enrutamiento, administración de formularios y comunicación cliente-servidor. También, provee un conjunto de herramientas que facilitan el desarrollo de aplicaciones web, como: compilación, depuración y testeo de código. Angular brinda escalabilidad en proyectos, desde aplicaciones de un desarrollador hasta aplicaciones de gran escala a nivel empresarial [27]. Tiene una gran comunidad que constantemente está aportando para mejorar las herramientas del *framework*.

En cada proyecto, se pueden generar interfaces, servicios, componentes, entre otros, mediante comandos en la consola de Node.JS. A continuación, se mencionan los esquemas que se usarán en el proyecto:

- **Interfaz:** Permite crear objetos con una estructura de datos específica. Ya que se obtienen objetos tipo Clave:Valor, se define en la interfaz el nombre de la clave y el tipo del valor para tener una mejor organización de los datos.
- **Servicio:** Es un proveedor de funcionalidad entre los datos y la aplicación. Los servicios son consumidos por los componentes y módulos de la aplicación. Brindan funciones para acceder a la información y realizar operaciones con los datos.

- **Componente:** Es un esquema que permite generar interfaces o plantillas en la aplicación web. Consta de tres archivos principales que permiten personalizar la vista:
  1. Archivo de vista HTLM (*component.html*, se codifica en HTML la vista para que pueda interactuar con el archivo de componente),
  2. Archivo de estilos CSS (*component.css*, carga la personalización en código CSS para el archivo de vista HTML, estilos),
  3. Archivo de componentes TS (*component.ts*, contiene la codificación *TypeScript* de eventos que gestionará la vista, lógica).

  Cada componente no puede inyectar módulos previamente creados. Utilizará todos los módulos declarados en el módulo raíz (módulo app: *app.module.ts*).
- **Módulo:** Se caracteriza por brindar una función en específico. Existen módulos provistos por Angular (ejemplo: *NgbModule*, *FormsModule*) que permiten diversificar la funcionalidad de la aplicación web. Angular permite crear módulos para orientarlos a las necesidades del proyecto, estos pueden incluir módulos predefinidos en Angular y de esta manera se puede personalizar la función de cada módulo.

  Tiene una funcionalidad similar a la de un componente, tiene los mismos archivos de codificación. Solo que, tiene un archivo adicional llamado *module.ts* en el cual se declaran los módulos a importar.

  Una aplicación web consta de muchos módulos y componentes que generan una vista con su respectiva funcionalidad.
- **Guard:** herramienta que se ejecuta en el momento que se accede a una ruta hacia un determinado componente. Mediante una función (llamada *canActivate*), permite verificar si la vista del componente se debe cargar o no. Es muy utilizado para proteger rutas y evitar accesos no autorizados.
- **Pipe:** Sirve para transformar los datos a visualizar en la vista. Comúnmente, es usado para generar filtros de búsqueda.

2.1.2.2.3. Firebase

Firebase es una plataforma creada por Google, que provee funciones para el desarrollo de aplicaciones, mejorar la calidad de desarrollo y brinda estadística de datos aplicada en analítica y predicciones [28]. Provee algunas herramientas para aportar al desarrollo de aplicaciones, como: almacenamiento en la nube (*Cloud Firestore*), *hosting*, configuración de *hosting*, autenticación y funciones de *BackEnd*.

Para el almacenamiento, Firebase provee dos servicios, las cuales son: *Cloud Firestore* y *Realtime Database*. En el presente trabajo de titulación se trabajará con *Cloud Firestore*.

Cloud Firestore

Es una base de datos *NoSQL* flexible, escalable, alojada en la nube con el fin de almacenar y sincronizar datos para el desarrollo tanto del lado del cliente como en el lado del servidor. Es aplicado en el desarrollo en servidores, dispositivos móviles y aplicaciones Web. Mantiene los datos sincronizados en la aplicación del usuario mediante objetos de escucha en tiempo real [29]. Ofrece integración con otros productos de Firebase y Google Cloud. Con esta herramienta se puede acceder a la base de datos con *SDKs* nativos de Google desde las aplicaciones desarrolladas en iOS, Android y Web. Además, brinda compatibilidad con *NodeJS*.

Modelo de datos de Cloud Firestore

Cloud Firestore es una base de datos NoSQL orientada a documentos [30]. A diferencia de una base de datos SQL, las cuales tienen un orden de almacenamiento estructurado por tablas y filas, ésta almacena los datos de forma ordenada en documentos, que se organizan por colecciones (conjunto de documentos). Los documentos almacenan datos con pares clave-valor.

Documentos

En la base de datos *Cloud Firestore*, la unidad de almacenamiento es el documento. Un documento contiene campos con valores asignados. Cada documento se identifica con un nombre o un identificador. En la Figura 2.14 se observa un documento que representa a un *estudiante* con sus datos tipo clave: valor.

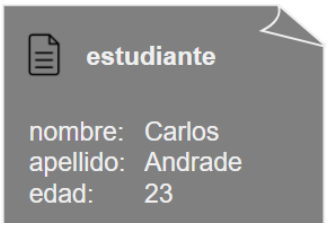


Figura 2.14. Ejemplo de documento en Firebase.

En la Figura 2.15 se observa un documento de *estudiante* con sus datos complejos tipo clave: valor. En este caso, el valor también puede contener un objeto tipo clave: valor; que se le conoce como objetos complejos anidados [30].

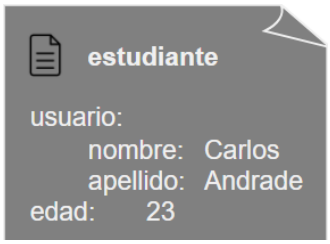


Figura 2.15. Ejemplo de documento con objeto complejo.

Entre las características de *Cloud Firestore* destacan:

- Flexibilidad: El modelo de datos de *Cloud Firestore* permite crear estructuras de datos flexibles y jerárquicas. Con esto, se almacenan los datos en documentos, organizados en colecciones. Los documentos pueden ser objetos anidados simples o complejos, y pueden estar organizados en subcolecciones.
- Consultas expresivas: se puede usar consultas para recuperar documentos específicos individualmente o para recuperar todos los documentos de una colección que coincidan con los parámetros de búsqueda. Las consultas pueden contener varios filtros en cadena y combinar filtros de criterio de orden. También, se indexan de forma predeterminada, por lo que se obtiene únicamente el conjunto de resultados.
- Actualizaciones en tiempo real: *Cloud Firestore* usa sincronización para actualizar los datos de cualquier dispositivo conectado. Sin embargo, se puede usar para ejecutar consultas de recuperación únicas y sencillas de manera eficiente [30].

Firebase Authentication

Firebase Authentication proporciona servicios de *BackEnd*, *Kits* de Desarrollo de Software y bibliotecas ya elaboradas, para autenticar a los usuarios en una determinada aplicación. Brinda el servicio de autenticación mediante contraseñas, números telefónicos, autenticación por Google, Facebook y Twitter [31]. Con esto, se controla el acceso de quien puede hacer uso de funciones determinadas en la aplicación. Se integra fácilmente con los otros servicios de Firebase.

Firebase Hosting

Firebase Hosting proporciona un *hosting* seguro, rápido y eficiente para las aplicaciones web de contenido dinámico y estático. Es un servicio de *hosting* de contenido web a aplicaciones con nivel de producción [32].

Firebase Hosting entrega el contenido de la web mediante una conexión segura. Incluye SSL (*Secure Sockets Layer*) sin necesidad de configuración. Aloja el contenido dinámico y

estático con facilidad de publicación. Además, admite todo tipo de contenido para *hosting*, como: archivos CSS y HTML, API's o los microservicios desarrollados en NodeJS. Provee un emulador para visualizar los cambios realizados en la aplicación antes de publicarlos. También, proporciona un servicio para la integración con GitHub [32].

Cloud Storage

Es un servicio de base de datos orientado a archivos. La base de datos está diseñada para almacenar y compartir contenido generado por usuarios, como fotos o videos. El sistema de Cloud Firestore está diseñado para aplicaciones de alta demanda multimedia brindando alta velocidad de subida y bajada de archivos. Está basado en la misma tecnología que impulsa aplicaciones como *Spotify* y *Google Photos*. En seguridad, el almacenamiento en la nube de Firebase proporciona accesos diferenciados y fáciles de configurar. Se puede otorgar acceso según el identificador de usuario, el nombre del archivo, el tamaño, el perfil y otros metadatos [33].

2.1.2.2.4. Visual Studio Code

Visual Studio Code es un editor de código fuente ligero, multiplataforma (Windows, Linux, MacOS). Incluye soporte para *JavaScript*, *TypeScript* y Node.JS y brinda extensiones para el desarrollo en lenguajes de programación conocidos, como: C ++, C #, Java, Python, PHP, Go, entre otros [34]. Además, da la posibilidad de incluir extensiones como idiomas, temas, depuradores, comandos y más. Permite a la comunidad publicar extensiones para mejorar su flujo de trabajo.

2.1.2.2.5. Arduino IDE

Arduino IDE (*Integrated Development Environment*) es un software de código abierto para la codificación en placas de desarrollo tipo Arduino. Cuenta librerías, bibliotecas, funciones de depuración y comunicación serial [35]. Tiene una amplia comunidad, que constantemente va creando y actualizando librerías para la implementación de módulos orientados a aplicaciones electrónicas. Brinda alta integración con placas de desarrollo diferentes a las propietarias, algunas como ESP32, ESP8266, NodeMCU, Wemos D1 mini, entre otros.

2.1.2.3. Herramientas para interfaz móvil

En esta sección se detallarán las herramientas usadas para la implementación del Módulo Móvil. Este módulo contiene las mismas funciones que el Módulo Web (Figura 2.16), por lo que será una aplicación móvil web (*Web App*) instalable en el sistema operativo Android.

Figura 2.16. Estructura del Módulo Móvil.

Se usará la base de código del Módulo Web conjuntamente con las herramientas mencionadas a continuación.

2.1.2.3.1. Ionic

Ionic es un *framework* para el desarrollo de aplicaciones, provee un conjunto de herramientas de código abierto para crear aplicaciones móviles y de escritorio de alto rendimiento utilizando tecnologías web HTML, CSS y *JavaScript* (o *TypeScript*), con integración de *frameworks* para el desarrollo de aplicaciones como: Angular, React y Vue [36]. Ionic usa el lenguaje de programación *JavaScript* o *TypeScript* para el desarrollo. En el presente trabajo de titulación se usará Ionic con integración del *framework* Angular.

Capacitor es una herramienta con la cual se puede convertir aplicaciones hibridas de Ionic en nativas móviles. Con Capacitor, se puede acceder a los *Kits* de Desarrollo de Software (SDK) nativos de cada plataforma para realizar aplicaciones orientadas a dispositivos móviles. Capacitor se adapta tanto a proyectos nuevos como a proyectos ya desarrollados [37].

2.1.2.3.2. Android Studio IDE

Android Studio IDE es un software para el desarrollo de aplicaciones en la plataforma de Android. Brinda un entorno de desarrollo unificado para la plataforma, se puede codificar aplicaciones para todos los dispositivos Android. Integra un emulador para ejecutar aplicaciones durante el proceso de codificación [38].

2.2. FASE DE DISEÑO

2.2.1. DESARROLLO DEL TABLERO KANBAN

Se ha desarrollado un tablero Kanban para la fase de diseño del prototipo. En la Figura 2.17 se observa este tablero representado por 3 columnas principales que representan los flujos de trabajo de cada fase del proyecto.

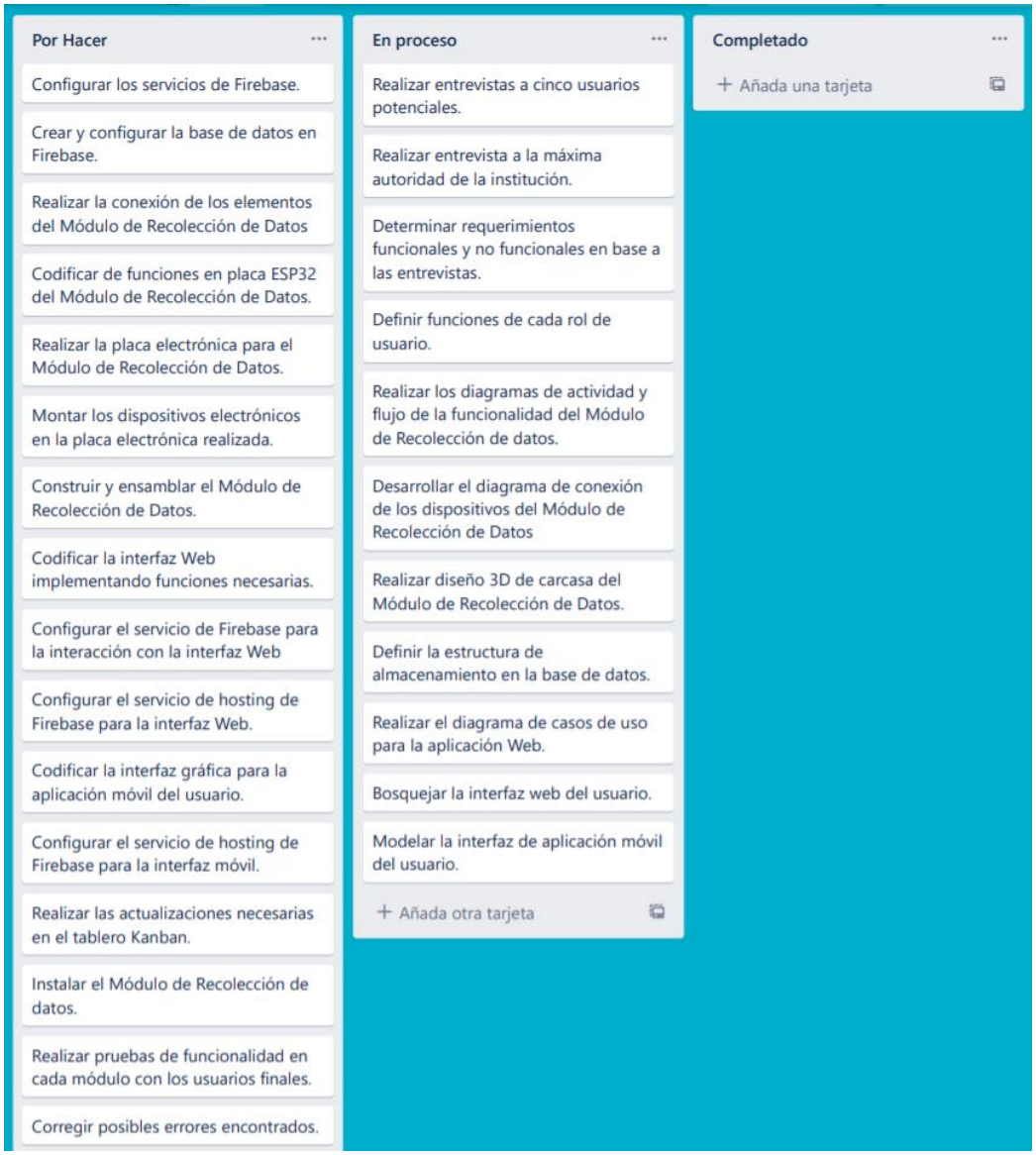


Figura 2.17. Tablero Kanban de Metodología.

En primer lugar, se encuentra la columna “Por Hacer” que detalla todas las actividades que están pendientes para su desarrollo, en este caso, las actividades de las fases de implementación y pruebas. En segundo lugar, se encuentra la columna “En proceso” que detalla las actividades que se encuentran en desarrollo respecto a la fase correspondiente, en este caso, la fase de diseño. Por último, se encuentra la columna “Completado” que hace referencia a las actividades finalizadas.

2.2.2. ANÁLISIS DE REQUERIMIENTOS

En esta sección, se detallarán los requerimientos tanto funcionales como no funcionales del sistema en base al análisis de entrevistas desarrolladas a la máxima autoridad de la institución y a usuarios potenciales.

2.2.2.1. Entrevista a usuarios y a la autoridad de la institución

Tabla 2.11. Preguntas de entrevista realizada a la máxima autoridad de la institución.

Pregunta	Análisis
¿De qué forma controla la asistencia y hora de entrada/salida del personal a la institución?	La directora menciona que actualmente no tiene un método oficial para realizar un control detallado de asistencia, atrasos, salida temprana y horas extra de la institución. Detalla que se ha dado tolerancia en este aspecto. Y que únicamente se tomaba acciones en caso de que haya faltas o atrasos de varios minutos.
¿Este método de control de asistencia y hora de entrada/salida le causa problemas?	La directora mencionó que no ha sido efectivo. Primero, porque no se tiene un control total. Segundo, porque en caso de que exista un inconveniente de asistencia, atrasos u horas extra, debía llevar un registro escrito y que en ocasiones se olvidaba en registrarlo.
¿Su personal trabaja horas extra? ¿Cómo registra estas horas?	La directora indicó que su personal trabaja horas extra. Detalló que recibe la notificación de la cantidad de horas de un determinado empleado y lo guarda en un registro escrito (cuaderno). Estas notificaciones las recibe por medio de conversaciones, llamada o mensaje de texto.
¿Usted cree qué, una aplicación digital y un componente electrónico para identificar al personal con su huella digital solucione los problemas?	La directora indicó que si es necesario implementar una aplicación. Ya que, con esto, tendría los registros digitalizados y podría acceder desde el Internet en cualquier momento. Por otro lado, indicó que solucionaría el problema de falta de control de asistencia en el personal de la institución.
Para la aplicación digital ¿De qué forma desearía observar la información de asistencia y hora de entrada/salida del personal?	La directora mencionó que le gustaría observar un resumen diario de su asistencia, si el usuario llegó atrasado o a tiempo. Además, visualizar la hora de entrada y salida del personal, la cantidad de horas que ha trabajado y la cantidad de horas extras (si el usuario cumple). Por último, visualizar un resumen mensual de las horas trabajadas y horas extra.
Para la aplicación digital ¿Desearía visualizar datos relacionados con justificaciones por inasistencia o llegada tarde a la institución?	La directora indicó que si desease observar las justificaciones en caso de que exista alguna novedad en la asistencia del usuario. Que desea almacenar el motivo por la cual existe esa novedad.
En el componente electrónico para identificar al usuario, ¿desearía que cuente con otro método para identificar al personal, aparte del biométrico por huella digital?	La directora mencionó que el único método que desea para identificar al personal es el biométrico por huella digital. Con esto, se asegura de que cada empleado pueda generar su propio registro de asistencia.
¿Cuáles son sus expectativas con respecto a la facilidad de uso de esta aplicación?	La directora mencionó que espera una aplicación fácil de manejar, que sea intuitiva. Espera que la información mostrada sea fácil de interpretar.
¿Cuáles son sus expectativas en cuanto a rendimiento del sistema?	La directora indicó que desearía un sistema que no tenga mucho retardo en mostrar la información.
¿Cuáles serían los requisitos de acceso a la información?	La directora mencionó que espera desearía que solo los usuarios autorizados puedan ver toda la información recolectada. Y los usuarios no autorizados, visualicen información limitada.
¿Cuáles son sus expectativas con respecto al tiempo de entrenamiento para el uso del sistema?	La directora mencionó que espera que el tiempo de familiarización con la aplicación sea lo más corto posible. Con una hora estaría bien.

Los requerimientos funcionales y no funcionales fueron obtenidos mediante entrevistas a la máxima autoridad de la institución y a cinco usuarios potenciales.

La entrevista con la autoridad de la institución tuvo como objetivo analizar el problema de control de asistencia en la actualidad y las preguntas realizadas a los usuarios potenciales fueron orientadas para obtener información acerca de las expectativas que tienen acerca del prototipo y las funcionalidades con las que podría contar el prototipo. Las preguntas realizadas en la encuesta se adjuntan en el ANEXO A.

Tabla 2.12. Lista de preguntas de entrevista realizada a 5 empleados de la institución.

Preguntas	Análisis
¿De qué forma controla la asistencia y hora de entrada/salida del personal a la institución?	El personal entrevistado mencionó que no se aplica ningún método de control de asistencia a la institución. En el caso de que exista una falta o salida anticipada de la institución, esta es notificada a la directora mediante llamada o mensaje de texto. En el caso de que un atraso sea de un tiempo consideradamente alto, es notificado por intermedio de la administradora académica.
¿Usted cree que la falta de control de asistencia influye en su nivel de puntualidad?	En esta pregunta, 3 de las 5 personas entrevistadas mencionaron que si influye.
¿Tiene problemas en registrar la cantidad de horas extra que usted ha cumplido?	El personal entrevistado mencionó que si se tiene problemas con el registro y notificación de horas extra. Detallaron que en ocasiones este proceso es por mensaje y que a veces se le debe recordar más de una vez a la directora.
¿Usted cree qué, una aplicación digital y un componente electrónico para identificar al personal con su huella digital solucione los problemas de puntualidad y registro de horas extra?	El personal entrevistado estuvo de acuerdo en que la aplicación de este trabajo de titulación en la institución brinde mayor nivel de responsabilidad en la puntualidad y facilite el registro de horas extra.
¿Cuáles son sus expectativas con respecto a la facilidad de uso de esta aplicación?	El personal entrevistado mencionó que desearía una aplicación simple y fácil de usar.
¿Cuáles son sus expectativas en cuanto a rendimiento del sistema?	El personal entrevistado indicó que desearía un sistema que no tenga mucho retardo en mostrar la información.
¿Cuáles serían los requisitos de acceso a la información?	El personal entrevistado mencionó que desearía visualizar la información de los registros de asistencia. Con la capacidad de generar justificaciones por asistencia, horas extra, falta y salida temprana de la institución.
¿Cuáles son sus expectativas con respecto al tiempo de entrenamiento para el uso del sistema?	El personal entrevistado mencionó que espera que el tiempo de familiarización con la aplicación sea lo más corto posible. Con una hora estaría bien.

El análisis realizado a partir de la entrevista con la autoridad de la institución, MSc. Paola Flores, se muestran en la Tabla 2.11 y el análisis de las entrevistas realizadas a 5 trabajadores de la institución se muestra en la Tabla 2.12.

En base al análisis realizado en las entrevistas se puede concluir que, tanto el personal administrativo como el personal de apoyo y docencia de la institución requiere un sistema de control que ayude con el registro de asistencia y horas laborales.

Al aplicar un método como registros escritos en cuadernos, se corre el riesgo de que pierda o modifique los mismos. Por lo tanto, digitalizar la información y almacenarla en la nube resultaría más eficiente.

2.2.2.2. Requerimientos funcionales

Los requerimientos funcionales (RF) definen los servicios o funciones que proveerá el prototipo. Deben definir los servicios solicitados por el usuario consistentemente, es decir, sin ningún tipo de contradicciones [39]. En la Tabla 2.13 se listan los requerimientos para el Módulo de Recolección de Datos.

Tabla 2.13. Requerimientos para Módulo de Recolección de Datos.

RF	Descripción
RF01	Identificar al personal por huella dactilar. A cada usuario se le asignará un identificador único de huella digital.
RF02	Generar registros de entrada y salida en base al siguiente patrón: primer registro será de entrada, segundo registro: salida, tercer registro: entrada, etc.
RF03	Registrar lectura temperatura corporal. Únicamente en el primer registro.
RF04	Mostrar en pantalla la hora ingresada y brindar la posibilidad de que el usuario pueda aceptar o cancelar la generación del registro.
RF05	Agregar huella digital del personal nuevo y actualizar plantilla de huella en caso de que se requiera.

Tabla 2.14. Requerimientos para el Rol Administrador en el Módulo Web.

RF	Descripción	Rol De Usuario	
		Administrador	Empleado
RF06	Ingresar al sistema mediante usuario y contraseña. Permitir recuperar contraseña	X	X
RF07	Añadir, modificar, visualizar y eliminar personal	X	
RF08	Mostrar información de asistencia y horarios de entrada y salida	X	X
RF09	Mostrar cantidad de horas cumplidas y horas extra	X	X
RF10	Mostrar registros de temperatura	X	X
RF11	Añadir justificación por horas extra y horas trabajadas	X	X
RF12	Mostrar resumen mensual de horas extra y trabajadas	X	X
RF13	Aceptar o rechazar solicitudes de justificación	X	
RF14	Visualizar y editar preferencias del sistema	X	

En la Tabla 2.14 se listan los requerimientos de funcionamiento tanto para el Módulo Web como para el Módulo Móvil dependiendo del rol de usuario Administrador y Empleado.

2.2.2.3. Requerimientos no funcionales

Los requerimientos no funcionales (RNF) son las restricciones de los servicios o funciones que ofrece el prototipo, algunas como: la fiabilidad, la respuesta en el tiempo y la capacidad de almacenamiento [39]. Estos se muestran en la Tabla 2.15.

Tabla 2.15. Requerimientos no Funcionales del Sistema.

Métrica	Requerimiento	Descripción
Usabilidad	RNF01	Los módulos diseñados deben ser fáciles e intuitivos de usar.
Escalabilidad	RNF02	El prototipo debe adaptarse a cambios a futuro, como aumento de personal.
Disponibilidad	RNF03	El prototipo debe contar con completa disponibilidad
Portabilidad	RNF04	La interfaz web será capaz de visualizarse en un navegador. El módulo móvil correrá bajo el sistema operativo Android.
Autenticación	RNF05	El acceso a los módulos web y móvil deberá ser controlado por autenticación mediante correo y contraseña

2.2.2.4. Funciones de cada rol de usuario

Se han definido dos roles para el personal: administrador y empleado. A continuación, se detalla la funcionalidad de cada rol:

- **Rol: Administrador**

En el rol *Administrador* se podrá realizar una gestión CRUD (*Create, Read, Update, Delete*) de los usuarios (personal) con la capacidad de configurar el horario de entrada/salida y el tipo de rol de usuario. Además, tendrá la capacidad de aceptar o rechazar justificaciones, con el fin de registrar horas extra y horas de trabajo justificadas por falta.

El usuario Administrador podrá visualizar los datos de registros de todos los usuarios del sistema.

- **Rol: Empleado**

El usuario con rol *Empleado* podrá únicamente visualizar los datos relacionados con su persona.

La información que podrá visualizar será la siguiente:

- Información de asistencia del personal.
- Cantidad de horas cumplidas.
- Cantidad de horas extras.

- Visualizar sus justificaciones generadas.
- Información de hora entrada y salida.
- Registro de temperatura.

Esta información podrá ser filtrada por nombre de usuario (en el caso de usuarios con rol administrador) y fecha de registro, para así, facilitar la búsqueda de registros.

2.2.3. DISEÑO DEL MÓDULO DE RECOLECCIÓN DE DATOS

En esta sección se detallará la etapa de diseño del Módulo de Recolección de Datos, estructura de registros, funcionalidades, diagramas de conexión y diseño del contenedor del Módulo.

2.2.3.1. Estructura de datos registros recolectados

Este Módulo, recolectará información de registros del usuario que se identifique mediante su huella digital; cada usuario tendrá un identificador tipo número asociado con su plantilla de huella. Una vez identificado el usuario, se procede a la construcción de registro. Este registro llevará los siguientes datos:

- **Número de registro:** Mostrará la cantidad de registros que se han generado en un día. Empezando desde el número 0.
- **Identificador de usuario:** Identificador provisto por los servicios de Firebase (*Cloud Firestore*) cuando se crea un usuario en el sistema (Módulo Web/Móvil); será el identificador del documento de este usuario. Este identificador será alfanumérico y aleatorio de una longitud por defecto de 20 caracteres. Ejemplo: "*5SeF4rjWi3mc4AmjVCpy*".
- **Identificador de registro:** identificador de tipo texto que será construido mediante el identificador de usuario (provisto por los servicios Firebase) más la fecha del registro en formato "aaaammdd". Ejemplo: "*202201135SeF4rjWi3mc4AmjVCpy*".
- **Nombre del usuario:** Almacenará el nombre de usuario. Esta acción se la realiza para no tener que iterar la lista de usuarios.
- **Estado de asistencia:** determinará si el usuario está atrasado, si ha faltado o si ha llegado a tiempo. Será de tipo *array* para dar la posibilidad de agregar otro estado de asistencia como: sin salida o salida temprana.
- **Hora:** Almacenará las horas de trabajo del usuario que generará el registro.
- **Hora:** determinará la hora en que se ha generado un registro de un determinado usuario. Sera de tipo *array*.
- **Temperatura:** campo en donde se almacenará el valor de la temperatura, se lo tomará únicamente cuando el usuario genere su primer registro; será de tipo *string*.

- **Horas trabajadas:** mostrará el cálculo de horas que han transcurrido durante un registro de entrada y uno de salida.

Cuando el usuario genere su primer registro, se creará un documento en *Cloud Firestore* con estos datos y se irá actualizando conforme el usuario genere más registros. Con esto, se estructuran los datos de registros mediante la comunicación del Módulo y *Cloud Firestore*; en el caso de que se cuente con conexión a Internet.

Los registros que no se suben a Internet por falta de conexión se guardarán en la memoria SD en texto plano. Para un primer registro de entrada, los datos se guardarán como se muestra en la Figura 2.18. Los datos obtenidos serán almacenados en una cadena de texto y estarán separados por el carácter guión bajo (_).

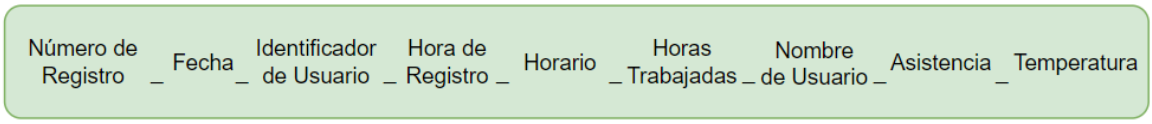
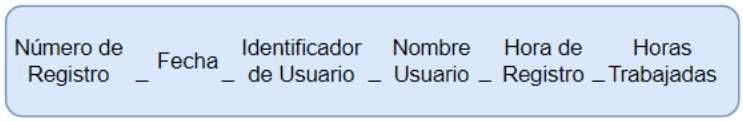


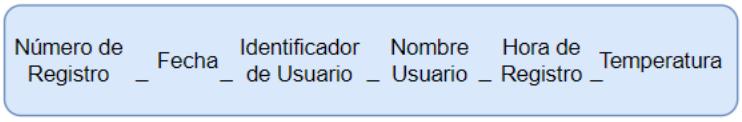
Figura 2.18. Estructura de almacenamiento de datos en memoria SD para un registro de entrada.

Una vez se haya generado el primer registro, los siguientes registros se guardarán conforme a la estructura presentada en la Figura 2.19.

Esta estructura permitirá dividir los campos mediante el carácter "_". Con esto, se obtiene la información para construir el objeto que se va a crear (en el caso de un registro de entrada, numRegistro igual a 0) o actualizar (en el caso de que numRegistro sea mayor que 0). Se creará una función que permita subir estos registros pendientes a la base de datos cuando se cuente con conexión a Internet.



(a)



(b)

Figura 2.19. Estructura de almacenamiento de datos en memoria SD para un registro de salida (a) y entrada (b).

2.2.3.2. Sistema de archivos en SD

Para garantizar el correcto funcionamiento del módulo, cuando disponga o no de Internet, se crearán archivos con las siguientes funciones:

- *Usuarios.txt*: archivo en donde se almacenarán los datos de la colección *usuarios* obtenidos desde Firebase. El Módulo leerá este archivo al momento de encenderse para cargar la información de los usuarios en código.
- *Preferencias.txt*: archivo en donde se almacenarán los datos de la colección preferencias obtenidos desde Firebase. El Módulo actualizará este archivo al momento de encenderse.
- *registrosSinSubir.txt*: archivo en donde se almacenará la información de los registros en caso de que no exista conexión a Internet. Este registro se guardará en base al formato explicado en la sección 2.2.3.1.
- *registrosDiarios.txt*: archivo en donde se almacenarán los identificadores de registros recolectados por día, seguido de la hora de generación del registro y las horas acumuladas de trabajo. El contenido de este archivo se vaciará cada día. A continuación, se listan dos ejemplos de registros diarios:
  - "*202201135SeF4rjWi3mc4AmjVCpy_7:30:12_0:00:00*": El registro con id *202201135SeF4rjWi3mc4AmjVCpy* ha sido generado a las 7:30:12 am y lleva 0:00:00 horas de trabajo debido a que será un registro de entrada y empezará a contar las horas laborales desde ese momento.
  - "*202201135SeF4rjWi3mc4AmjVCpy_10:40:24_3:10:12*". El registro con id *202201135SeF4rjWi3mc4AmjVCpy* ha sido generado a las 10:40:24 am y lleva 3:10:12 horas acumuladas de trabajo en este día; será un registro de salida, contabilizará las horas transcurridas con el registro anterior y las almacenará en la etiqueta. Las horas laborales no se acumularán hasta que se genere un nuevo registro de entrada.

2.2.3.3. Diagramas de actividad del Módulo de Recolección de datos

Se han realizado diagramas de actividad para describir el funcionamiento de cada Módulo.

Las funciones que ejecutará el Módulo de Recolección de Datos en el momento que este encienda (función *Setup*) se detallan en la Figura 2.20. Entre sus funciones están: inicializar la comunicación con los módulos, conectarse a Internet y cargar los datos.

El ciclo de ejecución de la función *loop()* se muestra en la Figura 2.21, En primer lugar, se determinará si se debe visualizar el estado de conexión o la fecha y hora. Esto se logra mediante el valor de una variable que cambiará cuando el usuario pulse un botón físico que se instalará en el Módulo. Además, mediante otro botón se podrá verificar si el Módulo debe crear, actualizar o eliminar una plantilla de huella digital. Por último, el sensor de huella digital verificará si se detecta una huella para generar un registro.

El funcionamiento general del Módulo de Recolección de Datos (teniendo en cuenta debe existir al menos un usuario registrado) se describe en la Figura 2.22. Primero, el usuario que desee registrar su entrada/salida, deberá colocar su dedo en el lector de huellas. El módulo comparará la huella digital leída con sus datos almacenados. En caso de no encontrar la huella digital, se mostrará un mensaje en pantalla "*Usuario no encontrado. Favor, ingrese nuevamente*". Por otro lado, si el módulo encontró coincidencia, procederá a registrar la fecha y hora en la que se encontró coincidencia. El Módulo registrará la hora de registro en base a la cantidad de registros que se encuentren en el archivo *registrosDiarios.txt*; de modo que la hora deberá ser un *array* y la cantidad de registros será el índice del *array*. Se mostrará un mensaje para que el usuario se acerque al sensor de temperatura para su registro. Este sensor se activará cuando el usuario se aproxime, por lo que se usará un sensor de proximidad infrarrojo. Si detecta un valor incorrecto, el módulo insistirá en la toma de temperatura por un determinado tiempo. Después, se formará el registro para ser subido a *Cloud Firestore* en caso de que el Módulo cuente con Internet o caso contrario, ser guardado en la tarjeta SD (en el archivo *registrosSinSubir.txt*).

El procedimiento que realiza el Módulo en caso de que se tenga registros sin subir en el archivo *registrosSinSubir.txt* se detalla en la Figura 2.23. Como en el proceso anterior, el Módulo guarda registros de entrada/salida con o sin Internet, es necesario enviar los registros pendientes a la base de datos. Para esto, se diseña una función para subir los registros que se ejecutará al iniciarse el Módulo o cuando se presione un determinado botón para realizar este proceso manualmente. Cuando se ejecute esta función, el módulo verificará si existe conexión a Internet y empezará a leer el archivo *registrosSinSubir.txt*; si encuentra un registro, lo subirá.

El procedimiento para el registro de un nuevo usuario, actualizar una huella digital y eliminar un determinado usuario se muestra en la Figura 2.24. El usuario con rol Administrador deberá acceder desde el Módulo Web/Móvil para generar una solicitud al Módulo de Recolección de Datos para que realice alguna de estas funciones. Para esto, se creará un documento, llamado *Preferencias*, en la base de datos *Cloud Firestore* que permita gestionar cada una de las funciones mencionadas con los campos *puedeCrear*, *puedeActualizar* y *puedeEliminar*.

El procedimiento para crear, actualizar y eliminar una plantilla de huella en el Módulo se muestra en la Figura 2.25. En el caso de que se desee crear un usuario, el administrador deberá realizar el proceso de registro del usuario. Automáticamente después, se modificará un campo en la base de datos (*puedeCrear*) que indique al Módulo que debe crear una plantilla de huella digital. Con esto, desde el Módulo se solicitará al usuario que se ingrese dos veces su huella digital. Posteriormente, se guardará el identificador de huella digital en

la base de datos y en memoria local del respectivo usuario. Finalmente, se mostrará un mensaje de proceso satisfactorio en el Módulos de Recolección de Datos.

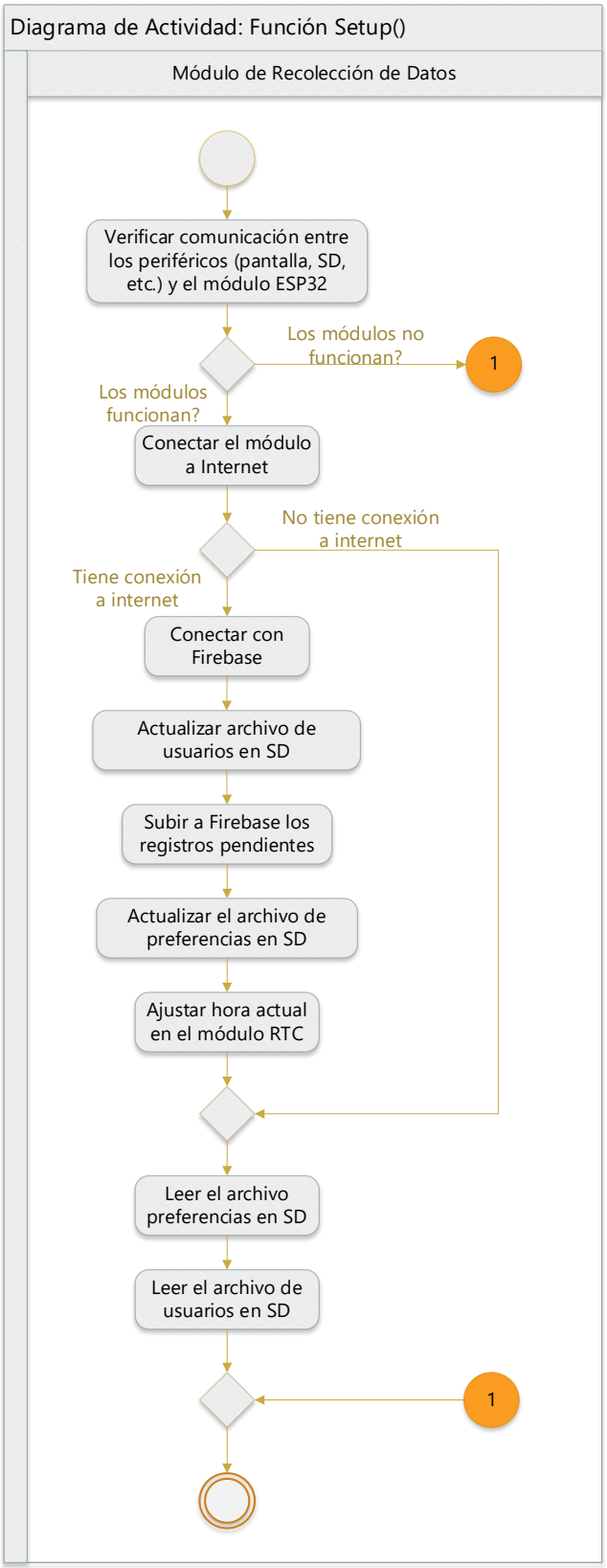


Figura 2.20. Funciones de *Setup*().

Si se desea actualizar, se deberá seleccionar el usuario a actualizar. Con esto, se modificará el campo (*puedeActualizar*) que permita actualizar una huella digital con la información necesaria para efectuar la actualización. Se debe eliminar la huella actual y crear una nueva con el mismo identificador de huella digital que tenía el usuario.

Se podrá eliminar una huella digital únicamente cuando el administrador elimine un usuario. Hecho esto, el campo *puedeEliminar* notificará al Módulo que debe eliminar la plantilla de un usuario.

