ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE FORMACIÓN DE TECNÓLOGOS

DESARROLLO DE APLICACIÓN MÓVIL Y WEB PARA GESTIÓN ASISTENCIA DE EMPLEADOS EN EMPRESAS EN CRECIMIENTO

BACKEND

TRABAJO DE INTEGRACIÓN CURRICULAR PRESENTADO COMO REQUISITO PARA LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO SUPERIOR EN DESARROLLO DE SOFTWARE

LIZBETH LEONELA GARCIA TIRIRA

DIRECTORA: ING. MAYRA ISABEL ALVAREZ JIMENEZ

DMQ, septiembre 2022

CERTIFICACIONES

Yo, LIZBETH LEONELA GARCIA TIRIRA declaro que el trabajo de integración curricular 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.

LIZBETH LEONELA GARCIA TIRIRA

lizbeth.garcia@epn.edu.ec

liz.leonela21@gmail.com

Certifico que el presente trabajo de integración curricular fue desarrollado por LIZBETH LEONELA GARCIA TIRIRA, bajo mi supervisión.

ING. MAYRA ISABEL ALVAREZ JIMENEZ

DIRECTORA

mayra.alvarez@epn.edu.ec

DECLARACIÓN DE AUTORÍA

A través de la presente declaración, afirmamos que el trabajo de integración curricular aquí descrito, así como el (los) producto(s) resultante(s) del mismo, son públicos y estarán a disposición de la comunidad a través del repositorio institucional de la Escuela Politécnica Nacional; sin embargo, la titularidad de los derechos patrimoniales nos corresponde a los autores que hemos contribuido en el desarrollo del presente trabajo; observando para el efecto las disposiciones establecidas por el órgano competente en propiedad intelectual, la normativa interna y demás normas.

LIZBETH LEONELA GARCIA TIRIRA

DEDICATORIA

El presente proyecto se lo dedico a mi bella madre, quien me supo guiar en cada etapa de esta carrera, quien me dio el ánimo suficiente para no decaer en el trayecto, por confiar en mis capacidades cuando me hacía falta un empujoncito y por brindarme todas las herramientas suficientes para culminar una etapa más en mi vida profesional. Es gratificante poder culminar una meta más y cumplir con una expectativa más que mi madre tiene hacia mí.

LIZBETH LEONELA GARCIA TIRIRA

AGRADECIMIENTO

Agradezco a todas las personas que de cierta forma me han acompañado en esta pequeña travesía, a mis compañeros quienes alegraron mis días y me brindaron el apoyo cuando lo necesitaba, a mis profesores por motivarme a seguir con la carrera y también a la “Escuela Politécnica Nacional” por cumplir con mi educación y dejarme ser parte de tan bonita experiencia.

LIZBETH LEONELA GARCIA TIRIRA

ÍNDICE DE CONTENIDO

RESUMEN

El presente proyecto hace referencia al desarrollo del componente Backend cuyo objetivo es llevar un control de horas y actividades de trabajo de los empleados de empresas en crecimiento, esto quiere decir que la lógica del sistema web y móvil cumple con registrar usuarios con diferentes roles, tales como administrador, gerente o empleado, según corresponda. Los tres usuarios pueden iniciar sesión a través de un correo electrónico y contraseña desde la interfaz web, pero solo los usuarios con rol empleado tienen acceso a la interfaz móvil, en ambos casos deben verificar su correo para empezar con el flujo del sistema. Además, se debe mencionar que cada uno de los usuarios tienen la capacidad de modificar su información personal y en caso del usuario con rol empleado, tiene la función para registrar su actividad laboral según: hora de inicio, hora de receso, hora final del receso y hora final de la jornada. La lógica del sistema web y móvil realiza el cálculo diario y mensual de horas que los usuarios con rol empleado han trabajado.

Los usuarios tienen la opción de realizar consultas, ingreso o actualización de registros de la base de datos acorde a la necesidad establecida en la recopilación de requerimientos. La información reposa en una base de datos no relacional que maneja colecciones y documentos, que proporciona Firestore Database. Los archivos o imágenes se almacenan en el Storage y la gestión de usuarios se establece en Authentication de Firebase, plataforma de Google.

PALABRAS CLAVE: Firebase, Firestore, Empleados, Jornada laboral.

ABSTRACT

This project refers to the development of the Backend component, the objective is to keep track of hours and work activities of employees of growing companies. This means that the logic of the web and mobile system complies with register users with different roles, such as administrator, manager or employee, as appropriate. All three users can login via email and password from the web interface but only users with employee role have access to the mobile interface. In both cases, users must verify their email to start with the system flow. In addition, it should be mentioned that each of the users has the ability to modify their personal information and in the case of the user with employee role, this user has the function to record their work activity according to: start time, break time, end break time and end time of the day. The logic of the web and mobile system performs the daily and monthly calculation of hours that users with the employee role have worked.

Users have the option to make inquiries, to enter or to update database records, this according to the need established in the compilation of requirements. The information rests in a non-relational database that handles collections and documents, which are provided by the Firestore Database. The files or images are saved in the Storage and user management is set to Firebase Authentication, Google Platform.

KEYWORDS: Firebase, Firestore, employees, working hours.

1 DESCRIPCIÓN DEL COMPONENTE DESARROLLADO

Las pequeñas empresas tienen la costumbre de usar el software de hojas de cálculo, Excel, para ingresar información personal de sus empleados, controlar las horas de trabajo e incluso la usan para llevar la contabilidad. Este método es factible hasta que la empresa inicia con el crecimiento operacional, de productos, de personal, de líneas de negocio o de clientela [1].

Una empresa que inicia su etapa de crecimiento debe optar por invertir en recursos tecnológicos, este tipo de recursos se adaptan a las empresas pequeñas según sus necesidades y logra resultados similares a los que gozan grandes corporaciones, tales como: buen posicionamiento ante la competencia, agrega valor a la entidad, sube la participación en el mercado, transforma el proceso de negocio y reduce costos [2].

En este caso el problema a resolver es cuando el personal de una empresa empieza a incrementar y es necesario que haya un encargado quien gestione la información de cada empleado, controle el horario de entrada y salida, los recesos entre horas, los permisos para faltar al trabajo, las multas por llegar tarde a la jornada laboral.

La mejor elección ante el incremento de empleados es optar por un software de recursos humanos (Software ERP, Enterprise Resource Planning), esta es una herramienta tecnológica para gestionar personal donde su principal función es administrar y planificar la gestión del personal [3]. Según el tamaño de la empresa se escoge un tipo de software ERP, es decir, para compañías grandes por ejemplo se usa SAP o Microsoft Dynamics. Sin embargo, si el negocio se identifica como pequeña empresa o en crecimiento podría optar por un ERP en la nube, como es el caso de Open ERP [4]. Tomando en cuenta el punto de vista técnico y funcional, la herramienta mencionada, se encarga de la planificación y gestión de recursos empresariales [5]. El presente proyecto integra dos módulos acoplados para una empresa en crecimiento: gestión de empleados y gestión de horas laboradas.

Para iniciar con un software ERP, es fundamental el desarrollo correcto del Backend, este es el encargado de la lógica del sistema y es necesario para que la aplicación web y móvil tenga buena conexión con la base de datos, de esta manera el intercambio de información no genera conflictos y así el usuario final tiene una buena experiencia con la navegación dentro del sistema.

Ante la problemática expuesta, se elige el desarrollo de Backend como servicio (BaaS, Backend as a Service) porque este ya ofrece módulos pre desarrollados para que se

puedan usar mediante algunas funciones específicas y así manejar correctamente la interfaz del usuario. De esta manera el desarrollo de Frontend es mejor estilizado y acorde a las necesidades del cliente, los costos son menores, la lógica del sistema es más manejable y permite un desarrollo de Backend en menos tiempos [6].

Como se menciona anteriormente, la nube es el mejor lugar para almacenar información de empresas que seleccionan nuevas herramientas tecnológicas. Por lo tanto, Firebase es el Backend como servicio y se define como una base de datos no SQL donde se puede almacenar, acceder y sincronizar los datos al instante, sin la necesidad de ocupar grandes servidores que ejecutan código de Backend [7].

1.1 Objetivo general

Desarrollar el Backend para el control de empleados de empresas en crecimiento acorde a las necesidades del cliente.

1.2 Objetivos específicos

1. OE1 Identificar las necesidades y requerimientos del sistema para estructurar correctamente la lógica del Backend.
2. OE2 Diseñar la base de datos no SQL del sistema para usar estructuras JSON con clave asociada.
3. OE3 Implementar Firebase Authentication en el sistema web y móvil para que el usuario pueda iniciar sesión como administrador, gerente o empleado.
4. OE4 Implementar consultas a la base de datos no relacional de Firebase para que el usuario interactúe con el sistema recibiendo y enviando información relevante.

1.3 Alcance

Las empresas en crecimiento de personal necesitan un software ERP para ingresar la información de sus empleados, cargar documentos importantes, como hoja de vida o documento de identidad, registrar hora de ingreso a la jornada laboral, la hora de salida, el receso para la comida, las vacaciones aprobadas y el estado dentro de la empresa, es decir, activo o inactivo.