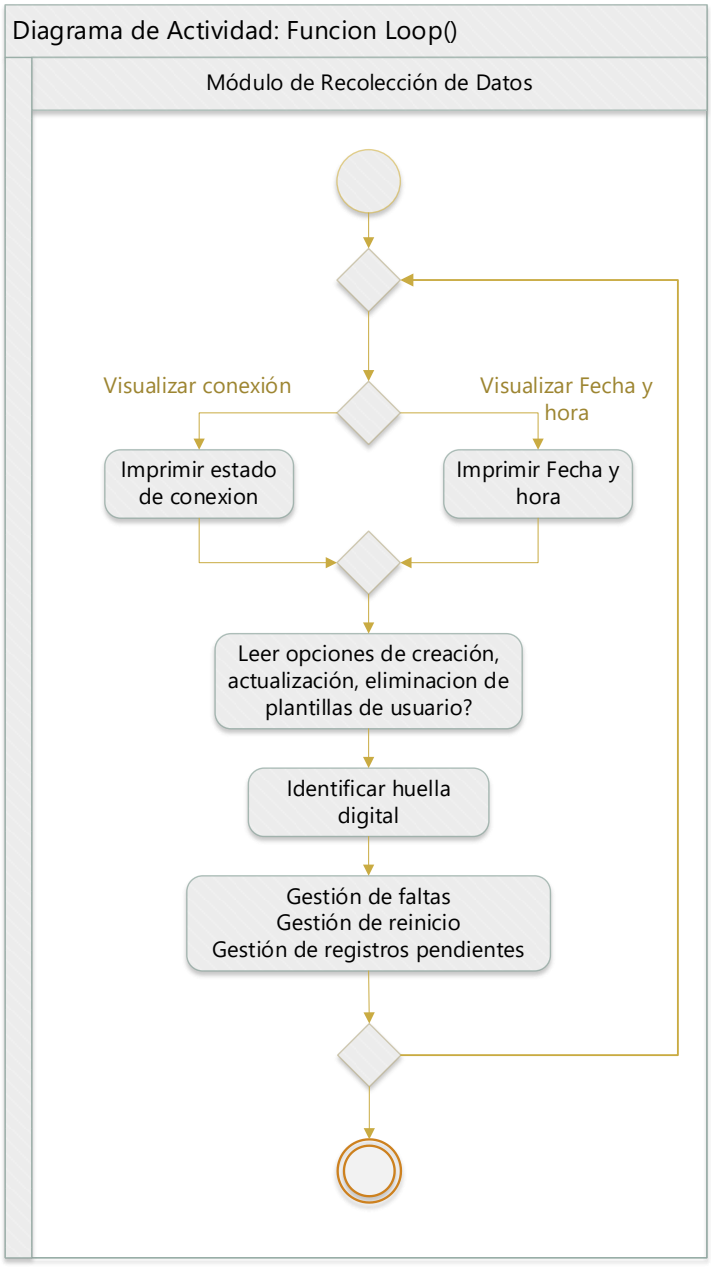


Figura 2.21. Función *Loop*().

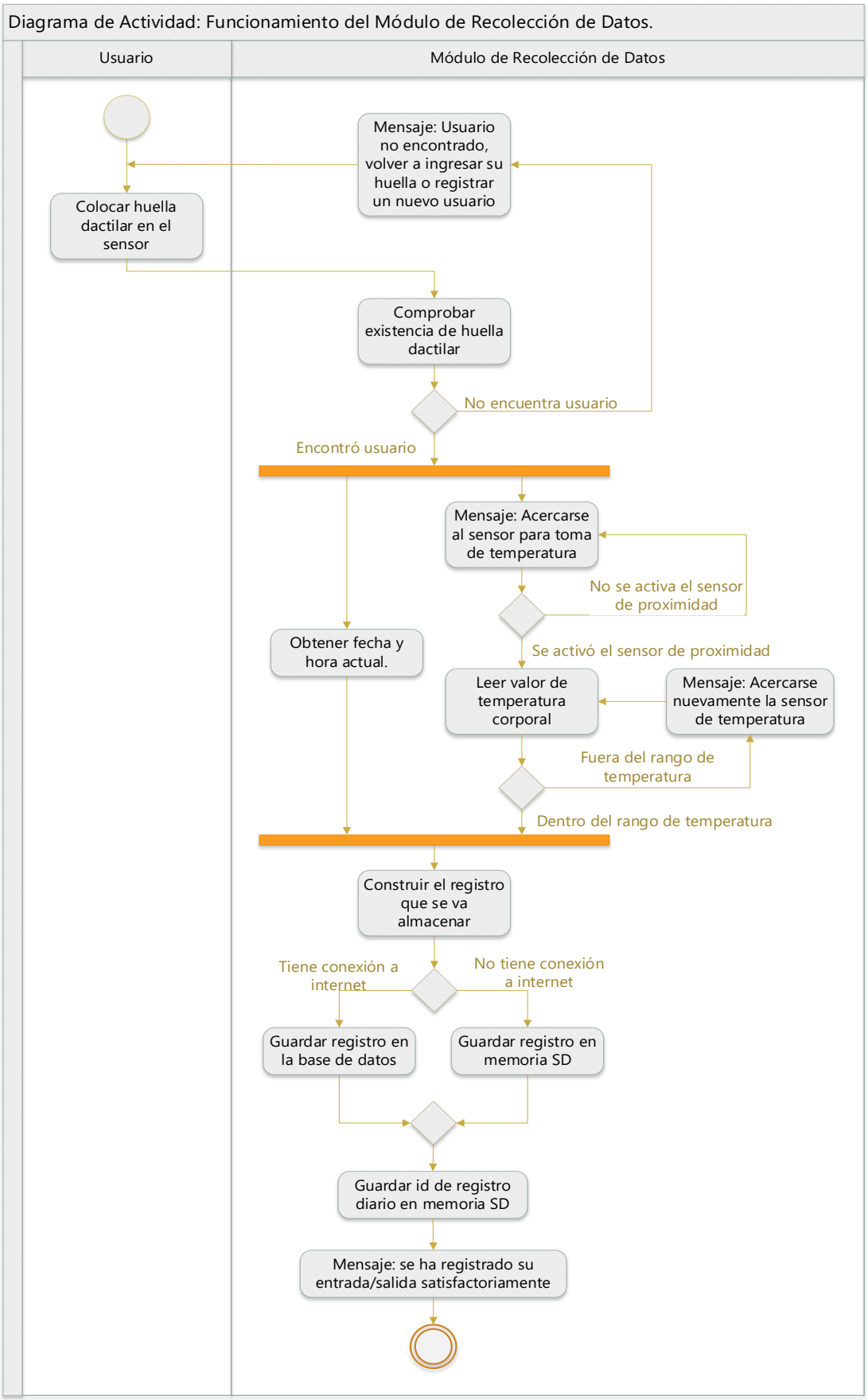


Figura 2.22. Diagrama de Actividad de funcionamiento Módulo Recolección de Datos.

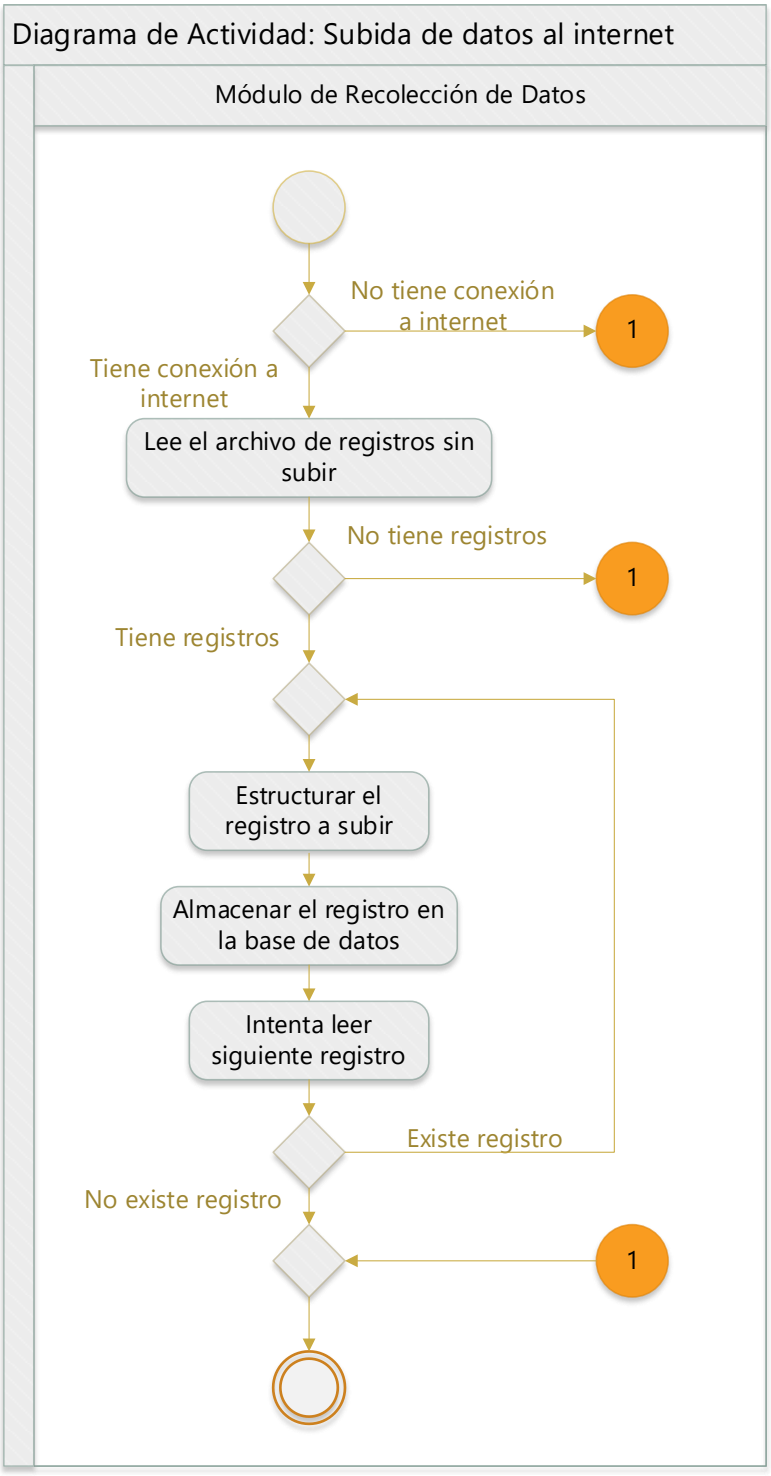


Figura 2.23. Diagrama de Actividad para subida de datos pendientes.

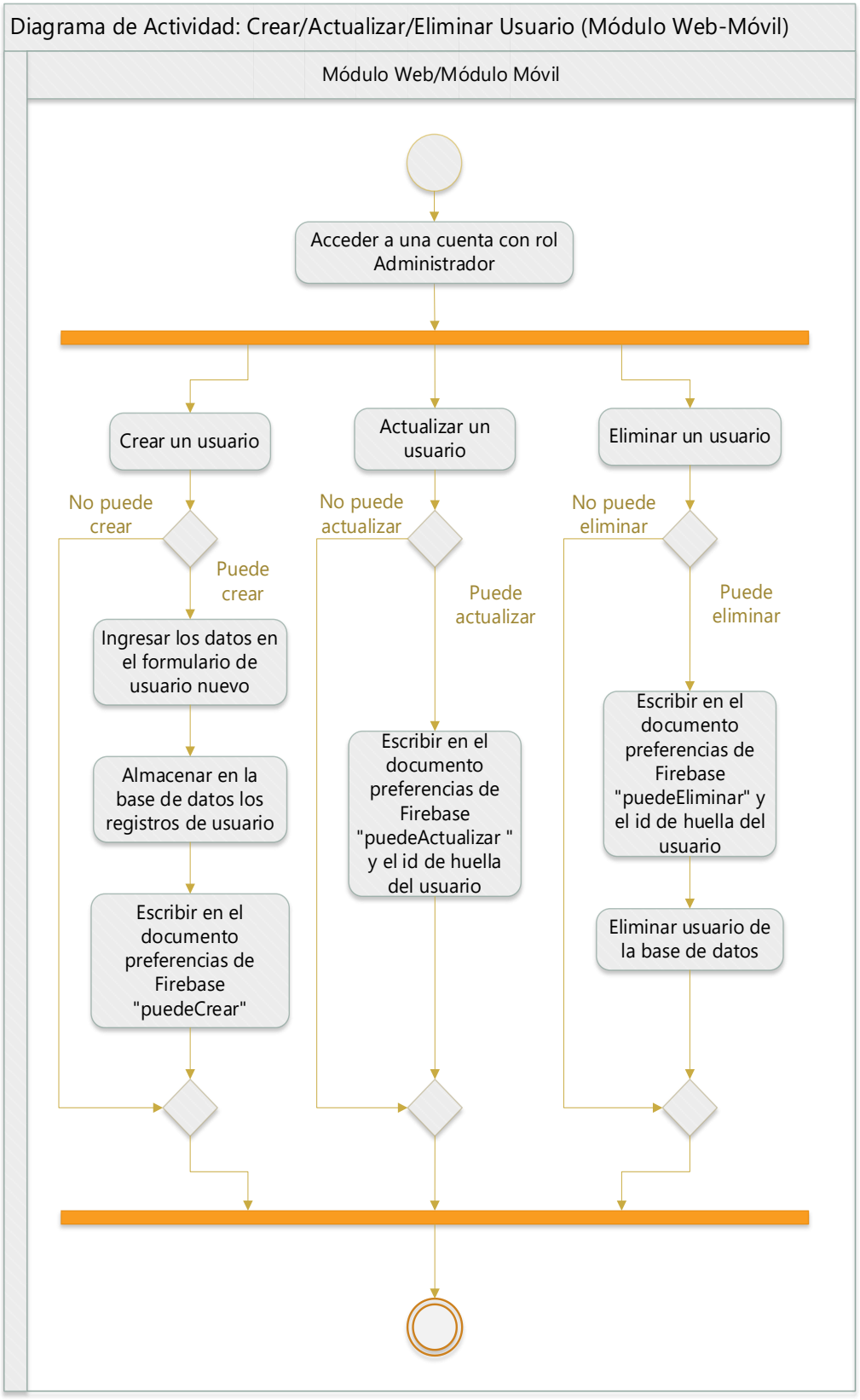


Figura 2.24. Diagrama de Actividad: Crear/actualizar/eliminar usuario módulo Web.

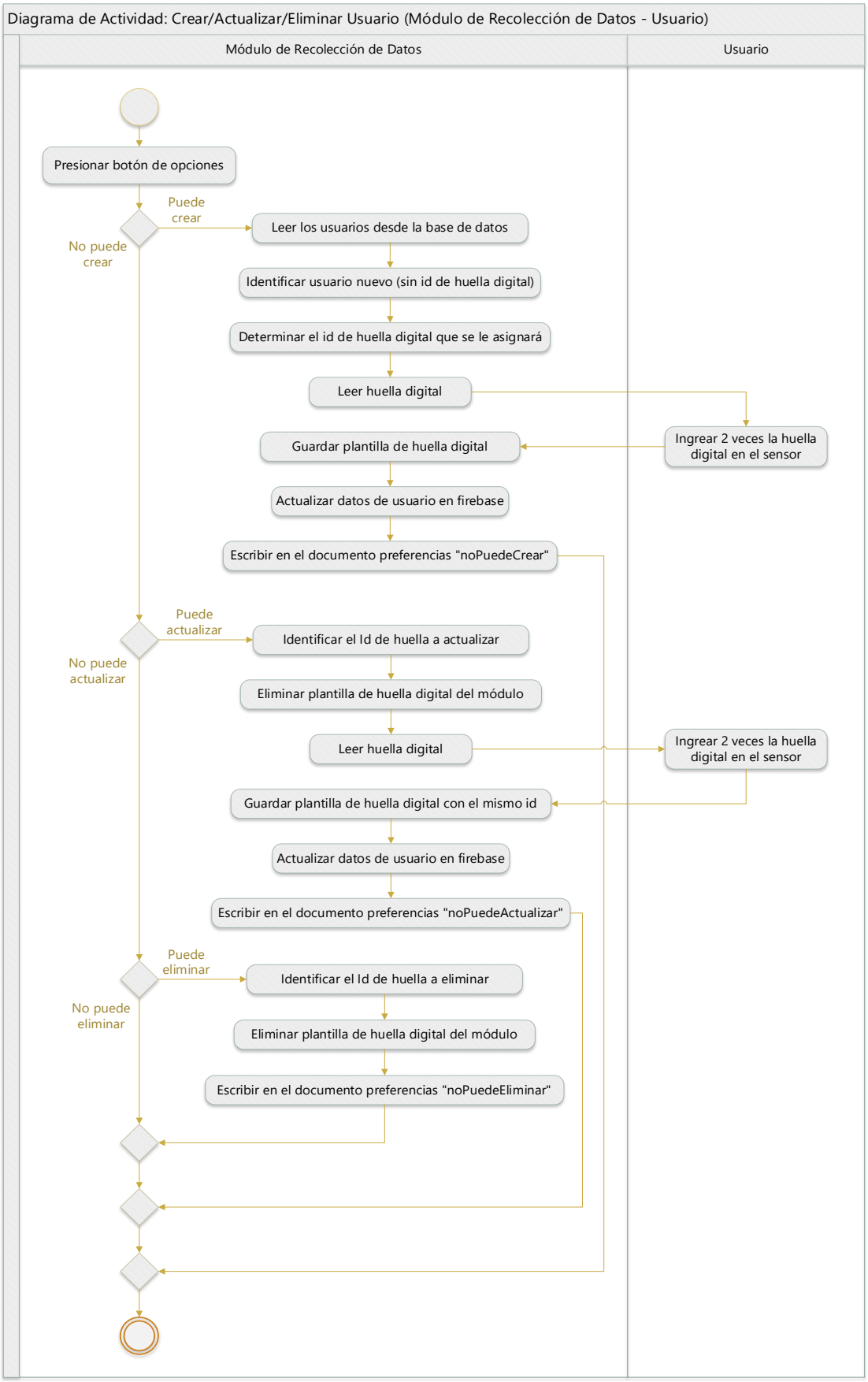


Figura 2.25. Diagrama de Actividad: Crear/actualizar/eliminar Módulo datos de usuario.

2.2.3.4. Diagrama de conexión del Módulo de Recolección de Datos

Para que el sistema continúe funcionando en caso de un corte de luz, se usará una batería y un módulo cargador de energía basado en el circuito integrado TP4056. El TP4056 un chip que se encarga de rectificar el voltaje de entrada y gestionar la carga de una batería; este módulo alimenta al sistema. Mientras se tiene alimentación por USB, el chip mantiene cargada a la batería. En el caso de que se corte la alimentación por USB, el módulo TP4056 alimentará el sistema mediante la batería. En la Figura 2.26 se muestra la conexión del módulo TP4056 con una batería y la alimentación hacia el módulo ESP32. Se tiene un OUT+ como VCC y OUT- como GND para todos los dispositivos del Módulo de Recolección de Datos.

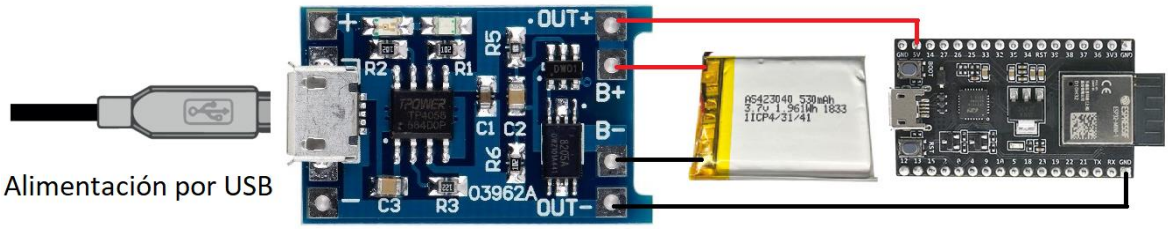


Figura 2.26. Diagrama de conexión para fuente de alimentación.

Como el Módulo va a funcionar con o sin Internet, es indispensable que cuando no se tenga conexión a Internet, sea capaz de seguir tomando datos de fecha y hora. Para esto, se conectará un módulo reloj DS1307 a la placa ESP32. Es un módulo RTC (*Real-Time Clock*) que proporciona datos en horas, minutos, segundos, día, mes y año; para tener la hora actual primero se debe guardar en memoria la hora en la que debe iniciar. Cuenta con una ranura para introducir una batería y así pueda seguir ejecutando el reloj cuando se corte la energía principal en su totalidad. Se comunica hacia la placa mediante una interfaz I2C. En la Figura 2.27 se muestra el módulo DS1307.

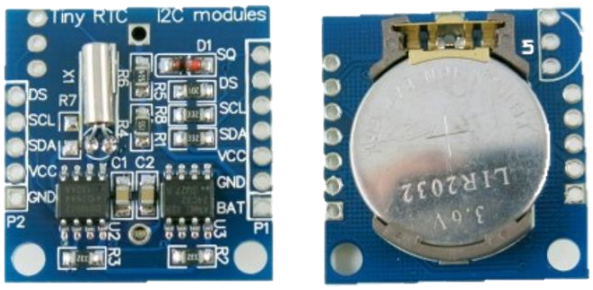


Figura 2.27. Módulo DS1307.

Se usará un sensor de proximidad para activar la lectura del sensor de temperatura. El mismo que, se activará cuando detecte un objeto cerca y permitirá que el sensor de temperatura emita una lectura, de lo contrario, esperaría hasta que el usuario se acerque

al sensor. Este sensor de proximidad emitirá un 1 o 0 lógico por su pin V_{out} dependiendo si detecta o no un obstáculo mediante sus diodos. En la Figura 2.28 se muestra el sensor de proximidad que se usará para realizar la función mencionada.

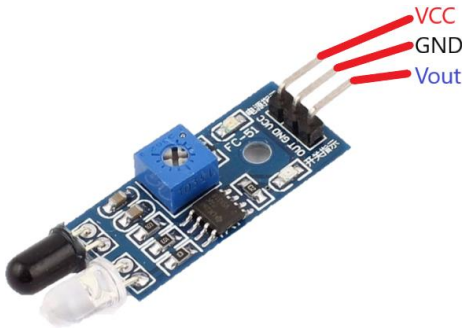


Figura 2.28. Sensor de proximidad.

En la Figura 2.29 se muestra la conexión de todos los dispositivos involucrados en el sistema. Se incluyen todos los módulos electrónicos mencionados además de 4 pulsadores para la gestión de: cambio de visualización de datos en la pantalla, leer opciones de base de datos en el caso de crear, actualizar, eliminar una huella digital y para aceptar y cancelar la creación de un registro. Además, se incluye un pulsador de reinicio manual del Módulo en el caso de que se requiera.

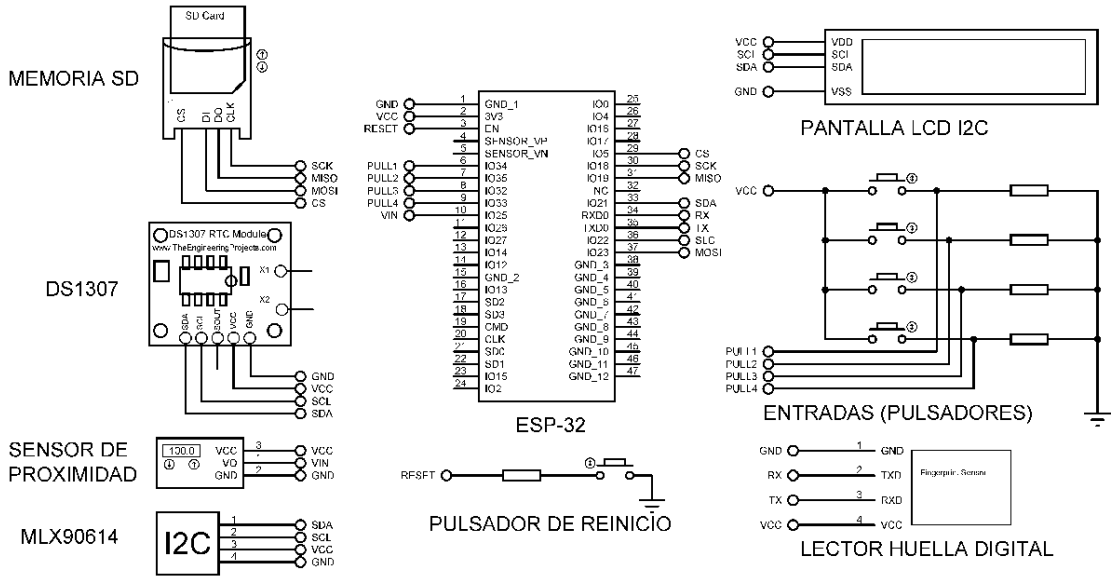


Figura 2.29. Diagrama de conexión del Módulo de Recolección de Datos.

2.2.3.5. Diseño 3D de carcasa del Módulo de Recolección de Datos

Para el diseño de la carcasa, primero se realizó un diseño en 2D para tener en cuenta la distribución de los dispositivos electrónicos involucrados en el sistema. En la Figura 2.30

se muestra la distribución de los dispositivos en un plano frontal, plano lateral y plano superior.

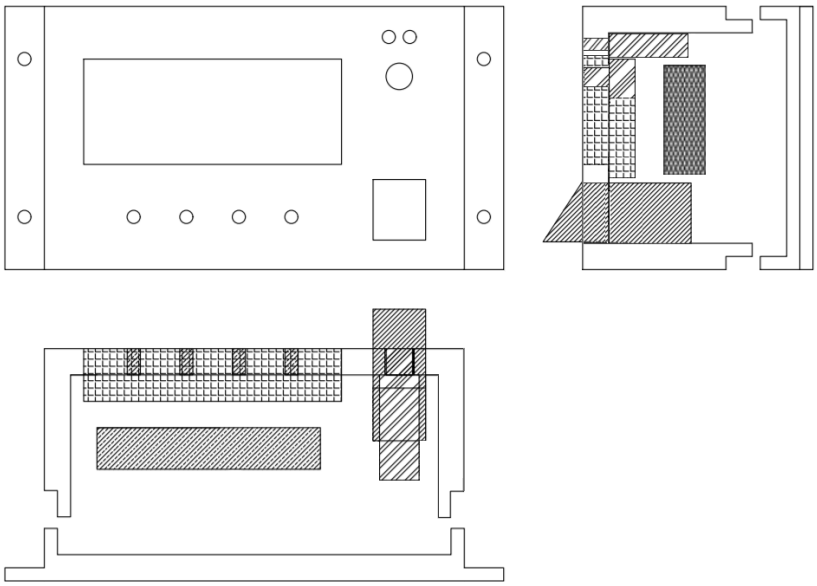


Figura 2.30. Diagrama 2D de Diseño de Carcasa.

En la Tabla 2.16 se puede observar la relación del plano 2D con los dispositivos electrónicos. Con esto, se tiene mayor percepción de la ubicación de estos dispositivos cuando se ensamble este Módulo.

Tabla 2.16. Fondo de dispositivos electrónicos en plano 2D.

Fondo	Dispositivo Electrónico
	Placa electrónica
	Batería
	Pantalla LCD
	Sensor de proximidad
	Sensor de temperatura
	Pulsadores (PULL)
	Sensor de huella dactilar

En base a esto, se ha realizado un modelo en 3D mostrado en la Figura 2.31 de una carcasa contenedora de los dispositivos electrónicos que intervienen en el Módulo de Recolección de Datos.

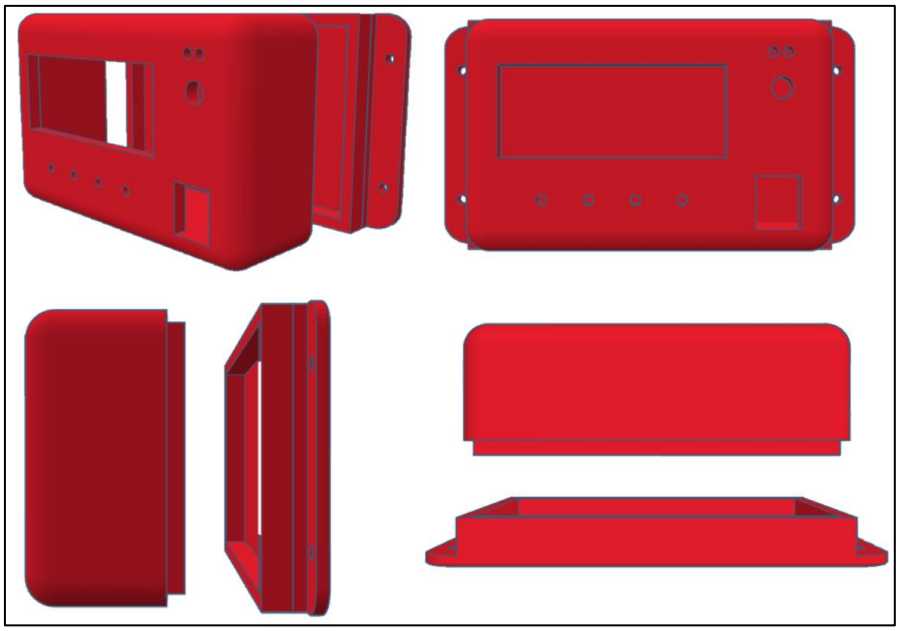


Figura 2.31. Vistas de la carcasa 3D para el Módulo de Recolección de Datos.

Además, se diseñó una base en 3D para sujetar la placa electrónica en la carcasa del Módulo y así evitar que se produzcan contactos involuntarios por movimiento o caída. Esta base se muestra en la Figura 2.32.

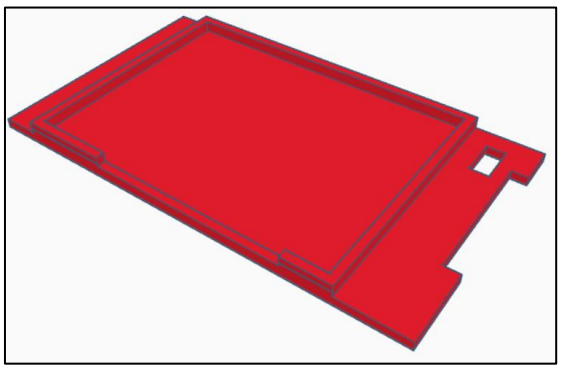


Figura 2.32. Base para la placa electrónica.

2.2.4. DISEÑO DE MÓDULO WEB

En esta sección se detallarán las fases de diseño del Módulo Web. Se explicará la arquitectura de software, estructura de modelo de base de datos. Se desarrollarán los diagramas de casos de uso para la aplicación web y se bosquejarán sus interfaces.

2.2.4.1. Arquitectura de Software

En la Figura 2.33 se puede observar cómo será la estructura de la aplicación web y móvil que constituyen los Módulos planteados. La aplicación seguirá el patrón arquitectónico MVC (ver 1.3.2), con esto, se divide la funcionalidad de la aplicación en tres roles principales. Se tiene la vista en donde estarán los módulos y componentes creados en la

aplicación de Angular para gestionar el diseño y funcionalidad de las interfaces web/móvil. El Controlador constará de los servicios de Firebase (en particular el servicio de Autenticación y la comunicación con la base de datos) y servicios creados en Angular para la integración de la Vista con el modelo. Por último, el Modelo que describirá la estructura de datos a utilizar mediante interfaces en Angular y la interconexión con la base de datos de Firebase *Cloud Firestore*; esto para la integración de datos. Todo esto, bajo el *hosting* en Firebase.

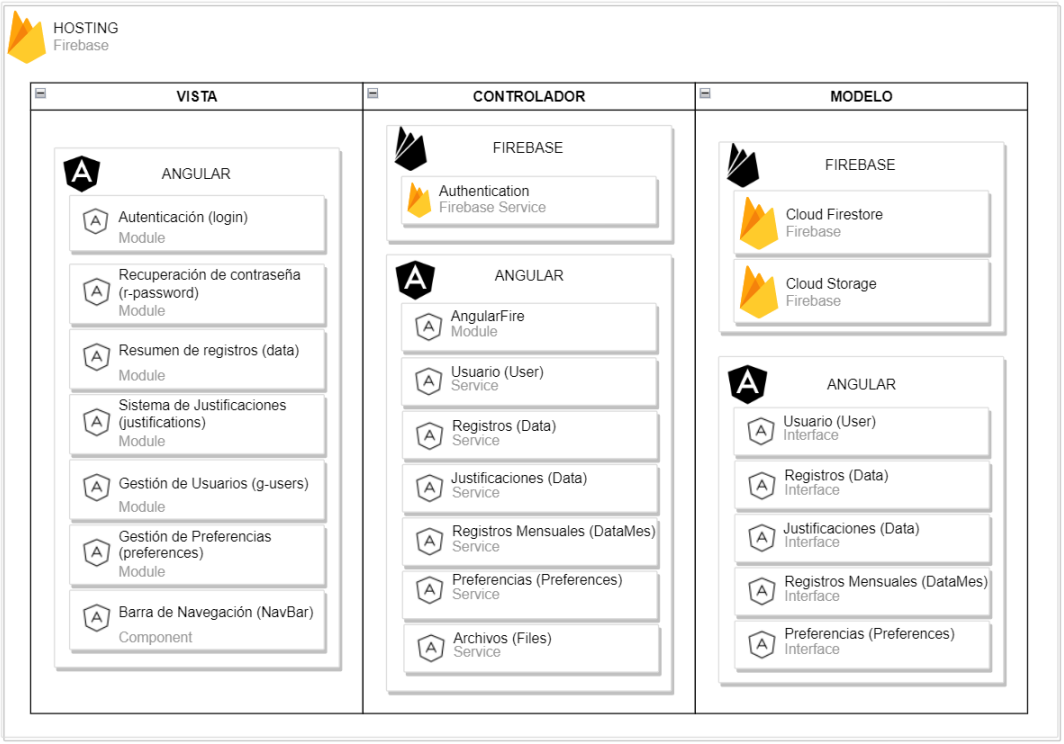


Figura 2.33. Arquitectura del Módulo Web/Móvil.

2.2.4.2. Estructura de almacenamiento en la base de datos

Como se va a usar una base de datos documental no relacional, se ha propuesto trabajar con los siguientes documentos: Usuarios, Registros, Justificaciones y Preferencias; esto permitirá almacenar de forma organizada todos los datos necesarios para el funcionamiento del sistema. A continuación, se muestra la estructura de los documentos mencionados:

❖ Documento para Usuarios

Para los datos de *usuarios*, se tomará en cuenta lo siguiente:

- **Uid:** identificador de usuario.
- **Nombre:** nombre completo del usuario.
- **Descr:** descripción del usuario

- **Email:** *email* de registro usado para inicio de sesión.
- **Rol:** rol de usuario. Será Administrador o Empleado.
- **HoraIn:** hora en que el usuario debe ingresar a la institución y generar un primer registro. En caso de que el usuario trabaje por horas en ciertos días, este campo será “00:00” o “12:00” am, así el Módulo de Recolección de Datos no emitirá faltas.
- **HorasDeTrabajo:** se definirá el tiempo (en horas y minutos) que el usuario debe cumplir en su jornada de trabajo.
- **IdHuella:** Identificador asociado con su número de huella dactilar en el Módulo de Recolección de Datos.

Los datos se almacenarán en la base de datos de la siguiente forma:

```
Usuarios: {
        uid: string                 (Identificador de usuario)
        nombre: string              (Nombre del usuario)
        descr: string               (Número de cédula del usuario)
        email: string               (Email de usuario)
        rol: string                 (Rol del usuario)
        horaIn: string              (Hora entrada)
        horasDeTrabajo: string      (Hora laboral)
        idHuella: number            (Identificador de huella dactilar)
}
```

❖ Documento para Registros

Para los datos de *Registros* se tendrán los siguientes campos:

- **Identificador del registro:** identificador que será generado en base a la fecha de registro seguido del identificador del usuario. Creado en el Módulo de Recolección de Datos.
- **Asistencia:** será de tipo *array*. En el primer campo será: Atraso, Falta, Presente. En el segundo campo podrá almacenar: Salida Temprana, Horas Extra.
- **Hora:** será un *array* en donde se almacenarán las horas en las cuales un usuario ha generado su respectivo registro. La hora de entrada y salida se calcularán con el primer registro y el último registro con índice impar (índice 1, 3). El índice será provisto por el número de registros que se encentren en el archivo *registrosDiarios.txt*. Serán almacenados en formato timestamp.

- **Horario:** almacenará el horario de trabajo del usuario que generó el registro.
- **Temperatura:** el valor de temperatura del usuario en grados Celsius. Será de tipo *array* ya que almacenará todas las temperaturas de los registros de entrada.
- **Horas Trabajadas:** cantidad de horas trabajadas por día. Este valor se calculará mediante registros pares empezando por el registro 0, es decir, el tiempo transcurrido entre el registro 0 y 1, luego, tiempo transcurrido entre registro 2 y 3 y así, sucesivamente.
- **Horas Extras:** en el caso de que exista, se almacenará la cantidad de horas extra en el día si se supera el valor de horas trabajadas en el día.
- **Justificaciones:** este campo existirá únicamente en el caso de que se genere una justificación. Es de tipo *array* y almacenará los identificadores de las justificaciones relacionadas con este registro.
- **IdUsuario:** identificador del usuario generó la justificación.
- **Usuario:** nombre del usuario.

Este registro se creará cuando el usuario genere su ingreso y se actualizará cuando el usuario genere más registros por medio del Módulo de Recolección de Datos.

Los datos se almacenarán en la base de datos de la siguiente forma:

```
Registros: {
        Id: string                  (Identificador del registro)
        Asistencia: array           (Estado de asistencia del registro)
        hora: array                 (Array de horas en formato timestamp)
        horario: string             (Cantidad de horas de trabajo del usuario)
        Temperatura:                (Valores de temperatura corporal)
        horasExtra: string          (Cantidad de horas extra)
        horasTrabajadas: string     (Cantidad de horas trabajadas)
        justificaciones: array      (Array de ids de justificación)
        IdUsuario: string           (Id del usuario que pertenece el registro)
        usuario: string             (Usuario que pertenece el registro)
}
```

❖ Documento para Justificaciones

Se tendrá un documento para almacenar las justificaciones con los siguientes campos:

- **Identificador de justificación:** identificador que se generará en base a un id aleatorio más un prefijo del tipo de justificación.

- **IdRegistro:** almacenará el identificador del registro que se está justificando.
- **IdUsuario:** identificador del usuario que generó la justificación.
- **Usuario:** nombre de usuario.
- **Tipo:** si la justificación generada es por falta, atraso, salida temprana, para validar horas extra, día extra, si el usuario no generó una salida.
- **Mensaje:** se almacenará la razón por la cual se justifica.
- **MotivoR:** se almacenará la razón por la cual se rechaza la justificación.
- **Fecha:** se almacenará la fecha del Registro que se está justificando.
- **Estado:** indicará si la justificación esta por revisar (solicitada), si ha sido rechazada o ha sido aceptada.
- **horaJustificada:** almacenará la cantidad de horas justificadas (extra o trabajadas) en el caso de que sea aceptada la justificación. El usuario administrador será quien determine cuántas horas se van a justificar.
- **Url:** link del documento de respaldo para la justificación.
- **NombreDoc:** se almacenará el nombre del documento de justificación subido.

Los datos se almacenarán en *Cloud Firestore* de la siguiente forma:

```
Justificaciones: {
        Id: string                  (Identificador de la justificación)
        idRegistro: string          (Identificador del registro)
        idUsuario: string           (Identificador del usuario)
        Usuario: string             (Nombre Usuario)
        Tipo: string                (Falta/Atraso/HorasExtra/SalidaTempr)
        Mensaje: string             (Motivo o razón)
        MotivoR: string             (Motivo de rechazo en la justificación)
        Fecha: string               (Fecha en formato timestamp)
        Status: string              (Solicitado/Rechazado/Aceptado)
        horaJustificada: string     (Horas extra/trabajadas justificadas)
        url: string                 (Url del documento de respaldo)
        nombreDoc: string           (Motivo de rechazo en la justificación)
}
```

❖ Documento para Preferencias

Para configuraciones del sistema, se tendrá un documento llamado *Preferencias*. En este documento se guardará lo siguiente:

- **ToleranciaIn:** es el margen de tolerancia, en minutos, para que el sistema marque un registro como atrasado.
- **ToleranciaOut:** es el margen de tolerancia, en minutos, para que el sistema permita generar justificaciones por horas extra después que genere una salida.
- **PuedeCrear:** registro que indicará al Módulo de Recolección de Datos si debe crear una plantilla de huella digital para un usuario nuevo.
- **PuedeActualizar:** registro que indicará al Módulo de Recolección de Datos si debe actualizar una plantilla de huella digital para un usuario determinado.
- **PuedeEliminar:** registro que indicará al Módulo de Recolección de Datos si debe eliminar una plantilla de huella digital.
- **PuedeActualizarUsuarios:** campo que indicará si el Módulo de Recolección de Datos requiere actualizar su lista de usuarios.
- **IdHuellaAcción:** registro que indicará al Módulo de Recolección de Datos que identificador de plantilla debe actualizar o eliminar.
- **horaRegistroFaltas:** registro que indica al Módulo de Recolección de Datos la hora (en formato HHmmss) en la cual se marcará como falta a los usuarios que no hayan registrado una entrada.
- **numHuellasRegistradas:** campo que almacenará la cantidad de huellas registradas en el sistema (incluyendo las eliminadas). Servirá para dar un identificador a una huella digital nueva.
- **timeServer:** registro que almacenará la hora actual del servicio de Firebase. Este registro se actualizará en base a una petición del Módulo de Recolección de Datos cada vez que este se encienda y tenga conexión a Internet. Seguido de esto, el Módulo actualizará la fecha y hora del reloj RTC.
- **EmailAdmin:** en este registro se almacenará el *email* del Administrador del sistema con el fin de que el usuario relacionado con este *email* no se pueda eliminar de la base de datos.

Estos datos se guardarán en un único documento dentro de la colección *Preferencias* de la siguiente manera:

```
Preferencias: {
    toleranciaIn: number          (Minutos de tolerancia)
    toleranciaOut:number          (Minutos de tolerancia, horas extra)
    puedeCrear: boolean           (Si se debe crear una huella)
    puedeActualizar: boolean      (Si se debe actualizar una huella)
```

puedeEliminar: boolean (Si se debe eliminar una huella)
puedeActualizarUsuarios: boolean (Si se debe eliminar una huella)
idHuellaAcción: number (Huella para actualizar/eliminar)
numHuellasRegistradas: number (Número de huellas registradas)
timeServer: timestamp (Hora para actualizar módulo RTC)
horaRegistroFaltas: number (Hora para generar faltas)

}

Por último, en la Figura 2.34 se muestra el diagrama de clases que detalla la estructura de los documentos mencionados anteriormente.

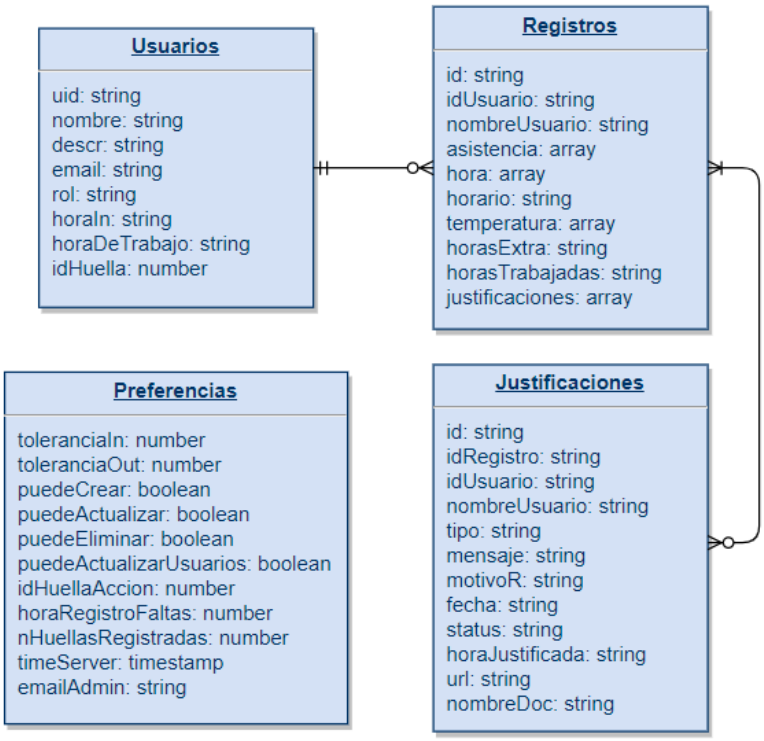


Figura 2.34. Diagrama del Clases para Módulo Web y Móvil.

2.2.4.3. Diagrama de casos de uso para la aplicación Web

Se ha diseñado un diagrama de casos de uso incluyendo las funcionalidades disponibles para el usuario con rol Administrador y Empleado. El diagrama se observa en la Figura 2.35.

El usuario Administrador, tendrá la posibilidad de iniciar al sistema, registrar un nuevo empleado, actualizar los datos del empleado, revisar todos los registros de asistencia, información de horas extra, información de horas trabajadas, registros de hora de entrada/salida y podrá generar justificaciones propias en caso de que se requiera; la información a visualizar será de todos los usuarios registrados en el sistema. Además, el

usuario Administrador podrá revisar las solicitudes de justificación y con el debido criterio, aceptar o rechazar la justificación. Estas justificaciones serán creadas en base a su necesidad (atraso, falta, horas extra, día extra, sin registro de salida o salida temprana).

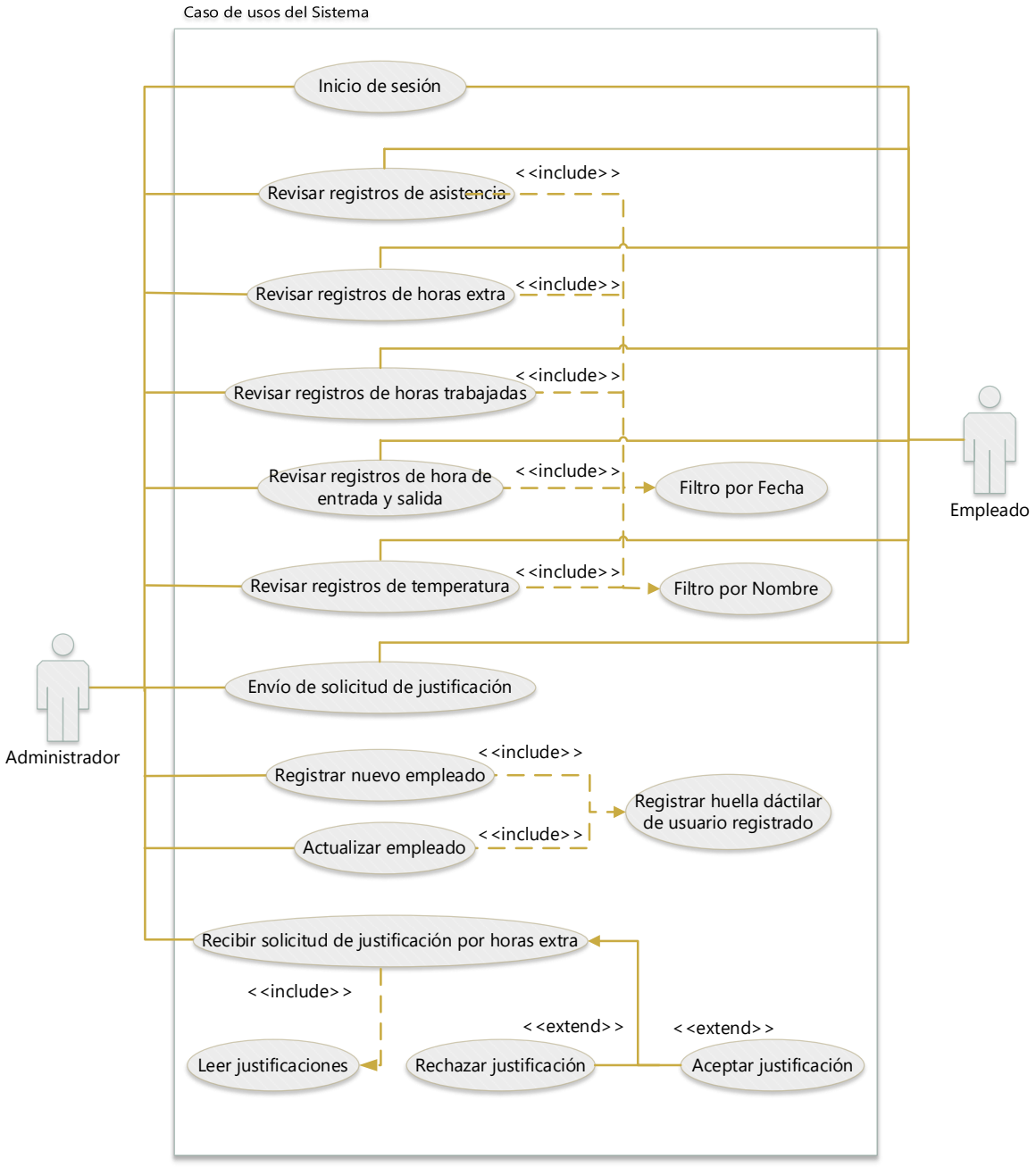


Figura 2.35. Diagrama de Casos de Uso del Sistema.

El usuario Empleado, tendrá la posibilidad de iniciar al sistema y revisar únicamente sus registros generados: registros de asistencia, de horas extra, de horas trabajadas, de hora de entrada y salida, de temperatura. Tendrá la posibilidad de generar solicitudes para que se justifique inasistencias, atrasos, salida temprana, registros sin salida, validación de horas y días extras

2.2.4.4. Bosquejo de la interfaz web del usuario

En el diseño de la interfaz del usuario, se ha tomado en cuenta las vistas principales en el sistema, las cuales son: inicio de sesión, visualización de registros, visualización de justificaciones y visualización de usuarios.

En primer lugar, se ha bosquejado una pantalla de inicio de sesión para que el usuario pueda ingresar sus credenciales y acceder al sistema. Se ha colocado un botón para el inicio de sesión y un botón para brindar una función de recuperación de contraseñas. Esta interfaz se muestra en la Figura 2.36.

Figura 2.36. Interfaz Inicio de Sesión (Web).

En la parte superior pantalla, se tendrá un NavBar que permitirá navegar por las distintas funcionalidades del sistema. Este permanecerá en todas las ventanas de la aplicación.

Se tendrá una interfaz para visualizar la lista de todos los registros. Esta interfaz se puede observar en la Figura 2.37. En esta ventana se podrá filtrar por nombre y fecha; el usuario podrá realizar búsquedas orientadas a sus necesidades. En esta interfaz, se podrán visualizar los registros diarios y el resumen de registros mensuales. Si un registro requiere una justificación, se visualizará un botón que permita crear una nueva. Para esto se tendrá una interfaz en donde el usuario pueda crear una justificación y detallar las circunstancias por la cual se solicita.

Para el usuario Administrador, se tendrá una interfaz en la que se podrá visualizar únicamente la información de las justificaciones generadas por los usuarios, Figura 2.38. Se mostrarán las justificaciones en un formato de tarjeta. El usuario podrá aceptar o rechazar la solicitud en base a su criterio.

Figura 2.37. Interfaz para Visualización de Registros (Web).

Figura 2.38. Interfaz para Justificaciones (Web).

Para el usuario Administrador, se tendrá una interfaz en donde se visualizarán todos los usuarios registrados, Figura 2.39. El usuario Administrador podrá crear, editar y eliminar usuarios.

Figura 2.39. Interfaz para Gestión de Usuarios (Web).

2.2.5. DISEÑO DEL MÓDULO MÓVIL

En esta sección se mencionará el diseño del Módulo Móvil. Únicamente se diseñará una interfaz para dispositivos móviles, por lo demás, se tendrá un funcionamiento similar al Módulo Web.

En el diseño de la interfaz del usuario en la aplicación móvil, se ha bosquejado una pantalla de inicio de sesión para que el usuario pueda ingresar sus credenciales y acceder al sistema, Figura 2.40. Además, se ha colocado un botón para brindar una función de recuperación de contraseña.

Figura 2.40. Interfaz de Inicio de Sesión (Móvil).

Debido a que se tiene un espacio pequeño, comparado con la interfaz web, se ha creado un botón que redirige a una página para visualizar todos los detalles del registro, esto para ambos casos mencionados anteriormente.

Se ha diseñado una interfaz para cada informe, las cuales son: Resumen de Asistencia por día, Resumen de asistencia por mes, Figura 2.41.

En esta interfaz se desplegará la lista de registros diarios, mostrará el usuario, la fecha y un botón en donde desplegará una ventana con el resto de información. El usuario podrá justificar en esta ventana en el caso que sea necesario; para esto se tendrá una interfaz en donde el usuario pueda crear una justificación, pueda seleccionar el día y la circunstancias por la cual solicita justificación.

También, se podrá ver el resumen del mes, detallando las horas extras y las horas trabajadas. En ambos casos, se manejarán filtros de nombre y fecha.

Se tendrá una interfaz para que el usuario Administrador pueda ver las solicitudes de justificación Figura 2.42. Se tendrá la opción de visualizar todas las justificaciones generadas por los empleados. El usuario Administrador podrá aceptar o rechazar la solicitud en base a su criterio.

Figura 2.41. Interfaz Asistencia e Informe de Entrada y Salida (Móvil).

Figura 2.42. Interfaz para Justificaciones (Móvil).

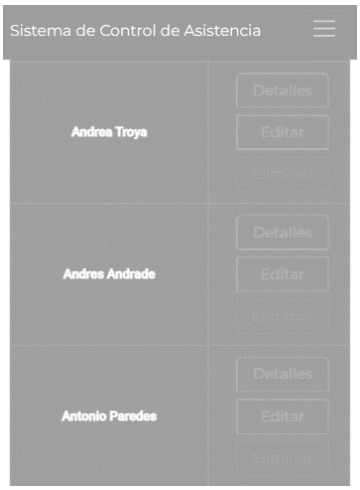


Figura 2.43. Interfaz para Gestión de Empleados (Móvil).

Para el usuario Administrador, se tendrá una interfaz en donde se visualizarán todos los usuarios registrados Figura 2.43. El usuario administrador podrá crear, ver, editar y eliminar usuarios.

2.3. IMPLEMENTACIÓN

Al inicio de este capítulo, se detallarán las actividades a realizar mediante el tablero Kanban. Se instalarán las herramientas necesarias para la implementación de los módulos propuestos. Se codificará las funcionalidades del Módulo de Recolección de Datos mediante el entorno de desarrollo integrado de Arduino [40]. Se codificará el Módulo Web y Módulo Móvil en el editor de código Visual Studio Code siguiendo el patrón de diseño MVC. Toda la implementación se desarrollará en base a lo planteado en la etapa de diseño.

2.3.1. ACTUALIZACIÓN DEL TABLERO KANBAN

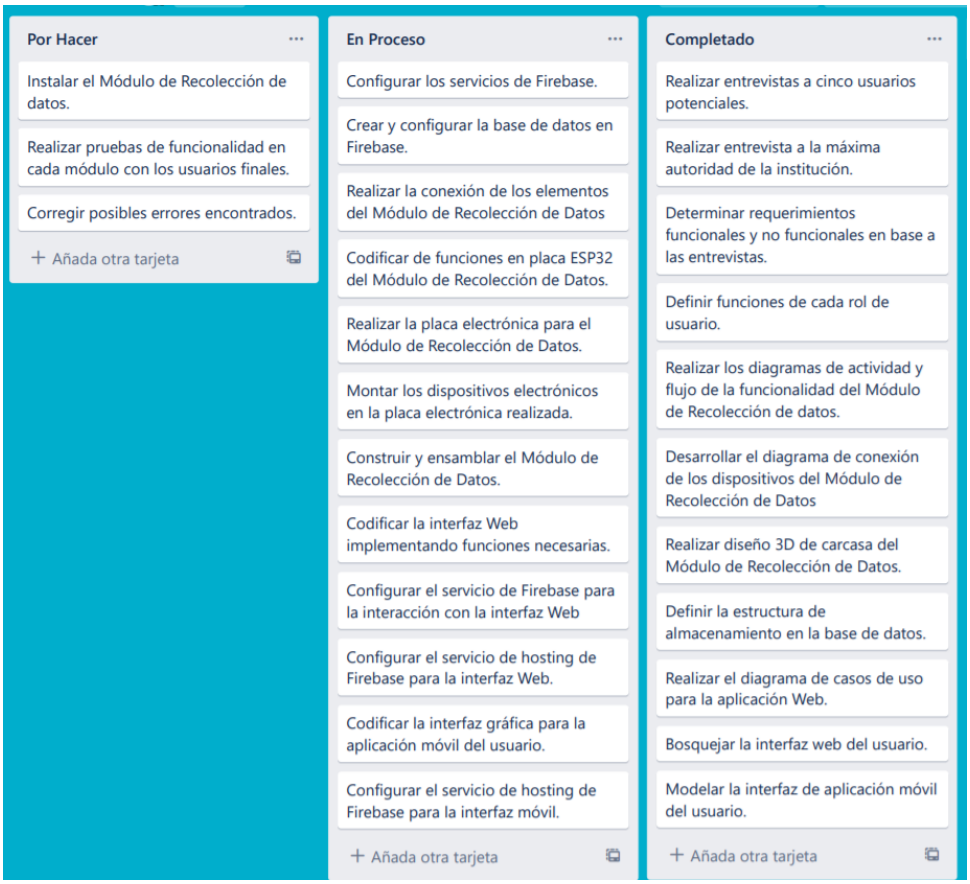


Figura 2.44. Actualización de tablero Kanban para fase de implementación.

En la Figura 2.44 se observa la actualización del tablero Kanban. En primer lugar, se encuentra la columna “Por Hacer” que detalla todas las actividades que están pendientes para su desarrollo, en este caso, las actividades de la fase de Pruebas de Funcionamiento. En segundo lugar, se encuentra la columna “En proceso” que detalla las actividades que se encuentran en desarrollo respecto a la fase correspondiente, en este caso, la fase de Implementación. Por último, se encuentra la columna “Completado” que hace referencia a las actividades finalizadas, en este caso, las actividades de la fase de Diseño.

2.3.2. CONFIGURACIÓN DE SERVICIOS DE FIREBASE

Para la configuración de los servicios, es necesario tener una cuenta de Google. Mediante cualquier navegador, se puede acceder a los servicios de Firebase en *firebase.google.com*. Para crear un nuevo proyecto en Firebase, se selecciona *comenzar* (ver Figura 2.45) y después en *agregar un nuevo proyecto*.

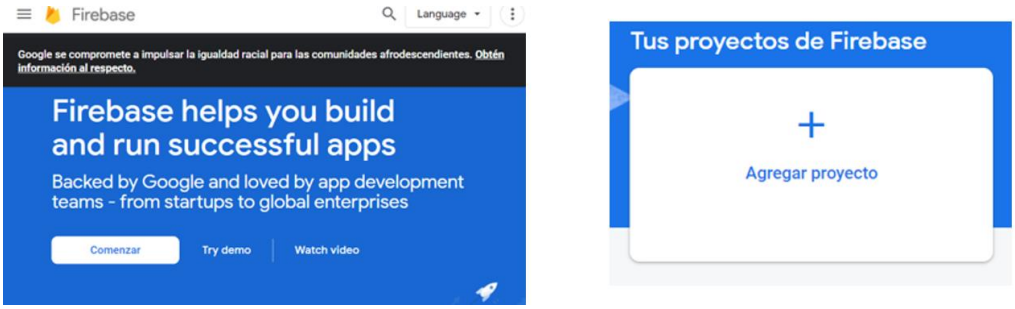


Figura 2.45. Creación de un proyecto en Firebase.

Se creará un nuevo proyecto llamado *SCA_CDICI* (Sistema de Control de Asistencia de CDICI). Dentro de este proyecto se podrá acceder a las funcionalidades y servicios que brinda Firebase (ver Figura 2.46).

Figura 2.46. Definición del nombre de proyecto en Firebase.

En la consola de Firebase, se selecciona en *Firestore Database* y se accede a la opción desplegada Crear base de datos (ver Figura 2.47). De esta forma se crea una base de datos a la que se enlazará a la aplicación web y móvil.

Figura 2.47. Creación de la base de datos *Cloud Firestore* en Firebase.

Cloud Firestore brinda la posibilidad de personalizar el acceso a sus datos, con esto, se orienta la privacidad y acceso a las necesidades del proyecto. En este caso, se debe actualizar las reglas de *Cloud Firestore* para limitar el acceso de lectura/escritura

únicamente a usuarios autenticados y así asegurar que los datos sean privados; se denegarán todas las operaciones de lectura y escritura de terceros. Para esto, se edita las reglas de *Cloud Firestore* de la forma en que se detalla en el Código 2.1.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

Código 2.1. Reglas para lectura y escritura en *Cloud Firestore*.

Así mismo, se inició el servicio de Cloud Storage para almacenar archivos como justificaciones o logos de la institución. En el Código 2.2 se muestran las reglas que se configuraron para el acceso a archivos solo a personas autenticadas los servicios de Firebase.

```
rules_version = '2';
service firebase.storage {
  match /b/{bucket}/o {
    match /{allPaths=**} {
      allow read, write: if request.auth!=null;
    }
  }
}
```

Código 2.2. Reglas para subida y bajada de archivos en Cloud Storage

2.3.3. CONEXIONES DEL MÓDULO DE RECOLECCIÓN DE DATOS

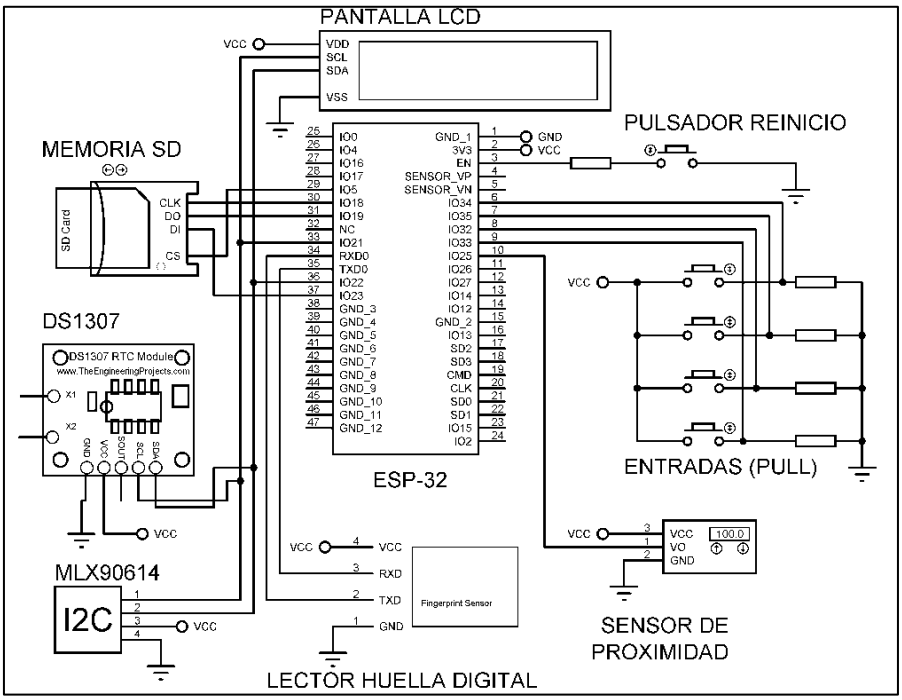


Figura 2.48. Conexión de elementos en el módulo electrónico.

Para realizar la codificación del Módulo, es necesario la interconexión de todos los dispositivos electrónicos involucrados. En la Figura 2.48 se muestra la conexión de los elementos tal y como se detalló en la Figura 2.29.

2.3.4. CODIFICACIÓN DEL MÓDULO DE RECOLECCIÓN DE DATOS

En esta sección, se menciona la estructura de codificación del programa en la placa ESP-32. Partiendo desde las librerías, estructuras, variables globales y funciones. La codificación de este Módulo se encuentra en el ANEXO B.

2.3.4.1. Librerías

Las librerías son porciones de códigos que permiten añadir nuevas funcionalidades al programa que se está desarrollando; cada librería está orientada a cubrir una funcionalidad en específico. Generalmente se usan para potenciar la aplicación y brindar compatibilidad con módulos externos a la placa de desarrollo. En la Tabla 2.17 se detallan las librerías usadas en este prototipo.

Tabla 2.17. Librerías utilizadas en la codificación del Módulo de Recolección de Datos.

Nombre	Descripción	Versión
Adafruit_Fingerprint.h	Biblioteca Arduino para el uso de un módulo lector de huellas dactilares.	2.0.7
SoftwareSerial.h	Librería para acceder a la comunicación serial en pines digitales. Usado en la comunicación del lector de huellas dactilares.	1.0
LiquidCrystal_I2C.h	Librería para el controlar pantallas LCD con comunicación I2C.	1.1.2
SD.h	Gestiona información de la tarjeta SD. Además, permite leer y escribir archivos almacenados.	1.2.4
RTClib.h	Librería para el uso de módulos RTC (RealTimeControl) DS1307, DS3231, PCF8523, PCF8563	2.0.2
Adafruit_MLX90614.h	Librería para el uso del sensor MLX90614. Brinda funciones para la lectura de temperatura.	2.1.3
WiFiManager.h	Administrador de configuración Wifi. Se conecta automáticamente a la última red registrada, si no la encuentra, crea un portal de configuración web en donde se ingresa las credenciales de la red Wifi.	2.0.5
Firebase_ESP_Client.h	Librería para la comunicación entre las funciones y servicios de Firebase con la placa de desarrollo ESP32.	3.14.3
addons/TokenHelper.h	Librería para administrar el proceso de generación de tokens de Firebase. Usado para verificar si se pudo autenticar el usuario.	1.0
ESP32Ping.h	Librería que brinda funciones para realizar *ping* a un host remoto y comprobar conexión a Internet.	1.7

2.3.4.2. Variables Globales y Estructuras

Se crearon estructuras, constantes y variables globales para la gestión de los datos en código. A continuación, se detalla cada una de ellas.

2.3.4.2.1. Estructuras

Se han creado un tipo de datos compuesto para facilitar el manejo de datos en el código.

Estas estructuras permiten almacenar los datos de una manera ordenada y así organizar los datos en categorías determinadas, como: datos de usuarios, datos de registro e información de fecha y hora.

Las estructuras que se crearon en este Módulo para el manejo de datos se describen en la Tabla 2.18.

Tabla 2.18. Estructuras definidas para el manejo de datos en código.

Nombre	Datos		Descripción
	Nombre	Tipo	
Usuario	Id	String	Estructura para almacenar los datos de usuarios leídos desde Firebase y usarlos en código.
	nombreUsuario	String	
	idHuella	Int	
	horaIn	String	
	horasDeTrabajo	String	
Fecha	Hoy	DateTime	Estructura para extraer la fecha actual en formato *DateTime*, fecha y hora en cadenas de caracteres.
	timeToSet	DateTime	
	fecha	Char[12]	
	hora	Char[10]	
Registro	path	String	Estructura para organizar el registro que se va a guardar. El path será la dirección del documento en *Cloud Firestore* y la mascará tendrá los valores a modificar si es un registro ya creado. Se almacenará el id de registro, el id del usuario, el nombre del usuario a quien pertenece el registro, su estado de asistencia, hora de registro, horas trabajadas, número de registro y temperatura.
	Id	String	
	idUsuario	String	
	usuario	String	
	asistencia	String	
	hora	String	
	numRegistro	Int	
	horasTrabajadas	String	
	temperatura	String	
RegistroDiario	numRegistro	Int	Almacenará la cantidad de registros que se han generado en el día. Hora de inicio contiene la hora de generación del registro. Y horasTrabajadas contiene el número total de horas trabajadas.
	horaInicio	String	
	horaTrabajadas	String	

2.3.4.2.2. Variables Globales

Se declararán variables globales para la gestión de los datos en todas las funciones del programa.

Estas variables se mencionan y describen en la Tabla 2.19.

Tabla 2.19. Descripción de variables globales definidas en código (Parte 1 de 3).

Nombre	Tipo	Descripción
Wm	WiFi Manager	Variable para acceder a las funciones de la librería *WifiManager*. Gestiona la conexión a Internet
Auth	Firebase Auth	Se define la variable *FirebaseAuth* gestionar el proceso de autenticación con los servicios de Firebase.
Config	Firebase Config	Se define la variable *FirebaseConfig* para acceder a los datos de configuración y generación de tokens por autenticación de Firebase.
dataMillis	unsigned long	Variable para obtener el tiempo (en milisegundos) a partir de la ejecución de una parte determinada del código.
dataResetMillis	unsigned long	Tiempo transcurrido a partir de una notificación de desconexión de wifi. Se realiza esta acción para que el Módulo vuelva a conectarse a la red Wifi después un reinicio.
dataTryFirebase	unsigned long	Tiempo transcurrido para que el Módulo vuelva a intentar subir un registro a Firebase después de un error al subir datos.
dataResetFirebase	unsigned long	Tiempo transcurrido después de un error al subir datos a Firebase. Con esto, se reiniciará el Módulo si no logra subir los registros pendientes para intentar reconectar a Firebase.
horaReinicio	unsigned long	Cantidad de milisegundos que deberá pasar para que el módulo se reinicie por perdida de conexión Wifi.
debeReiniciar	bool	Variable que indica si el módulo se debe reiniciar tras perdida de conexión Wifi; se marca como *true* para reiniciar. En el caso de que el Módulo logre reconectarse la red, esta variable vuelve a *false*.
reset	Int[4]	*Array* en donde se almacenarán las horas (en formato HHmmss) para gestionar el reinicio periódico.
debeReiniciarFirebase	Bool	Variable que indica si debe reiniciar el Módulo por un error de subida de datos a Firebase. Será *true* después de que ha transcurrido un determinado tiempo controlado por *dataResetFirebase*.
registrosPendientes	Bool	Variable que indica si existen registros guardados en SD. Su valor es *true* cuando se guarda un registro en memoria SD y vuelve a *false* cuando se suben los registros a Firebase.

Tabla 2.19 Descripción de variables globales definidas en código (Parte 2 de 3).

Nombre	Tipo	Descripción
numPagLCD	Int	Variable que contendrá el número de página en la pantalla LCD. 0: impresión de fecha y hora. 1: impresión de estado de conexión a Internet y la dirección IP. Valor que se modificará cuando se pulse el botón PULL2
rtc	RTC_DS3231	Variable para acceder a las funciones de la librería RTC para el módulo DS3231. Permite obtener la hora actual del módulo.
diasDeLaSemana	String[7]	*Array* en donde se almacenarán los nombres de los días de la semana.
tipoDeAsistencia	String[3]	*Array* en donde se almacenarán los nombres los tipos de asistencia que tendrá el sistema: Atraso, falta, presente.
Lcd(0x27,20,4)	LiquidCrystal	Variable para acceder a las funciones de la librería *LiquidCrystal* con el protocolo I2C. Se define la dirección para la comunicación I2C (0x27) y las dimensiones de la pantalla (20 x 4 caracteres).
mySerial(16, 17)	Software Serial	Variable para asignar pines TX y RX para comunicación UART. Se definen los pines 16 y 17 ya que son pines digitales TX y RX usados para la comunicación con el lector de huella digital.
Finger	Adafruit Fingerprint	Variable para acceder a las funciones de la librería *Adafruit_fingerprint*. Se inicializa con la variable serial mencionada anteriormente.
miArchivo	File	Variable para acceder a las funciones de la librería SD. Sirve para abrir y cerrar archivos para su lectura o modificación.
idUsuario	String	Variable para almacenar el identificador de usuario. Funciona únicamente cuando se debe crear, actualizar o eliminar una plantilla de huella digital en el sistema.
idFP	uint8_t	Variable para almacenar el identificador de huella digital. Funciona únicamente cuando se debe crear, actualizar o eliminar una plantilla de huella digital en el sistema.
Mlx	Adafruit MLX90614	Variable para acceder a las funciones de la librería *Adafruit_MLX90614* compatible con el sensor de temperatura corporal.
toleranciaIn	Int	Variable donde se almacenará el tiempo (minutos) de tolerancia para marcar un registro como atraso. Leído de Firebase.
numHuellas Registradas	Int	Variable donde se almacenará la cantidad de huellas digitales guardadas en el sistema. Valor leído desde la Firebase. Servirá para asignar un id de huella digital a un usuario nuevo.
Timeserver	String	Variable donde se almacenará el dato de fecha y hora actual del servicio de Firebase.
Usuarios	Usuario [ ]	*Array* de la estructura usuarios en donde se extraerán los datos leídos en el archivo *usuarios.txt* en la tarjeta SD.

Tabla 2.19 Descripción de variables globales definidas en código (Parte 3 de 3).

Nombre	Tipo	Descripción
horaRegistroFaltas	Int	Variable que almacenará la hora máxima (en formato HHmmss) para generar un registro. Posterior a esta hora, se marcarán registros faltantes como falta. Leído de Firebase.
generar RegistroDiario	Boolean	Variable que permitirá verificar y generar registros faltantes.
puede ActualizarFecha	Boolean	Variable que permitirá actualizar o no la hora del módulo RTC.

2.3.4.2.3. Constantes

Se declararán tipos de datos de valor fijo durante la ejecución del programa para evitar que su valor cambie. Se crearon las constantes detalladas en la Tabla 2.20.

Tabla 2.20. Descripción de constantes definidas en código.

Nombre	Descripción
API_KEY	Llave del proyecto para la aplicación de Firebase en el código.
FIREBASE_PROJECT_ID	Identificador de proyecto de Firebase. Este Id es generado automáticamente cuando se crea un proyecto.
USER_EMAIL	Email del usuario para autenticación con Firebase y *Cloud Firestore*.
USER_PASSWORD	Contraseña del usuario para autenticación con Firebase y *Cloud Firestore*.
NUM_USUARIOS	Cantidad de máxima de usuarios que se registrarán en el sistema.
PULL_1	Pin del pulsador encargado de leer el documento Preferencias.
PULL_2	Pin del pulsador encargado de cambiar de página de la pantalla LCD.
PULL_3	Pin del pulsador encargado de aceptar la generación de un registro.
PULL_4	Pin del pulsador encargado de cancelar la generación de un registro
SENSOR_1	Contendrá el número de pin en el que se conectará la salida del sensor de proximidad.
NUM_MAX_REGISTRO	Cantidad máxima de registros que un usuario podrá generar en un día.
REMOTE_HOST	Contendrá el dominio del *host* al cual se realizará *ping* para confirmar la conexión a Internet del módulo.

2.3.4.3. Funciones

Se han codificado funciones para brindar operabilidad al Módulo de Recolección de Datos y han sido nombradas en base a utilidad en el ciclo de ejecución.

Las funciones con tipo de retorno se listan en Tabla 2.21 y las funciones tipo *void* sin retorno se mencionan en la Tabla 2.22.

Las descripciones de las funciones, tipos de retorno y argumentos de entrada se detallan en el ANEXO C.

Tabla 2.21. Funciones con tipo de retorno.

Función	Tipo	Retorno
cargarRegistrosFaltantes	boolean	True: si tiene registros pendientes False: si no tiene registros pendientes
crearDocumentoFirebase	Boolean	True: si se ha creado correctamente False: si ha fallado la creación
actualizarDocumentoFirebase	Boolean	True: si se ha actualizado correctamente False: si ha fallado la actualización
agregarArrayFirebase	Boolean	True: si se ha actualizado correctamente False: si ha fallado la actualización
generarDatosRegistro	String	Identificador de registro generado
getRegistroDiario	RegistroDiario	Estructura *RegistroDiario* con los valores atraídos del último identificador de usuario almacenado en el archivo *registrosDiarios.txt.*
calculoHorasTrabajadas	String	Total de horas trabajadas desde el primer registro de entrada
getUsuario	Usuario	Estructura de los datos del usuario que generó un registro
getRegistro	Registro	Estructura de los datos de registro
getFecha	Fecha	Estructura con los datos de fecha recibidos
generarEstadoAsistencia	String	Estado de asistencia (Presente, Falta, Atraso o Día Extra)
leerTemperatura	String	Valor de temperatura (en el caso de que el usuario no se acerque al sensor, devolverá un valor de: “-”)
guardarRegistroFirebase	String	Identificador del registro guardado
guardarRegistroSD	String	Identificador del registro guardado
solicitarHuella	int	Identificador de plantilla de huella digital
guardarHuellaFirebase	Boolean	True: pudo guardar el identificador False: no pudo guardar el identificador
eliminarHuellaDigital	Boolean	True: pudo remover el identificador False: no pudo remover el identificador
getSubCadena	String	Subcadena encontrada dependiendo de un carácter de separación y su posición

Tabla 2.22. Funciones tipo Void sin retorno

Función	Utilidad
Setup	Iniciar y configurar los dispositivos y servicios para el funcionamiento del Módulo.
leerPreferencias	Leer los valores del archivo *preferencias.txt*
verificarArchivosDiarios	Verificar si los archivos diarios pertenecen solo al día presente.
generarRegistrosFaltantes	Genera faltas a usuarios que no han registrado una entrada
setModuloRTC	Actualiza la hora en el módulo RTC
Loop	Ejecutar continuamente el código principal esperando algún cambio para acceder a las funcionalidades del programa
gestionReinicio	Gestionar los reinicios del Módulo
gestionFaltas	Gestionar la hora de registro de faltas
gestionRegistrosPendientes	Gestionar carga de registros pendientes
imprimirConexion	Imprimir el estado de conexión en la pantalla LCD
imprimirFechaLCD	Imprimir información de fecha y hora en la pantalla LCD
getIDHuella	Identificar si se ha ingresado una huella en el lector y obtiene su identificador
guardaRegistroDiario	Almacenar en el archivo *registrosDiarios.txt* el identificador de un determinado registro
leerOpciones	Leer el documento de *preferencias/sistema* en Firebase para determinar si se debe realizar una acción
obtenerUsuariosFirebase	Leer la colección de *usuarios* y almacenarlos en el archivo *usuarios.txt* de la memoria SD
getDataUsuarios	Cargar en código la información del archivo *usuarios.txt*
crearPlantillaHuella	Crear una plantilla de huella digital en la memoria del lector

2.3.4.4. Ciclo de ejecución del programa

En la Figura 2.49 se mostrará el ciclo de ejecución para el programa desarrollado. Este ciclo corresponde a la ejecución de las funciones que intervienen en la función principal *loop()*; se asume una ejecución limpia con todos los procesos completados. Se puede observar que, dentro de esta función, continuamente se está evaluando las funciones de gestión de reinicio, gestión de registros pendientes y gestión de faltas.

Además, se evaluará la variable global *numPagLCD* para determinar el contenido que debe mostrar el Módulo en pantalla. Si la variable es 0, mostrará la fecha y hora (función: *imprimirFechaLCD*) y si la variable es 1, mostrará el estado de conexión y la IP con la que se está conectando a la red Wifi (función: *imprimirConexion*).