La sección del Backend propone el uso de la API REST para el inicio de sesión de los empleados y administración, usando la herramienta de Firebase Authentication. Para que un empleado tenga acceso al sistema, el gerente, encargado de recursos humanos, es

quien genera una cuenta con estado activo para que el empleado pueda acceder desde la interfaz web o móvil con las credenciales mencionadas y desde su dispositivo puede registrar su hora de entrada, hora de receso y salida del trabajo. Este intercambio de información se refleja en el sistema web dirigido para el gerente de la empresa, los empleados tienen el acceso a su información desde la página web y el sistema móvil donde podrá registrar las horas y actividades de trabajo. El usuario con rol de administrador es el encargado de gestionar la página web.

En la interfaz web se visualiza la lista de empleados, su estado dentro del trabajo, las horas trabajadas y las opciones de crear, actualizar o eliminar registros en caso de roles administrador o gerente, además de la opción de aceptar o denegar los permisos del empleado para ausentarse del trabajo. El acceso a la base de datos y administración de la misma se da por las peticiones de solicitudes REST y los métodos set, get, update o remove, según corresponda.

1.4 Marco Teórico

Para un mejor entendimiento del contenido del presente trabajo a continuación mencionamos de forma general conceptos clave que involucran el desarrollo del Backend del sistema de control de empleados propuesto.

Metodología

Las metodologías en el desarrollo de software son útiles para trabajar en equipo y se pueda obtener una correcta organización y distribución de trabajo, eso hace que el desarrollo sea eficaz y también productivo. Estas metodologías son la combinación de métodos y técnicas para diseñar soluciones alternas tomando en cuenta los distintos factores que implica el desarrollo de un producto, de esta manera un equipo puede distribuirse las funciones de manera más ordenada y equilibrada. En el caso de que no esté clara la metodología al momento de desarrollar un sistema, este se tornará más complejo y tendrá muchos conflictos y errores, lo que causará un mal resultado [8].

Las metodologías se consideran un pilar para poder iniciar con un proyecto de software de carácter profesional, de esta manera reflejan capacidad de los desarrolladores y la experiencia de los que están al mando. Al usarlas la documentación refleja el proceso y el resultado puede ser justificado sin ningún problema. El conocimiento de las mismas logra un correcto levantamiento de información y un eficaz desarrollo del sistema. La metodología escogida es capaz de comprender el problema, hallar distintas soluciones,

aplicarlas y realizar las pruebas respectivas para solucionar posibles errores y completar con la calidad, elemento indispensable al momento de la entrega final del producto [9].

Actualmente se distinguen dos tipos de metodologías: las metodologías tradicionales y las metodologías agiles.

Metodología Ágil

Para comprender la diferencia entre metodologías agiles y tradicionales, se realiza un cuadro comparativo donde se mencionan sus características principales, véase Tabla I.

Tabla I. Cuadro comparativo entre metodología ágil y tradicional [10].

Metodología ágil	Metodología tradicional
Son susceptibles al cambio y a las habilidades del equipo.	Son rígidas y con documentación exhaustiva.
El objetivo es generar buena relación con el usuario final y este colabore con el grupo de desarrollo.	El objetivo es cumplir con el proyecto que se define al inicio del desarrollo.
El equipo de desarrollo es el factor principal para establecer las necesidades.	Los ciclos se establecen en cascada.
La documentación debe tener valor directamente con el producto final.	El proceso debe seguir según la planificación.
Se valora la capacidad de respuesta ante los cambios.	Se entrega el producto tal cual se establece al inicio de la planificación, no tolera los cambios.

Para aumentar la probabilidad de éxito en el desarrollo de software generalmente se utilizan metodologías ágiles, ya que permite particionar el proyecto de modo que los objetivos se van alcanzando de a poco y según como se vayan dando los avances el desarrollo se va adaptando a los cambios necesarios. Este tipo de metodología ayuda a encontrar errores fácilmente y se resuelven tomando en cuenta al cliente y esto provoca también una rápida retroalimentación. Según la necesidad del cliente se establece prioridades en las funciones, de esta manera el producto se puede lanzar más pronto y el usuario logra un retorno de inversión de forma más temprana. Por lo mencionado, se garantiza un progreso constante. El inconveniente que se puede presentar es al inicio porque no se estima correctamente el tiempo ni el campo económico [11].

En la actualidad existen muchos métodos que comparten la filosofía Agile. En la Tabla II, se mencionan algunas metodologías ágiles:

Tabla II. Metodologías ágiles [12].

Scrum	**Kanban**	**Extreme Programming**
Divide al proyecto en Sprint para ser revisados de forma individual y constante.	Utiliza objetos visuales para conocer el estado del proceso.	El proyecto se divide en fases y se realiza un análisis completo.
Los objetivos son de acuerdo a las necesidades del usuario.	Los objetivos contemplados contienen 3 estados: pendiente, en proceso y realizado.	El objetivo es responder con brevedad al cambio gracias a la comunicación constante.
Cada Sprint presenta u prototipo de producto o servicio.	Las tareas no se designan sin antes haber terminado la anterior.	Las fases mantienen una retroalimentación continua entre el cliente y el equipo de desarrollo.
Desarrolla soluciones de cualquier industria.	Desarrolla soluciones en base al trabajo en equipo y el flujo de tareas.	Desarrolla soluciones para proyectos que son cambiantes.

Metodología Scrum es la seleccionada por que divide al sistema en piezas pequeñas y fáciles de resolver para que haya un avance progresivo y no se acumulen las tareas. De esta manera todo el equipo de trabajo aporta y se organizan mientras están siendo supervisados por un encargado y existe la comunicación con el cliente para resolver conflictos, en caso de que existan.

Backend

El Backend es un sistema que maneja la información de los usuarios u otro tipo de sistemas que se manejan dentro de un mismo entorno empresarial. Un Backend es capaz de soportar aplicaciones de “back office”. El mencionado es el principal elemento de un sistema web y móvil [13].

La lógica y desarrollo de un Backend no es visible para el usuario final y este es consumido a través de la web o una aplicación móvil. Funciona al lado del servidor y genera respuestas ante las peticiones del cliente. Y, el lugar donde se almacena la información recolectada del usuario son las bases de datos, parte importante del Backend [14].

Firebase

Plataforma móvil proveniente de Google, su objetivo es facilitar y desarrollar aplicaciones de alta calidad donde el proceso sea rápido. Esta plataforma se encuentra en la nube y es capaz de detectar errores y cumple la función de testeo [15].

Firebase tiene la capacidad de gestionar los usuarios de las aplicaciones, en este caso, la información del jefe, encargado y los empleados de la empresa en crecimiento. Tiene la posibilidad de añadir nuestros usuarios mediante invitaciones o notificaciones. Tiene una interfaz intuitiva gracias a su API [15].

Firebase cuenta con múltiples servicios como:

- Authentication: tiene servicios de Backend para autenticar usuarios dentro de una app, usa contraseñas, numero de celular o plataformas como Google, Facebook o Twitter.
- Firestore Database: es una base de datos no SQL, es flexible, se encuentra en la nube y sincroniza y almacena datos.
- Realtime Database: la sincronización se da por objetos de escucha en tiempo real con soporte sin conexión a internet
- Storage: es un servicio para subir y descargar objetos grandes como por ejemplo archivos multimedia [7].

2 METODOLOGÍA

La metodología es el proceso que se realiza para tomar ciertas decisiones, estas son objetivas y se eligen para obtener los datos necesarios que se verán reflejados al final del estudio, para ello se usan diferentes herramientas, técnicas y métodos. Este último son todos aquellos procesos para recoger información nueva [16].

El presente trabajo muestra su proceso mediante el método experimental, este es un método empírico producto del ser humano que busca entre lo desconocido usando actividades transformadoras. El experimento forma parte del estudio donde se establece condiciones importantes y necesarias para que el objeto posea propiedades y relaciones útiles en la investigación [17].

Tomando en cuenta lo anterior, el experimento se encarga de separar el problema, es decir, el aumento de empleados de las empresas en crecimiento. Donde, las características de la empresa crean diferentes requerimientos para formar parte de la capa tecnológica. A continuación, se coloca al problema ante soluciones o situaciones alteras y se trata de controlarlas. Si se toma en cuenta la gestión de empleados y la necesidad de controlar el horario de trabajo de cada uno de ellos. El experimento busca un software ERP que se adapte al presupuesto de la empresa o desarrolla un sistema que solo se enfoque en los módulos que la empresa necesita, siempre tomando en cuenta el problema teórico. Por lo tanto, según los requerimientos del problema el experimento se va reproduciendo.

El marco de trabajo que se acopla para el desarrollo de este sistema es Scrum. Esta metodología se basa en ser flexible ante los cambios y los nuevos requerimientos que proponga el cliente durante el desarrollo del sistema. También se caracteriza por ser colaborativo y le da la importancia debida al cliente. Sus pilares son la transparencia, inspección y adaptación. La primera hace referencia a que todos los implicados en el proyecto tienen una visión global del problema y conocen el estado del proyecto, la segunda, se destaca por que existe un seguimiento constante para supervisar el avance y confirmar el flujo del trabajo. La última, si los requerimientos iniciales no son bien definidos la metodología hace que el proyecto se ajuste al cambio [18].