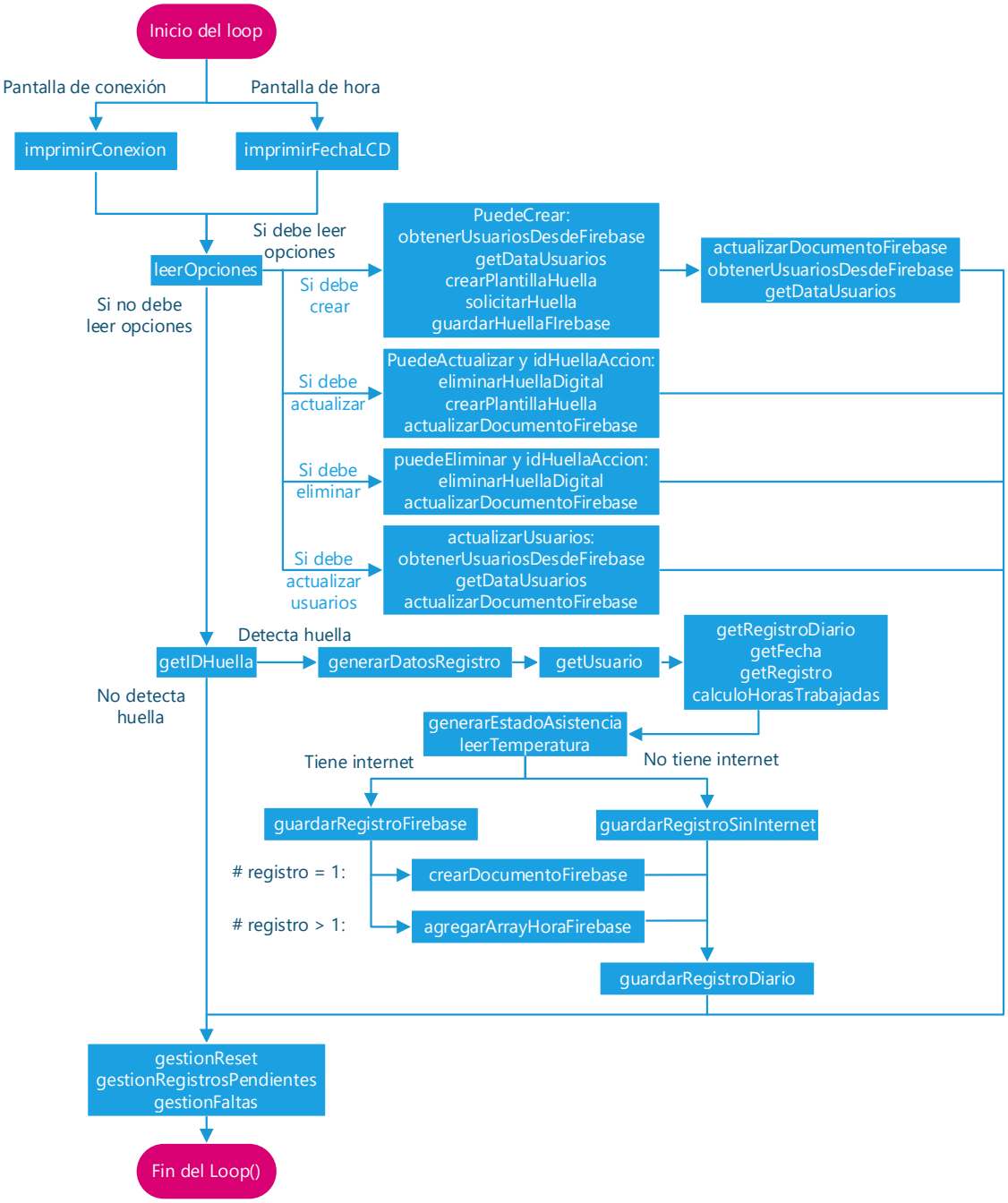


Figura 2.49. Ciclo de ejecución del programa para el Módulo de Recolección de Datos.

2.3.5. INTEGRACIÓN DEL MÓDULO CON LOS SERVICIOS DE FIREBASE

Se debe obtener la Clave de *API web* del proyecto creado para que el Módulo consuma los servicios de Firebase. Para esto, se debe acceder a la consola de Firebase, *Descripción general del Proyecto* y luego en *Configuración del proyecto*. Ahí se podrá observar las propiedades del proyecto, algunas como: *nombre del proyecto*, *ID del proyecto*, *número de proyecto*, *ubicación predeterminada de los recursos de Google Cloud* y *clave de API web*, como se muestra en la Figura 2.50. Los valores de *ID del proyecto* y *clave de API Web*

deben ser almacenados en las constantes *FIREBASE_PROJECT_ID* y *API_KEY* respectivamente; constantes definidas en la sección 0.

Figura 2.50. Propiedades del proyecto en Firebase.

En la Figura 2.50 (b) se ha ocultado información para evitar vulnerabilidades en el sistema.

2.3.6. MONTAJE DE DISPOSITIVOS EN PLACA ELECTRÓNICA

La placa electrónica desarrollada para la interconexión de los dispositivos electrónicos que conforman este Módulo se presenta en la Figura 2.51 y Figura 2.52.

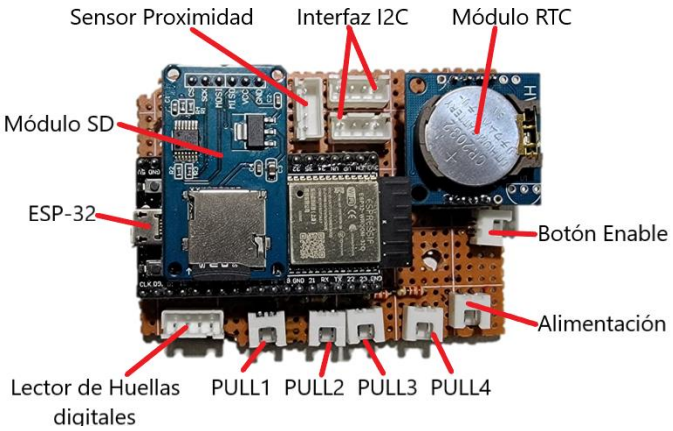


Figura 2.51. Vista superior de placa electrónica del Módulo de Recolección de Datos.

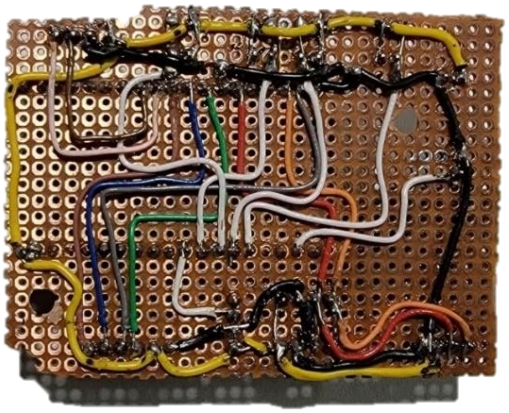


Figura 2.52. Vista inferior de placa electrónica en el Módulo de Recolección de Datos.

Para su construcción, se ha utilizado una baquelita perforada donde se ha unido cada dispositivo electrónico, tal y como se define en la Figura 2.29.

2.3.7. ENSAMBLE DEL MÓDULO DE RECOLECCIÓN DE DATOS

El Módulo de Recolección de Datos ya ensamblado conforme a la distribución presentada se muestra en la Figura 2.53 y la carcasa en la Figura 2.31. Para esta construcción, se ha colocado los dispositivos electrónicos en su lugar y se ha acoplado la placa electrónica para la interconexión de los mismos.

Figura 2.53. Módulo de Recolección de Datos ensamblado.

Internamente, se incluyó la base para la placa electrónica (Figura 2.32) que está sujeta en el módulo de la pantalla LCD. El interior del Módulo de Recolección de Datos se muestra en la Figura 2.54.

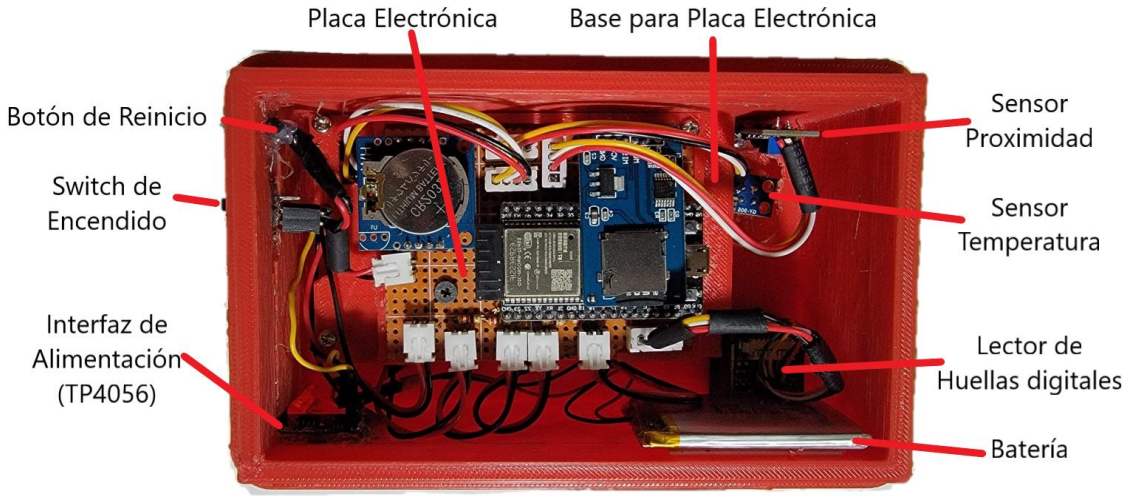


Figura 2.54. Interior del Módulo de Recolección de Datos

2.3.8. CREACIÓN DE LA APLICACIÓN WEB (MÓDULO WEB)

El desarrollo de la aplicación web se realizó en una maquina con las siguientes características: Sistema Operativo Windows 10, memoria RAM de 16, almacenamiento SSD 512 GB y procesador Intel i7-10750H.

2.3.8.1. Aplicaciones necesarias

En primer lugar, se debe instalar Node.JS desde su página oficial [41]. Es necesario bajar el instalador y ejecutarlo en el equipo. La versión que se ha usado para este prototipo es: v14.16.0. Después, se debe instalar el editor de código Visual Studio Code [42]. Para esto, se accede a la página oficial del editor, se debe descargar el instalador y ejecutarlo. La versión usada para este prototipo es: v1.64.

2.3.8.2. Comandos

Una vez instaladas las dos aplicaciones mencionadas, se debe abrir el editor de código Visual Studio Code y crear un nuevo terminal mediante la siguiente combinación de teclas: *Ctrl+Mayús+*`.

En la Tabla 2.23 se muestran los comandos y su descripción de los comandos que fueron ejecutados para la instalación de Angular [43], creación el proyecto del Módulo Web, agregar características e incluir herramientas.

La opción "*–save*" hace que el gestor de archivos de Node.JS incluya estos paquetes dentro del archivo de dependencias del proyecto.

Los comandos para la generación de esquemas en Angular se muestran en la Tabla 2.24.

Tabla 2.23. Comandos a ejecutar para la creación del proyecto e instalación de dependencias.

Orden	Comando	Descripción	Versión
1	npm install -g @angular/cli	Instalación del framework Angular	12.2.10
2	ng new mweb	Creación del proyecto en Angular llamado *mweb*.	No aplica
3	ng add @angular/fire	Agrega *AngularFire* al proyecto (Servicios de Firebase)	6.1.5
4	npm install bootstrap –save	Instalación de estilos *Bootstrap*	5.1.1
5	npm install jquery –save	Instalación de *jquery* (librería de *JavaScript*)	3.6.0
6	npm install @popperjs/core –save	Instalación de *popperJs* (crear etiquetas con información HTML en código JS)	2.10.1
7	npm install rxjs	Instalación de *RXJS* (gestión de eventos y uso de objetos observables)	6.6.0

Tabla 2.24. Lista de comandos para generar elementos en Angular.

Esquema Angular	Comando
Generar Interfaz	ng generate interface RUTA/NOMBRE_INTERFAZ
Generar Clase	ng generate class RUTA/NOMBRE_CLASE
Generar Servicio	ng generate service RUTA/NOMBRE_SERVICIO
Generar Guard	ng generate guard RUTA/NOMBRE_GUARD
Generar Pipe	ng generate pipe RUTA/NOMBRE_PIPE
Generar Módulo	ng generate module NOMBRE_MÓDULO -m=app —route NOMBRE_RUTA
Generar Componente	ng generate component NOMBRE_COMPONENTE -m=app —route NOMBRE_RUTA

2.3.8.3. Instalación de estilos

Se instalaron estilos en el proyecto para brindar personalización a las vistas *HTML*. Se usó la herramienta de *Bootstrap*, es un *framework* para la modificación de interfaces web. Entre sus funciones están, dar estilos a los elementos visuales de una interfaz y permitir que la misma se adapte al tamaño de la pantalla en que se presenta. La funcionalidad de *Bootstrap* en proyectos basados en Angular es complementada por *jquery* y *popperJs*; librerías instaladas en el proyecto únicamente para el funcionamiento de *Bootstrap*. La instalación de este *framework* se realizó mediante el gestor de paquetes de Node.JS.

Como último paso para poder aplicar los estilos, se debe especificar la ruta de 4 archivos para que Angular pueda usar las funciones de *Bootstrap*. Se debe modificar el archivo *angular.json* añadiendo las direcciones de las dependencias en los campos *styles* y *scripts* como se muestra en el Código 2.3.

```
"styles": [
  "node_modules/bootstrap/dist/css/bootstrap.min.css",
  "src/styles.css"
],
"scripts": [
  "node_modules/jquery/dist/jquery.min.js",
  "node_modules/@popperjs/core/dist/umd/popper.min.js",
  "node_modules/bootstrap/dist/js/bootstrap.min.js"
]
```

Código 2.3. Inyección de estilos de *Bootstrap* al proyecto creado.

2.3.9. CONFIGURACIÓN DEL SERVICIO DE FIREBASE

En esta sección, se explica cómo se configura un proyecto de Firebase para consumir sus servicios en una aplicación web desarrollada en Angular.

El código del Módulo Web se encuentra en el ANEXO D.

2.3.9.1. Autenticación

Los servicios de Firebase brindan opciones de autenticación por: Google, Facebook, número telefónico, correo electrónico y contraseña, entre otros. Para este prototipo, se usó autenticación por correo electrónico y contraseña. En la Figura 2.55 se muestra la activación de este método de autenticación en el proyecto seleccionado. Para activar este servicio, se accedió a la consola de Firebase, en la opción *Authentication/Sign-in-method*.

En la sección *Authentication/Sing-In-method/Proveedores* de la consola de Firebase, se puede observar los métodos de autenticación habilitados en el proyecto.

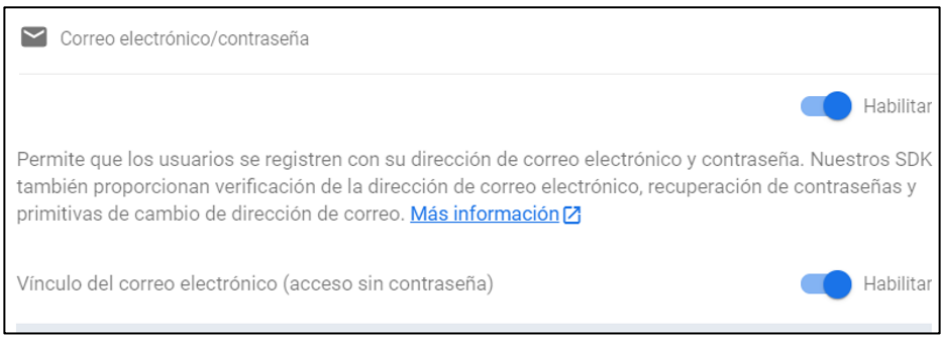


Figura 2.55. Habilitar servicio de autenticación en Firebase por usuario y contraseña.

2.3.9.2. Configuración de Aplicación en Firebase

Para configurar los servicios de Firebase en Angular, se creó un nuevo proyecto Web en *Descripción general del proyecto* y *Configuración del proyecto*. En *Tus apps* se selecciona *agregar una app* para vincular Firebase con la aplicación web. Se selecciona la plataforma a la cual se brindará el servicio (aplicación web) y se coloca el nombre por el cual Firebase la identificará, en este caso se llamará *SCA_CDICI_WEB*. Esto se muestra en la Figura 2.56.

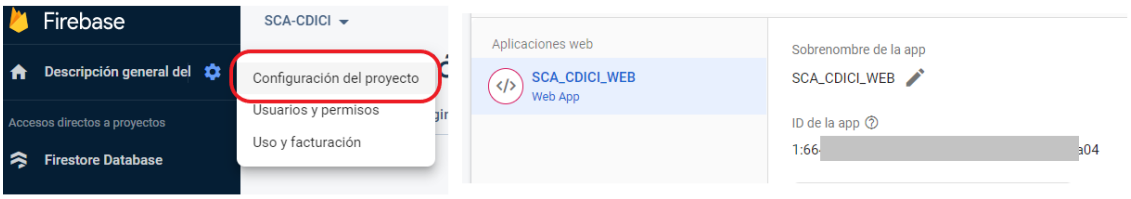


Figura 2.56. Configuración de proyecto creado en Firebase.

Con esto, se debe generar un objeto de configuración de Firebase para vincular la aplicación web. En la Figura 2.57 se puede observar que este objeto contiene identificadores y claves, los cuales permiten que la aplicación web de Angular pueda consumir los servicios de Firebase.

Figura 2.57. Objeto de configuración del proyecto.

Se ha restringido parte de la Figura 2.57 para evitar posibles accesos no deseados en la aplicación web.

Este objeto se incluyó en el archivo *environments/environments.ts,* como se muestra en el Código 2.4.

Se ha restringido parte del Código 2.4 para evitar posibles accesos no deseados en la aplicación web.

```
src > environments > TS environment.prod.ts > ...
1   export const environment = {
2     production: true,
3     // For Firebase JS SDK v7.20.0 and later, measurementId is optional
4     firebaseConfig: {
5       apiKey: 'AIz 1vw',
6       authDomain: ' firebaseapp.com',
7       databaseURL: 'https:// .firebaseio.com',
8       projectId: ' ',
9       storageBucket: ' .appspot.com',
10      messagingSenderId: '66 1',
11      appId: '1:6642 04',
12      measurementId: 'G- W',
13    },
14  };
```

Código 2.4. Configuración de variables de entorno en proyecto web para enlazar con los servicios de Firebase.

2.3.10. CODIFICACIÓN LA INTERFAZ WEB

En esta sección, se mencionan los pasos para la implementación del proyecto en Angular. Se explican los esquemas: módulos, componentes, servicios, *guards*, *pipes* y clases, los cuales fueron creados para brindar funcionalidad a este Módulo.

2.3.10.1. Módulos de Angular

Un módulo en Angular es un contenedor donde se declaran componentes necesarios para brindar mayor funcionalidad a la aplicación. En el Código 2.5 se muestra la definición del módulo de la aplicación en general llamada *AppModule*.

```
@NgModule({
  declarations: [AppComponent, NavbarComponent],
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,
    AngularFireModule.initializeApp(environment.firebaseConfig),
    AngularFireAuthModule,
    FormsModule,
    NgbModule,
  ],
  providers: [AuthService, AngularFirestore],
  bootstrap: [AppComponent],
})
```

Código 2.5. Módulos de Angular incluidos en la aplicación desarrollada.

Para definir este módulo raíz, se agregan los siguientes argumentos:

2.3.10.1.1. Declaraciones

Se declaran componentes, tuberías y directivas para utilizarlos en el módulo actual.

- **AppComponent:** Componente raíz, componente donde se ejecutará todos los elementos de la aplicación web
- **NavBarComponent:** Componente para visualizar una barra de navegación. Este componente estará presente durante toda la ejecución de la aplicación Web.

2.3.10.1.2. Importaciones

Se utilizan para importar módulos de soporte. Permite importar otros módulos para utilizarlos en el módulo actual.

- **BrowserModule**: Exporta la infraestructura necesaria para la correcta ejecución de las aplicaciones desarrolladas en Angular. Este módulo se incluye por defecto en todas las aplicaciones generadas a partir del comando *ng new mweb*.
- **AppRoutingModule**: Es un módulo de enrutamiento de aplicaciones que permite configurar nombres de rutas para cada módulo o componente y así acceder a cualquiera de ellas mediante código.
- **ReactiveFormsModule:** Exporta la infraestructura y las directivas requeridas para formularios reactivos, haciéndolos disponibles para su importación por parte de NgModules que importan este módulo.

- **AngularFireModule:** Es un módulo Ng necesario para usar cualquier servicio de Firebase. Usa como argumento el objeto copiado en el archivo *environments/environments.ts* mostrado en la sección 2.3.9.2.
- **AngularFIreAuthModule:** Es un módulo individual orientado únicamente en la autenticación por Firebase.

2.3.10.1.3. Providers

Se utilizan para inyectar los servicios requeridos por componentes, directivas, tuberías que sean necesarias para el módulo actual.

- **AuthService**: Servicio creado para la autenticación de los usuarios.
- **AngularFirestore**: le permite trabajar con *Cloud Firestore*, la nueva base de datos emblemática para el desarrollo de aplicaciones móviles.

En la aplicación web, se usarán módulos brindados por Angular y módulos creados para personalizar cada interfaz de usuario de acuerdo a su función.

2.3.10.2. Interfaces

Se han creado interfaces para almacenar y organizar los datos que gestiona el Módulo Web.

Estas interfaces almacenarán: información de usuario, información de registro, información de justificación, datos agrupados por mes y preferencias del sistema.

Interfaz de Usuario

La interfaz para gestionar los datos de usuario se llama *User*.

Para la creación de esta interfaz, se ha ejecutado el siguiente código en la terminal:

- ng generate interface models/User

```
export type Roles = 'EMPLEADO' | 'ADMINISTRADOR';

export interface User {
  uid: string;
  email: string;
  role: Roles;
  name: string;
  descr?: string;
  horaIn: string;
  horasDeTrabajo: string;
  idHuella?: number;
}
```

Código 2.6. Definición de la interfaz *User*.

Se tiene un *uid* para almacenar el identificador de usuario, el *email*, rol de usuario, nombre completo, cédula, la hora de entrada, horas de trabajo diario y su identificador de huella

digital. Además, se crea un tipo *Roles* que almacenará EMPLEADO o ADMINISTRADOR para tener un campo role con valor tipo Roles.

Interfaz de Registros

La interfaz para obtener los datos de los registros recolectados se llama *Data*.

Para la creación de esta interfaz, se ha ejecutado el siguiente código en la terminal:

- ng generate interface models/Data

```
export type asistencia = 'PRESENTE' | 'ATRASO' | 'FALTA'
  | 'SALIDA TEMPRANA' | 'SIN SALIDA' | 'DIA EXTRA';

export interface Data {
  id: string;
  usuario: string;
  idUsuario: string;
  asistencia: asistencia[];
  hora: string[];
  horasExtra?: string;
  horasTrabajadas?: string;
  temperatura?: string[];
  justificaciones?: string[];
  horasDeTrabajo: string;
  horario?: string;
}
```

Código 2.7. Definición de la interfaz *Data*.

Dentro de esta interfaz, se definirá un tipo de dato *asistencia* el cual podrá contener los siguientes valores: PRESENTE, ATRASO, FALTA, SALIDA-TEMPR, SIN-SALIDA o DÍA EXTRA.

Además, se almacenan los datos determinados en la definición de la colección *Registros* (2.2.4.1).

Interfaz para Justificaciones

La interfaz para las justificaciones se llamó *Justification.*

Para la creación de esta interfaz, se ha ejecutado el siguiente código en la terminal:

- ng generate interface models/Justification

Se definirá un tipo de dato, de nombre *tipo*, el cual describe lo que se está justificando, es decir: atraso, falta, horas extra, salida temprana, si el usuario no ha generado una salida, si el usuario no cumplió sus horas de trabajo. Se tendrá un tipo de dato *status* que almacenará el estado de la justificación, puede ser: aceptado, rechazado o solicitado. Estos campos facilitarán el cálculo de las horas trabajas y horas extra en el caso de que una justificación sea aceptada.

Además, se almacenan los datos determinados en la definición de la colección *Justificaciones* (2.2.4.1).

```typescript
export type tipo = 'ATRASO' | 'FALTA' | 'HORAS_EXTRA'
  | 'SALIDA_TEMPR' | 'SIN_SALIDA' | 'DIA_EXTRA';

export type status = 'ACEPTADO' | 'RECHAZADO' | 'SOLICITADO';

export interface Justification {
  id: string;
  tipo: tipo;
  mensaje: string;
  motivoR: string;
  status: status;
  idUsuario: string;
  idRegistro: string;
  usuario: string;
  fecha: any;
  horaJustificada?: string;
  urlLink?: string;
  nombreDoc?: string;
}
```

Código 2.8. Definición de la interfaz *Justification*.

Interfaz de Preferencias

La interfaz para gestionar las preferencias del sistema se llama *Preferences*.

Para la creación de esta interfaz, se ha ejecutado el siguiente código en la terminal:

- ng generate interface models /User

```typescript
export interface Preference {
  toleranciaIn: number;
  toleranciaOut: number;
  horaRegistroFaltas: number;
  numeroDeHuellasRegistradas: number;
  puedeCrear: boolean;
  puedeActualizar: boolean;
  puedeEliminar: boolean;
  puedeActualizarUsuarios: boolean;
  idHuellaAccion: number;
  timeServer: string;
  emailAdmin: string;
}
```

Código 2.9. Definición de la interfaz *Preference*.

Se almacenan todos los datos determinados en la definición del documento *preferencias/sistema* (2.2.4.1).

Interfaz para Registros Mensuales

Por último, se tiene una interfaz, llamada *DataMes*, para los datos de cada usuario por mes.

Para la creación de esta interfaz, se ha ejecutado el siguiente código en la terminal:

- ng generate interface models/DataMes

```
import { Data } from './data.interface';

export interface DataMes {
  usuario: string;
  idUsuario: string;
  fecha: string;
  data: Data[];
  horasExtra: string;
  horasTrabajadas: string;
  numAtrasos: number;
  numFaltas: number;
  numSalidasTempranas: number;
}
```

Código 2.10. Definición de la interfaz *DataMes*.

Se tendrá el identificador y nombre del usuario, fecha (el mes y año correspondiente al resumen), un arreglo del tipo Data en el cual se almacenarán todos los registros del mes, el valor total de horas extra y horas trabajadas en el mes. Todos estos datos se calculan en el Controlador en base a los datos obtenidos desde *Cloud Firestore*.

2.3.10.3. Clase

Se han creado clases para complementar la función de los módulos y reutilizar código mediante la herencia de funciones.

Clase Validador de Roles

Clase de nombre *RoleValidator* que permite identificar si un determinado usuario tiene el rol *Administrador* o *Empleado*.

Para la creación de esta clase se ha ejecutado el siguiente código en la terminal:

- ng generate class controllers/class/roleValidator

```
import { User } from 'src/app/shared/models/user.interface';

export class RoleValidator {
  //función que devuelve true si el rol del usuario es EMPLEADO
  isEmpl(user: User): boolean {
    return user[0].role === 'EMPLEADO';
  }

  //función que devuelve true si el rol del usuario es ADMIN
  isAdmin(user: User): boolean {
    return user[0].role === 'ADMINISTRADOR';
  }
}
```

Código 2.11. Definición de la clase *RoleValidator*.

Como se muestra en el Código 2.11, esta clase contiene 2 funciones que comparan el campo *role* del usuario que llega como argumento de entrada. La función *isAdmin* compara el campo *role* usando un operador tripe-igual (operador que compara sin forzar el tipado) con la cadena *ADMINISTRADOR*. La función *isEmpl* compara el campo *role*, usando un operador tripe-igual, con la cadena *EMPLEADO*. En el caso de que se cumpla la condición, retornará un valor "*true*", caso contrario retornará un valor "*false*".

Clase para Gestión de Hora y Fecha

Clase de nombre *TimeConverter* que permite convertir una fecha y hora en formato timestamp a formato *string*, su definición se muestra en el Código 2.12.

Para la creación de esta clase se ha ejecutado el siguiente código en la terminal:

- Ng generate class controllers/class/timeConverter

Esta clase contiene 2 funciones. La primera es *timeConverter*, la encargada en convertir la fecha y hora a formato *string*. Además, se define un argumento *idFecha* para seleccionar si se desea visualizar la hora y fecha, solo la fecha o solo la hora.

```
1   export class TimeConverter {
2     //Convierte fecha y hora de formato Timestamp a String
3     //el idFecha indica si la funcion deve devolver:
4     //0: hora y fecha
5     //1: solo fecha
6     //2: solo hora
7   >   public timeConverter(UNIX_timestamp, idFecha) {···
51    }
52    //función para sumar 2 horas (hora1+hora2)
53  >   addHoras(hora1: String, hora2: String): string {···
77    }
78    //función para restar 2 horas (hora1-hora2)
79  >   subtractHoras(hora1: string, hora2: string): string {···
103   }
104   //funcion para comparar 2 horas. Devuelve:
105   //-1: si hora1 es mayor a hora 2
106   //0: si son iguales
107   //1: si hora1 es menor a hora 2
108 >   compareH(hora1: String, hora2: String): number {···
129   }
130   //Convierte un formato HHmmss a HH:mm:ss
131 >   convertString(time: number): string {···
133   }
134   //Convierte un formato HH:mm:ss a HHmmss
135 >   convertNumber(time: string): number {···
139   }
```

Código 2.12. Definición de la clase *TimeConverter*.

Se definen las funciones llamadas *addHours* y *substractHours* para sumar y restar, respectivamente, 2 horas en formato *string*. La primera función es utilizada para sumar el total de horas trabajadas y horas extra al mes y la segunda función es utilizada para determinar si se ha cumplido el total de horas trabajadas.

Se definen funciones para convertir formatos de horas en enteros o texto y para comparar dos horas.

2.3.10.4. Servicios

Se han creado servicios para la interacción entre los servicios de Firebase y los módulos creados. Estos servicios cumplen una determinada función respecto a la interfaz y al módulo que se ha creado.

Servicio de Autenticación

Se crea el servicio *authService* que contiene las funciones necesarias relacionadas con la autenticación al sistema. Este servicio hereda las funciones de la clase *RoleValidator*. Para la creación de este servicio, se ha ejecutado el siguiente código en la terminal:

- ng generate service controllers/services/auth

En el Código 2.13 se puede observar las variables y el constructor de la función. Se define un objeto observable, la cual es llamado *userDataFirebase* (línea 16), para almacenar la información del usuario autenticado.

```
12  export class AuthService extends RoleValidator {
13    //se almacenan los datos de Cloud Firestore del usuario loggeado
14    public currentUser$: Observable<User>;
15    //se almacenan los datos del usuario loggeado (servicio de autenticación de Firebase)
16    public userDataFirebase$: Observable<any>;
17    //usuario loggeado.
18    public user: any;
19
20    //Constructor:
21    //se declara el servicio AngularFireAuth para acceder a las características Authentication Firebase
22    //el servicio AngularFirestore para acceder a las características de Cloud Firestore.
23    constructor(public afAuth: AngularFireAuth, private afs: AngularFirestore) {
24      //funcion para acceder a las funciones del padre (RoleValidator)
25      super();
26      //variable que almacena el estado del usuario que se autenticó por el servicio de Firebase
27      this.userDataFirebase$ = afAuth.authState;
28      //variable que almacenará el registro de CloudFirestore del usuario autenticado
29      //se compara el email del usuario autenticado con los registros de user en la base de datos
30      this.currentUser$ = afAuth.authState.pipe(
31        switchMap((user) => {
32          if (user) {
33            this.user = user;
34            return this.afs
35              .collection<User>('usuarios', (ref) =>
36                ref.where('email', '==', user.email)
37              )
38              .valueChanges();
39          }
40          return of(null);
41        })
42      );
43    }
```

Código 2.13. Definición de variables y constructor del servicio *AuthService*.

Mediante la función *authState* del módulo *AngularFireAuth*, obtenemos un objeto tipo *Firebase.user* que contiene a información del usuario autenticado por el servicio de *Firebase Authentication*; entre esta información está su *email*. Con este *email*, se realiza una consulta a la base de datos de *Cloud Firestore* para obtener la colección correspondiente del usuario (línea 30 al 38).

Se define un objeto observable tipo *User*, la cual es llamada *currentUser$* (línea 14), para obtener los datos del usuario autenticado desde *Cloud Firestore*; se usa una consulta que comparará el email del usuario obtenido en *userDataFirebase*.

El constructor de este servicio obtendrá los valores de *userData* y *user$*. Ambas variables se obtendrán desde el método *authState* y su tipo de datos será un objeto observable.

En el Código 2.14 se observa las funciones definidas para este servicio. Se tendrá una función para restablecer la contraseña (línea 43), se toma como argumento de entrada el email del usuario y mediante el servicio de Firebase *AngularFireAuth* se realiza el proceso con un correo de recuperación.

Se tendrá una función para login (línea 51), se toma como argumento de entrada el email y la contraseña del usuario; mediante el servicio de Firebase *AngularFireAuth*, se realiza el proceso respectivo. Se define una función para crear un nuevo usuario en el servicio de *Firebase Authentication* (línea 64). Solo se podrá crear un nuevo usuario siempre y cuando se haya iniciado sesión desde un usuario Administrador. Por último, se define una función para cambiar la contraseña de un usuario autenticado (línea 88).

```
45    //función para resetear contraseña de autenticación
46  > async resetPassword(email: string): Promise<void> {…
52    }
53    //función para gestionar el login.
54  > async login(email: string, password: string): Promise<any> {…
65    }
66    //función para crear una cuenta nueva a un usuario en FirebaseAuthentication
67  > async register(email: string, password: string): Promise<any> {…
77    }
78    //función para cerrar sesión
79  > async logout() {…
86    }
87    //función para cambiar la contraseña de un usuario autenticado
88  > async onChangePassword(pass1: any): Promise<any> {…
93    }
```

Código 2.14. Definición de las funciones del servicio *AuthService*.

Servicio para Gestión de Usuario

Se crea el servicio *userService* que contiene las funciones necesarias para la gestión de usuarios registrados en el sistema.

Para la creación de este servicio, se ha ejecutado el siguiente código en la terminal:

- ng generate controllers/service/user

En el Código 2.15 se muestra la definición de este servicio. Se define un objeto observable tipo *User[]*, llamado *user$* (línea 13), el cual contendrá los datos recuperados desde la colección *usuarios* en *Cloud Firestore*. En el caso de que el usuario autenticado no tenga rol de administrador, se leerán los datos de este usuario. Para esto, se usa el objeto observable tipo *User[]* llamado *userById$* (línea 17).

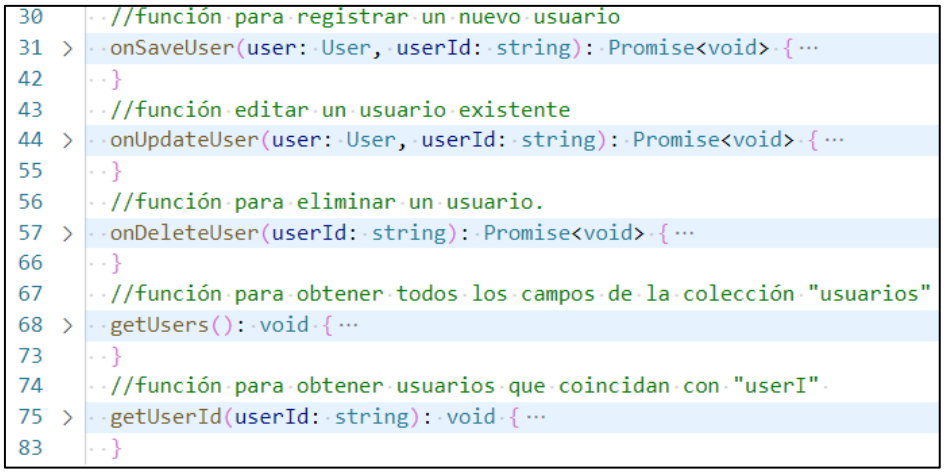
Se definen dos objetos *AngularFirestoreCollection*, con referencia al tipo de datos definido en la interfaz *User* llamados *usersCollection* y *usersCollectionById* (línea 15 y 19, respectivamente), para gestionar el acceso de lectura y escritura a *Cloud Firestore* para los dos objetos observables mencionados anteriormente. En el constructor del servicio se define el servicio de *Cloud Firestore*, llamado *AngularFirestore*.

Las funciones definidas en este servicio se muestran en el Código 2.16. El servicio define una función llamada *onSaveUser* (línea 31) que permite registrar un nuevo usuario. La función *onUpdateUser* (línea 44), permite modificar la información del usuario. Se crea una función para eliminar un determinado usuario en base a su id llamado *onDeleteUser* (línea 57).

Se define una función llamada *getUsers* (línea 68) que permite recuperar los datos de la colección de *users*; los datos recuperados se guardan en el objeto *user$*. La función *getUserId* obtiene los datos de un determinado usuario; se realiza una consulta por id. Los datos recuperados se guardan en el objeto *userById$* (línea 75).

```
11  export class UserService {
12    //observable en donde estarán los usuarios obtenidos por CloudFirestore
13    user$: Observable<User[]>;
14    //se crea un objeto para gestionar el acceso al Cloud Firestore
15    private usersCollection: AngularFirestoreCollection<User>;
16    //observable en donde estarán los datos obtenidos por CloudFirestore en base al id de usuario
17    userById$: Observable<User[]>;
18    //se crea un objeto para gestionar el acceso al Cloud Firestore base al id de usuario
19    private usersCollectionById: AngularFirestoreCollection<User>;
20
21    //Constructor: se declara el servicio AngularFirestore para acceder a Cloud Firestore
22    constructor(private readonly afs: AngularFirestore) {
23      //se define el nombre de la colección que se leerá en el Cloud Firestore
24      //se aplica un filtro de orden por nombre de forma ascendente
25      this.usersCollection = this.afs.collection<User>('users', (ref) =>
26        ref.orderBy('name', 'asc')
27      );
28    }
```

Código 2.15. Definición de variables y constructor del servicio *UserService*.



Código 2.16. Definición de las funciones del servicio *UserService*.

Servicio para Gestión de Registros

Se crea el servicio *dataService* que contiene las funciones necesarias relacionadas con la gestión de los registros de los usuarios.

Para la creación de este servicio, se ha ejecutado el siguiente código en la terminal:

- ng generate service controllers/services/data

Este servicio permitirá obtener y modificar los registros recolectados, además, incluirá funciones para acceder a las justificaciones ligadas con estos registros.

En el Código 2.17 se muestra la definición de este servicio conjuntamente con su constructor y las variables necesarias para ejercer funcionalidad en el mismo.

Se define un objeto observable tipo *Data[]*, llamado *data$* (línea 14), el cual contendrá los datos recuperados desde la colección "*registros*" en *Cloud Firestore*. En el caso de que el usuario autenticado no tenga rol de Administrador, se leerán los registros relacionados con este usuario. Para esto, se usa el objeto observable tipo *Data[]* llamado *dataById$* (línea 17).

Se definen dos objetos *AngularFirestoreCollection*, con referencia al tipo de datos definido en la interfaz *Data* llamados *datasCollection* y *datasCollectionById* (línea 20 y 23, respectivamente), para gestionar el acceso de lectura y escritura a *Cloud Firestore* para los dos objetos observables mencionados anteriormente.

En el constructor del servicio se inyecta el servicio de *Cloud Firestore AngularFirestore*, llamado *afs*. Dentro del constructor, se inicializa por defecto el objeto *datasCollection* llamando a los documentos de la colección *registros* mediante un filtro de orden descendente del campo *hora*.

```
11  export class DataService {
12    //observable en donde estarán los datos obtenidos por CloudFirestore
13    //Se almacenarán todos los registros de usuarios (para rol Administrador)
14    data$: Observable<Data[]>;
15    //observable en donde estarán los datos obtenidos por CloudFirestore
16    //Se almacenarán los registros del usuario autorizado (para rol Empleado)
17    dataById$: Observable<Data[]>;
18
19    //se crea un objeto para gestionar el acceso al Cloud Firestore
20    private datasCollection: AngularFirestoreCollection<Data>;
21    //se crea un objeto para gestionar el acceso al Cloud Firestore
22    //se realizará una consulta de registros en base al id de usuario
23    private datasCollectionById: AngularFirestoreCollection<Data>;
24
25    //Constructor: se declara el servicio AngularFirestore para acceder a
26    //las caracteristicas de Cloud Firestore
27    constructor(private readonly afs: AngularFirestore) {
28      //se define el nombre de la colección que se leerá en el Cloud Firestore
29      //se aplica un filtro de orden por fecha
30      this.datasCollection = this.afs.collection<Data>('registros', (ref) =>
31        ref.orderBy('hora', 'desc')
32      );
33    }
```

Código 2.17. Definición de variables y constructor del servicio *DataService*.

En el Código 2.18 se muestra la definición de las funciones necesarias para la operatividad de este servicio. Se tiene una función llamada *onEditDataJust* (línea 36) la cual se encargará de actualizar el campo *justificaciones* del registro en el caso de que se genere una justificación. Además, se define una función llamada *onUpdateDataJust* (línea 58) para actualizar los campos *horasExtra* y *horasTrabajadas* del registro en el caso de que se acepte o se rechace una justificación.

```
35    //funcion que editará el campo justificaciones del registro cuando se crea una justificación
36  > onEditDataJust(edit: Data, idJust: string): Promise<void> {…
55    }
56    //funcion que editará el campo horasExtra y HorasTrabajadas del registro cuando se acepte
57    // o se rechace una justificación
58  > onUpdateDataJust(edit: any): Promise<void> {…
69    }
70    //función para obtener todos los campos de la colección "registros"
71  > getDatas(): void {…
76    }
77    //Se obtienen los documentos que coincidan con el argumento "userId" de la colección "registros"
78  > getDataId(userId: string): void {…
89    }
90    //Obtiene el registro de una justificacion
91  > getDataJustification(idRegistro: string): void {…
99    }
```

Código 2.18. Definición de las funciones del servicio *DataService*.

Se define una función llamada *getDatas* (línea 71) que permite recuperar los datos de la colección de *registros*; los datos recuperados se guardan en el objeto *data$*. La función *getDataId* (línea 78) obtiene solo los registros de un determinado usuario; se realiza una

consulta por identificador (*userId*). Los datos recuperados se guardan en el objeto *dataById$*. Por último, en la línea 91 se define una función para obtener un registro mediante su identificador. Esta función sirve para recuperar el registro de una justificación.

Servicio para Gestión de Justificaciones

Se crea el servicio *justificationService* el cual contiene las funciones necesarias relacionadas con la creación y gestión de justificaciones de registros. Para la generación de este servicio, se ha ejecutado el siguiente código en la terminal:

- ng generate service controllers/services/justification

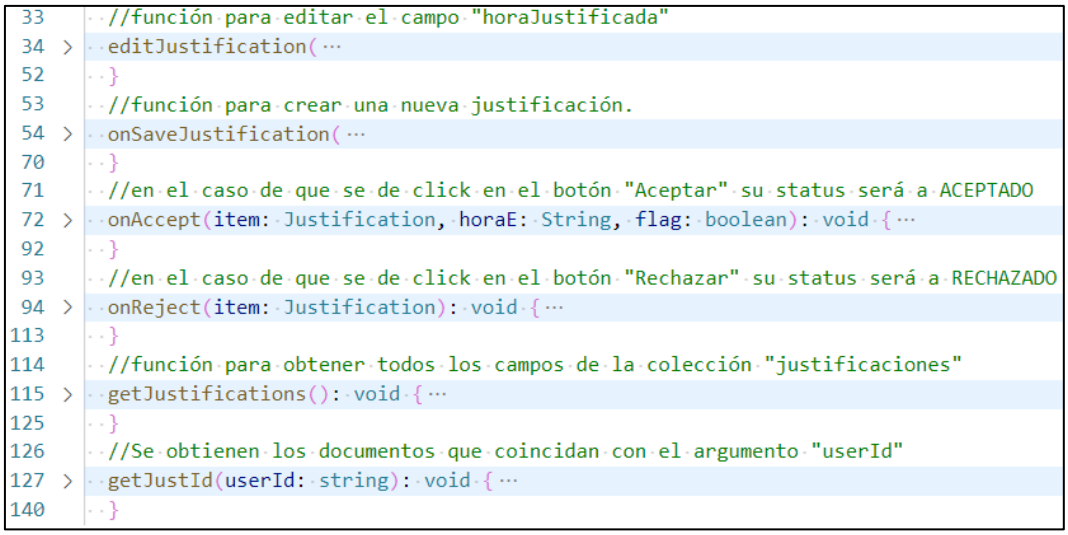
En el Código 2.19 se observa el servicio de justificaciones. Se define un objeto observable tipo *Justification[]*, llamado *justification$* (línea 17), el cual contendrá los datos recuperados desde la colección *justificaciones* en *Cloud Firestore*. En el caso de que el usuario autenticado no tenga rol de Administrador, se leerán los registros relacionados con este usuario, para esto, se usa el objeto observable tipo *Justification[]* llamado *justificationById$* (línea 19).

Se definen dos objetos *AngularFirestoreCollection*, con referencia al tipo de datos definido en la interfaz *Justification* llamados *justificationsCollection* y *JustificationsCollectionById* (línea 22 y 25, respectivamente), para gestionar el acceso de lectura y escritura a *Cloud Firestore* para los dos objetos observables mencionados anteriormente.

En el constructor del servicio se define el servicio de *Cloud Firestore*, llamado *AngularFirestore*, y se incluye el servicio de *Data* para modificar los registros en caso de que se genere una justificación. Se inicializa el objeto llamado *justificationsCollection* por defecto para que pueda acceder a los documentos de la colección *justificaciones*.

```
13  export class JustificationService {
14    //observable en donde estarán las justificaciones obtenidos por CloudFirestore
15    //se almacenarán todas las justificaciones (para el rol Administrador)
16    justification$: Observable<Justification[]>;
17    //observable en donde estarán los datos obtenidos por CloudFirestore
18    //se almacenarán las justificaciones de un determinado usuario (para el rol Empleado)
19    justificationById$: Observable<Justification[]>;
20
21    //se crea un objeto para gestionar el acceso al Cloud Firestore
22    private justificationsCollection: AngularFirestoreCollection<Justification>;
23    //se crea un objeto para gestionar el acceso al Cloud Firestore
24    //se realizará una consulta de justificaciones en base al id de usuario
25    private justificationsCollectionById: AngularFirestoreCollection<Justification>;
26
27    //Constructor: se declara el servicio AngularFirestore para acceder a las características de
28    //Cloud Firestore y el servicio Data para actualizar la informacion de registros
29    constructor(private readonly afs: AngularFirestore, private dataSvc: DataService) {
30      this.justificationsCollection = this.afs.collection<Justification>('justificaciones');
31    }
```

Código 2.19. Definición de variables y constructor del servicio *JustificationService*.



Código 2.20. Definición de las funciones del servicio *JustificationService*.

En el Código 2.20 se observan las funciones declaradas en este servicio. Se define una función llamada *editJustification* (línea 34) que permite editar el campo *horaJustificada* en caso de que la justificación sea aceptada o rechazada. Además, actualizará el campo *horasExtra* u *horasTrabajadas* del registro.

En el caso de que se desee generar una nueva justificación, se tiene la función llamada *onSaveJustification* (línea 54).

Para aceptar o rechazar una justificación, se tienen las funciones *onAccept* (línea 72) y *onReject* (línea 94) respectivamente; estas funciones modificarán el campo *status* de la justificación.

Se define una función llamada *getJustifications* (línea 115) que permite recuperar los datos de la colección de *justificaciones*; los datos recuperados se guardan en el objeto *justification$*. La función *getJustId* (línea 127) obtiene solo las justificaciones de un determinado usuario; se realiza una consulta por identificador (*userId*); los datos recuperados se guardan en el objeto *justificationById$*.

Servicio para Gestión de Preferencias

Se crea el servicio *preferenceService* que contiene las funciones necesarias para la obtención y edición de las preferencias del sistema. Solo los usuarios con rol *Administrador* podrán hacer uso de este servicio. Para su creación se ha ejecutado el siguiente código en la terminal:

- ng generate service controllers/services/preferece

```
13  export class PreferenceService {
14    preference$: Observable<Preference>;
15    //objeto que almacenará una copia de los datos del observable justification$
16    preference: Preference[];
17    //se crea un objeto para gestionar el acceso al Cloud Firestore
18    private preferencesCollection: AngularFirestoreDocument<Preference>;
19    constructor(private readonly afs: AngularFirestore) {
20      this.preferencesCollection = afs.collection<Preference>('preferencias').doc('system');
21      this.getPreferences();
22    }
23
24    //función para registrar un nuevo usuario o editar un usuario existente
25  > onSavePreference(pref: Preference): Promise<void> { ···
35    }
36    //función para obtener las preferencias de la colección definida usersCollection
37  > private getPreferences(): void { ···
41    }
42  }
```

Código 2.21. Definición del servicio *PreferenceService*.

En el Código 2.21 se puede observar el constructor, las variables y las funciones necesarias para la operabilidad del servicio. Se define un objeto observable tipo *Preference[]*, llamado *preference$* (línea 14), el cual contendrá los datos recuperados desde la colección "*preferencias/sistema*" en *Cloud Firestore*. Se definen un objeto *AngularFirestoreCollection*, con referencia al tipo de datos definido en la interfaz *Preference* llamado *preferencesCollection* (línea 18) para gestionar el acceso de lectura y escritura a *Cloud Firestore*.

En el constructor del servicio, se inicializa el objeto *preferencesCollection* (línea 20) y se obtienen los datos de preferencia mediante la función *getPreferences()* (línea 37).

Además, se tiene una función, llamada *onSavePreference* (línea 25), para editar los campos del documento preferencias.

Servicio para Cargar Archivos de Justificaciones

Se crea el servicio *filesService* para gestionar la subida de archivos de justificaciones. Para su creación se ha ejecutado el siguiente código en la terminal:

- ng generate service controllers/services/files

En el Código 2.22 se puede observar el constructor, las variables y las funciones necesarias para la operabilidad del servicio. Se define un objeto observable tipo *String* (línea 11) el cual contendrá el *url* del archivo que se desea subir a *Cloud Storage*.

En el constructor del servicio, se inyecta el servicio de *Cloud Storage* (línea 14).

Por último, en la línea 16 se define la función que permite subir un archivo al *Storage* de Firebase.

```
export class FilesService {
  //Observable que contendrá el url del archivo que se subió a Cloud Storage
  urlFile: Observable<string>;

  //Se inyecta el servicio de Cloud Storage
  constructor(private storage: AngularFireStorage) {}
  //Función que permite cargar un archivo a Cloud Storage, retorna su url
  uploadFile(path: any, file: any): Promise<string> {…
  }
}
```

Código 2.22. Definición del servicio *filesService*

2.3.10.5. Guards

Se han creado *guards* para limitar el acceso a usuarios no autorizados, con esto, se brinda mayor seguridad y protección de datos; brindará un valor booleano que indicará si se debe cargar o no la interfaz a la que se intenta acceder.

En el caso de *AuthGuard*, permite visualizar las páginas solo a usuarios autenticados. *AdminGuard* habilita las páginas orientadas a usuarios con rol *Administrador*. *LogInGuard* bloquea la página de *login* cuando el usuario está autenticado.

Guard de Autenticación

Para restringir las rutas de usuarios que no estén autenticados, se ha creado un *guard* llamado *auth.guard.ts*. Para la creación de este *guard*, se ha ejecutado el siguiente código:

- ng generate guard controllers/guards/auth

En este *Guard* se verifica si el objeto *userData$* existe. Este objeto está definido en el servicio *authService* (2.3.10.4, servicio de autenticación) y contiene los datos del usuario autenticado por Firebase. En el caso de que no exista, se redirige a la interfaz de login (ruta de */login*) y se devuelve un valor *false*. De otro modo, se devuelve un valor *true* y la interfaz cargará sin problema. Esto se muestra en el Código 2.23.

```
export class AuthGuard implements CanActivate {
  constructor(private authSvc: AuthService, private router: Router) {}
  //función que se ejecuta antes de cargar la vista del componente
  canActivate(): Observable<boolean> {
    //compara si en el servicio authService existe un usuario loggeado
    //en el caso de que no existe, se redirige a la ruta de login y retorna false
    //de lo contrario, devuelve true y el componente carga la vista correspondiente
    return this.authSvc.userDataFirebase$.pipe(
      map((user) => {
        if (!user) {
          this.router.navigate(['/login']);
          return false;
        }
        return true;
      })
    );
  }
}
```

Código 2.23. Definición del *Guard AuthGuard.*

Guard de Login

Se ha creado un *guard* llamado *auth.guard.ts* para restringir el acceso a la interfaz de login (ruta de *login*) cuando un usuario se encuentre autenticado.

Para la creación de este *guard*, se ha ejecutado el siguiente código en la terminal:

- ng generate guard controllers/guards/login

```
10  export class LoginGuard implements CanActivate {
11    constructor(private authSvc: AuthService, private router: Router) {}
12    canActivate(): Observable<boolean> {
13      //compara si en el servicio authService existe un usuario loggeado
14      //en el caso de que existe, se redirige a la ruta de data y retorna false
15      //caso contrario, devuelve true y el componente carga la vista correspondiente.
16      return this.authSvc.userDataFirebase$.pipe(
17        map((user) => {
18          if (user) {
19            this.router.navigate(['/data']);
20            return false;
21          }
22          return true;
23        })
24      );
25    }
26  }
```

Código 2.24. Definición del *Guard LoginGuard.*

Se verifica si el objeto *userData$* existe (línea 18). En el caso de que exista un usuario autenticado, se redirige a la interfaz de registros (ruta de *"/data"*, línea 19) y se devuelve un valor *false* para no cargar el módulo. De otro modo, se devuelve un valor *true* y la interfaz cargará sin problema. Esto se muestra en el Código 2.24.

Guard de Usuario Administrador

Para restringir las rutas a usuarios que no estén autorizados, se ha creado un *guard* llamado *admin.guard.ts*.

Para la creación de este *guard*, se ha ejecutado el siguiente código en la terminal:

> ng generate guard controllers/guards/auth

En este *Guard* se verifica si el objeto *user$* tiene como rol *ADMINISTRADOR*. Este objeto está definido en el servicio *authService* (2.3.10.4, Servicio de Autenticación) y contiene los datos del registro *usuarios* en *Cloud Firestore* del usuario autenticado. En el caso de que su rol sea Administrador, se devuelve un valor *true* y la interfaz a la cual se quiere acceder cargará sin problema. De otro modo, se redirige a la interfaz donde se muestra la información de los registros (ruta de */data*) y se devuelve un valor *false* para evitar que la interfaz cargue. Esto se muestra en el Código 2.25.

```
15  export class AdminGuard implements CanActivate {
16    constructor(private authSvc: AuthService, private router: Router) {}
17    //función que se ejecuta antes de cargar la vista del componente
18    canActivate(): Observable<boolean> {
19      //compara si en el servicio authService existe un usuario loggeado
20      //y si el usuario se envia a la funcion isAdmin definida en RoleValidator.ts
21      //si el usuario es Admin devolverá true y el componente carga la vista correspondiente
22      //en caso contrario retornará false, se mostrará un mensaje de acceso denegado.
23      //y se redirige a la ruta de 'data' (única ruta que tiene acceso un usuario EMPLEADO)
24      return this.authSvc.currentUser$.pipe(
25        map((user) => {
26          if (this.authSvc.isAdmin(user)) {
27            return true;
28          } else {
29            this.router.navigate(['data']);
30            return false;
31          }
32        })
33      );
34    }
35  }
```

Código 2.25. Definición del *Guard AdminGuard*.

2.3.10.6. Pipes

Se han creado *pipe's* para la incluir filtros y herramientas de paginación. Esta herramienta analiza datos semejantes entre los campos de un objeto y una entrada de texto en la vista.

Pipe para Filtrado de Nombres

Se ha creado un *pipe* para filtrar datos en base al nombre del usuario. Se ha ejecutado el siguiente código en la terminal para la creación del *pipe*:

- ng generate pipe controllers/pipe/filterUser

En el Código 2.26 se observa la lógica de este *pipe*. Este se relaciona con una entrada de texto en la vista del módulo. Toma como argumentos de entrada el arreglo de registros (*value*) y la entrada de texto (*args*) y en cada llamada la función; se recorre el arreglo comparando un determinado campo del registro y entrada de texto mediante la función *indexOf()*, en el caso de que exista coincidencia, se añade el registro correspondiente a un arreglo de respuesta (línea 23). Se retorna el arreglo de respuesta si es diferente a vacío, caso contrario, se retorna *null*. Se retorna todo el arreglo de registros si el argumento *args* contiene menos de 2 caracteres (línea 17). En este pipe se analiza la variable *usuario* (para documentos de *Registros* y *Justificaciones*, línea 19) y *name* (para documentos de *Usuarios*); se coloca el signo "?" para comparar únicamente si existe el campo.

```
6  export class FilterUserPipe implements PipeTransform {
7    //objeto donde se guardará toda la data (value)
8    public data: any = [];
9    transform(value: any, args: any): any {
10     //en el caso de que se escriba menos de 2 caracteres se retornará value (todos los resultados)
11     if (args.length < 2) return value;
12     //caso contrario se asigna el valor de value a data para acceder a su propiedad foreach()
13     this.data = value;
14     //se define un array que contendrá todos los resultados que coincidan
15     const resultData = [];
16     this.data.forEach((item: any) => {
17       //mediante el foreach se obtiene cada item y se analiza el nombre de usuario
18       if (
19         item.usuario?.toLowerCase().indexOf(args.toLowerCase()) > -1 ||
20         item.name?.toLowerCase().indexOf(args.toLowerCase()) > -1
21       ) {
22         //si encuentra coinciencia, se agrega al array definido
23         resultData.push(item);
24       }
25     });
26     //si el array de resultados contiene items, se lo retorna
27     if (resultData.length > 0) return resultData;
28     //caso contrario se retorna null.
29     return null;
30   }
31 }
```

Código 2.26. Definición de *Pipe FilterUser.*

Pipe para Filtrado de Fechas

Se ha creado un *pipe* llamado *FilterDate* para filtrar la fecha de los datos con la entrada de texto respectiva. En el Código 2.27 se muestra su lógica, su funcionamiento es similar al pipe descrito anteriormente. Analiza el campo fecha y el campo hora; para este último es necesario asegurarse de que exista el campo (línea 27), y después analizar el contenido del mismo con la función *indexOf()* (línea 29).

Para la creación del *pipe*, se ha ejecutado el siguiente código en la terminal:

- ng generate pipe controllers/pipe/filterDate

```
19     this.data.forEach((item: any) => {
20       //mediante el foreach se analiza cada item
21       //se analiza el usuario, la fecha y la asistencia(si existe)
22       if (item.fecha?.indexOf(args) > -1) {
23         //si encuentra coinciencia, se agrega al array definido
24         resultData.push(item);
25       }
26       //verifica si existe la variable hora (array).
27       if (item.hora) {
28         //Si existe la variable, analiza el campo fecha de hora
29         if (
30           this.timeConverter(item.hora[0]['seconds'], 1)?.indexOf(args) > -1
31         ) {
32           //si encuentra coinciencia, se agrega al array definido
33           resultData.push(item);
34         }
35       }
36     });
```

Código 2.27. Definición de *Pipe FilterDate*.

Pipe para Filtrado de Tipo, Estatus y Asistencia

Se ha creado un *pipe* para filtrar los campos de tipo, estatus y asistencia de justificaciones y registros. Para la creación del *pipe*, se ha ejecutado el siguiente código en la terminal:

- ng generate pipe controllers/pipe/filterDataMes

Su función similar al *pipe FilterUser*; la diferencia está en que esta analiza los campos tipo, asistencia y estatus. Esto se muestra en el Código 2.28.

```
17      this.data.forEach((item: any) => {
18        //mediante el foreach se obtiene la informacion, se analiza el tipo, status y asistencia
19        if (
20          item.tipo?.toLowerCase().indexOf(args.toLowerCase()) > -1 ||
21          item.status?.toLowerCase().indexOf(args.toLowerCase()) > -1 ||
22          item.asistencia[0]?.toLowerCase().indexOf(args.toLowerCase()) > -1 ||
23          item.asistencia[1]?.toLowerCase().indexOf(args.toLowerCase()) > -1
24        ) {
25          //si encuentra coinciencia, se agrega al array definido
26          dataResult.push(item);
27        }
28      });
```

Código 2.28. Definición de Pipe *FilterStatus*.

Pipe para Paginación

Se ha creado un *pipe* de para la paginación y mostrar elementos de 10 en 10, con nombre *paginationPipe*. Se ha ejecutado el siguiente código en la terminal:

- ng generate pipe controllers/pipe/pagintion

Retorna una copia de parte del *array* de *Data* mediante la función *slice()* (línea 11). La variable *number* permite limitar el inicio y el fin del *array*. Esto se muestra en el Código 2.28.

```
7   export class PaginationPipe implements PipeTransform {
8     transform(data: Data[], page: number = 0): Data[] {
9       if (data) {
10        const filter = data.filter((item) => item.usuario);
11        return filter.slice(page, page + 10);
12      }
13    }
14  }
```

Código 2.29. Definición del Pipe *Pagination*.

2.3.10.7. Módulos

Se han creado módulos para la gestión de las vistas. Al usar módulos, se gestiona sus propios componentes evitando que se sobrecargue el módulo raíz o principal (*appModule)* y se importan únicamente los módulos requeridos para el funcionamiento del mismo. En la Tabla 2.25 se muestran los módulos utilizados para la autenticación y reinicio de

contraseña del sistema. Para la creación de los módulos, se ha ejecutado el siguiente código en la terminal:

- ng generate module views/NOMBRE_MÓDULO -m=app —route NOMBRE_MÓDULO

Tabla 2.25. Módulos para propósitos de autenticación.

Módulo	Descripción	Requerimiento
login	Es creado para brindar acceso al sistema a usuarios registrados. También, brinda la posibilidad de redireccionar al módulo de recuperación de contraseña.	RF06
r-password	Brinda la posibilidad al usuario de recuperar su contraseña mediante un correo de recuperación.	RF06

En la Tabla 2.26 se muestran los módulos de propósito general para todos los usuarios registrados en el sistema.

Por último, en la Tabla 2.27 se muestran los módulos con acceso limitado a usuarios con rol *Administrador*; para su activación se usará el guard *AdminGuard.*

Tabla 2.26. Módulos de uso general para ambos roles de usuarios.

Módulo	Descripción	Requerimiento
navbar	Permite navegar por los distintos componentes de la aplicación.	RFN01
data	Componente para visualizar la información de los usuarios. La información se muestra por: Resumen de asistencia, registros diarios y registros mensuales.	RF08, RF09 RF10. RF12
new-j	Permite agregar una nueva justificación.	RF11
details-j	Permite mostrar la información del registro. Permite crear una justificación si el registro lo requiere. Además, muestra las justificaciones generadas.	RF11

Tabla 2.27. Módulos para usuarios con rol Administrador.

Módulo	Descripción	Requerimiento
g-users	Despliega el listado de todos los usuarios registrados en el sistema.	RF07
edit-user	Permite editar la información del usuario.	RF07
details-user	Permite mostrar la información del usuario	RF07
justification	Brindar las funcionalidades para visualizar todas las justificaciones registradas en el sistema	RF13
preferences	Muestra las preferencias del sistema. Permite modificar campos como: tolerancia de entrada, tolerancia de salida y hora de registro de falta.	RF14

2.3.10.8. Ventanas de avisos

Se ha creado ventanas modales con distintos mensajes para la visualización de avisos al usuario. Las ventanas modales son codificadas en HTML en el archivo *component.ts* del módulo.

Para usar las ventanas modales en Angular, se debe importar el módulo *NgbModal*. Este módulo es provisto por *Bootstrap* para Angular y que permite gestionar el manejo de una ventana modal mediante una etiqueta.

Una vez importado el módulo *NgbModal*, se puede abrir un modal, como se muestra en el Código 2.30 línea 70, teniendo en cuenta que los argumentos de entrada de la función *open()* son: etiqueta del modal (con nombre: *contenido*) y el tamaño de la ventana (*sm*, *lg*, *xl*). Además, se puede cerrar un modal con *dismissAll()*, como se muestra en el Código 2.30 línea 75.

```
69    onClickModal(contenido: any) {
70      this.modal.open(contenido, { size: 'lg' });
71    }
72
73    onClickModalAccept(confirm: any) {
74      this.onAccept(this.just, this.tiempo, this.flag);
75      this.modal.dismissAll();
76      this.modal.open(confirm, { size: 'lg' });
77    }
```

Código 2.30. Código para gestionar una ventana modal.

La codificación de la ventana modal, dentro del módulo *justification*, usado para notificar al usuario que los datos se han guardado correctamente se muestra en el Código 2.31.

```
122  <ng-template #confirm let-modal>
123    <div class="modal-header">
124      <h4 class="modal-tittle">Hecho!</h4>
125      <button
126        class="close"
127        aria-label="close"
128        type="button"
129        (click)="modal.dismiss()"
130      >
131        <span aria-hidden="true">&times;</span>
132      </button>
133    </div>
134    <div class="modal-body">
135      <p>Datos guardados correctamente</p>
136    </div>
137    <div class="modal-footer">
138      <button type="button" class="btn btn-primary" (click)="modal.dismiss()">
139        ACEPTAR
140      </button>
141    </div>
142  </ng-template>
```

Código 2.31. Codificación de la ventana modal para confirmación.

En la etiqueta *ng-template*, se define un identificador de nombre *#confirm* (línea 122) para saber que modal mostrar en código del componente del módulo. Se codifica la estructura del modal, en este caso, tendrá un título, un botón de cerrar en su esquina superior derecha, un mensaje de “Datos guardados correctamente” y un botón de “Aceptar” para cerrar el modal.

El resultado de la ventana modal mencionada anteriormente se muestra en la Figura 2.58.

Figura 2.58. Visualización de ventana modal de confirmación.

2.3.10.9. Gestión de rutas

El principal archivo gestor de rutas en el proyecto es *app-routing.module.ts*. Este es el archivo que se genera al crear el proyecto y es el punto de partida de la aplicación. Gracias a los comandos de creación de los respectivos módulos, se pudo crear instancias de rutas automáticamente en este archivo.

Las rutas definidas en la aplicación se agregan los *Guards* (función *canActivate* de los *guards* definidos en 2.3.10.6) necesarios para el acceso a ellas. Esto se muestra el Código 2.32.

```
7   const routes: Routes = [
8     { path: '', redirectTo: '/login', pathMatch: 'full' },
9     {
10      path: 'login', canActivate: [LoginGuard],
11      loadChildren: () =>
12        import('./auth/login/login.module').then((m) => m.LoginModule),
13    },
14    {
15      path: 'r-password',
16      loadChildren: () =>
17        import('./auth/r-password/r-password.module').then(
18          (m) => m.RPasswordModule
19        ),
20    },
21    {
22      path: 'g-usuarios', canActivate: [AuthGuard, AdminGuard],
23      loadChildren: () =>
24        import('./g-usuarios/g-usuarios.module').then((m) => m.GUsuariosModule),
25    },
```

Código 2.32. Gestión de rutas para los componentes creados.

2.3.11. CONFIGURACIÓN EL SERVICIO DE HOSTING

Para acceder al servicio de *Hosting* en Firebase, es necesario iniciar sesión con una cuenta de Google, para esto, se ejecuta el comando "*firebase login"* en la consola del Visual Studio Code. Una vez iniciada la sesión, con el comando "*firebase init"* se inician los servicios de Firebase; se verifica la ruta del proyecto. Se selecciona la opción *Hosting: Configure files for Firebase Hosting*. Esto se muestra en la Figura 2.59.

Luego, se selecciona el proyecto en Firebase al cual se vinculará la aplicación web.

Por último, se ejecuta el comando "*firebase deploy"* para que el proyecto de Angular se publique en la web mediante el hosting de Firebase.

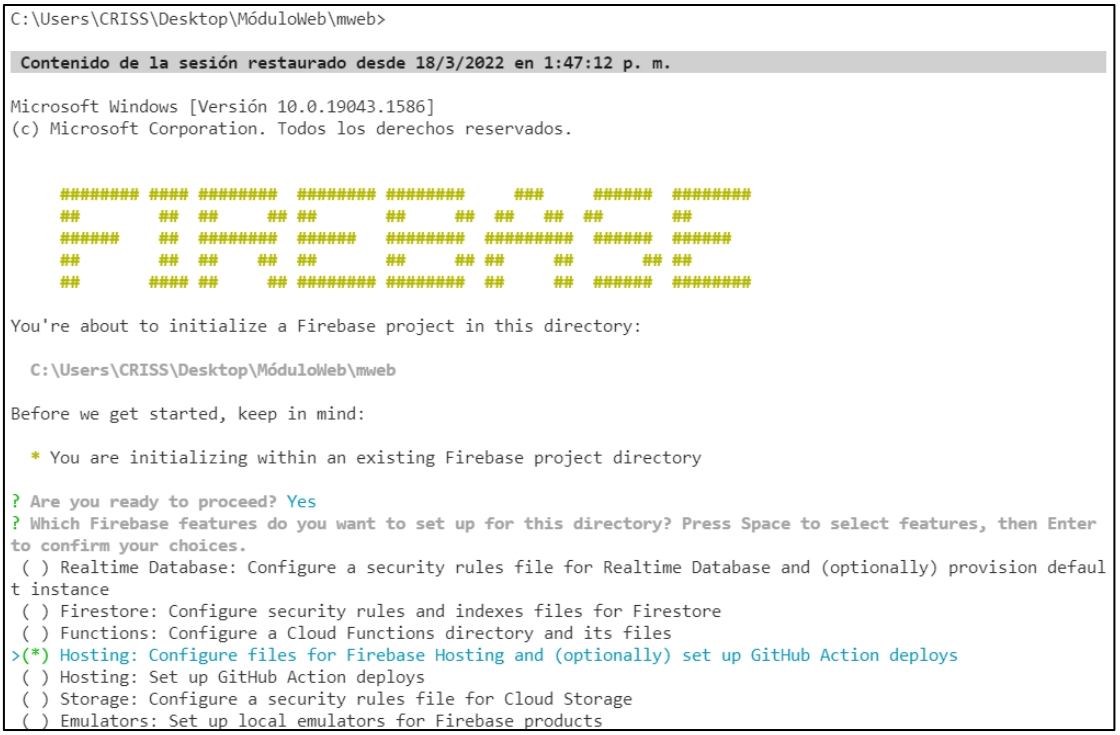


Figura 2.59. Asistente de configuración para servicios de Firebase en Angular.

2.3.12. CODIFICACIÓN LA INTERFAZ GRÁFICA MÓDULO MÓVIL

En esta sección se detallan los pasos para la creación de la interfaz gráfica del Módulo Móvil. Se muestran los códigos necesarios a ejecutar en el terminal de la aplicación y los cambios necesarios para acoplar la interfaz web a una interfaz móvil. El código del Módulo Móvil se encuentra en el ANEXO E.

2.3.12.1. Creación del proyecto a partir de la aplicación Web

Los comandos que se deben ejecutar para la creación del Módulo Móvil y para agregar características al proyecto se muestran en la Tabla 2.28. Se debe abrir el editor de código Visual Studio Code y crear un nuevo terminal.

El tercer comando, ejecutará un asistente de configuración del proyecto al que se le aplicará el framework Ionic [44]; se ejecutará en consola. Se selecciona el *framework* con el que se desarrolló el proyecto web, en este caso Angular.io, y se coloca el nombre de la aplicación de Ionic, esto se muestra en la Figura 2.60.

Figura 2.60. Asistente de configuración de Ionic.

Tabla 2.28. Lista de comandos para generar y configurar el Módulo Móvil a partir del Módulo Web.

Orden	Comando	Descripción	Versión
1	npm install -g @ionic/cli	Instalación de Ionic	5.0.0
2	ng add @ionic/angular	Agrega el Framework de Ionic en el proyecto existente	No Aplica
3	ionic init	Inicializa el framework de Ionic	No Aplica
4	ng config cli.defaultCollection @ionic/angular-toolkit	Cambio de colección de esquemas (de Angular a Ionic)	No Aplica

2.3.12.2. Modificaciones necesarias

En esta sección se mencionan los cambios necesarios que se deben hacer en la aplicación móvil (Módulo Móvil) para su correcto funcionamiento. Debido a que las pantallas de los teléfonos móviles son limitadas en su tamaño, es necesario aplicar reducir la información desplegada en las vistas para acoplar el contenido en la pantalla. Para esto, se usarán elementos de Ionic y las herramientas que este *framework* brinda.

2.3.12.2.1. Etiqueta de aplicación de Ionic

La etiqueta <*ion-app*> permite cargar el componente principal de Ionic en la aplicación. Esta etiqueta inserta el contenido del componente principal, el cual contiene el resto de las páginas (módulos o componentes) de la aplicación.

```
1  <ion-app>
2    <app-navbar></app-navbar>
3    <ion-content>
4      <router-outlet></router-outlet></div>
5    </ion-content>
6  </ion-app>
```

Código 2.33. Etiqueta Ion en el componente principal *app-component*.

Además, el uso de esta etiqueta permite agregar elementos de Ionic en las vistas. Por ejemplo: iconos, contenedores, botones, *cards*, entre otros. Para la implementación de esta etiqueta, se añadió en la vista del módulo raíz o principal de la aplicación (*app.component.html*) como se muestra en el Código 2.33.

2.3.12.2.2. Páginas

Fue necesario realizar modificaciones a la aplicación web para adaptar la vista del resumen mensual a una interfaz móvil, para esto, se creó una página. La página en Ionic permite visualizar información mediante una interfaz que se acople al tamaño de pantalla de un dispositivo móvil.

Se ha creado una página con nombre *DataMesPage* para visualizar toda la información del resumen mensual. Este elemento se generó con el siguiente comando:

- ionic generate page views/pages/DataMesPage

Para visualizar la página *DataMesPage* se usará el módulo *ModalController* el cual permite gestionar la apertura de la página como si fuera una ventana modal. El paso de argumentos hacia la página creada se gestiona en el Código 2.34. En la línea 132 se envía como argumento el registro mensual (*item)* desde el módulo "*data"* a una variable de nombre *dataMes* que se encuentra definida en la página creada (Código 2.35, línea 12). Una vez obtenido el valor de *dataMes* en la página, se muestra su contenido en la vista.

Por último, se define la función para cerrar el modal con la función *dismiss()* (Código 2.35, línea 16).

```
128   async generatePageDataMes(item: DataMes) {
129     const modal = await this.modalControler.create({
130       component: DataMesPage,
131       componentProps: {
132         dataMes: item,
133       },
134     });
135     return await modal.present();
136   }
```

Código 2.34. Creación de ventana modal con paso de argumentos en Ionic.

```
10  export class DataMesPage {
11    //variable en donde se almacenará el registro mensual enviado como argumento
12    @Input() dataMes: DataMes;
13
14    constructor(private modalController: ModalController) {}
15
16    closeModal() {
17      this.modalController.dismiss();
18    }
19  }
```

Código 2.35. Definición de la página *DataMesPage*.

2.3.13. CREACIÓN DE LA APLICACIÓN MÓVIL INSTALABLE

Para la creación de una aplicación en Android (*apk*), es necesario instalar Android Studio desde su página oficial [45]. En la Tabla 2.29 se muestran los comandos que se deben ejecutar en un terminal de Visual Studio Code dentro de la carpeta del proyecto para la inclusión de la herramienta Capacitor [46] en el proyecto.

Tabla 2.29. Lista de comandos para generar una aplicación instalable.

Orden	Comando	Descripción	Versión
1	npm install @capacitor/core	Instalación e integración de Capacitor en el proyecto de angular.	3.4.1
2	npm install @capacitor/cli --save-dev		3.4.1
3	npm install @capacitor/android	Instalación de la plataforma Android en el proyecto.	3.4.1
4	npx cap init	Inicialización del CLI de Capacitor.	No Aplica
5	npx cap add android	Integración de Android en el proyecto	No Aplica
6	ionic build	Compilación del proyecto basado en Ionic y Angular.	No Aplica
7	npx cap open android	Apertura de Android Studio con el proyecto compilado.	No Aplica

Una vez ejecutados estos comandos, se abrirá Android Studio. Se podrá observar el proyecto compilado para Android con la ayuda de Capacitor, tal y como se muestra en la Figura 2.61.

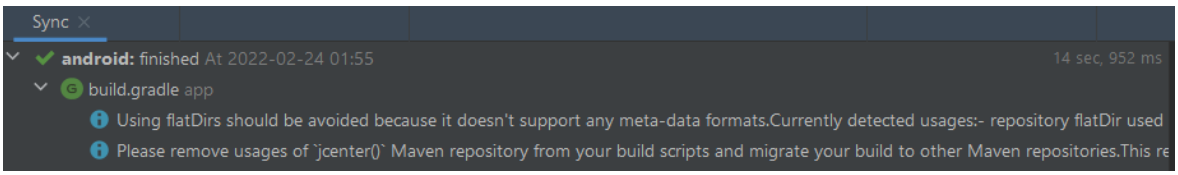


Figura 2.61. Compilación de proyecto en Android Studio.

Los archivos de la aplicación compilada se muestran en la Figura 2.62. Al afirmar que la aplicación se ha compilado con éxito, se puede ejecutar el proyecto en un simulador de Android para la búsqueda de errores funcionales y de visualización.

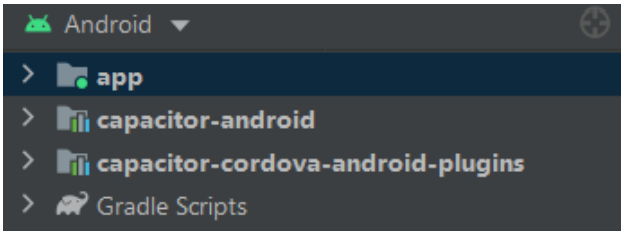


Figura 2.62. Archivos de Android Studio.

Mediante Android Studio, se desarrolló un archivo instalable para la plataforma de Android. El proceso para obtener un archivo *APK* se muestra en la Figura 2.63. En la consola de Android Studio se notificará la generación de este paquete instalable y se podrá acceder a la carpeta donde se encuentra este archivo.

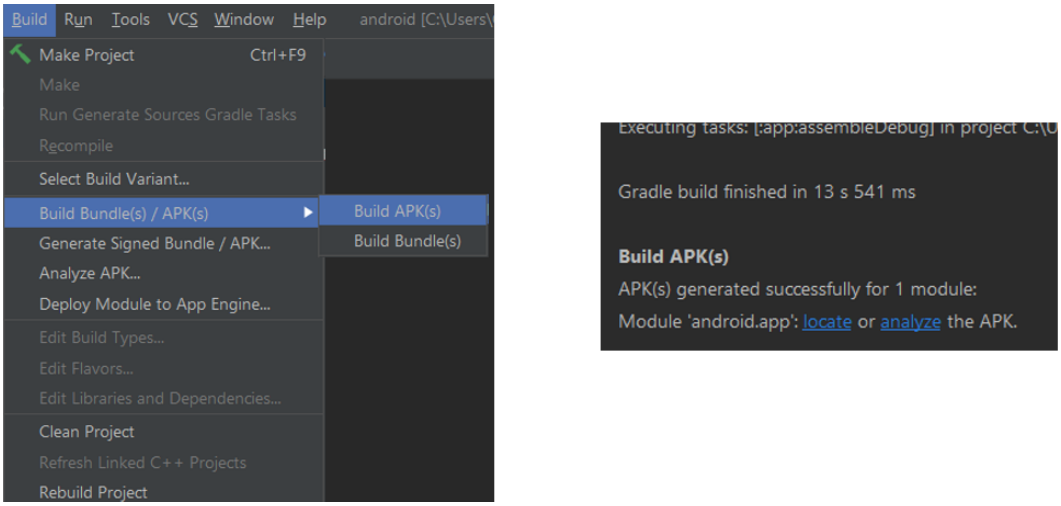


Figura 2.63. Proceso para generar archivo *APK*.

Una vez generado el paquete instalable, se procede a instalar en los dispositivos que requieran usar este Módulo. El instalador de la aplicación Móvil se encuentra en el ANEXO F.

3. RESULTADOS Y DISCUSIÓN

3.1. ACTUALIZACIÓN DEL TABLERO KANBAN

En la Figura 3.1 se observa la actualización del tablero Kanban. Se encuentra la columna “En proceso” que detalla las actividades que se encuentran en desarrollo respecto a la fase correspondiente, en este caso, la fase de implementación. Por último, se encuentra la columna “Completado” que hace referencia a las actividades finalizadas, en este caso, las actividades de la fase de diseño.