2.1 Metodología de Desarrollo

La metodología Scrum hace que el cliente sienta entusiasmo y logra observar cómo va incrementando el trabajo gracias a la entrega continua de Sprint. Estas son las iteraciones o divisiones que tienen un objetivo pequeño que cumple con una funcionalidad específica que requiere el cliente y esta se clasifica según una prioridad alta, media o baja. Por lo

tanto, esta metodología logra cumplir con expectativas que posee el cliente desde el inicio, mejora la calidad del software por que pasa por diferentes revisiones, mejora la productividad porque los integrantes son se auto organizan y distribuyen equitativamente las tareas, estima mejor los tiempos porque hay entregas continuas de las diferentes funcionalidades [19].

Esta metodología posee un equipo, eventos y artefactos Scrum que se mencionan a continuación.

Roles

El equipo Scrum está conformado por un equipo de desarrollo de 3 hasta 9 personas, adicional el Scrum Master y Product Owner, cada uno con una función y responsabilidad determinada según el rol que cumpla.

- **Product Owner:** Es quien determina el valor del producto, es decir, se encarga de establecer el trabajo importante y focaliza la lógica del negocio. Su objetivo es gestionar presupuestos, contratar al equipo de desarrollo, puede aportar al proyecto y debe tener la capacidad de dar valor a cada Sprint [20]. Este rol lo asume la Ing. Mayra Alvarez quien se encarga de aclarar las ideas de la lógica del negocio.

- **Scrum Master:** Es el mentor quien motiva al equipo de desarrollo que siga el modelo Scrum y también se lo considera como el líder del proyecto, encargado de que los implicados cumplan los objetivos hasta completar el último Sprint [21]. Este rol también lo asume la Ing. Mayra Alvarez quién supervisa la adecuada elaboración de las tareas y funciones que debe cumplir cada individuo. Adicionalmente, se encarga de ayudar a solucionar inconvenientes a lo largo del desarrollo.

- **Developer Team:** En cada entregable, el equipo es responsable de convertir el Product Backlog en un avance con una funcionalidad representativa dentro del producto final, cada integrante debe asignarse tareas, fijar una fecha de entrega y entregar dentro del plazo [22]. El producto final de este proyecto se divide en Backend, Frontend y Aplicación móvil, por lo tanto, el equipo de desarrollo está conformado por 3 estudiantes y cada uno es responsable de su componente, es decir, Lizbeth García, Kevin Veliz y Mateo Vera, respectivamente.

En la Tabla III, se muestra el equipo Scrum establecido para el presente proyecto con los respetivos roles.

Tabla III. Equipo Scrum

Rol	Integrante
Product Owner	Ing. Mayra Alvarez
Scrum Master	Ing. Mayra Alvarez
Developer Team	Srta. Lizbeth García Sr. Kevin Veliz Sr. Mateo Vera

Artefactos

Los artefactos Scrum son todos los elementos que aseguran transparencia y son el registro de evidencia de proceso que se está llevando a cabo, esto afirma la calidad y productividad de este proyecto [23].

Recopilación de Requerimientos

Esta recopilación es el proceso donde se pueden identificar todas las necesidades que posee el negocio y se proponen soluciones alternas que involucra al Product Owner y al Scrum Master con el objetivo de cumplir con la expectativa del cliente [24]. Los requerimientos del sistema se obtienen tras haber realizado una investigación sobre el problema que presentan las empresas en crecimiento y esto se analizó con el equipo de desarrollo para plantear los requerimientos que se presentan en la sección Anexo en la parte de Recopilación de requerimientos, específicamente en la Tabla XI. A continuación, en la Tabla IV, Tabla IV. se pueden observar un ejemplo de los requerimientos.

Tabla IV. Recopilación de requerimientos

ID-RR	Enunciado del ítem
RR-01	Como usuario administrador, necesita una autenticación de rol para iniciar sesión.
RR-02	Como usuario administrador, necesita registrar usuarios con roles empleado o gerente para llevar un registro de los mismos.
RR-03	Como usuario administrador, necesita disponer de la información de los usuarios del sistema para poder editar, eliminar o actualizar los datos en caso de que sea necesario.

Historias de Usuario

Las historias de usuario son descripciones sencillas establecidas por el cliente del sistema, estas mencionan a los usuarios que intervienen en el sistema, la funcionalidad que debe poseer y el motivo por el que se debe desarrollar la misma [25]. A continuación, en la Tabla V, se muestra un ejemplo de historia de usuario, el resto de las historias de usuario se visualizan en la sección Anexo**,** a partir de la Tabla XII.

Tabla V. Historia de usuario Nro. 1

HISTORIA DE USUARIO	
Identificador (ID): HU01	**Usuario:** Administrador
Nombre de Historia: Servicio para inicio de sesión según el rol de administrador	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Iteración asignada: 1	
Responsable: Lizbeth García	
Descripción: El usuario administrador ingresa al sistema y la interfaz se personaliza según el rol indicado para que pueda observar cierta información o actualizarla.	
Observación: Este usuario es capaz de realizar cualquier acción y conceder todos los permisos necesarios dentro del sistema.	

Product Backlog

El Product Backlog es una lista donde se establecen prioridades de cada funcionalidad que se establece en las historias de usuario [25]. El

Product **Backlog** detallado del sistema se visualiza en la sección Anexo, en la Tabla XXVI. Los tres primeros ítems mencionados en el Product Backlog se muestran en la Tabla VI.

Tabla VI. Tres primeros ítems del Product Backlog

ID-HU	Historia de usuario	Iteración	Estado	Prioridad
HU01	Servicio para inicio de sesión según el rol de administrador	1	Completa	Alta
HU02	Almacenar en la base de datos un usuario con rol empleado	1	Completa	Alta
HU03	Consultar en la base de datos la información del empleado	2	Completa	Media

Sprint Backlog

Los Sprint son ciclos de tiempos pequeños ya planificados en el que se culmina una tarea con una funcionalidad respectiva, este periodo puede durar aproximadamente dos semanas. Entonces, el Sprint Backlog es un lista tareas que se presenta en un tablero donde se puede identificar los posibles inconvenientes que pueden pausar el avance [26]. En la Tabla VII, se muestra un ejemplo del Sprint Backlog del presente proyecto, un mayor detalle del mismo se encuentra en el Anexo, Tabla XXVII.

Tabla VII. Sprint 0 del Proyecto Mainsfot

ID-SB	Nombre	ID-HU	HISTORIA DE USUARIO	TAREAS	TIEMPO ESTIMADO
SB-000	Configuración del entorno de desarrollo.	N/A	N/A	● Crear un proyecto dentro de Firebase para guardar la información necesaria. ● Configurar el entorno de desarrollo en React para validar los scripts de registro y consulta en el entorno web. ● Configurar el entorno de desarrollo en React Native para validar los scripts de registro y consulta en el entorno móvil.	10H

2.2 Diseño de interfaces (mockups)

El presente proyecto se enfoca en el proceso de Backend del sistema web y móvil para el control de empleados de empresas en crecimiento, por lo tanto, no hace uso de herramientas para el diseño de interfaces, pero si presenta la interfaz gráfica como ejemplo visual para el sistema web y la aplicación móvil.

Sistema Web

La Fig. 1 presenta la interfaz web del inicio de sesión para empezar con el flujo del sistema dependiendo del rol del usuario.

Fig. 1. Inicio de sesión web

Aplicación Móvil

La Fig. 2 presenta la interfaz móvil del inicio de sesión para empezar con el flujo de la aplicación tomando en cuenta que es exclusiva para usuarios con rol “Empleado”.

Fig. 2. Inicio de sesión móvil

2.3 Diseño de la arquitectura

La organización de un sistema para la gestión del personal de una empresa y sus horarios laborales requiere la selección de un patrón arquitectónico, de esta manera se define la forma de trabajo dentro del sistema y así se indica un diseño de la estructura global del mismo. Un buen diseño arquitectónico logra complementar correctamente la interfaz con la ingeniería de requerimientos [27].

Arquitectura de Datos

La arquitectura de datos es un conjunto de herramientas y productos con el fin de gestionar información. También define el proceso para capturar datos, transformarlos y la forma en cómo se entregan al usuario final. Además, logra identificar a las personas quienes consumen la información. El flujo que debe seguir es desde los consumidores de datos a la fuente de los mismos [28].

A continuación, se presenta el esquema de la base de datos, la Fig. 3, representa una de las primeras colecciones donde se manejan los cargos a través del usuario administrador, la Fig. 4, representa la colección de roles desde donde se valida el acceso a diferentes rutas y permisos según el usuario que vaya a iniciar sesión.