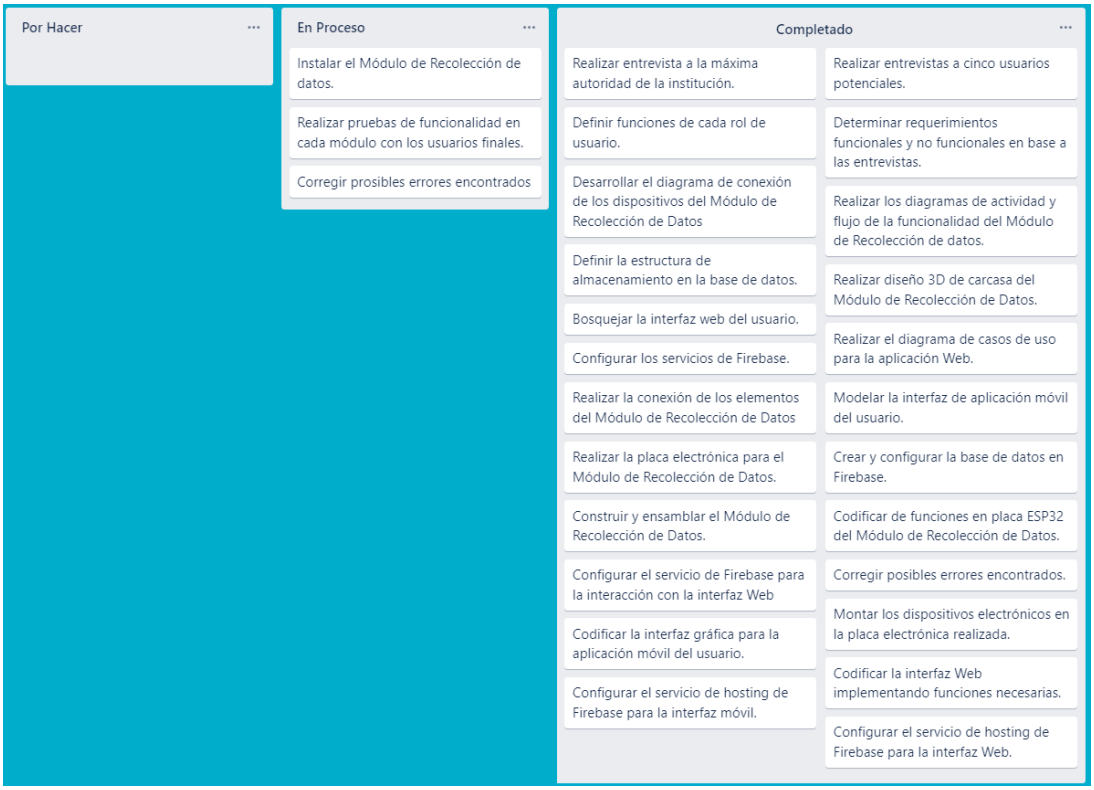


Figura 3.1. Actualización del tablero Kanban para resultados.

3.2. PRUEBAS DE FUNCIONAMIENTO

En esta sección se detallarán los resultados de las pruebas de funcionamiento de todos los módulos planteados: Módulo de Recolección de datos, Módulo Web y Módulo Móvil. Para el primer Módulo, se realizarán pruebas de identificación del personal en base a su huella digital, creación y actualización de plantillas de huellas digitales, generación de múltiples registros. Para los Módulos Web y Móvil, se realizarán pruebas de funcionamiento en visualización de registros, gestión de usuarios y gestión de justificaciones.

Para realizar algunas pruebas, se desarrollaron los procesos detallados en el manual de usuario adjunto en el ANEXO G.

3.2.1. FUNCIONAMIENTO DE MÓDULOS

En primer lugar, se realizaron pruebas básicas a cada uno de los módulos con el fin de verificar su correcto funcionamiento. En esta sección se detallarán estas pruebas.

3.2.1.1. MÓDULO WEB

Para comprobar el funcionamiento de este módulo, se creó un usuario de autenticación desde la consola de Firebase; esto se muestra en la Figura 3.2.

Figura 3.2. Usuario para prueba de autenticación

Se creó un documento de usuario. Su estructura se muestra en la Figura 3.3.

Figura 3.3. Documento en Cloud Firestore del usuario creado para verificación

Al momento de realizar esta prueba, no existían registros. Se modificaron los documentos de Cloud Firestore para visualizar información en el módulo. Esta información se detalla en la Figura 3.4.

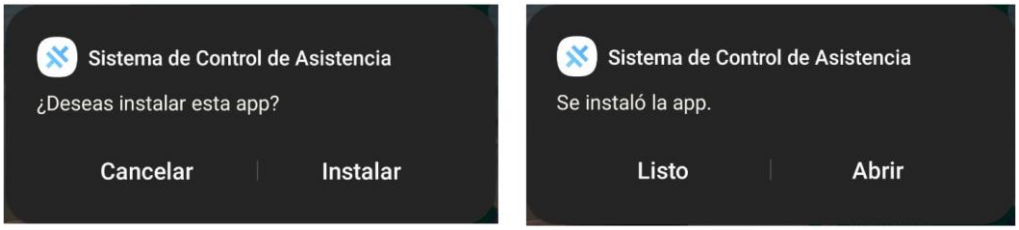
Figura 3.4. Registro de prueba en Cloud Firestore.

Después, se accedió al URL del módulo Web y se inició sesión en el sistema. En la Figura 3.5 se muestra el funcionamiento del módulo. Se pudo visualizar la información creada en cada una de las pestañas.

Figura 3.5. Visualización del registro creado en el Módulo Web

3.2.1.2. MÓDULO MÓVIL

Para comprobar el funcionamiento de este módulo, se instaló la aplicación adjunta en el ANEXO F. En la Figura 3.6 se muestra el proceso de instalación.



(a) (b)

Figura 3.6. Instalación del Módulo Móvil

Una vez instalada la aplicación, se inició sesión en el sistema. En la Figura 3.7 se muestra el funcionamiento del módulo. Se pudo visualizar la información creada en cada una de las pestañas.

Figura 3.7. Visualización del registro creado en el Módulo Móvil

3.2.1.3. MÓDULO DE RECOLECCIÓN DE DATOS

Se cargaron *sketch's* para realizar las pruebas individuales del módulo. Estos se encuentran adjuntos en el ANEXO H. A continuación, se detallan las pruebas realizadas.

3.2.1.3.1. Conexión a Internet y Firebase

Se cargó el *sketch* "*prueba Firebase.ino*" para verificar la conexión a Internet y subida de datos a *Cloud Firestore*. En la Figura 3.8 se muestra el resultado de esta prueba.

```
-> Creando documento... ok
-> {
->   "name": "projects/sca-cdici/databases/(default)/documents/prueba/1",
->   "fields": {
->     "Resultado": {
->       "stringValue": "Operación Correcta"
->     }
->   },
->   "createTime": "2022-05-30T19:57:11.647254Z",
->   "updateTime": "2022-05-30T19:57:11.647254Z"
-> }
->
->
-> Obteniendo documento... ok
-> {
->   "name": "projects/sca-cdici/databases/(default)/documents/prueba/1",
->   "fields": {
->     "Resultado": {
->       "stringValue": "Operación Correcta"
->     }
->   },
->   "createTime": "2022-05-30T19:57:11.647254Z",
->   "updateTime": "2022-05-30T19:57:11.647254Z"
-> }
```

Figura 3.8. Prueba de conexión a Internet en el Módulo de Recolección de Datos

3.2.1.3.2. Pantalla

Se cargó el *sketch* "*prueba Pantalla.ino*" para verificar el correcto funcionamiento de las pantallas y los pulsadores. En la Figura 3.9 se muestra el resultado de esta prueba al observar que el Módulo imprime en pantalla un mensaje.

Figura 3.9. Prueba de funcionamiento de pantalla LCD.

3.2.1.3.3. Lector de huella digital

Se cargó el *sketch* "*prueba Sensor Huella.ino*" para verificar el correcto funcionamiento del lector de huellas digitales. En la Figura 3.10 se puede ver el resultado de esta prueba. Al

pulsar el botón *Opciones (PULL1)*, el Módulo guarda una plantilla de huella con identificador 1. Con esto, se comprueba el funcionamiento del lector al ingresar la huella almacenada.

```
-> Iniciando Pantalla LCD
-> Iniciando Sensor de Huella Digital: OK
->
-> ID Huella: 1
```

Figura 3.10. Prueba de funcionamiento del lector de huellas digitales

3.2.1.3.4. Sensor de temperatura

Para verifica el correcto funcionamiento del sensor de huellas digitales, se cargó el *sketch* "*prueba Sensor Temperatura.ino*". En la Figura 3.11 se muestra el resultado de esta prueba al observar que se imprime en pantalla el valor de temperatura actual en el sensor y si se detecta un objeto cerca mediante el sensor de proximidad.

```
-> Iniciando Pantalla LCD
-> Iniciando Sensor de temperatura: OK
-> ================================================
-> Midiendo Temperatura...
-> Ambiente: 18.25
-> Objeto: 33.84
->
-> Midiendo Temperatura...
-> Ambiente: 18.47
-> Objeto: 26.59
```

Figura 3.11. Prueba de sensor de temperatura.

En la Tabla 3.1 se tabulan los valores del sensor de temperatura frente a los valores medidos por un termómetro digital, donde se presenta el porcentaje de error y la variación entre estas dos lecturas.

Tabla 3.1. Resultado de pruebas con el sensor de temperatura.

Prueba	Medida (°C)		Error (%)
	Módulo	Termómetro	
1	36.35	36.58	0.6%
2	35.87	35.31	1.6%
3	36.72	35.63	3.1%
4	34.72	34.67	0.1%
5	35.52	36.2	1.9%

3.2.1.3.5. Inicialización del Módulo

Se cargó el programa que brida la funcionalidad al Módulo (ANEXO B). Al encender el Módulo, se pudo comprobar su funcionamiento en base al documento *preferencias/sistema* en *Cloud Firestore*. Este documento actualizará el campo *timeServer* a la hora en que el

módulo se conectó con los servicios de Firebase, esto se muestra en la Figura 3.12. Además, se modificó el campo *puedeCrear* en el valor *true* para que el Módulo pueda crear una plantilla de huella digital del usuario creado en la sección 3.2.1.1.

Figura 3.12. Inicialización del Módulo de Recolección de Datos.

3.2.1.3.6. Funcionamiento General

En esta sección se detalla el funcionamiento básico del módulo después de que este se ha encendido. Se muestran los resultados de visualización de pantallas, lectura de información por medio de la memoria SD, funcionamiento sin fuente de alimentación para el caso de corte de energía y gestión de reinicio cuando se haya perdido conexión a la red definida.

Encendido del Módulo

El Módulo debe mostrar la información de día, fecha y hora en pantalla una vez los módulos hayan iniciado sin ningún error. En la Figura 3.13 se muestra esta información en la pantalla del Módulo.

Figura 3.13. Funcionamiento del Módulo

Así mismo, si se pulsa el botón Hora/Wifi (PULL2), se obtendrá la siguiente vista en donde se informará el estado de conexión de red, la dirección IP con la que se está conectado y si existe un tiempo para el reinicio programado.

(a) (b)

Figura 3.14. Pantalla de conexión a Wifi

Lectura de información

Se comprobó si el Módulo está leyendo correctamente la información de los documentos de usuarios y preferencias desde Cloud Firestore. En la Figura 3.15 se muestra el contenido del archivo *usuarios.txt* de la memoria SD una vez el Módulo haya inicializado correctamente. El archivo contiene la lista de usuarios en formato *JSON*.


```
{
  "documents": [
    {
      "name": "projects/sca-cdici/databases/(default)/documents/usuarios/MJ5pOvht9hlSw1mUaouV",
      "fields": {
        "uid": {
          "stringValue": "MJ5pOvht9hlSw1mUaouV"
        },
        "idHuella": {
          "integerValue": "1"
        },
        "horaIn": {
          "stringValue": "00:00"
        },
        "horasDeTrabajo": {
          "stringValue": "00:00"
        },
        "name": {
          "stringValue": "Cristhian Morocho"
        }
      },
      "createTime": "2022-03-29T06:40:41.485539Z",
      "updateTime": "2022-04-05T01:15:05.666412Z"
    }
  ]
}
```


Figura 3.15. Contenido del archivo *usuarios.txt* en memoria SD.

El contenido del archivo *preferencias.txt* de la memoria SD, en el cual se despliega el contenido del documento preferencias en formato *JSON,* se muestra en la Figura 3.16.

```
{
  "name": "projects/sca-cdici/databases/(default)/documents/preferencias/sistema",
  "fields": {
    "timeServer": {
      "timestampValue": "2022-04-1T22:26:51.975Z"
    },
    "numeroDeHuellasRegistradas": {
      "integerValue": "1"
    },
    "toleranciaIn": {
      "integerValue": "5"
    },
    "horaRegistroFaltas": {
      "stringValue": "230000"
    }
  },
  "createTime": "2022-01-23T14:43:08.077649Z",
  "updateTime": "2022-04-1T22:26:52.014451Z"
}
```


Figura 3.16. Contenido del archivo *preferencias.txt* en memoria SD.

Funcionamiento Sin Alimentación

Para comprobar esta función, se verificó el correcto funcionamiento del Módulo con su fuente de alimentación. Después, se procedió a desconectar la fuente y apagar el *Access Point* para simular un corte de energía eléctrica. En la Figura 3.17 se muestra el Módulo sin conexión de alimentación funcionando correctamente. El Módulo continuó funcionando sin alimentación por un tiempo de 1 hora con 45 minutos.

Figura 3.17. Verificación de funcionamiento sin alimentación.

Gestión de reinicio

En base a la función gestionReinicio del *sketch* principal del Módulo se programaron reinicios periódicos, reinicio por desconexión de la red wifi y por desconexión a Firebase. En la Figura 3.18 se muestra el mensaje que se produce en consola previo a un reinicio.

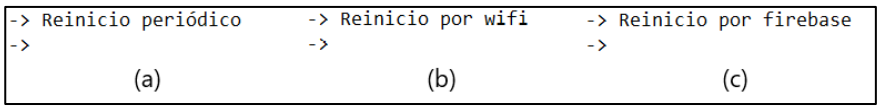


Figura 3.18. Verificación de reinicio del Módulo

3.2.2. GESTIÓN DE USUARIOS

Para esta sección, se detallarán los resultados de las pruebas de creación, lectura, actualización y eliminación de usuarios. Estas acciones implican modificación en los archivos y plantillas de huella digital almacenadas en Módulo de Recolección de Datos.

3.2.2.1. Creación de Usuarios y Creación de Cuenta

En primer lugar, desde el Módulo Web/Móvil se debe crear un usuario nuevo desde la cuenta administrador. Una vez ingresado al sistema, se verifica el correcto funcionamiento del proceso de creación con los datos de un determinado usuario.

3.2.2.1.1. Creación de Usuarios y Habilitación de cuenta en el Módulo Web/Móvil

Para la creación de un determinado usuario, se accedió a la pestaña *Usuarios* en el Módulo. luego se pulsó el botón *Crear Nuevo Usuario*. En la Figura 3.19 se puede observar este proceso en el Módulo Web (a) y Móvil (b).

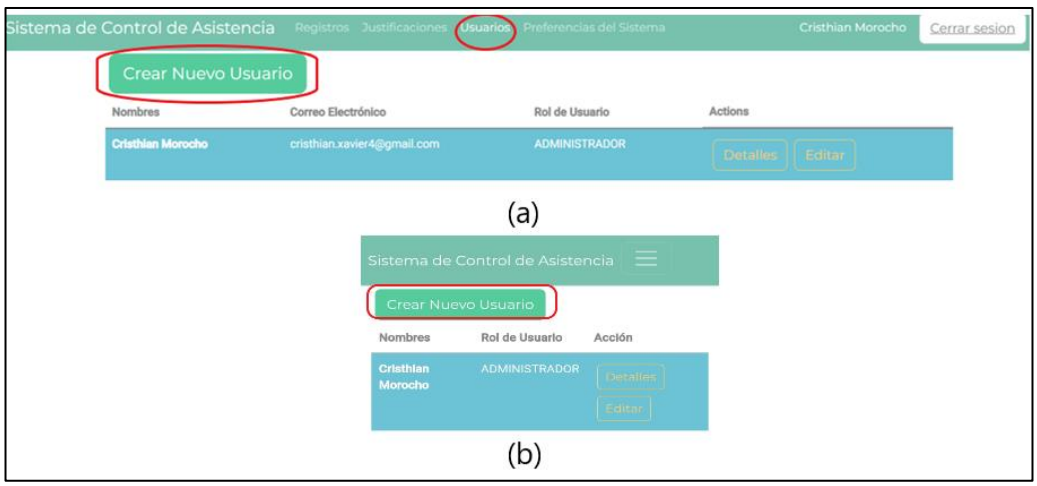


Figura 3.19. Proceso de creación de usuario.

Se desplegó una interfaz para ingresar los datos del usuario nuevo. En esta interfaz se llenaron los campos con la información del usuario a ingresar.

En la Figura 3.20 se muestra la interfaz de usuario nuevo con los datos ingresados.

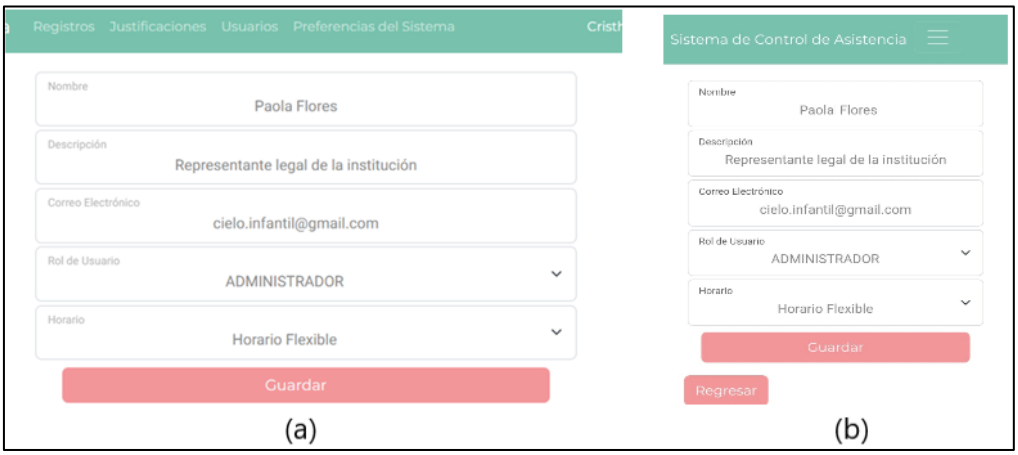


Figura 3.20. Proceso de creación de usuario (información).

Al pulsar el botón Guardar, el usuario se crea y se desplegará una ventana modal donde solicitó una contraseña para crear una cuenta para este usuario. Esta ventana se puede observar en la Figura 3.21.

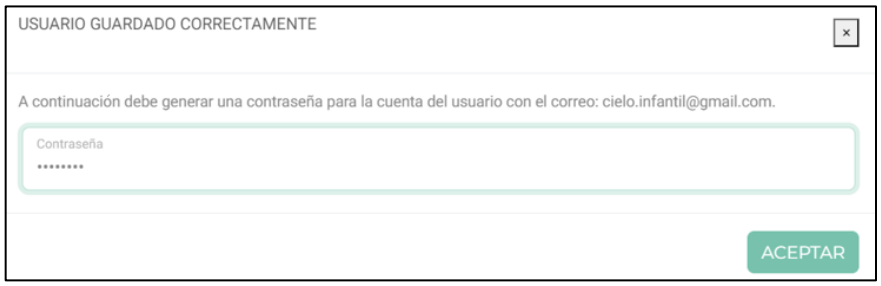


Figura 3.21. Creación de cuenta para el usuario nuevo.

Una vez ingresada la contraseña y pulsado el botón Aceptar, se cerró la sesión del usuario donde se creó este usuario. A continuación, se autenticó con las credenciales del usuario. En la Figura 3.22 se puede observar el ingreso de las credenciales del nuevo usuario.

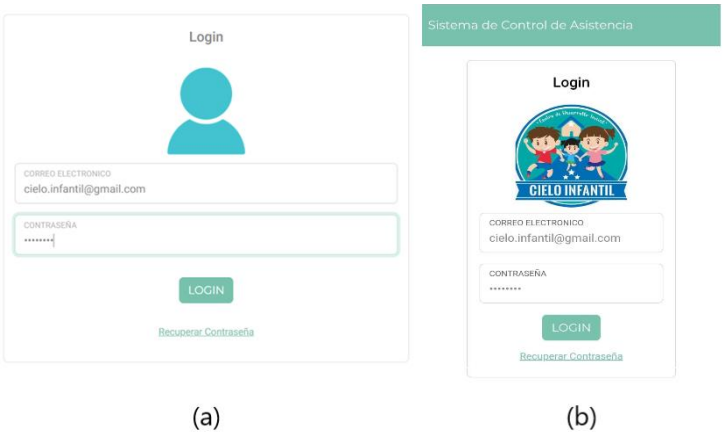


Figura 3.22. Pantalla de inicio de sesión.

Una vez autenticado con este usuario, se puede ver que junto al botón Cerrar sesión se muestra el nombre del nuevo usuario. Se accedió a la pestaña Usuarios y se verificó la creación de este usuario. En la Figura 3.23 se puede observar los usuarios registrados en el sistema.

Figura 3.23. Interfaz del usuario nuevo.

Este proceso se repitió para todo el personal de la institución.

Además, en la Figura 3.24 se muestra la creación de un nuevo documento con los datos de usuario en *Cloud Firestore* y la inclusión de un nuevo correo para el proceso de autenticación mediante los servicios de Firebase.

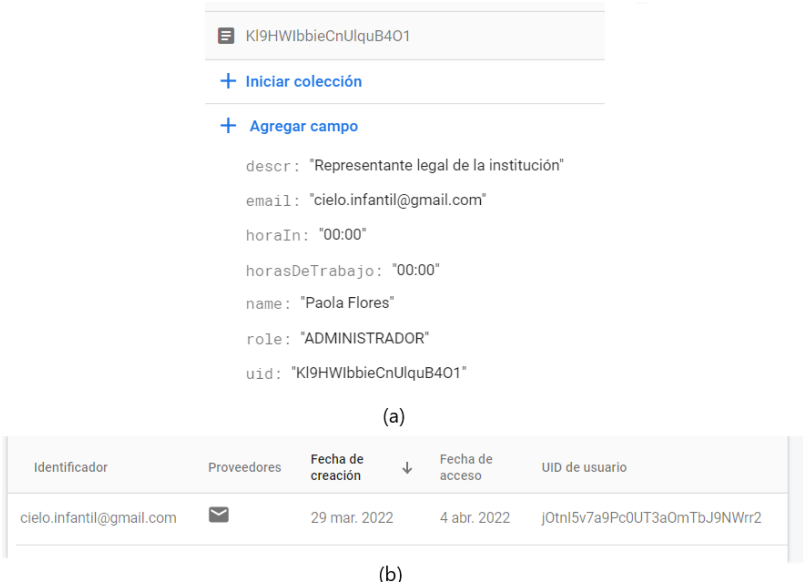


Figura 3.24. Documento de *Cloud Firestore* y usuario en *Firebase Authentication* del usuario creado.

Por último, si este proceso se realizó con éxito, en la pestaña *Preferencias del Sistema* se mostrará un mensaje de que el Módulo de Recolección de Datos debe crear una plantilla de huella digital nueva, tal y como se muestra en la Figura 3.25.

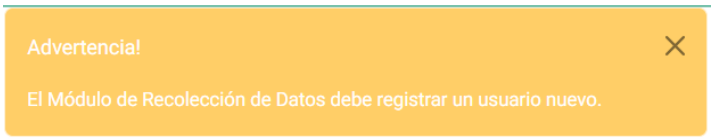


Figura 3.25. Notificación en interfaz Preferencias.

Después de crear un nuevo usuario, no se podrá crear otro hasta que se almacene su huella digital.

En el caso de que no se haya generado la cuenta en el proceso de creación de usuario, se implementó una opción que permita generar una cuenta en *Firebase Authentication*. Para generar una cuenta, se accede a la sección de *Preferencias* y se pulsa el botón de *Generar Cuenta.* Se mostrará una ventana modal como se visualiza en la Figura 3.26.

Para verificar el funcionamiento de la nueva cuenta, se procede a iniciar con las credenciales de la nueva cuenta generada o en la consola de Firebase mostrando los usuarios almacenados.

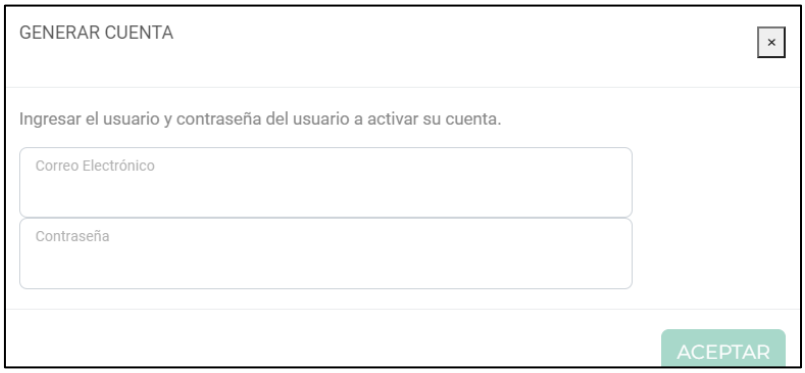


Figura 3.26. Generación de cuenta.

3.2.2.1.2. Creación de Plantilla en el Módulo de Recolección de Datos

Una vez creado el usuario en el Módulo Web/Móvil, se procede a almacenar la plantilla de su huella digital. Para esto, se pulsó el botón *Opciones* (PULL1) para que el Módulo verifique qué acción debe realizar, en este caso, crear usuario. En la Figura 3.27 se muestra la respuesta del Módulo para esta acción.

Figura 3.27. Documento de Preferencias al crear un usuario nuevo.

El Módulo identificó el usuario nuevo (usuario que no contaba con identificador de huella digital) leyendo los datos de *Cloud Firestore*. Al encontrar el usuario, mostró en pantalla el nombre del mismo, tal y como se muestra en la Figura 3.28.

(a) (b)

Figura 3.28. Identificación de usuario nuevo.

El Módulo solicitó 2 veces la huella digital del usuario. En primer lugar, se ingresó dos huellas digitales diferentes, a lo cual el Módulo respondió con un mensaje de error. Este mensaje se muestra en la Figura 3.29. En el caso de que no exista coincidencia, volverá a solicitar que se ingrese la huella digital.

Figura 3.29. Aviso de huellas no coincidentes.

Después, se ingresaron dos huellas iguales, el módulo las comparó y las almacenó en memoria una vez verificado que las huellas coinciden. En la Figura 3.30 se muestra el resultado del proceso mencionado.

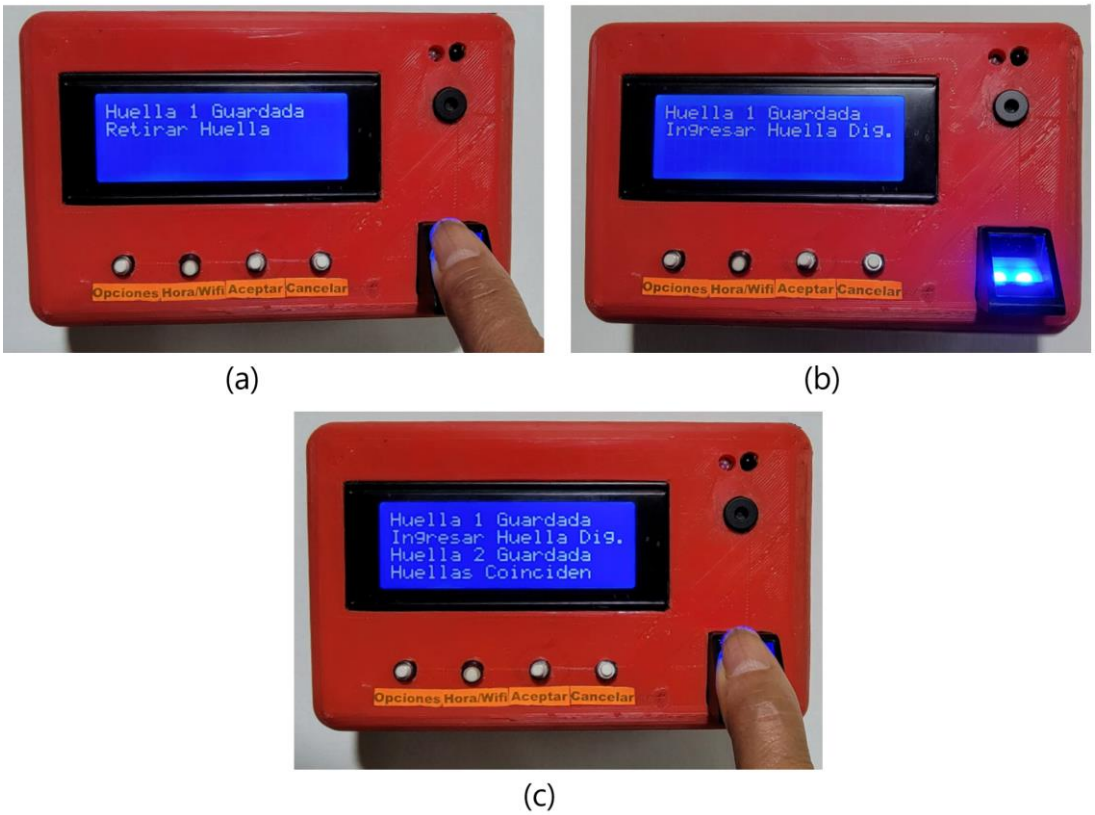


Figura 3.30. Proceso de creación de plantilla de huella digital nueva.

Una vez creada la plantilla, se verificó este proceso ingresando la huella digital del usuario creado y comprobando que el Módulo identifique el nombre de la persona. En la Figura 3.31 se muestra la coincidencia que encontró el Módulo después de crear la plantilla.

Figura 3.31. Identificación de usuario mediante huella digital.

3.2.2.2. Actualización de Usuarios

Para verificar el funcionamiento de actualización de usuarios, se ingresó al sistema con una cuenta de administrador. Se realizaron dos procedimientos: actualizar datos de usuario y actualizar plantilla digital de usuario.

3.2.2.2.1. Actualización de Datos de Usuario

Este proceso se realizó accediendo a la pestaña *Usuarios* y pulsando el botón *Editar* en el usuario al cual se desea modificar, como se muestra en la Figura 3.32.

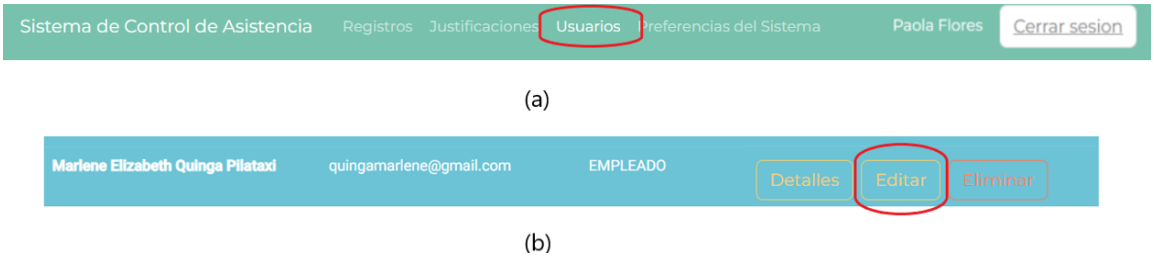


Figura 3.32. Interfaz de gestión de usuarios.

Se desplegó una interfaz con los datos del usuario seleccionado. En esta interfaz, se modificó el horario del usuario. Las modificaciones se muestran en la Figura 3.33.

Figura 3.33. Modificaciones para el usuario seleccionado.

En la Figura 3.34 se puede observar la ventana modal de confirmación de este proceso.

Para verificar el funcionamiento de actualización de datos, se accedió a la pestaña *Usuarios* y se pulsó el botón *Detalles* del usuario modificado. Se mostró una interfaz con los datos del usuario evidenciando la modificación realizada. Además, se pudo constatar el cambio efectuado en el documento de *Cloud Firestore* del usuario modificado (Figura 3.35).

Figura 3.34. Aviso de éxito en la operación.

Figura 3.35. Verificación de cambio en la información.

3.2.2.2.2. Actualización de Plantilla de Huella Digital

Para realizar este proceso, se accedió a la pestaña *Usuarios*, se seleccionó el usuario a modificar y se pulsó el botón *ACTUALIZAR HUELLA DIGITAL*. Al realizar este proceso, los campos del documento *Preferencias/sistema* destinados para la actualización de huella digital se modificaron para que en el Módulo se pueda realizar el procedimiento respectivo. Esto se presenta en la Figura 3.36.

En el Módulo de Recolección de Datos se pulsó el botón *Opciones* (*PULL1*) para acceder al identificador de huella que se va a actualizar. El Módulo eliminará la plantilla y creará una nueva plantilla con el mismo identificador. Este proceso se puede observar en la Figura 3.37.

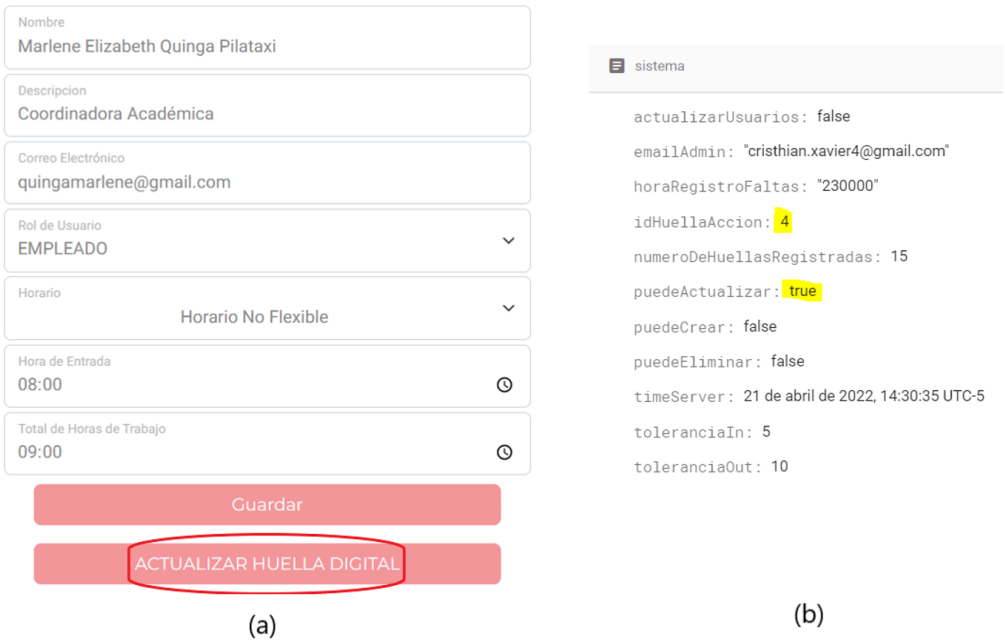


Figura 3.36. Proceso de actualización de huella y documento preferencias.

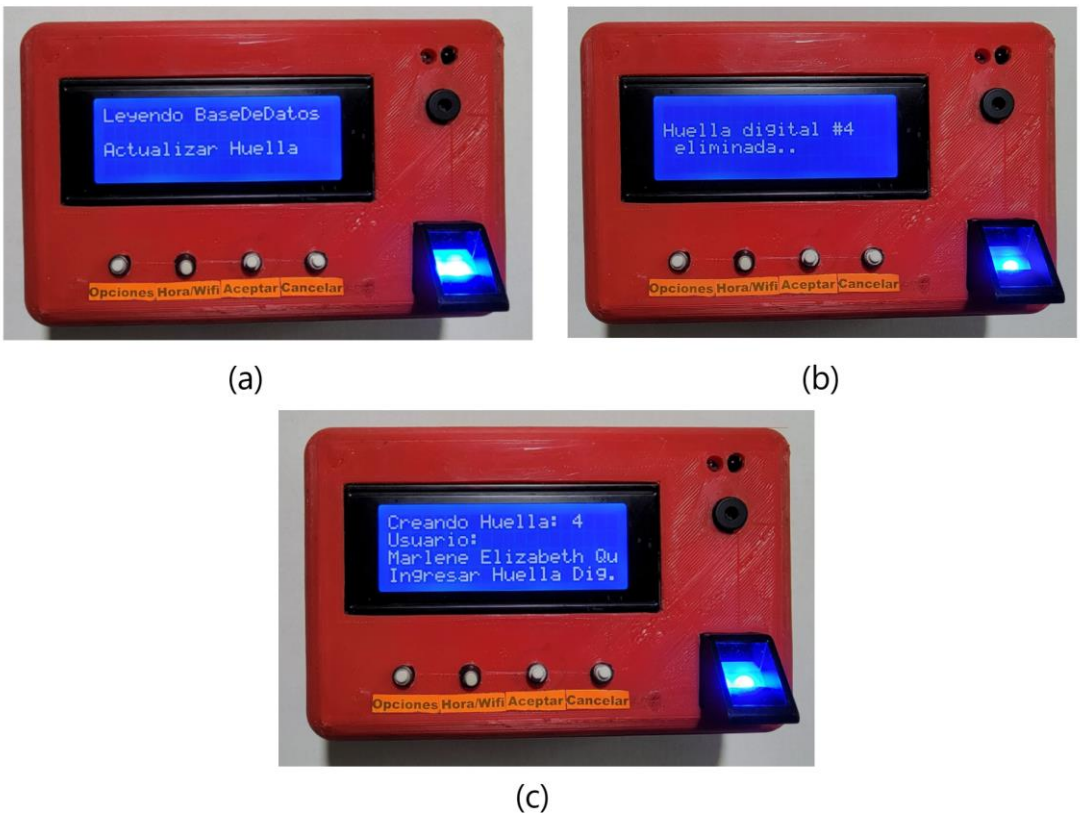


Figura 3.37. Proceso de actualización de plantilla de huella digital en el Módulo.

Luego de este proceso, se solicitó que ingrese 2 veces la huella digital tal y como se muestra en la Figura 3.38.

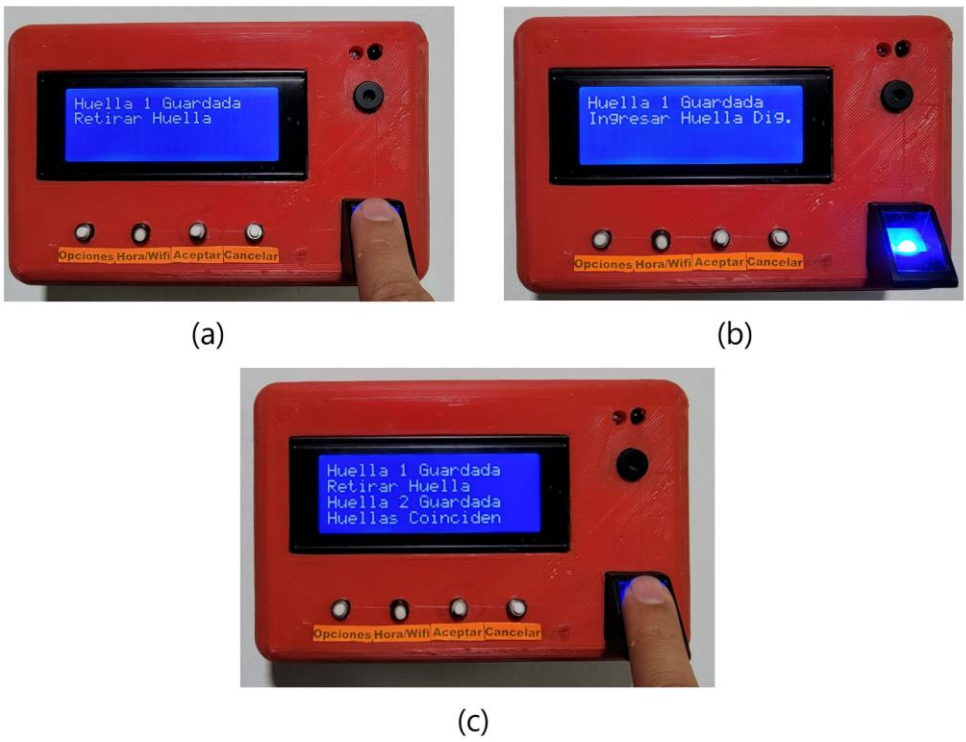


Figura 3.38. Registro de nueva plantilla de huella digital.

Para verificar que se realice correctamente este proceso se ingresó la nueva huella y se identificó al usuario, como se ve en la Figura 3.39.

Figura 3.39. Identificación del usuario mediante su huella digital.

3.2.2.3. Eliminación de Usuarios

Para este proceso, se creó un usuario de prueba. Se realizó todo el proceso de creación de usuario y plantilla de huella digital para generar un registro a su nombre y verificar la creación del mismo en el Módulo Web/Móvil. En la Figura 3.40 se muestra el proceso de creación del usuario con su cuenta en el Módulo Web.

Para verificar la creación de usuarios, se accedió a la pestaña *Usuarios* y se pulsó en el botón *Detalles* del usuario. Se mostró una interfaz en donde se puede visualizar toda la información ingresada en la creación del usuario, como se presenta en la Figura 3.41.

Figura 3.40. Creación de un usuario de prueba.