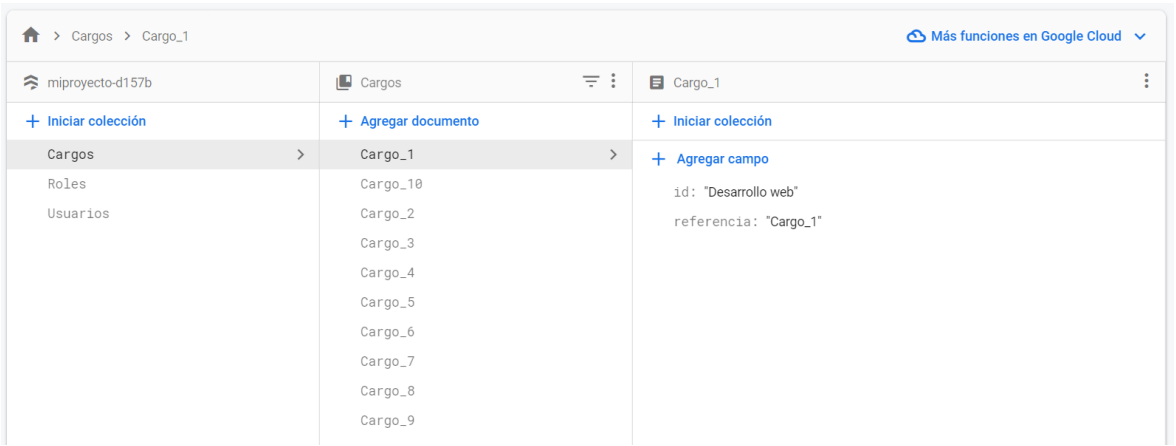


Fig. 3. Colección de cargos

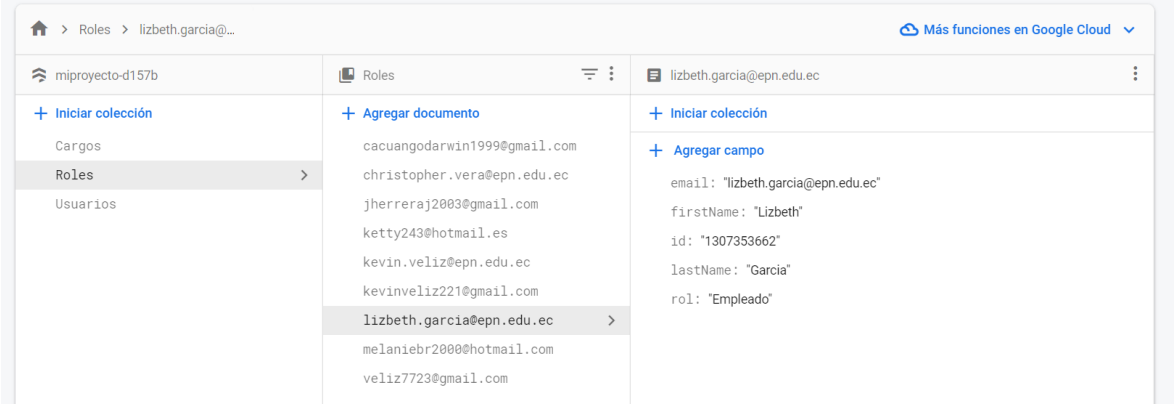


Fig. 4. Colección roles para control de inicio de sesión

En la Fig. 5, se muestra la colección donde se almacena la información de todos los usuarios registrados, la Fig. 6, es una subcolección registrada en el documento correspondiente a cada usuario y en caso de poseer el rol de empleado, es capaz de registrar su actividad laboral y registrar su documentación como se observa en la Fig. 7.

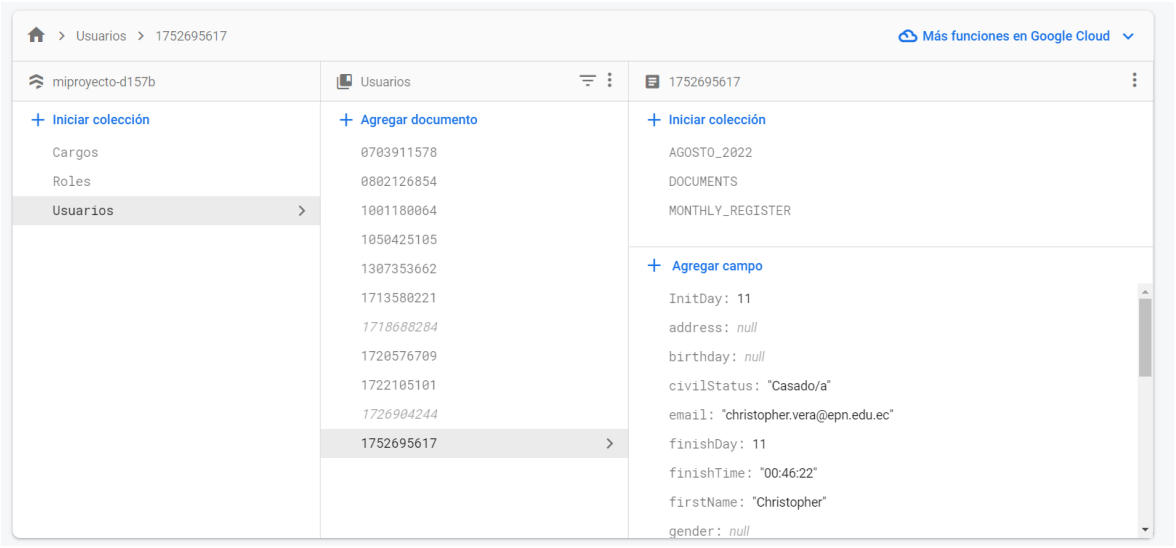


Fig. 5. Colección de Usuarios

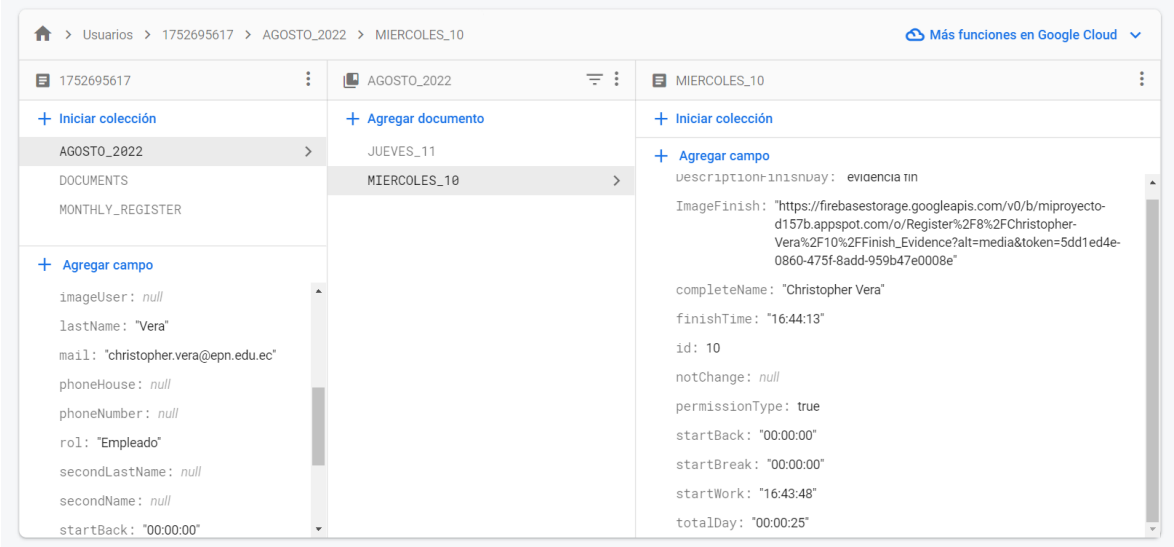


Fig. 6. Colección diaria del registro laboral del empleado

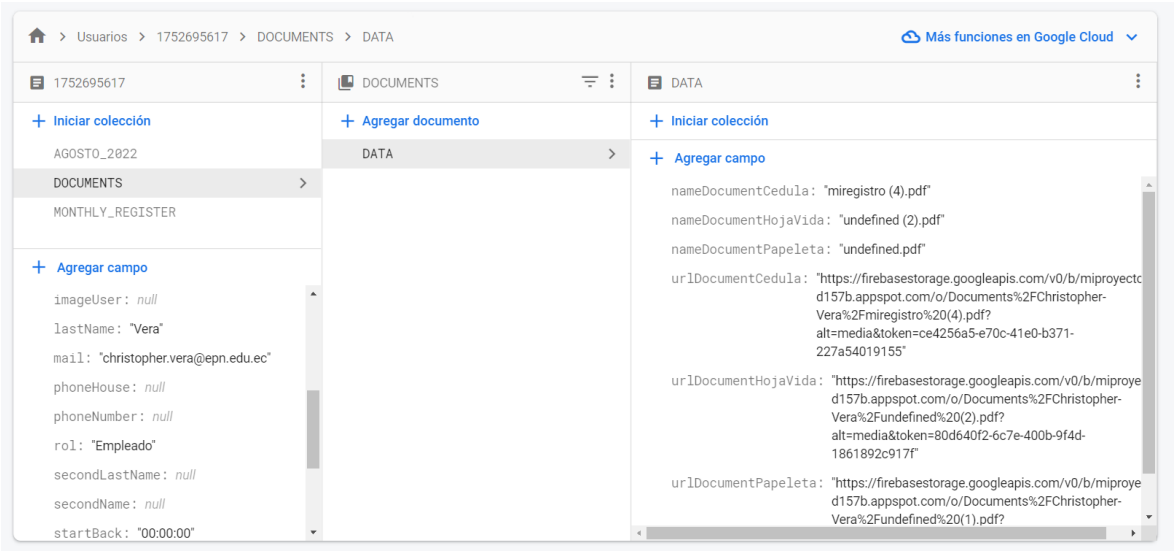


Fig. 7. Colección de documentos importantes del empleado

Finalmente, la Fig. 8, corresponde a los campos del registro total de las horas de trabajo de un empleado, es decir, horas y actividades registradas mensualmente.

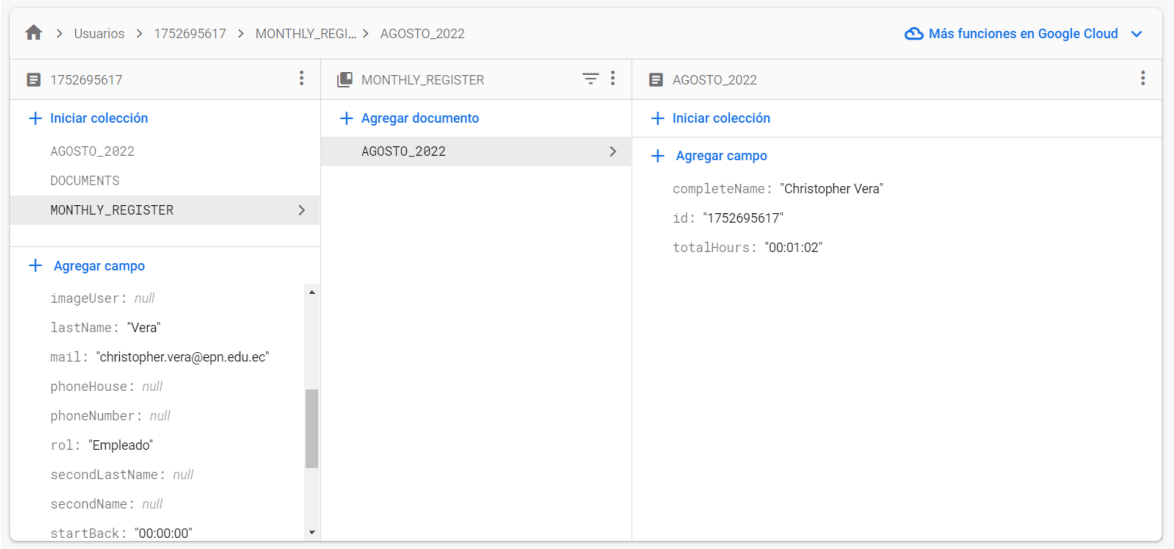


Fig. 8. Colección mensual del registro laboral del empleado

Patrón arquitectónico

Los patrones de arquitectura hacen referencia a un esquema organizado cuya estructura es fundamental para sistemas de software. Los patrones buscan encontrar elementos reusables en el diseño de sistemas, evitan buscar soluciones a problemas que ya se han encontrado, estandarizan modelos para el diseño. Un patrón de arquitectura posee subsistemas, responsabilidades e interrelaciones, también se destaca por ser de un mayor nivel de abstracción [29] .

Para el presente proyecto se establece un patrón donde se presentan esquemas para definir las relaciones y de esta manera se vaya construyendo el sistema. La mejor opción de patrón arquitectónico es MVC (Model, Views & Controllers), esta arquitectura separa el código según la responsabilidad asignada y mantiene capas para tareas concretas [30]. Modelo-Vista-Controlador está diseñado para sistemas que usan interfaces de usuario, como es el caso de este proyecto que posee interfaz móvil y web.

Cada componente se enfoca en algo específico, es decir, modelo hace referencia a la lógica de los datos y como se almacenan en la base de datos, vista es la interfaz gráfica que puede observar el usuario final y finalmente, el controlador es el encargado de manejar los datos y decidir de qué manera se muestra la información [31].

Sistema Web

En la Fig. 9, se puede observar el gráfico del patrón arquitectónico del sistema web. En este gráfico se aprecian los tres componentes principales y la conexión existente entre el lenguaje de programación, sus librerías, la base de datos y la interfaz de usuario.

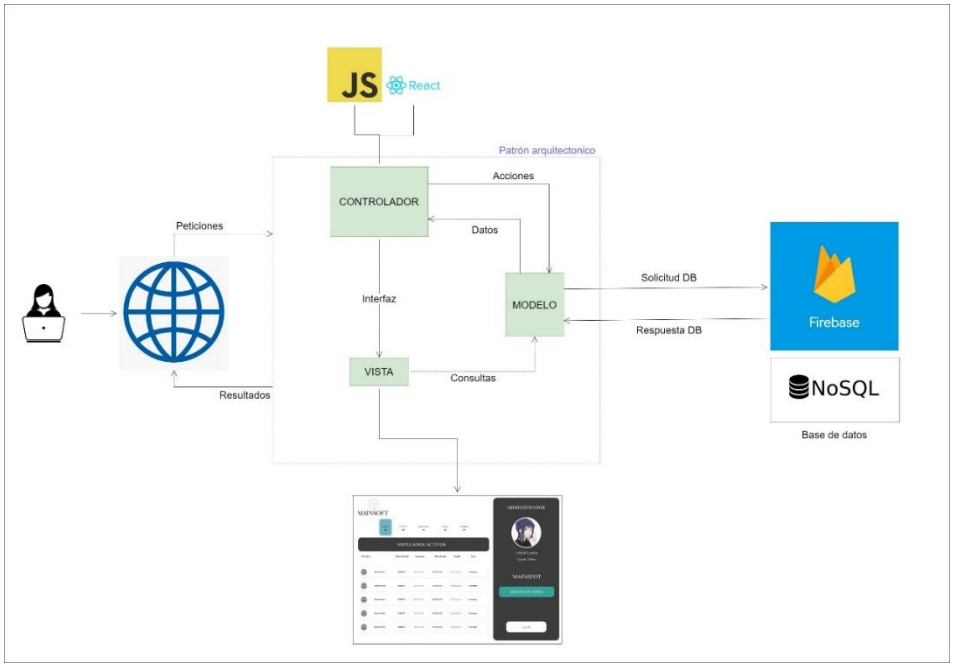


Fig. 9. Modelo-Vista-Controlador web

Aplicación Móvil

En la Fig. 10, se puede observar el gráfico del patrón arquitectónico del sistema móvil que empieza con el flujo a partir de una petición del usuario que es procesada por el controlador, es decir la lógica del programa, esta a su vez interactúa con la base de datos y finalmente muestra una respuesta a través de la vista en la aplicación móvil.

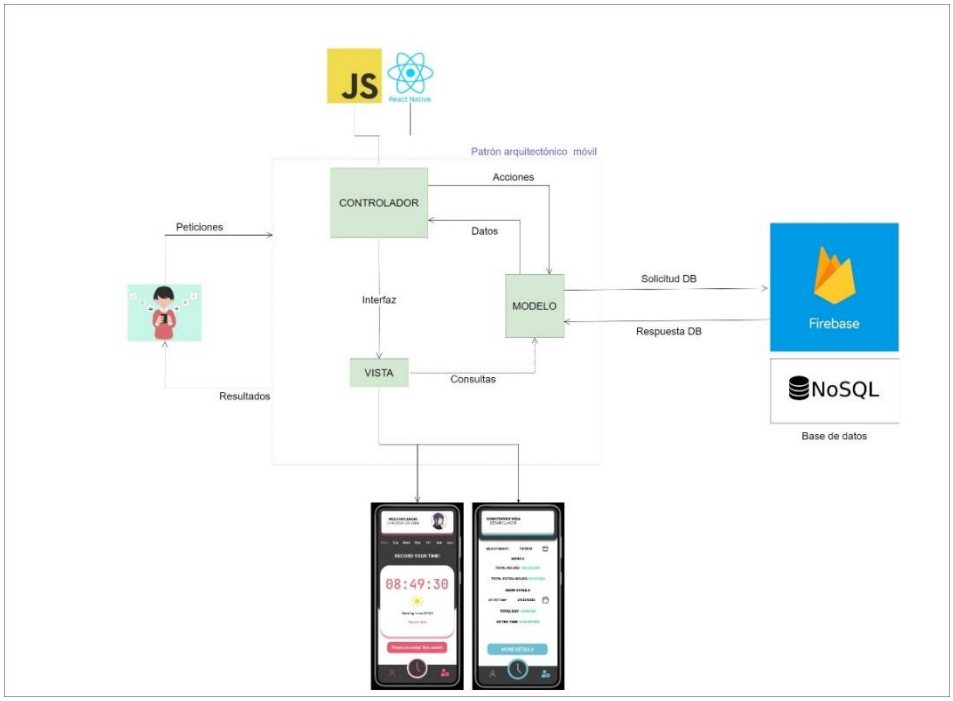


Fig. 10. Modelo-Vista-Controlador móvil

2.4 Herramientas de desarrollo

Las herramientas de desarrollo son aquellos programas que se usan con el fin de crear, mantener, depurara o gestionar un sistema [32]. El sistema para gestionar empleados y sus horarios laborales de empresas en crecimiento posee interfaz web y móvil. Para ambos sistemas se usa el mismo IDE (Entorno de Desarrollo Integrado) y mismo lenguaje de programación, es decir, Visual Studio Code y JavaScript, respectivamente.

Visual Studio Code es la mejor opción en este proyecto porque posee soporte nativo para muchos lenguajes y entre ellos está, JavaScript. Este editor de código permite configurar la vista dividida para ver y editar más de un código, además de contar con una terminal para ejecutar diferentes comandos. También cuenta con extensiones que se instalan dentro del programa [33]. Al usar esta herramienta, el código que se crea para consultar en la base de datos puede ser revisado para corregir algún tipo de error en la estructura.

Los sistemas web y móvil realizan consultas de lectura y escritura sobre la misma base de datos que se almacena en Firebase. Esta plataforma está disponible para web, Android y iOS. A parte de contar con una base de datos donde la información esta como JSON y son tipo No SQL, Firebase cuenta con autenticación de usuario para que sea posible ingresar al sistema que se está desarrollando mediante un correo y contraseña. Finalmente, el sistema ocupa el almacenamiento en la nube al momento de cargar documentos importantes o fotos de perfil desde la web o interfaz móvil.