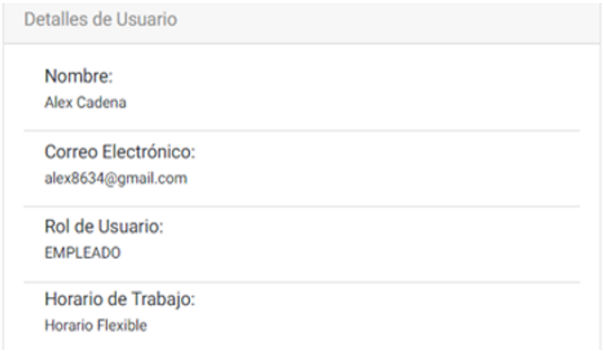


Figura 3.41. Verificación del proceso de creación del usuario de prueba.

En la Figura 3.42 se puede observar el proceso realizado para la creación de huella digital en el Módulo de Recolección de Datos.

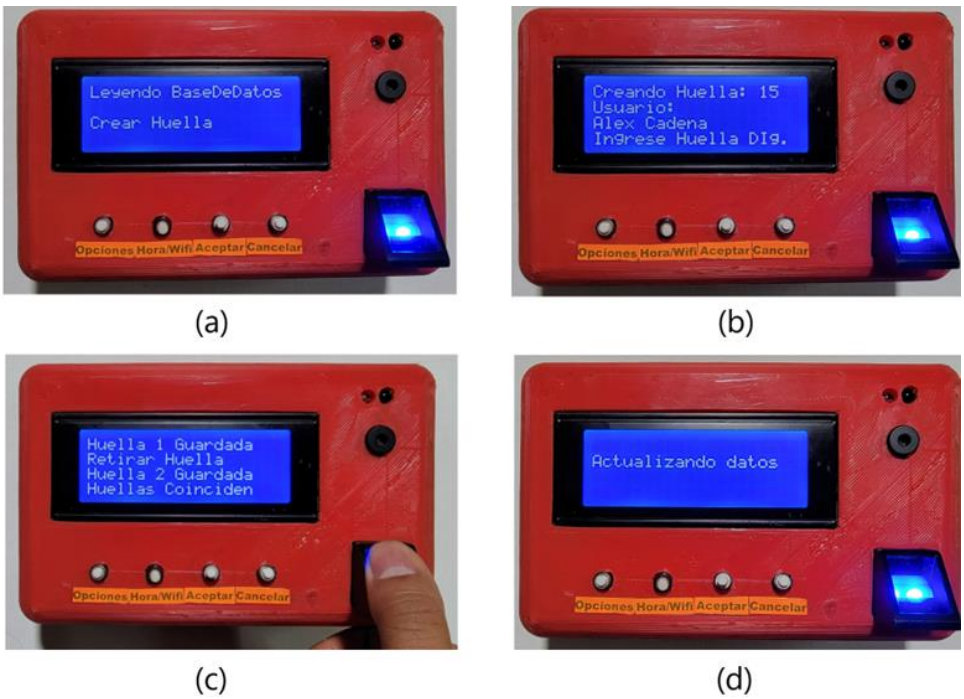


Figura 3.42. Proceso de creación de plantilla de huella para el usuario de prueba.

En la Figura 3.43 se puede observar el proceso realizado para la generación del registro del usuario creado.

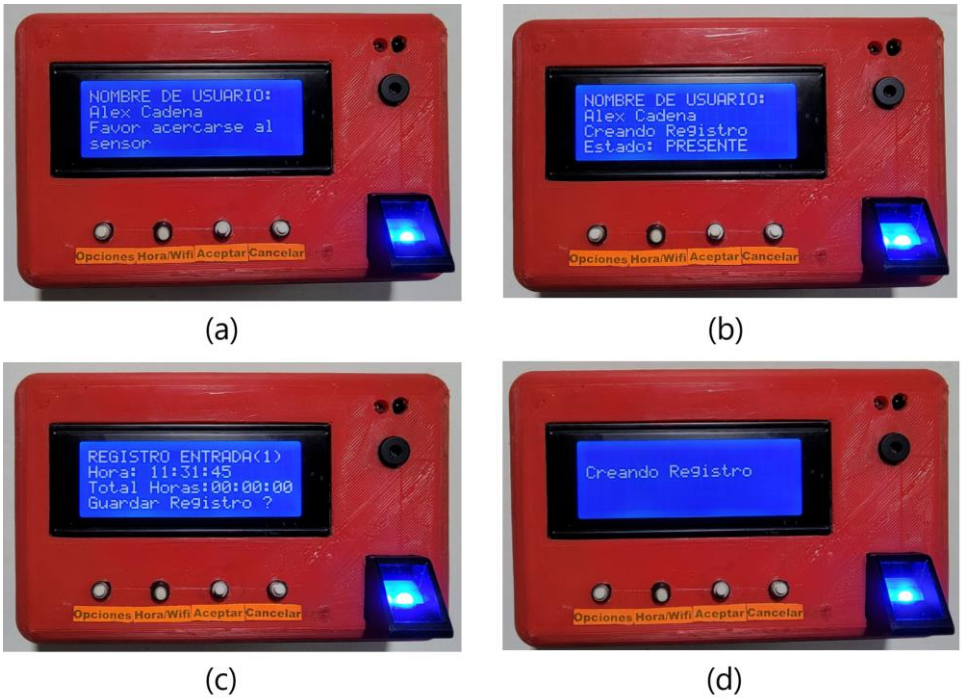


Figura 3.43. Proceso de generación de registro del usuario de prueba.

Para verificar la generación del registro, se accedió a la cuenta del usuario creado y se observó un registro creado. Este registro se muestra en la Figura 3.44.

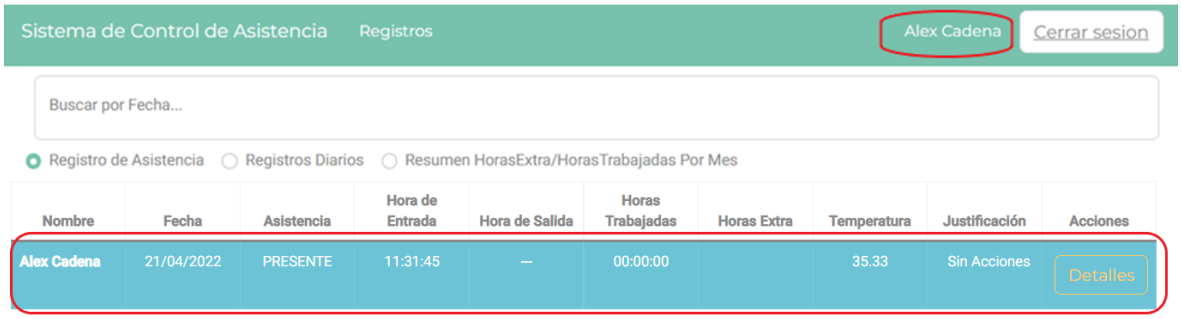


Figura 3.44. Visualización del registro generado por usuario de prueba.

Con esto, se pudo comprobar que el usuario estuvo activo y podía generar registros. Después, se accedió a la pestaña *Usuarios* en el Módulo Web/Móvil, se seleccionó el usuario que se va a eliminar y se pulsó el botón Eliminar tal y como se muestra en la Figura 3.45. Después de eliminar un usuario no se podrá eliminar otro.

En la Figura 3.46 se observa la ventana modal que se mostró al pulsar el botón *Eliminar*. Donde al pulsar la opción *Si* se confirmó la acción de eliminar un usuario.

Figura 3.45. Eliminación de usuario de prueba en el Módulo Web.

Figura 3.46. Mensaje de confirmación para eliminar el usuario de prueba.

Al aceptar la eliminación, se pudo constatar en la Figura 3.47 el cambio en el documento *Preferencias/sistema.* Los campos que se modificaron en este documento son: *idHuellaAccion* y *puedeEliminar*. Con el valor de estos campos, el Módulo de Recolección de Datos sabrá que identificador de huella debe eliminar.

Figura 3.47. Documento de Preferencias después de eliminar el usuario de prueba.

En el Módulo de Recolección de Datos, se pulsó el botón Opciones para acceder al identificador de huella del usuario que se eliminó, como se muestra en la Figura 3.48.

(a) (b)

Figura 3.48. Proceso de eliminación de plantilla de huella digital.

Para verificar el proceso de eliminación, se ingresó nuevamente la huella del usuario. El Módulo no fue capaz de identificar la plantilla de huella digital y mostró el mensaje presentado en la Figura 3.49.

Figura 3.49. Ingreso de la plantilla luego de eliminar un usuario.

3.2.3. TOMA DE REGISTROS

En esta sección se mencionarán los procedimientos que se realizaron para la verificación de la toma de registros en el Módulo de Recolección de Datos.

3.2.3.1. Identificación del Personal

Una vez agregados los usuarios en el sistema, el Módulo de Recolección de Datos será capaz de identificar a un determinado usuario mediante su huella dactilar; este proceso es independiente de la conexión a Internet.

En primer lugar, se realizaron pruebas de identificación del personal mediante el sensor de huella dactilar. Este proceso se realizó con una muestra de 8 empleados registrados, quienes generaron un registro de entrada y salida. En la Tabla 3.2 se muestra el resultado de los intentos exitosos (×) y fallidos (✓) en la identificación del personal que se realizó durante 5 días diferentes.

Tabla 3.2. Pruebas de lector de huellas digitales.

#	Registro	Empleado							
		1	2	3	4	5	6	7	8
1	Entrada	✓	✓	✓	✓	✓	✓	✓	✓
	Salida	✓	✓	✓	✓	✓	✓	×✓	✓
2	Entrada	✓	✓	✓	✓	✓	✓	✓	✓
	Salida	✓	✓	××✓	✓	✓	×✓	✓	✓
3	Entrada	✓	✓	✓	✓	✓	✓	✓	×✓
	Salida	✓	✓	✓	✓	✓	✓	×✓	✓
4	Entrada	✓	✓	✓	✓	✓	××✓	✓	✓
	Salida	✓	✓	✓	×✓	✓	✓	✓	✓
5	Entrada	✓	×✓	✓	✓	✓	✓	✓	✓
	Salida	×✓	✓	×✓	✓	✓	✓	✓	×✓

En la Tabla 3.3, se tabula el resumen de los datos obtenidos, se detalla el porcentaje de acierto y error en los intentos al identificar el usuario.

Tabla 3.3. Resumen de pruebas de lector de huellas digitales

#	Intentos	Intentos exitosos	Intentos fallidos	% acierto	%error
1	17	16	1	94.12%	5.88%
2	19	16	3	84.21%	15.79%
3	18	16	2	88.89%	11.11%
4	19	16	3	84.21%	15.79%
5	17	16	1	94.12%	5.88%
Promedio				89.12%	10.88%

Con estos datos realizados en la prueba, se obtuvo un 89.12% de éxito en la identificación del personal cuando el usuario ingrese su huella digital.

3.2.3.2. Generación de Registros de Entrada y Salida

En esta sección se detallan las estructuras de los registros de entrada/salida. Se analizó las estructuras de los registros cuando el Módulo tenía conexión a Internet y cuando la había perdido.

3.2.3.2.1. Estructura de registros

Una vez se identifique el usuario, inmediatamente se generarán los datos para construir el registro. En el caso de que sea registro de entrada (registro impar), se solicitará al usuario

la medición de su temperatura, por lo cual, debe acercarse al sensor. En la Figura 3.50 se muestra la toma de temperatura para un registro de entrada.

(a) (b)

Figura 3.50. Toma de temperatura por el Módulo de Recolección de Datos.

En la Figura 3.51 se muestra la salida de la consola serial cuando se genera un primer registro de entrada.

```
-> Creando documento...
-> {
->         "fields": {
->                 "id": {
->                         "stringValue": "2022041213AZjvHI4Gb1kFau79Kf"
->                 },
->                 "idUsuario": {
->                         "stringValue": "13AZjvHI4Gb1kFau79Kf"
->                 },
->                 "usuario": {
->                         "stringValue": "Erika Dayana Basantes Mancero"
->                 },
->                 "temperatura": {
->                         "arrayValue": {
->                                 "values": [
->                                         {
->                                                 "stringValue": 35.24
->                                         }
->                                 ]
->                         }
->                 },
->                 "asistencia": {
->                         "arrayValue": {
->                                 "values": [
->                                         {
->                                                 "stringValue": "ATRASO"
->                                         }
->                                 ]
->                         }
->                 },
->                 "horasTrabajadas": {
->                         "stringValue": "00:00:00"
->                 },
->                 "hora": {
->                         "arrayValue": {
->                                 "values": [
->                                         {
->                                                 "timestampValue": "2022-04-12T13:24:34Z"
->                                         }
->                                 ]
->                         }
->                 }
->         }
-> }
```

Figura 3.51. Estructura del primer registro de entrada

En la Figura 3.52 se muestra la salida de la consola serial en el momento que se genera un registro de entrada diferente al primero.

```
-> Actualizando Array Horas del documento: registros/2022041213AZjvHI4Gb1kFau79Kf
-> {
->         "fields": {
->                 "hora": {
->                         "arrayValue": {
->                                 "values": [
->                                         {
->                                                 "timestampValue": "2022-04-12T13:25:21Z"
->                                         }
->                                 ]
->                         }
->                 }
->         }
-> }
->
-> OK!
->
-> Actualizando Array Temperatura del documento: registros/2022041213AZjvHI4Gb1kFau79Kf
-> {
->         "fields": {
->                 "temperatura": {
->                         "arrayValue": {
->                                 "values": [
->                                         {
->                                                 "stringValue": "36.21"
->                                         }
->                                 ]
->                         }
->                 }
->         }
-> }
```

Figura 3.52. Estructura de registros de entrada.

En la Figura 3.53 se muestra la salida de la consola serial cuando se genera un registro de salida.

```
-> Actualizando Array Horas del documento: registros/2022041213AZjvHI4Gb1kFau79Kf
-> {
->         "fields": {
->                 "hora": {
->                         "arrayValue": {
->                                 "values": [
->                                         {
->                                                 "timestampValue": "2022-04-12T13:24:34Z"
->                                         }
->                                 ]
->                         }
->                 }
->         }
-> }
->
-> OK!
->
-> Actualizando Horas Trabajadas en el documento: registros/2022041213AZjvHI4Gb1kFau79Kf
-> {
->         "fields": {
->                 "horasTrabajadas": {
->                         "stringValue": "09:23:51"
->                 }
->         }
-> }
->
-> OK!
```

Figura 3.53. Estructura de registros de salida.

Así mismo, se realizaron pruebas cuando el Módulo no contaba con conexión a Internet. En este caso, los registros se guardaban en el archivo *registrosSinSubir.txt* de la memoria SD. En la Figura 3.54 se muestra la salida de la consola serial cuando se genera el primer registro de entrada (a) y la estructura del registro en el archivo *registrosSinSubir.txt* (b).

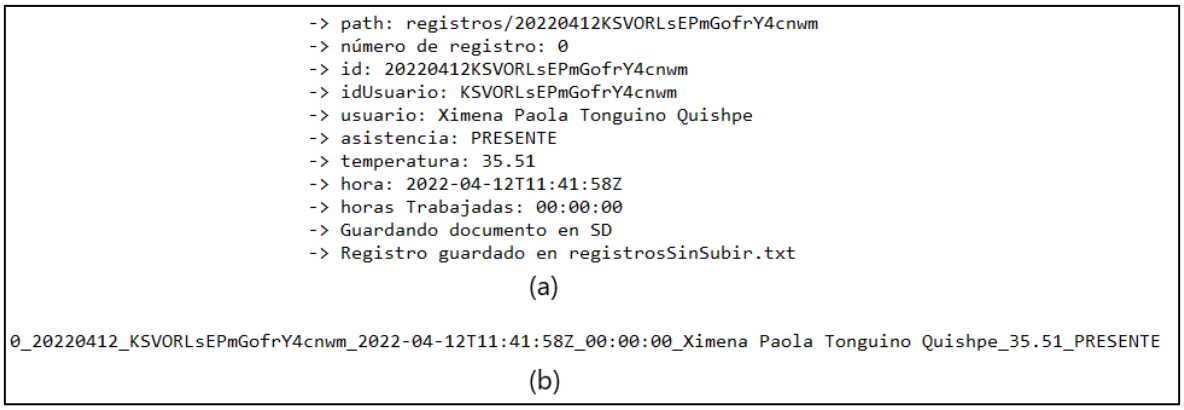


Figura 3.54. Estructura del primer registro de entrada en memoria SD

En la Figura 3.55 se muestra la salida de la consola serial en el momento que se genera un registro de entrada diferente al primero (a) y la estructura del mismo en el archivo *registrosSinSubir.txt* (b).

En la Figura 3.56 se muestra la salida de la consola serial cuando se genera un registro de salida (a) y la estructura de este en el archivo *registrosSinSubir.txt* (b).

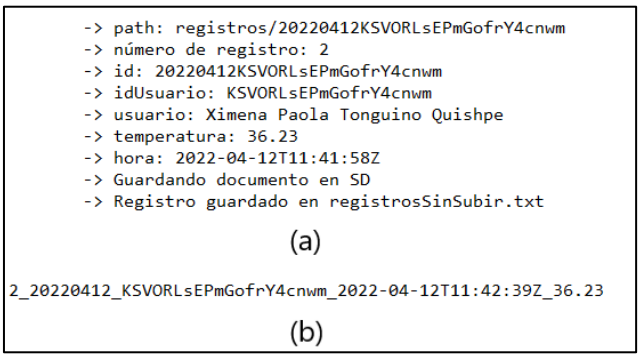


Figura 3.55. Estructura de los registros de entrada en memoria SD.

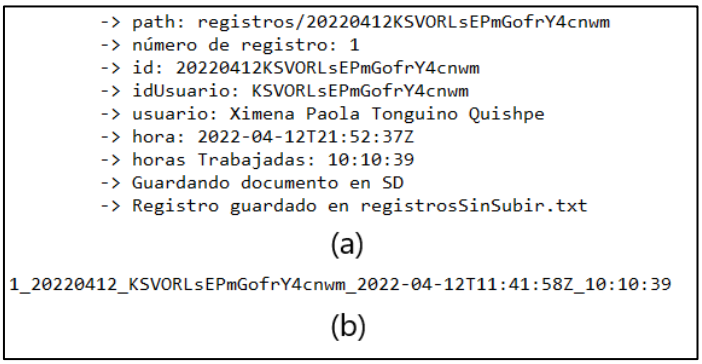


Figura 3.56. Estructura de los registros de salida en memoria SD.

3.2.3.2.2. Tiempo de Creación de Registro

Se midieron los tiempos de respuesta del Módulo cuando se creaban los registros de entrada y salida. Durante esta prueba, el Módulo contaba con conexión a Internet.

En la Tabla 3.4 se tabulan los tiempos medidos desde que se ingresaba la huella de un usuario, hasta que se mostraba el mensaje de “Registro guardado correctamente”.

Tabla 3.4. Tiempos de creación de registros con conexión a Internet.

#	Registro	Empleado							
		1	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	Entrada	25.62	24.06	25.44	27.80	27.38	27.59	24.83	27.46
	Salida	19.38	18.41	19.63	20.26	18.85	19.45	18.64	20.04
2	Entrada	24.03	26.07	25.12	24.10	25.63	25.30	26.49	26.33
	Salida	20.56	18.16	18.90	18.92	20.53	20.14	20.94	18.42
3	Entrada	25.87	26.86	24.94	25.81	27.08	26.53	26.23	27.92
	Salida	20.70	19.13	18.04	18.26	19.64	19.98	19.43	19.26
4	Entrada	24.84	24.98	25.52	26.12	24.16	24.76	27.92	26.32
	Salida	19.59	18.91	19.88	19.55	18.88	20.86	18.67	18.68
5	Entrada	26.91	25.35	26.59	24.89	26.21	27.70	27.99	24.39
	Salida	20.37	18.94	18.11	18.48	19.97	21.00	20.42	18.10

En la Tabla 3.5 se muestran los tiempos promedio de creación de los registros de entrada y salida medidos anteriormente.

Tabla 3.5. Tiempo promedio de creación de registros con conexión a Internet.

#	Registro	Promedio (s)
1	Entrada	26.2725
	Salida	19.3325
2	Entrada	25.38375
	Salida	19.57125
3	Entrada	26.405
	Salida	19.305
4	Entrada	25.5775
	Salida	19.3775
5	Entrada	26.25375
	Salida	19.42375
Promedio	Entrada	25.9785
	Salida	19.402

Se repitió el mismo proceso cuando el Módulo no contaba con conexión a Internet. En la Tabla 3.6 se tabulan los tiempos medidos.

Se tiene un tiempo promedio de 25.97 segundos en el registro de entrada y 19.40 segundos en un registro de salida cuando el Módulo tiene conexión a Internet.

Tabla 3.6. Tiempos en segundos de creación de registros sin conexión a Internet.

#	Registro	Empleado							
		1	2	3	4	5	6	7	8
1	Entrada	21.38	23.13	22.62	20.86	23.77	23.39	22.88	21.7
	Salida	16.17	16.4	14.1	14.16	15.19	16.12	14.04	14.44
2	Entrada	20.61	21.63	23.23	20.67	22.26	23.91	23.95	24
	Salida	16.04	14.49	16.7	17	15.11	15.12	14.44	16.55
3	Entrada	22.97	21.42	21.22	22.12	23.64	23.67	22.71	22.37
	Salida	14.99	15.3	15.51	14.2	15.83	14.9	16	14.99
4	Entrada	23.41	22.24	23.8	22.13	21.16	22.56	22.38	22.87
	Salida	16.16	15.85	15.13	15.54	15.57	16.44	15.21	14.6
5	Entrada	23.46	21.79	22.01	23.04	21.92	22.01	22.41	20.53
	Salida	15.11	14.96	14.38	15.3	15.27	15.76	15.12	14.61

En la Tabla 3.7 se muestran los tiempos promedio de creación de los registros de entrada y salida de los tiempos medidos en la Tabla 3.6.

Tabla 3.7. Tiempo promedio de creación de registros sin conexión a Internet.

#	Registro	Promedio (s)
1	Entrada	22.46625
	Salida	15.0775
2	Entrada	22.5325
	Salida	15.68125
3	Entrada	22.515
	Salida	15.215
4	Entrada	22.56875
	Salida	15.5625
5	Entrada	22.14625
	Salida	15.06375
Promedio	Entrada	22.44575
	Salida	15.32

Por otro lado, se tiene 22.44 segundos para la creación de un registro de entrada y 15.32 segundos para un registro de salida cuando el Módulo no cuente con conexión a Internet.

Un registro de salida se demora más ya que solicita al usuario la lectura de su temperatura corporal.

Con estos datos, se obtiene una idea del tiempo en que los usuarios emplearán para generar un determinado registro de asistencia.

3.2.3.3. Registros Diarios

```
registros/20220412RR2T4KhldwK91V21eTaE_06:40:11_00:00:00
registros/20220412KSVORLsEPmGofrY4cnwm_06:41:58_00:00:00
registros/20220412gY0gnWTAzIjMm071TsUQ_06:50:04_00:00:00
registros/20220412xrwg2wfrAi0hkUz88bCk_07:32:42_00:00:00
registros/20220412259chDhjVu6geyJXsKiP_07:50:14_00:00:00
registros/20220412QZqtiF6BxMnX14mExyZ1_07:56:05_00:00:00
registros/2022041231jJRiirgObRk7wdKwH5_07:57:32_00:00:00
registros/202204120fkU4xJ8eF3ddT4Q1hP8_08:06:02_00:00:00
registros/2022041213AZjvHI4Gb1kFau79Kf_08:24:34_00:00:00
registros/20220412G4Jab4dT4BGPta9sdgAZ_08:54:25_00:00:00
registros/2022041214NwhHDKkwWYNb4I3SrK_09:24:19_00:00:00
registros/20220412Kl9HWIbbieCnUlquB4O1_09:32:21_00:00:00
registros/202204120fkU4xJ8eF3ddT4Q1hP8_13:13:31_05:07:29
registros/20220412QZqtiF6BxMnX14mExyZ1_13:14:46_05:18:41
registros/20220412259chDhjVu6geyJXsKiP_13:28:47_05:38:33
registros/2022041231jJRiirgObRk7wdKwH5_13:33:32_05:36:00
registros/20220412KSVORLsEPmGofrY4cnwm_16:52:37_10:10:39
registros/20220412RR2T4KhldwK91V21eTaE_16:59:14_10:19:03
registros/20220412gY0gnWTAzIjMm071TsUQ_17:02:49_10:12:45
registros/20220412xrwg2wfrAi0hkUz88bCk_18:01:29_10:28:47
registros/2022041213AZjvHI4Gb1kFau79Kf_18:03:43_09:39:09
registros/20220412G4Jab4dT4BGPta9sdgAZ_18:17:22_09:22:57
registros/20220412Kl9HWIbbieCnUlquB4O1_18:54:27_09:22:06
registros/2022041214NwhHDKkwWYNb4I3SrK_18:55:16_09:30:57
```

Figura 3.57. Contenido del archivo *registrosDiarios.txt*.

Al finalizar la jornada laboral de la institución, se leyó el contenido del archivo *registrosDiarios.txt*. En la Figura 3.57 se muestra el contenido de este archivo y puede observar como el Módulo generó correctamente cada registro diario del personal.

3.2.3.4. Generación de Registros por Falta

Para comprobar la generación de faltas por el Módulo, se tomó un día y hora determinada en el cual se pidió a un usuario que no genere registros. Se visualizó el monitor serial en el momento en que se generó la falta para verificar la estructura del registro. Además, se tomaron los tiempos de creación de registros de faltas.

En esta sección, se detallan las pruebas de funcionamiento que se realizaron para verificar este proceso.

3.2.3.4.1. Estructura de registro

En el documento preferencias/sistema de Cloud Firestore, se definió la hora máxima en la que un determinado usuario puede generar un registro de entrada. Pasada esta hora, se generarán faltas para todos los usuarios que no hayan generado al menos un registro en

ese día y tengan un horario no flexible, es decir, cuenten con hora de entrada y horas de trabajo.

Figura 3.58. Estructura de registro de falta.

Para esta prueba, la hora de generación de faltas se definió a las 16:00. Por lo cual, se esperó a esa hora para que el Módulo genere faltas.

Una vez llegue la hora de generación de faltas, se verifica la creación del registro de falta en el monitor serial, el resultado se muestra en la Figura 3.58.

Después, el registro creado aparecerá en el archivo *registrosDiarios.txt*. En la Figura 2.57 se muestra el contenido de este archivo.

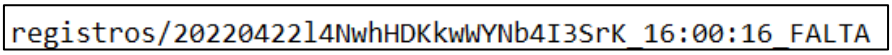


Figura 3.59. Registro diario de la falta generada.

Para la verificación de este proceso, se puede ver en la Figura 3.60 que en Cloud Firestore se ha creado un documento con los datos del usuario a quien se generó la falta.

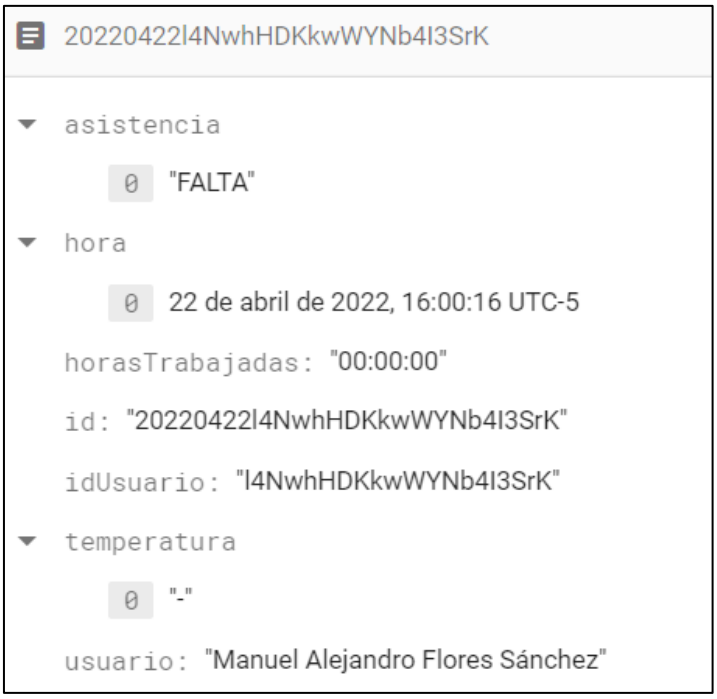


Figura 3.60. Estructura de falta en Cloud Firestore.

3.2.3.4.2. Tiempo de registro

Para realizar esta prueba, se cronometró el proceso de generación del registro de falta desde el momento en que la consola serial notificaba que se generaría una falta. Durante cada prueba, se fue modificando la hora máxima de generación de un registro y borrando los registros de faltas creados en la prueba anterior; se eliminan los registros diarios (en el archivo *registrosDiarios.txt*) y los documentos en *Cloud Firestore* para que el Módulo pueda generar nuevamente los registros faltantes. Se tomó el tiempo de 3 registros durante 5 pruebas.

En la Tabla 3.8 se tabulan los tiempos de generación de los registros de falta y se detalla el promedio total de este proceso.

Tabla 3.8. Tiempo de creación de faltas.

Prueba	Tiempo 1	Tiempo 2	Tiempo 3	Promedio
1	15.05	15.53	16.59	15.72
2	17.69	17.39	17.88	17.65
3	17.16	16.77	17.73	17.22
4	17.68	17.43	17.33	17.48
5	15.38	16.7	16.64	16.24
Promedio Total (seg)				16.86

Con este tiempo promedio, se resuelve el tiempo en que el Módulo de Recolección de Datos estará generando faltas. Este tiempo se multiplicará por los usuarios que no han generado un registro de asistencia.

3.2.4. VISUALIZACIÓN REGISTROS

En esta sección se detallarán los pasos que se realizaron para verificar el correcto funcionamiento de la visualización de los registros recolectados. Para esto, se usará el Módulo Web y Móvil. Es necesario ingresar a la página web y autenticarse con las credenciales de cada usuario. Hecho esto, se podrá acceder a las funcionalidades de los Módulos para su verificación.

3.2.4.1. Módulo Web

Una vez autenticado con un usuario, se da clic en *Registros*, luego de lo cual se mostrará la pestaña del resumen de asistencia, como se presenta en la Figura 3.61.

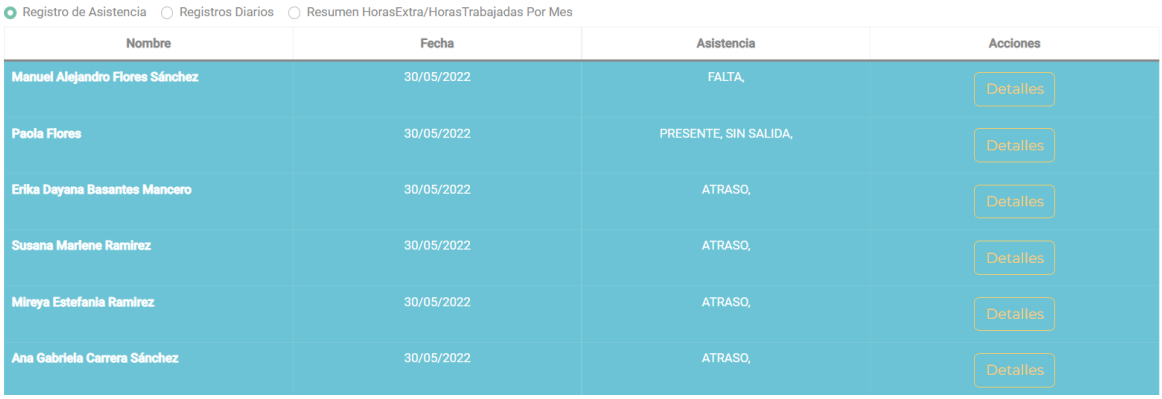


Figura 3.61. Visualización de registros de asistencia en el Módulo Web.

La segunda pestaña indicará los registros diarios. Esto se puede observar en la Figura 3.62. La última pestaña, presentará el resumen de registros mensuales. Esto se observa en la Figura 3.63.

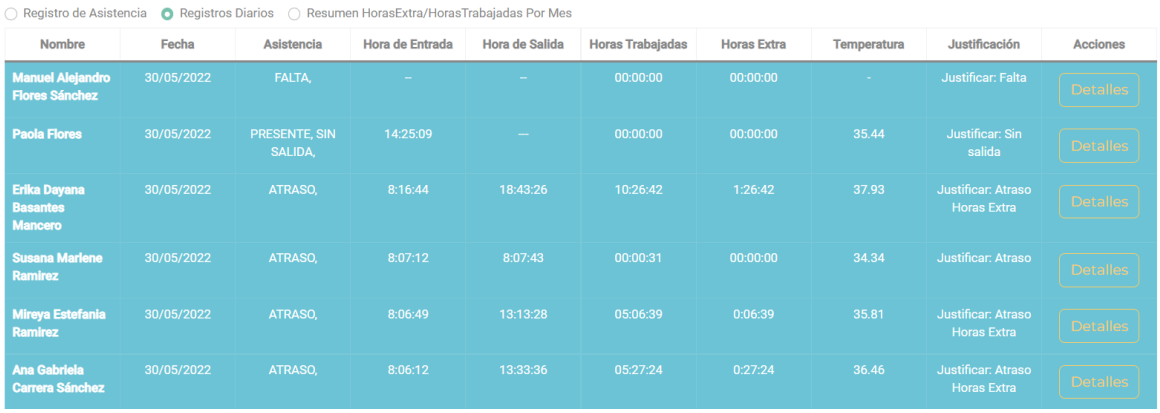
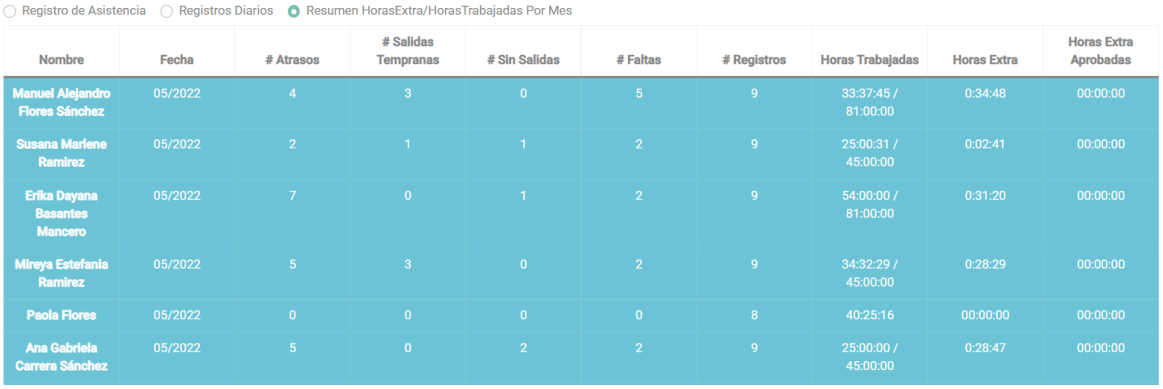


Figura 3.62. Visualización de registros diarios en el Módulo Web.



Registro de Asistencia Registros Diarios Resumen HorasExtra/HorasTrabajadas Por Mes

Nombre	Fecha	# Atrasos	# Salidas Tempranas	# Sin Salidas	# Faltas	# Registros	Horas Trabajadas	Horas Extra	Horas Extra Aprobadas
Manuel Alejandro Flores Sánchez	05/2022	4	3	0	5	9	33:37:45 / 81:00:00	0:34:48	00:00:00
Susana Marlene Ramirez	05/2022	2	1	1	2	9	25:00:31 / 45:00:00	0:02:41	00:00:00
Erika Dayana Basantes Mancero	05/2022	7	0	1	2	9	54:00:00 / 81:00:00	0:31:20	00:00:00
Mireya Estefania Ramirez	05/2022	5	3	0	2	9	34:32:29 / 45:00:00	0:28:29	00:00:00
Paola Flores	05/2022	0	0	0	0	8	40:25:16	00:00:00	00:00:00
Ana Gabriela Carrera Sánchez	05/2022	5	0	2	2	9	25:00:00 / 45:00:00	0:28:47	00:00:00

Figura 3.63. Visualización de registros mensuales en el Módulo Web.

3.2.4.2. Módulo Móvil

Debido a que en los dispositivos móviles se cuenta con un tamaño de pantalla reducida en comparación a una computadora, se visualizarán únicamente 2 pestañas. La primera presenta un resumen general de asistencia y la segunda incluye el resumen mensual de los registros.

La pestaña de resumen diario de asistencia y el detalle de una determinada asistencia se observa en la Figura 3.64 y en la Figura 3.65 se muestra un resumen mensual de asistencia juntamente con la página donde se obtiene toda esta información.

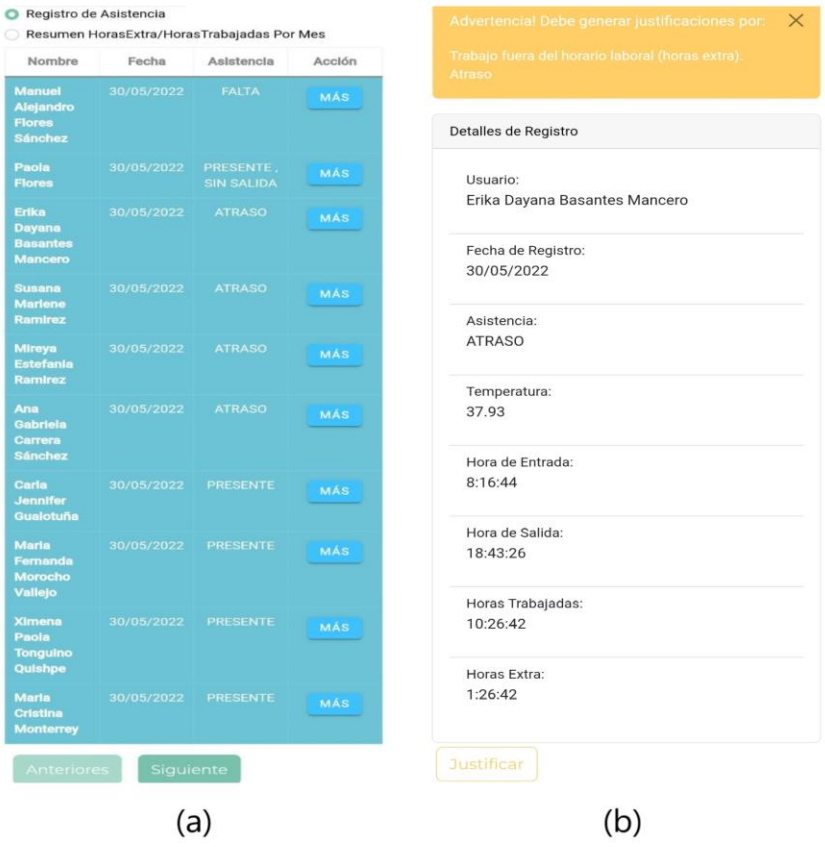


Figura 3.64. Visualización de registros diarios en el Módulo Móvil.

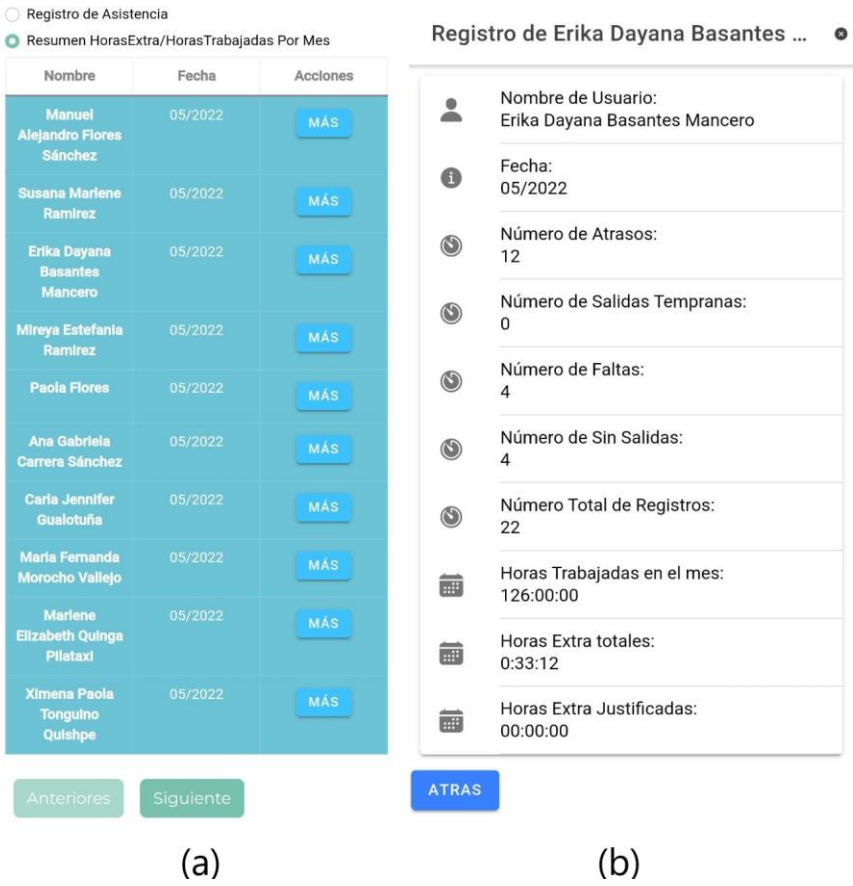


Figura 3.65. Visualización de registros mensuales en el Módulo Móvil.