Sistema Web

El sistema web se utiliza una herramienta distinta para el desarrollo de la interfaz y la implementación de los servicios, tal como se muestra en la Tabla VIII.

Tabla VIII. Herramienta de desarrollo web

Herramienta	Justificación
React	Esta herramienta se enfoca en el desarrollo de la interfaz web, pero dentro del componente Backend sirve para validar el código de consulta que requiere cada servicio, es decir, si el código esta correcto el acceso a la base de datos no da conflicto y la edición, actualización o registro de datos se completara con éxito, caso contrario el

	framework de React ayuda a revisar y corregir los errores.

Aplicación Móvil

En el sistema móvil se utilizan herramientas mencionadas en la Tabla IX, para validar que los servicios de consulta se hayan hecho correctamente.

Tabla IX. Herramientas de desarrollo móvil

Herramienta	Justificación
React Native	Este framework sirve para crear aplicaciones para Android y iOS y son ejecutadas en plataformas móviles. Para el componente Backend es importante tener este entorno para validar que el código de consulta a la base de datos se esté ejecutando de acuerdo a la lógica del negocio.
Expo Framework	Esta herramienta es útil dentro del Backend para revisar que la interfaz móvil y el flujo de datos no genere inconvenientes y al usuario final vea la información correcta.

Librerías

Para que exista conexión entre la base de datos de Firebase, Cloud Firestore, y la interfaz web o la aplicación móvil es necesario instalar la librería propia de Firebase dentro del proyecto para tener acceso a los diferentes métodos que posee la plataforma Firebase. A continuación, en la Tabla X se menciona las dos librerías necesarias.

Tabla X. Librerías de Firebase según el sistema

Sistema	Librería	Descripción
Web	AngularFire	A través de esta librería de angular que es compatible entre una plataforma web y Firebase se puede acceder a la información de la base de datos, es decir, realizar

		consultas en colecciones o documentos.
Móvil	React Native Firebase	Al implementar esta librería en el proyecto de aplicación móvil se asegura la conexión con Cloud Firestore, además esta es oficial de React Native .

3 RESULTADOS

En este punto, inicia el desarrollo del componente Backend y en cada Sprint se representa la función específica donde se detalla el comportamiento que sigue para traer la información de la base de datos, así como también la manera de cómo se guarda la información y como se estructuran los diferentes documentos y colecciones en Firebase.

3.1 Sprint 0. Configuración del ambiente de desarrollo

El objetivo del Sprint 0, es la configuración de la base de datos, en este caso la creación de un proyecto en Firebase para consumir sus servicios. Por otro lado, se debe configurar el entorno de desarrollo para ir realizando las pruebas respectivas de las consultas y escrituras sobre la base de datos, para esta parte se interactúa con React y React Native.

Lista de tareas:

- Tarea 1. Proyecto en Firebase.
- Tarea 2. Entorno de desarrollo para React.
- Tarea 3. Entorno de desarrollo para React Native.

Tarea 1. Proyecto en Firebase

Dentro de https://console.Firebase.google.com/ se crea un nuevo proyecto, se lo nombra de una manera identificable y se inicia el mismo mostrando sus credenciales para que pueda ser enlazado con el código, así como se muestra en la Fig. 11 y Fig. 12.

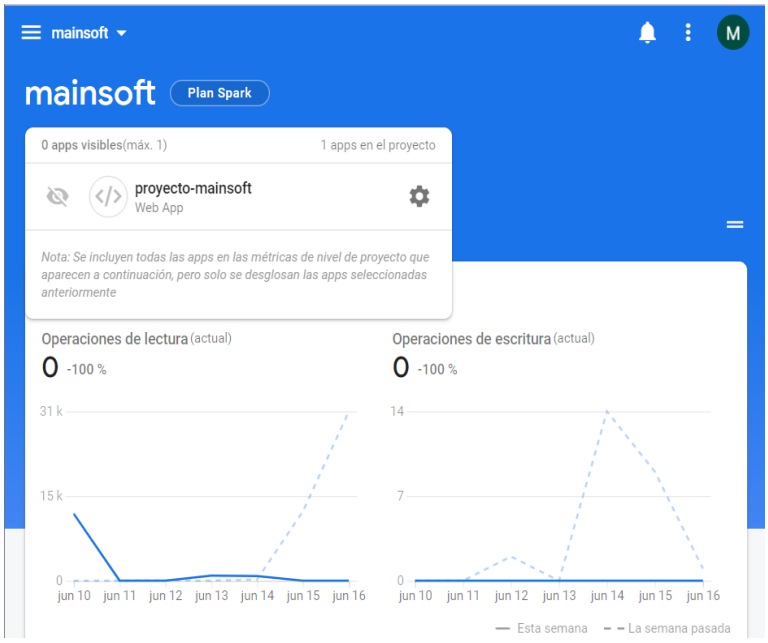


Fig. 11. Nuevo proyecto en Firebase

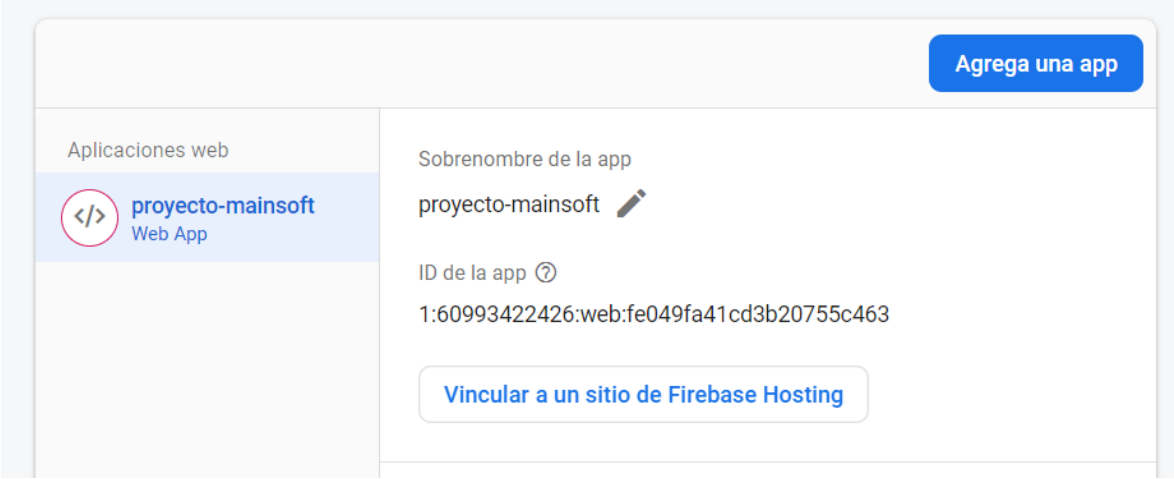


Fig. 12. Credenciales del nuevo proyecto de Firebase

Tarea 2. Entorno de desarrollo para React.

Para incorporar cada una de las consultas a la base de datos es necesario tener el entorno de desarrollo del sistema web para realizar las validaciones correspondientes y corregir errores en caso de que sea necesario. Por este motivo, se crea un proyecto de React donde se implementarán los servicios con una interfaz sencilla para que luego sean llamados por el sistema web con el diseño establecido. La estructura del proyecto se puede observar en la Fig. 13.

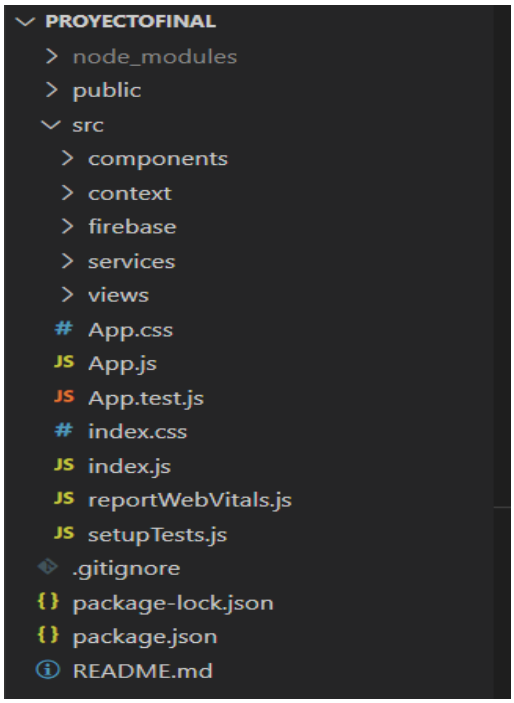


Fig. 13. Estructura del proyecto en React

Tarea 3. Entorno de desarrollo para React Native.

De la misma manera, en esta tarea se crea un proyecto con React Native, como se muestra en Fig. 14 para implementar las consultas a la base de datos para que sea validada la lógica junto con la interfaz móvil.

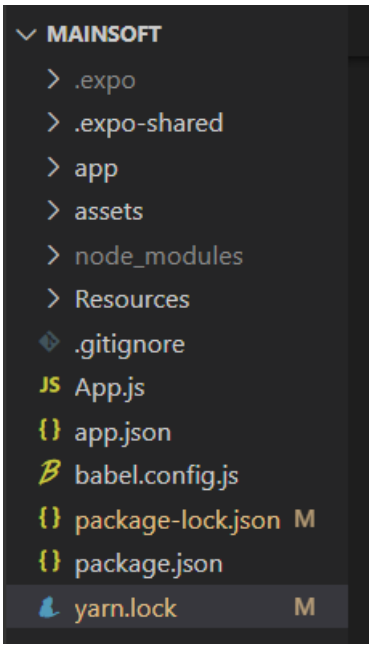


Fig. 14. Estructura del proyecto en React Native

Sprint 1. Autenticación

En este Sprint empieza el desarrollo del sistema y el objetivo es crear métodos para que el usuario pueda registrar nuevos usuarios y asignarle un rol, de esta manera empieza los nuevos registros en la base de datos.

Lista de tareas:

- Tarea 1. Registrar a un usuario.
- Tarea 2. Estructurar la base de datos.
- Tarea 3. Registrar roles del usuario en la base de datos.
- Tarea 4. Estructurar la base de datos del usuario.
- Tarea 5. Agregar función para inicio de sesión.

Tarea 1. Registrar a un usuario.

La lógica del negocio es tener un administrador por defecto dentro de la base de datos de Firebase que fue configurada en el Sprint 0, dentro de la interfaz del usuario que tiene el rol de administrador se puede llevar a cabo el registro de los usuarios tipo gerente o

empleado. La función se ejecuta cuando el administrador da clic en “Crear usuario” y se muestra la función en la Fig. 15.

```
const handleSubmit = (e) => {
  e.preventDefault();
  //console.log("Procesado form de registro", emailEmp, password);
  console.log("Usuario creado en Firebase");
  //navigate("/")
  createUserWithEmailAndPassword(auth2, emailEmp, password);

  try {
    const docuRef = doc(db, "Usuarios/" + id);
    setDoc(docuRef, {
      mail: emailEmp,
      rol: rol,
      firstName: name,
      secondName: null,
      lastName: lastname,
      secondLastName: null,
      id: id,
      phoneNumber: null,
      civilStatus: null,
      gender: null,
      birthday: null,
      address: null,
      phoneHouse: null,
      imageUser: null,
      workstation: cargo,
      workingState: "NOTWORKING",
    });

    const docuRef2 = doc(db, "Roles/" + emailEmp);
    setDoc(docuRef2, {
      email: emailEmp,
      rol: rol,
      firstName: name,
      lastName: lastname,
      id: id,
    });

    console.log("Usuario creado exitosamente");
    clicked();

  } catch (error) {
    console.log("Error de registro", error);
  }
};
```

Fig. 15. Código de handleSubmit del formulario de Registrar Usuario

La función inicia capturando todos los valores que se ingresaron en el formulario de registro, posterior a ello se usa una función de Firebase Authentication que permite crear nuevos usuarios usando un correo y una contraseña (createUserWithEmailAndPassword). Finalmente se usa la sentencia try y catch, en donde se accede a la base de datos y se setea un nuevo documento con un id único, en este caso la cédula del usuario y en el documento se detallan todos los atributos que debe poseer el usuario, los datos son los que se registran en el formulario y en caso de que no existan se colocan como null para en un proceso posterior puedan ser actualizados. A la par se crea otro documento que tiene como id el correo electrónico del usuario para luego poder acceder a la interfaz que

corresponda y validar el rol que posee cada usuario, puede ser administrador, gerente o empleado. En caso de que el registro sea exitoso se muestra un mensaje satisfactorio en la consola, caso contrario se muestra el error.

Tarea 2. Estructurar la base de datos.

La base de datos general cuenta con dos colecciones primarias: Usuarios y Roles, en la primera se almacena toda la información personal del usuario, la documentación que carga y su historial de horas de trabajo clasificada por mes, tal como se muestra en la Fig. 16.

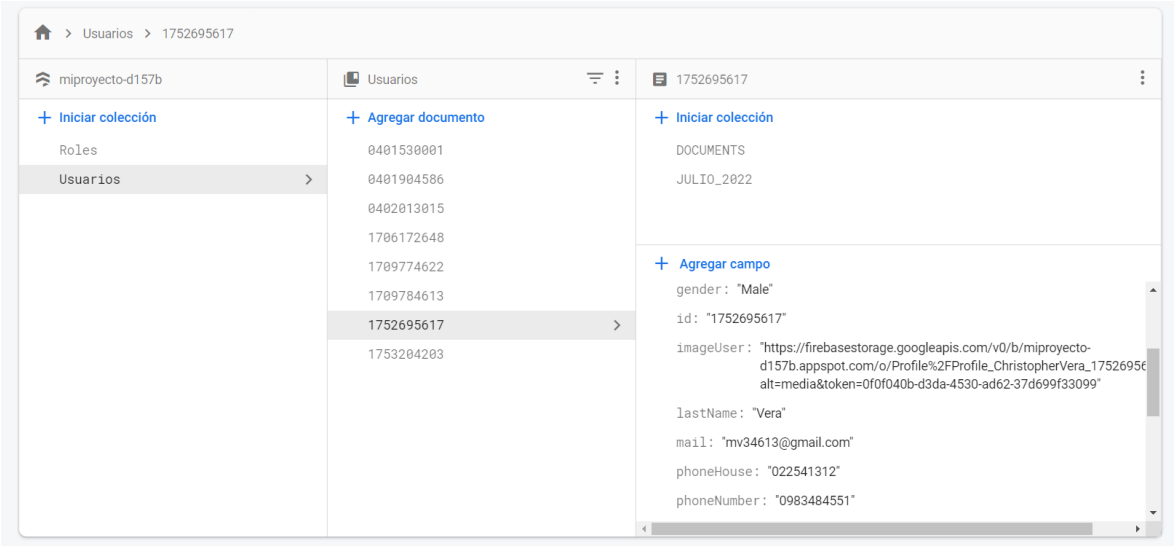


Fig. 16. Estructura de la colección principal de la base de datos

Tarea 3. Registrar roles del usuario en la base de datos.

Como se mencionó antes al registrar a un nuevo usuario, este genera dos colecciones y en este caso es importante mencionar los “Roles”. Aquí se guardan el nombre, apellido, cédula, correo del usuario y el rol que se selecciona en el formulario. A partir de esta colección se puede validar los roles del usuario y los privilegios que posee cada uno de ellos. En la Fig. 17. Se visualiza la segunda colección importante del proyecto.

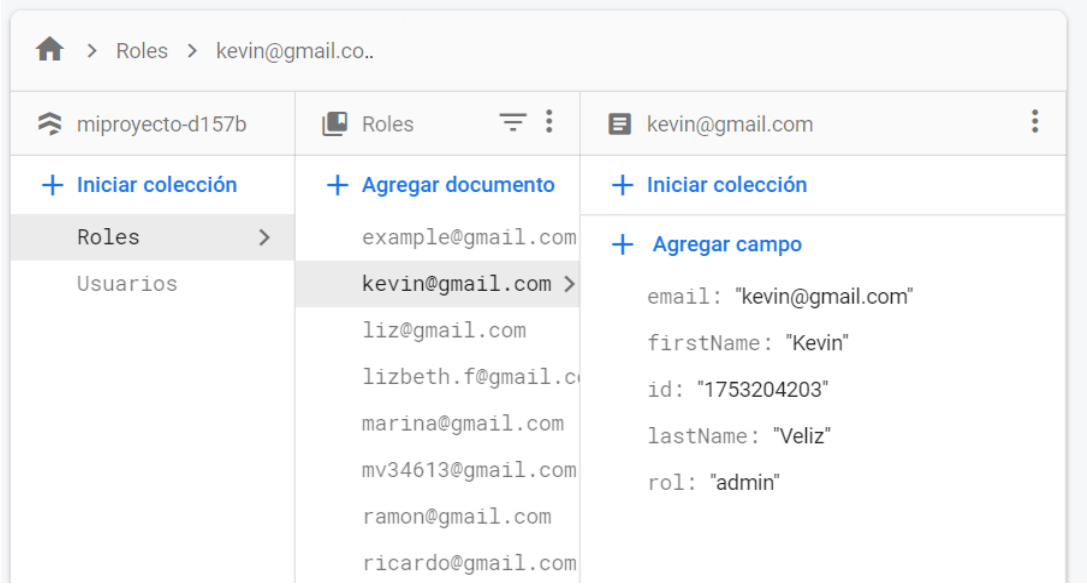


Fig. 17. Estructura de la colección secundaria de la base de datos.

Se debe mencionar que el usuario gerente y administrador son los únicos usuarios con permisos para crear usuarios tipo empleado o gerente.

Tarea 4. Estructurar la base de datos del usuario.

El usuario debe poseer todos los atributos que se muestran a continuación. Estos datos son la información general del usuario. Excepto el atributo InitDay que hace referencia al día actual que el usuario está trabajando y finishDay representa el último día de registro donde culminó su jornada de trabajo. En la Fig. 18 y Fig. 19, se muestran los atributos registrados.