3.2.5. GESTIÓN DE JUSTIFICACIONES

En esta sección, se detallarán los procesos de creación de justificaciones y su gestión por parte del personal administrativo para verificar su correcto funcionamiento.

3.2.5.1. Creación de Justificaciones

Un usuario puede generar una justificación si ha llegado tarde (atraso), si no ha asistido a trabajar (falta), si su horario laboral no fue completado (salida temprana), si ha trabajado fuera de su horario laboral (horas extra) o, si no ha generado un registro de salida (sin salida).

Para verificar este procedimiento se crearon justificación de un registro de atraso con horas extra. Para esto, se accedió al sistema en el Módulo Web, se ubicó el registro para ser justificado y se pulsó en el botón Detalles; se puede visualizar en la Figura 3.66.

En la Figura 3.67 se puede observar los datos del registro y dos mensajes que notifica al usuario que debe generar justificaciones. Para crear una justificación, se pulsó el botón llamado *Justificar*.

Figura 3.66. Selección de registro para justificación.

Figura 3.67. Proceso de justificación del registro.

Al pulsar el botón *Justificar*, se mostró la interfaz en donde el usuario debe llenar los datos relacionados con la justificación del registro, como se observa en la Figura 3.68.

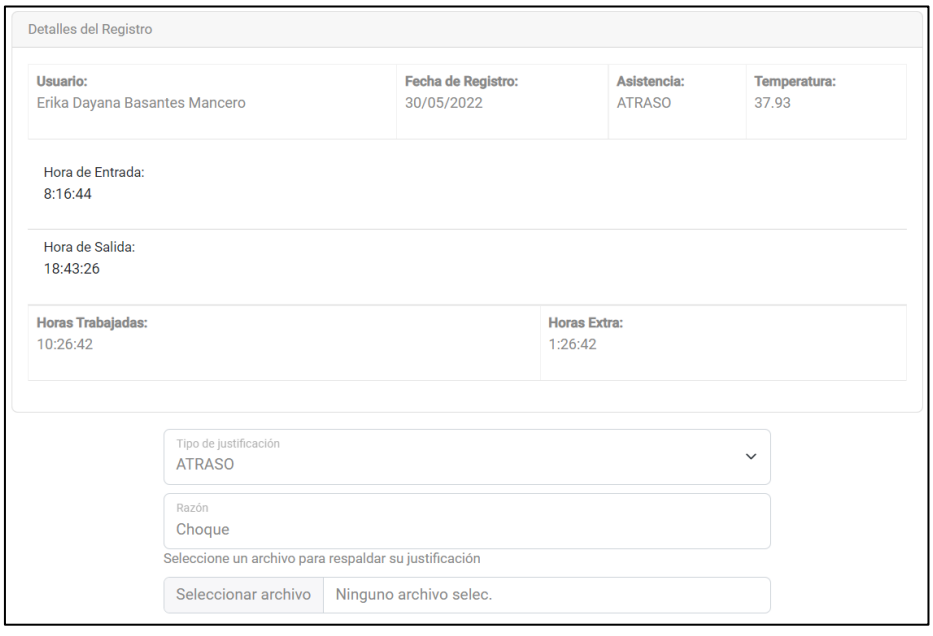


Figura 3.68. Creación de justificación en Módulo Web.

Una vez llenado los datos, se guardó la justificación. Para la justificación de horas extra, se procedió a realizar el mismo proceso en el Módulo Móvil; en la Figura 3.69 se muestra el resultado.

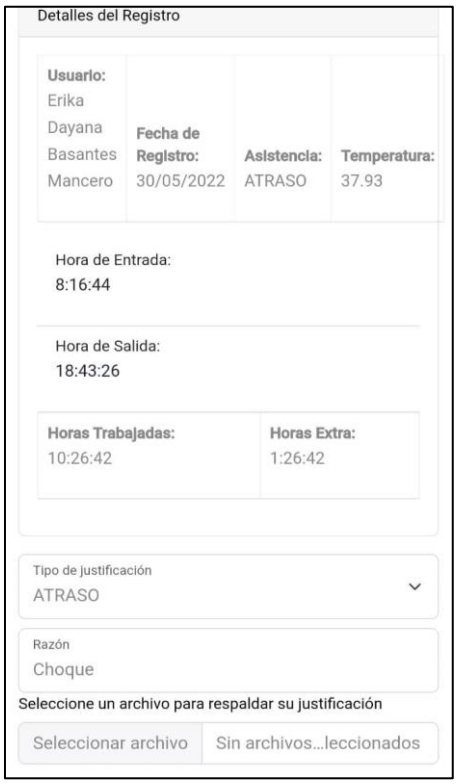


Figura 3.69. Creación de justificación en el Módulo Móvil.

Se seleccionó un archivo para respaldar la justificación. Este archivo solo puede ser tipo imagen con formato jpg, png o archivo formato pdf; estos tipos de formato se limitaron en la codificación de la funcionalidad del Módulo. Se pulsó en subir archivo y se guardó la justificación. Este proceso se puede ver en la Figura 3.70.

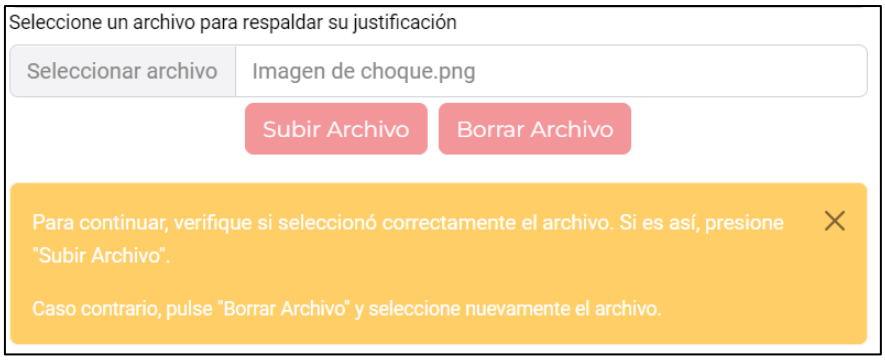


Figura 3.70. Subida de archivos para respaldo de justificación.

Después de subir el archivo, se podrá guardar la justificación. En la Figura 3.71 se muestra un hipervínculo con el *link* del archivo subido.

Figura 3.71. Link de archivo de justificación subido.

Para verificar este proceso, en la Figura 3.72 se puede observar que se accedió a los detalles del registro y se pudo observar que existen justificaciones relacionadas con el mismo.

Figura 3.72. Verificación del proceso de creación de justificación.

Al pulsar el hipervínculo se desplegará la imagen subida al crear la justificación. Esto se puede ver en la Figura 3.73.

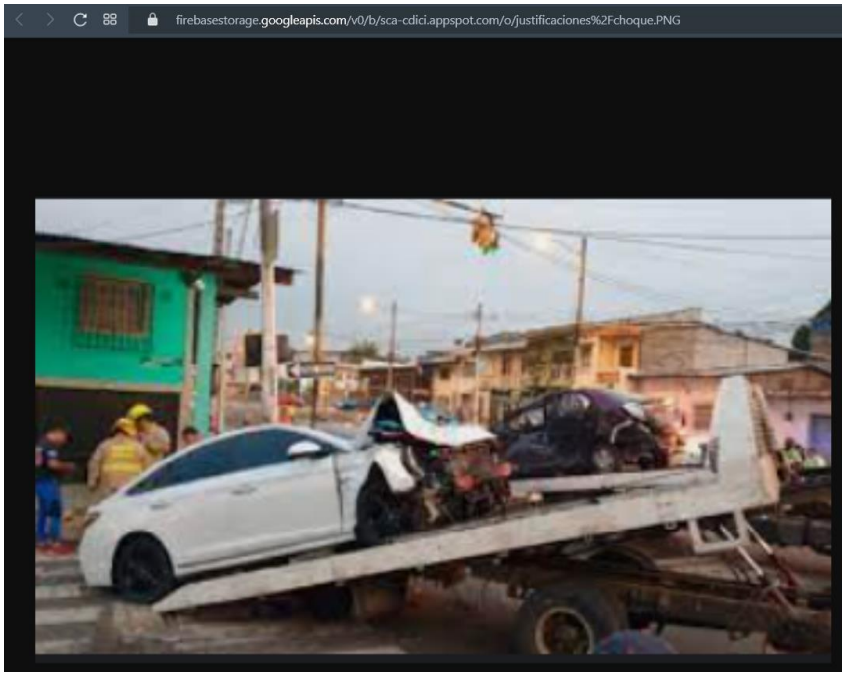


Figura 3.73. Acceso al archivo de justificación.

Además, en la Figura 3.74 se puede observar que se accedió al sistema con un usuario de rol *Administrador* y en la pestaña de Justificaciones se muestra esta justificación con estado en *Solicitado*.

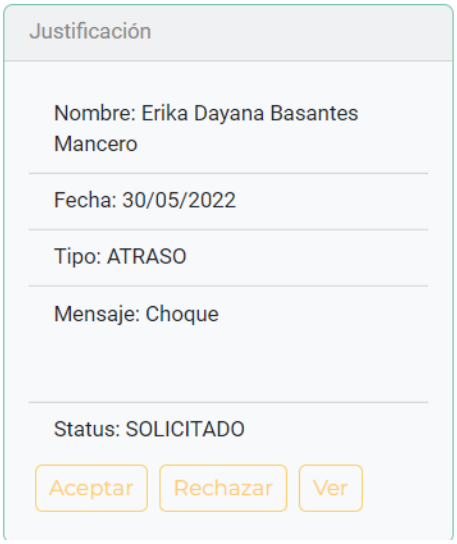


Figura 3.74. Tarjeta de justificación creada.

3.2.5.2. Gestión de Justificaciones para Usuario Administrador

Un usuario administrador puede aceptar o rechazar una solicitud de justificación según su criterio mediante el Módulo Web y Móvil.

Para verificar este procedimiento, se aceptaron y rechazaron las justificaciones anteriores para ver los efectos que causan estos hechos en el registro mensual. Para esto, se accedió a la pestaña de *Justificaciones* en el Módulo Web

En la Figura 3.75 se muestra el registro diario y el resumen del mes correspondiente al registro antes de aceptar una justificación.

Nombre	Fecha	# Atrasos	# Salidas Tempranas	# Sin Salidas	# Faltas	# Registros	Horas Trabajadas	Horas Extra	Horas Extra Aprobadas
Erika Dayana Basantes Mancero	05/2022	7	0	1	2	9	54:00:00 / 81:00:00	0:31:20	00:00:00

Figura 3.75. Resumen de registros mensuales antes de aceptar una justificación

En primer lugar, se aceptó la justificación de atraso. En la Figura 3.76 se observa que el *status* cambia.

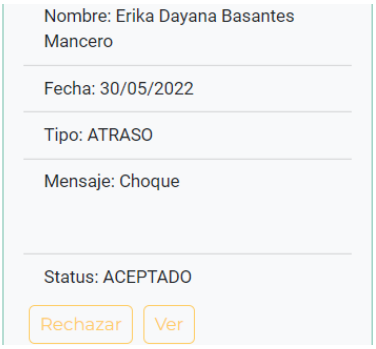


Figura 3.76. Estatus de la justificación cambia a Aceptado.

En la Figura 3.77 se muestran los cambios del registro diario y del resumen del mes.

Figura 3.77. Resumen de registros mensuales después de aceptar una justificación

Luego, se rechazó la justificación. Al pulsar el botón respectivo para rechazar la justificación, se mostró una ventana modal en la que se deberá ingresar el motivo por el rechazo de la justificación. En la Figura 3.78 se muestra la ventana modal.

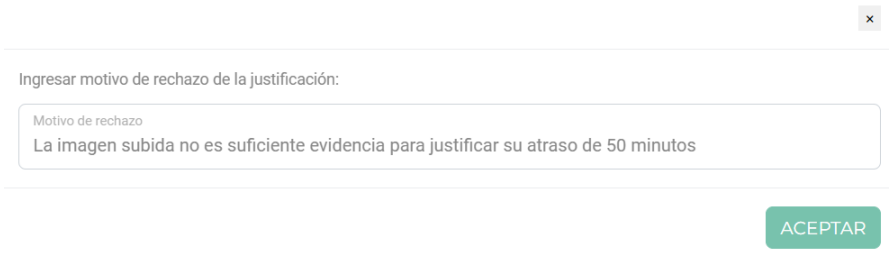


Figura 3.78. Ejemplo de rechazo de justificación

En la Figura 3.79 se puede observar que el estatus de la justificación ha cambiado a rechazado.

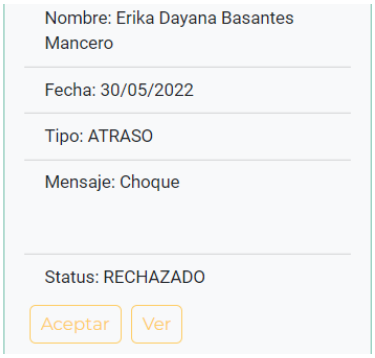


Figura 3.79. Estatus de la justificación se cambia a Rechazado.

En la Figura 3.80 se muestran los cambios del resumen del mes después de rechazar la justificación. Al rechazar una justificación, se contabilizará el atraso en el registro mensual.

Figura 3.80. Resumen de registros mensuales después de rechazar una justificación.

3.3. ANÁLISIS DE RESULTADOS

Se realizaron pruebas del prototipo con los usuarios para realizar el análisis de resultados mediante cuestionarios. Estas pruebas se realizaron a 2 usuarios con rol Administrador y 5 usuarios con rol Empleado.

Para los usuarios de rol Administrador, se seleccionó el personal administrativo de la institución y para usuarios con rol Empleado, se seleccionó parte del personal docente, de salud y cocina de la institución.

Tabla 3.9. Tareas realizadas por rol de usuario.

Rol de Usuario	Actividades
Administrador	• Instalación de la aplicación móvil (Módulo Móvil) • Iniciar sesión en el Módulo Web y Móvil. • Navegar por las interfaces del Módulo Web y Móvil • Agregar usuarios desde el Módulo Web y Móvil • Agregar plantillas de usuarios en el Módulo de Recolección de Datos • Visualizar usuarios desde el Módulo Web y Móvil • Editar usuarios desde el Módulo Web y Móvil • Actualizar plantillas de huellas digitales de usuarios en el Módulo de Recolección de Datos • Eliminar usuarios desde el Módulo Web y Móvil • Eliminar plantillas de huellas digitales de usuarios en el Módulo de Recolección de Datos • Visualizar. filtrar y buscar registros desde el Módulo Web y Móvil • Visualizar, filtrar y gestionar justificaciones (aceptar y rechazar) desde el Módulo Web y Móvil • Visualizar y modificar preferencias del sistema desde el Módulo Web y Móvil • Modificar contraseña personal • Generar registros desde el Módulo de Recolección de Datos.
Empleado	• Generar registros desde el Módulo de Recolección de Datos. • Instalación de la aplicación móvil (Módulo Móvil) • Iniciar sesión en el Módulo Web y Móvil. • Visualizar. filtrar y buscar registros desde el Módulo Web y Móvil • Generar justificaciones desde el Módulo Web y Móvil.

A cada usuario se le entregó Manual de Usuario (ANEXO G) con la finalidad de que sea capaz de operar el sistema en base a su rol.

En la Tabla 3.9 se muestran las actividades que cada usuario realizó para cumplir con las pruebas de usuario.

Se realizaron cuestionarios al personal que realizó las actividades para analizar el resultado de las pruebas de funcionamiento y evaluar la experiencia del usuario en el manejo del sistema. Se crearon dos cuestionarios, el primero se dirigió únicamente a los usuarios con rol Administrador y el segundo fue a todos los usuarios registrados en el sistema. Las preguntas realizadas en estos cuestionarios se encuentran adjuntos en el ANEXO I.

Tabla 3.10. Resultado del cuestionario realizado a los usuarios.

Preguntas	**Resultados**		
	Si	**No**	**Observaciones**
¿El Manual de Usuario fue de utilidad para facilitar la realización de las actividades solicitadas?	91.7%	8.3%	Una de las personas mencionó que las imágenes del manual estaban muy pequeñas
¿Se debería realizar algún cambio en el Manual de Usuario?	16.7%	83.3%	Dos usuarios mencionaron que se podrían cambiar unas imágenes en las cuales no se visualiza bien su finalidad
¿Pudo generar sus registros de asistencia con facilidad?	100%	0%	Todos los usuarios no tuvieron dificultades al generar un registro de asistencia en el Módulo de Recolección de Datos
¿Pudo visualizar sus registros de asistencia con facilidad en la interfaz web?	100%	0%	Los usuarios mencionaron que pudieron ver con facilidad los registros en la interfaz web
¿Pudo visualizar sus registros de asistencia con facilidad en la aplicación móvil?	91.7%	8.3%	Un usuario mencionó que le resultaba incómodo no visualizar la información detallada del registro
¿Pudo generar justificaciones con facilidad en la interfaz web?	91.7%	8.3%	Un usuario mencionó que tuvo dificultad en identificar cuando un archivo de justificación se subió
¿Pudo generar justificaciones con facilidad en la aplicación móvil?	91.7%	8.3%	Una de las personas indicó que no podía encontrar el documento en el explorador de archivos de su celular
¿Encontró algún error en la interfaz web?	75%	25%	Los usuarios mencionaron que algunas ventanas modales no se cerraban
¿Encontró algún error en la aplicación móvil?	75%	25%	Los usuarios mencionaron que algunas ventanas modales no se cerraban
¿Está conforme con el funcionamiento, desempeño y aplicación de este prototipo en la institución?	100%	0%	Los usuarios están de acuerdo con la implementación del prototipo ya que les permite visualizar sus registros en cualquier momento

El resultado del cuestionario realizado al personal se muestra en la Tabla 3.10 y en la Tabla 3.11 se muestra el resultado de las preguntas realizadas a los usuarios con rol Administrador.

Tabla 3.11. Resultado del cuestionario realizado a los usuarios con rol Administrador.

Preguntas	Resultados		
	Si	No	Observación
¿Pudo gestionar los usuarios con facilidad?	50%	50%	Uno de los dos usuarios administradores mencionó que tuvo dificultades con la actualización de una plantilla de usuario ya que al momento que quiso realizar este proceso el Módulo no contaba con conexión a internet.
¿Pudo gestionar los registros con facilidad?	100%	0%	Los usuarios pudieron visualizar los registros con facilidad
¿Pudo gestionar las justificaciones con facilidad?	100%	0%	Los usuarios mencionan que es fácil aceptar o rechazar una justificación

Tabla 3.12. Análisis de los resultados de las pruebas.

Observación	Análisis
Imágenes pequeñas en el manual de usuario	Se determina que se debe aumentar de tamaño las imágenes del manual de usuario. Además, se editarán las imágenes para indicar al usuario donde debe pulsar.
Un usuario mencionó que le resultaba incómodo no visualizar la información detallada del registro	Esto se debe al tamaño reducido en el dispositivo móvil. Para esto no se realizó ningún cambio. Se aconsejó acceder desde un navegador.
Un usuario mencionó que tuvo dificultad en identificar cuando un archivo de justificación se subió	Se ha realizado un cambio en la interfaz de nueva justificación. Se muestra una ventana modal cuando el archivo se ha subido al *Storage* de Firebase.
Una de las personas indicó que no podía encontrar el documento en el explorador de archivos de su celular	Para esto, no se realizó ningún cambio en el sistema. Se enseñó a la persona como acceder al sistema de archivos de su celular para poder identificar el archivo de justificación.
Los usuarios mencionaron que algunas ventanas modales no se cerraban	Se revisaron todas las instancias de las ventanas modales verificando su estructura y su correcto funcionamiento.
Uno de los dos usuarios administradores mencionó que tuvo dificultades con la actualización de una plantilla de usuario ya que al momento que quiso realizar este proceso el Módulo no contaba con conexión a Internet.	Se mostró un mensaje de “Sin conexión a Internet” cuando se pueda leer las opciones del documento preferencias/sistema desde el Módulo de Recolección de Datos por falta de conexión a Internet
Los usuarios están de acuerdo con la implementación del prototipo ya que les permite visualizar sus registros en cualquier momento	Se determina que el prototipo ha resuelto el problema de falta de control de asistencia en la institución.

El análisis de los problemas y novedades del sistema se muestran Tabla 3.12.

Con esto, se puede concluir que el sistema ha podido satisfacer la falta de control de asistencia y horario laboral en la institución. Se ha brindado un sistema que digitaliza los registros y permite visualizarlos con un navegador o desde un dispositivo Android con conexión a internet.

4. CONCLUSIONES Y RECOMENDACIONES

4.1. CONCLUSIONES

En este trabajo de titulación, se ha desarrollado un prototipo de sistema de control de asistencia mediante registro biométrico por huella digital para facilitar el registro de asistencia y la contabilidad de horas extra mediante justificaciones, cumpliendo con los objetivos planteados al inicio del proyecto. Para la implementación de este sistema, se adquirieron conocimientos del framework Angular y Ionic para el desarrollo de las interfaces visuales de cada Módulo, además del uso de los servicios de Firebase para dar funcionalidad al prototipo.

En base al desarrollo e implementación del prototipo, se puede concluir lo siguiente:

- Mediante las entrevistas realizadas, tanto al personal administrativo como al personal docente y de apoyo, se pudo plantear los requerimientos del prototipo y así se obtuvo una idea clara de las funciones con las que debía contar. Esto fue de gran ayuda en el momento de evaluar el correcto funcionamiento de cada Módulo involucrado en el sistema.

- Gracias a los estudios realizados de los componentes, dispositivos y herramientas involucradas en cada uno de los Módulos, se pudo analizar sus características y así orientar su funcionamiento a la aplicación del prototipo de un sistema de control de acceso.

- La codificación de las funciones en la placa de desarrollo ESP-32 brindan operabilidad al Módulo de Recolección de Datos. La codificación se realizó en base a librerías y bibliotecas de código abierto. Fue importante aprender y entender cómo trabajan las funciones de cada librería orientadas a una finalidad en específico, por ejemplo, las funciones implementadas en la lectura del sensor de temperatura MLX90614.

- La placa de desarrollo ESP-32 ha sido una gran opción para proyectos electrónicos a raíz de sus características de procesamiento y conectividad. Razón por la cual, su comunidad ha estado desarrollando librerías compatibles con módulos externos, como por ejemplo los módulos de Arduino. Con esto, la versatilidad de esta placa aumenta y permite acoplarse casi a cualquier módulo, dispositivo o sensor disponible en el mercado. Tal es el caso de la librería *Firebase_ESP_Client.h* la cual permite acceder a los servicios de Firebase de una forma simplificada y fácil,

como por ejemplo crear un documento en *Cloud Firestore* mediante variables estructuradas en formato *JSON* y funciones definidas en su librería. Así mismo, la librería *WifiManager.h* despliega servicio web que redirigirá a un portal de configuración en el cual se detectaran redes disponibles para que la placa se pueda conectar.

- Gracias al lector de huellas digitales, se pudo construir un sistema biométrico capaz de identificar al personal de la institución. Este lector utiliza procesamiento de señal digital para procesar imágenes digitales internas. También incluye la capacidad de administrar y actualizar su base de datos interna. El dispositivo se comunica mediante el protocolo *UART* o serial, brindando alta compatibilidad para que pueda usarse con cualquier microcontrolador o placa de desarrollo.
- Se pudo comprobar que el sensor de temperatura utilizado (MLX90614) permitió medir la temperatura corporal, sin embargo, es importante considerar la distancia a la que se encuentra la persona, ya que se puede aumentar el error en la medida.
- Firebase es una tecnología de Google que proporciona herramientas para potenciar la funcionalidad aplicaciones web y móviles. Gracias a su alta integración con varias tecnologías, se pudo enlazar Firebase con la placa de desarrollo ESP-32 para consumir los servicios que ofrece Google en un dispositivo portable. Se puede usar la placa ESP32 para conectarse e interactuar con proyectos de Firebase, esto implica que se puede controlar el ESP32 con Firebase desde Internet.
- Para este tipo de aplicación, la segmentación del personal en dos roles fue suficiente, ya que se cumple con las funciones del sistema. Así se dirige la información recolectada al usuario deseado. Es decir, se permite visualizar y gestionar los datos al usuario administrador y limitar las funciones al usuario empleado.
- Durante la fase de implementación, se logró producir un sistema funcional en el que intervienen cada uno de los elementos estudiados. Con esto, se demostró la fácil integración entre: Angular y Firebase, en el caso del Módulo Web; además de Ionic y Capacitor, en el caso del Módulo Móvil y, por último, la placa de desarrollo con los diferentes sensores y dispositivos utilizados, en el caso del Módulo de Recolección de Datos. Llegando a la conclusión de que este sistema es muy versátil y puede acoplarse a diversas aplicaciones.
- Mediante las pruebas de funcionamiento, se pudo verificar la correcta operación de cada Módulo de forma independiente y en conjunto. Se verificó el cumplimiento de

los objetivos y los requerimientos del sistema, con esto, se logró producir un prototipo a las necesidades del usuario.

- Al realizar las pruebas de funcionamiento, se determinaron algunos de los factores que influyen en el rendimiento del sistema. Uno de estos es la calidad de conexión entre el punto de acceso y el Módulo de Recolección de Datos. Fue influyente el ubicar el Módulo en un lugar en donde se cuente con una buena conexión a Internet, ya que esto reduce los retardos por subida de registros y así disminuir la cantidad de tiempo que el usuario espera al ser notificado que el registro se subió con éxito.
- La continua toma de registros mediante huellas digitales causa que el lector se llegue a opacar y ensuciar debido al sudor o humedad en los dedos. Esto origina errores en la lectura de huellas digitales y la identificación del personal, reduciendo así su rendimiento.
- Durante las pruebas, los usuarios aprendieron rápidamente cómo usar los Módulos gracias a las actividades solicitadas; estas se realizaron de manera correcta y precisa. Con esto, la fase de pruebas finalizó con éxito brindando un sistema listo para empezar funcionar.
- Gracias al cuestionario final, dirigido al personal registrado en el sistema, se evidencio que los usuarios están conformes con el prototipo mencionando que es necesario su aplicación en la institución.

En conclusión, la implementación del prototipo de sistema de control de asistencia ayudará a monitorear el horario laboral del personal de forma centralizada, teniendo en cuenta su asistencia, las horas extra, las horas de trabajo y la gestión de justificaciones. Además, brindará a todos los empleados la oportunidad de acceder a sus registros y monitorear sus horas de trabajo.

4.2. RECOMENDACIONES

- A pesar de que el Módulo de Recolección de Datos cuente con la función de reinicio automático por perdida de conexión Internet, es recomendable realizar un reinicio manual del Módulo al momento que se desee reconectar a la red Wifi.
- Se recomienda crear una función que permita subir una foto de perfil de cada usuario registrado en el sistema y almacenarlo en *Firebase Storage*. Esto con la finalidad de complementar el perfil de usuario.
- Es recomendable realizar pruebas minuciosas del sensor de temperatura. Esto con el fin de reducir el error de medición. Para esto, se recomienda modificar el valor de

la proporción de radiación térmica emitida por la superficie de la piel (*emissivity*) en el sensor de temperatura.

- Se recomienda, como trabajo a futuro, realizar un monitoreo en tiempo real del Módulo de Recolección de Datos mediante los servicios de Firebase. Con esto, se sabrá en el Módulo Web/Móvil cuando el Módulo de Recolección de Datos haya perdido conexión a Internet.
- En el caso de que se desarrolle un proyecto basado en este prototipo, revisar las versiones de las herramientas usadas en cada uno de los Módulos.
- Se recomienda construir una aplicación móvil híbrida capaz de ser compatible con el sistema operativo iOS y Android.
- En trabajos futuros, se recomienda aplicar el manejo de notificaciones en el Módulo Móvil.

5. REFERENCIAS BIBLIOGRÁFICAS

[1] Dragos-Paul Pop, Adam Altar, Designing an MVC Model for Rapid Web Application Development, Romanian-American University, Bucharest, Romania: Elsevier Ltd, 2014.

[2] Jon Galloway, Brad Wilson, K. Scott Allen, David Matson, Professional ASP.NET MVC 5, Indianapolis, Indiana. US: John Wiley & Sons, Inc., 2014.

[3] J. Tamariz, «Sistemas de Control de Asistencia,» EuroSoft, 26 Noviembre 2019. [En línea]. Available: https://solucioneseurosoft.com/tiempo-y-asistencia-news/sistemas-de-control-de-asistencia/. [Último acceso: 06 Abril 2022].

[4] CDI-CI, «Centro de Desarrollo Infantil "Cielo Infantil",» 2020. [En línea]. Available: http://www.cieloinfantil.com/index.php/quienes-somos. [Último acceso: 05 18 2021].

[5] E. Systems, «ESP32 (A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide-range of applications),» 2021. [En línea]. Available: https://www.espressif.com/en/products/socs/esp32. [Último acceso: 10 06 2021].

[6] E. Systems, «ESP32-DevKitC V4 Getting Started Guide,» 2019. [En línea]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html. [Último acceso: 10 06 2021].

[7] E. Systems, «SoCs (Espressif offers integrated, reliable and energy-efficient wireless SoCs),» 2020. [En línea]. Available: https://www.espressif.com/en/products/socs. [Último acceso: 10 06 2021].

[8] E. Systems, «ESP32 Series of Modules,» 2020. [En línea]. Available: https://www.espressif.com/en/products/modules/esp32. [Último acceso: 10 06 2021].

[9] E. System, «Development Boards,» 2020. [En línea]. Available: https://www.espressif.com/en/products/devkits. [Último acceso: 10 06 2021].

[10] E. Systems, «Datasheet: ESP32WROOM32E & ESP32WROOM32UE,» 2021. [En línea]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32e_esp32-wroom-32ue_datasheet_en.pdf. [Último acceso: 10 6 2021].

[11] L. Diéguez, «COMUNICACIONES SERIE EN ARDUINO: UART, I2C Y SPI,» Blog Kolwidi, 27 05 2019. [En línea]. Available: https://kolwidi.com/blogs/blog-kolwidi/comunicaciones-serie-en-arduino-uart-i2c-y-spi. [Último acceso: 10 06 2021].

[12] Jayant Mankar, Chaitali Darode, Komal Trivedi, Madhura Kanoje, Prachi Shahare, «REVIEW OF I2C PROTOCOL,» *International Journal of Research in Advent Technology,* vol. 2, nº 1, p. 474, 2014.

[13] F. Leens, «An Introduction to I2 and SPI Protocols,» *IEEE Instrumentation & Measurement Magazine ,* vol. 1, nº 1, p. 13, 2009.

[14] M. Santos, «Sensores de huellas dactilares,» Blog: HardZone, 20 4 2018. [En línea]. Available: https://hardzone.es/2018/04/20/sensores-huellas-dactilares/. [Último acceso: 10 6 2021].

[15] Biometricos.net, «Sensores de huellas dactilares,» 04 2018. [En línea]. Available: https://www.biometricos.net/2018/04/sensores-de-huellas-dactilares-como.html. [Último acceso: 10 06 2021].

[16] Adafruit, «Librería Adafruit Fingerprint Sensor (Documentación),» 12 Octubre 2021. [En línea]. Available: https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library/tree/master/documentation. [Último acceso: 07 Abril 2022].

[17] M. M. I. Systems, «Datasheet: MLX90614 family (Infra Red Thermometer),» [En línea]. Available: https://www.sparkfun.com/datasheets/Sensors/Temperature/MLX90614_rev001.pdf. [Último acceso: 10 06 2021].

[18] ElectroPeak, «SD Card Module with Arduino,» ARDUINO (Project Hub), 16 09 2019. [En línea]. Available: https://create.arduino.cc/projecthub/electropeak/sd-card-module-with-arduino-how-to-read-write-data-37f390. [Último acceso: 10 06 2021].

[19] Electropeak, «Micro SD TF Card Memory Shield Module,» 2019. [En línea]. Available: https://electropeak.com/micro-sd-tf-card-adapter-module. [Último acceso: 10 06 2021].

[20] J. G. Cobo, «Las pantallas LCD y Arduino,» 03 07 2018. [En línea]. Available: https://www.hwlibre.com/pantallas-lcd-arduino/. [Último acceso: 10 06 2021].

[21] HITACHI, «Datasheet: HD4478U,» 2018. [En línea]. Available: https://pdf1.alldatasheet.com/datasheet-pdf/view/63663/HITACHI/HD44780U.html. [Último acceso: 10 06 2021].

[22] B. naylampmechatronics, «LCD CON I2C, CONTROLA UN LCD CON SOLO DOS PINES,» Blog naylampmechatronics.com, 2016. [En línea]. Available: https://naylampmechatronics.com/blog/35_tutorial-lcd-con-i2c-controla-un-lcd-con-solo-dos-pines.html. [Último acceso: 10 06 2021].

[23] T. naylampmechatronics, «DISPLAY OLED 0.96" I2C 128*64 SSD1306,» 2019. [En línea]. Available: https://naylampmechatronics.com/oled/850-display-oled-096-i2c-12864-ssd1306.html. [Último acceso: 10 06 2021].

[24] Mike Cantelon, Marc Harter, T.J. Holowaychuk, Nathan Rajlich, Node.js in Action, Shelter Island, NY 11964: Manning Publications Co., 2014.

[25] B. A. Syed, Beginning Node.js, Spring Street, NY 10013: Springer Science+Business Media New York, Apress, 2014.

[26] Angular, «Introduction to the Angular Docs,» Angular, 2022. [En línea]. Available: https://angular.io/docs. [Último acceso: 25 07 2022].

[27] Angular.io, «Documentation: What is Angular?,» 08 03 2021. [En línea]. Available: https://angular.io/guide/what-is-angular. [Último acceso: 10 06 2021].

[28] Firebase, «Documentación de FIrebase,» 2021. [En línea]. Available: https://firebase.google.com/docs. [Último acceso: 10 06 2021].

[29] Firebase, «Documentación de FIrebase: Cloud Firestore,» 2021. [En línea]. Available: https://firebase.google.com/docs/firestore. [Último acceso: 10 06 2021].

[30] Firebase, «Documentación: Modelo de datos de Cloud Firestore,» 2021. [En línea]. Available: https://firebase.google.com/docs/firestore/data-model. [Último acceso: 10 06 2021].

[31] Firebase, «Documentación de Firebase: FIrebase Authentication,» 2021. [En línea]. Available: https://firebase.google.com/docs/auth. [Último acceso: 10 06 2021].

[32] Firebase, «Documentación de Firebase: Firebase Hosting,» 2021. [En línea]. Available: https://firebase.google.com/docs/hosting. [Último acceso: 10 06 2021].

[33] Firebase, «Cloud Storage,» Firebase, 2022. [En línea]. Available: https://firebase.google.com/products/storage. [Último acceso: 06 05 2022].

[34] V. S. Code, «Documentation for Visual Studio Code: Getting Started,» 2020. [En línea]. Available: https://code.visualstudio.com/docs. [Último acceso: 10 06 2021].

[35] Arduino, «Entorno de Desarrollo Integrado para Arduino,» [En línea]. Available: https://www.arduino.cc/en/software. [Último acceso: 07 Abril 2022].

[36] M. Netkow, «Overview: Ionic Framework - Angular,» Ionic Framework, 06 06 2019`. [En línea]. Available: https://ionicframework.com/docs/angular/overview. [Último acceso: 10 06 2021].

[37] C. CLI, «Capacitor: Cross-platform Native Runtime for Web Apps,» Capacitor CLI, 2022. [En línea]. Available: https://capacitorjs.com/docs. [Último acceso: 25 07 2022].

[38] D. d. Android, «Introducción a Android Studio,» Android, [En línea]. Available: https://developer.android.com/studio/intro. [Último acceso: 08 Abril 2022].

[39] I. Sommerville, «Requerimientos del software,» Universidad Veracruzana, 08 2015. [En línea]. Available: https://www.uv.mx/personal/fcastaneda/files/2015/08/F_Capitulo_5_Requerimientos_del_software.pdf. [Último acceso: 17 06 2021].

[40] Arduino.CC, «Downloads Arduino IDE,» Arduino.CC, 2022. [En línea]. Available: https://www.arduino.cc/en/software. [Último acceso: 26 07 2022].

[41] NodeJS, «Descargas de NodeJS,» NodeJS, 2022. [En línea]. Available: https://nodejs.org/es/download/. [Último acceso: 26 07 2022].

[42] Microsoft, «Download Visual Studio Code,» Microsoft, 2022. [En línea]. Available: https://code.visualstudio.com/download. [Último acceso: 26 07 2022].

[43] A. Docs, «Instalar la CLI de Angular,» Angular CLI, 2022. [En línea]. Available: https://docs.angular.lat/guide/setup-local. [Último acceso: 26 07 2022].

[44] I. Docs, «Installing Ionic,» Ionic Framework, 2022. [En línea]. Available: https://ionicframework.com/docs/intro/cli. [Último acceso: 26 07 2022].

[45] A. Studio, «Download Android Studio & App Tools,» Android, 2022. [En línea]. Available: https://developer.android.com/studio. [Último acceso: 26 07 2022].

[46] C. Docs, «Installing Capacitor,» CapacitorJS, 2022. [En línea]. Available: https://capacitorjs.com/docs/getting-started. [Último acceso: 26 07 2022].

[47] VISHAY, «Datasheet: OLED-128O064D-BPP3N00000,» 01 01 2021. [En línea]. Available: https://www.vishay.com/docs/37902/oled128o064dbpp3n00000.pdf. [Último acceso: 10 06 2021].

[48] Firebase, «Documentación de FIrebase: Cloud Functions para Firebase,» 2021. [En línea]. Available: https://firebase.google.com/docs/functions. [Último acceso: 10 06 2021].

[49] CDI-CI, «Centro de Desarrollo Infantil "Cielo Infantil",» 2020. [En línea]. Available: http://www.cieloinfantil.com/index.php/quienes-somos.

[50] Adafruit, «Adafruit-Fingerprint-Sensor-Library,» Adafruit, 12 Octubre 2021. [En línea]. Available: https://github.com/adafruit/Adafruit-Fingerprint-Sensor-Library. [Último acceso: 09 Febrero 2022].

6. ANEXOS

ANEXO A. **ENCUESTA REALIZADA A LA MÁXIMA AUTORIDAD DE LA INSTITUCIÓN Y A 5 EMPLEADOS**

ANEXO B. **CÓDIGO DEL MÓDULO DE RECOLECCIÓN DE DATOS**

ANEXO C. **DESCRIPCIÓN DE LAS FUNCIONES DEL MÓDULO DE RECOLECCIÓN DE DATOS**

ANEXO D. **CÓDIGO DEL MÓDULO WEB**

ANEXO E. **CÓDIGO DEL MÓDULO MÓVIL**

ANEXO F. **APLICACIÓN MÓVIL INSTALABLE DEL MÓDULO MÓVIL**

ANEXO G. **MANUAL DE USUARIO**

ANEXO H. **PROGRAMAS PARA LA VERIFICACIÓN DEL FUNCIONAMIENTO DEL MÓDULO DE RECOLECCIÓN DE DATOS**

ANEXO I. **CUESTIONARIOS REALIZADOS A LOS USUARIOS DEL SISTEMA**