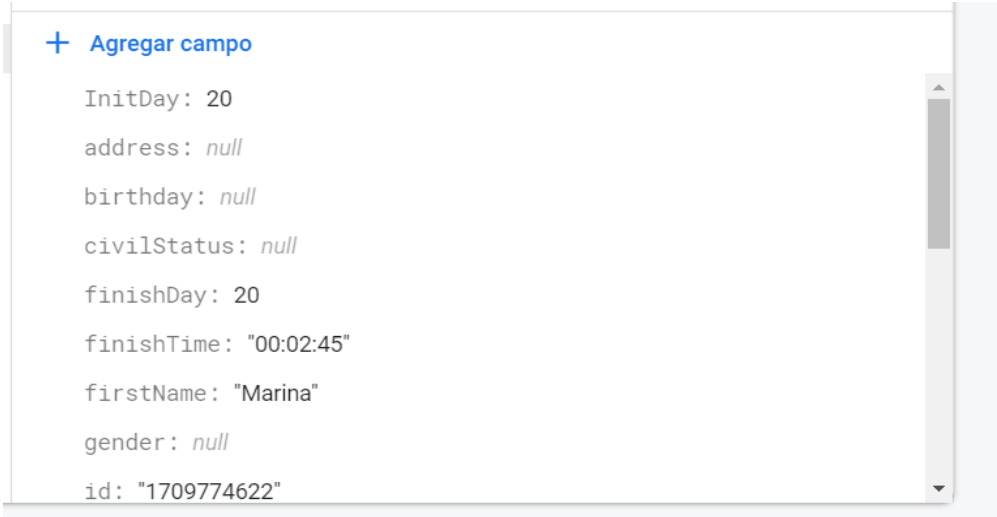


Fig. 18. Atributos del usuario

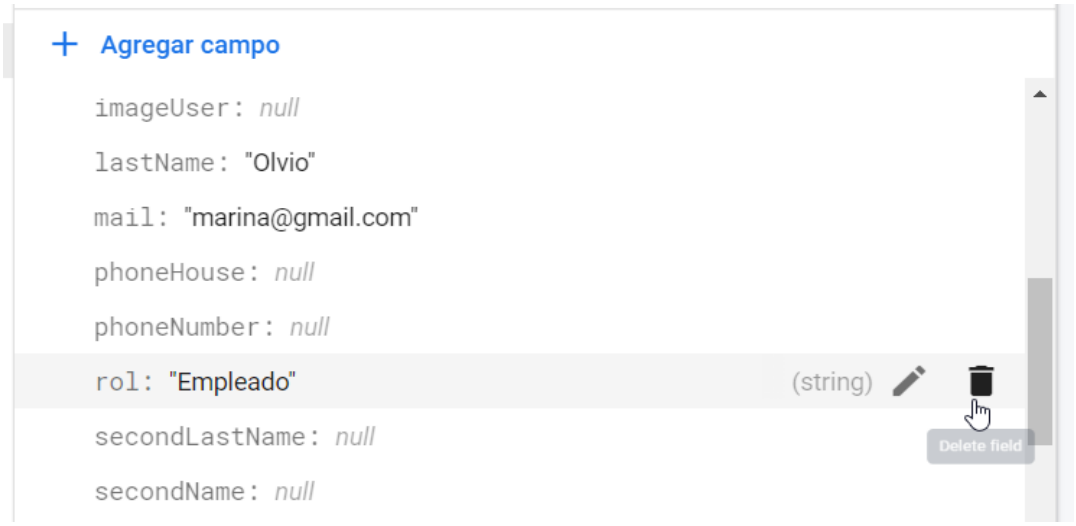


Fig. 19. Atributos del usuario

Los siguientes atributos que se muestran en la Fig. 20, hacen referencia a las horas que registra el usuario en su jornada laboral.

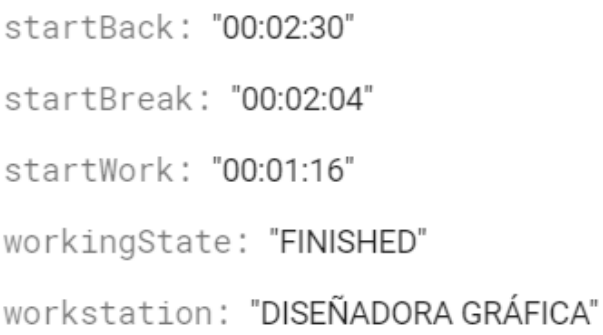


Fig. 20. Atributos secundarios del usuario

En startWork se almacena la hora exacta que registró el usuario al momento de iniciar su jornada laboral, startBreak almacena la hora cuando el usuario toma un receso, startBack es la hora en que el usuario retoma sus actividades, finishTime que se muestra anteriormente es la hora que finaliza la jornada el usuario, workingState es el estado que se actualiza conforme el usuario registre sus horas y workstation es el cargo que posee el usuario empleado dentro de la empresa.

Tarea 5. Agregar función para inicio de sesión.

El inicio de sesión de los usuarios se da a través de un pequeño formulario que especifica correo electrónico y contraseña. Entonces la función asincrónica que se ejecuta al enviar el formulario empieza capturando los dos campos. Primero empieza llamando a otra función que lee la base de datos usando el campo de correo electrónico del formulario y setea el rol que posee el usuario. En la Fig. 21, se representa la primera función a ejecutar.

```
async function actualizarRol(email) {
  const docRef2 = doc(db, "Roles/" + email);
  const docSnap2 = await getDoc(docRef2);
  rol.push(docSnap2.data().rol)
}
```

Fig. 21. Función que setea el rol del usuario

A continuación, la sentencia espera que se actualice el rol del usuario y posterior se da el inicio de sesión. Dependiendo el valor de la variable rol, el usuario es redirigido a la interfaz web que le corresponde. En caso de que haya un problema al iniciar sesión se muestra en consola un mensaje no satisfactorio tal y como se observa en la Fig. 22.

```
const handleSubmit = async (e) => {
  e.preventDefault();
  console.log("Usuario activo");

  try {
    await actualizarRol(email);
    await loginUser(email, password);

    if (rol[0] === "admin") {
      navigate("/admin");
    } else if (rol[0] === "Empleado") {
      navigate("/empleado");
    } else if (rol[0] === "Gerente") {
      navigate("/manager");
    }
  } catch (error) {

    console.log("Error de inicio de sesión", error);
  }
};
```

Fig. 22. Función que permite setear el rol e iniciar sesión

Se debe mencionar que la función de loginUser(email, password), se puede usar porque se debe declarar al inicio el uso de UseContext.

Línea de código: const {loginUser} = useContext(UserContext);

Este hook de contexto permite el acceso de variables o funciones que son globales en nuestro proyecto. Este se encuentra declarado en la carpeta context en el archivo userProvider, dentro de este archivo está declarada la función loginUser.

Línea de código: const loginUser =(email, password) => signWithEmailAndPassword(auth, email,password);

Sprint 2. Funciones principales del usuario Empleado

El objetivo de este Sprint es acceder a la información de los usuarios con rol empleado y darle las funciones necesarias, tales como, revisar o actualizar su información personal, cargar documentación que requiere la empresa y registrar su actividad diaria de horas de trabajo. A continuación, la lista de tareas a realizaren este Sprint.

- Tarea 1. Mostrar información del usuario
- Tarea 2. Modificar datos personales del usuario empleado
- Tarea 3. Cargar imagen de perfil del usuario
- Tarea 4. Cargar documentos importantes.
- Tarea 5. Registrar horas de trabajo del usuario.
- Tarea 6. Calcular las horas de trabajo de un día del usuario empleado.

Tarea 1. Mostrar información del usuario

El usuario empleado cuenta con un modal donde se le muestra la información principal que se completó al momento de registrarlo. Los demás campos están vacíos y tiene la posibilidad de editar su información.

La siguiente función, Fig. 23, es la que se usa para acceder a la base de datos, a la colección “Usuarios” y al documento que le corresponda según quien se haya autenticado en el sistema, luego trae la información del usuario y la setea en las variables de estado que le corresponda. Esta función se llama a penas se renderiza la página dentro del useEffect.

```
const informationFunction = async (cedula) => {
  const docRef = doc(db, "Usuarios/" + cedula);
  const docSnap = await getDoc(docRef);

  if (docSnap.exists()) {
    setUserImage(docSnap.data().imageUser);
    setFirsName(docSnap.data().firstName);
    setSecondName(docSnap.data().secondName);
    setLastName(docSnap.data().lastName);
    setSecondLastName(docSnap.data().secondLastName);
    setAddress(docSnap.data().address);
    setPhoneHouse(docSnap.data().phoneHouse);
    setPhoneNumber(docSnap.data().phoneNumber);
    setDate(docSnap.data().birthday);
    setGender(docSnap.data().gender);
    setCivilStatus(docSnap.data().civilStatus);
    setEmail(docSnap.data().mail)

  } else {
    console.log("No se pudo leer el documento");
  }
};
```

Fig. 23. Función que trae toda la información del usuario

La información al dispositivo móvil del usuario se la trae a través de la función getPersonalInformation(), que se muestra en Fig. 24 . Esta recibe una variable de estado tipo set y accede al documento correspondiente al usuario y almacena toda la información en un arreglo temporal que luego se setea en la variable que recibe la función.

```
export const getPersonalInformation = async (getInformations) => {
  const q = doc(global.dbCon, "/Usuarios/" + global.id);
  const docSnap = await getDoc(q);
  let personalInformation = [];
  personalInformation.push(docSnap.data());
  getInformations(personalInformation);
};
```

Fig. 24. Traer la información del empleado

Tarea 2. Modificar datos personales del usuario empleado

La información del usuario se muestra en un formulario dentro del modal y los campos vacíos se pueden completar con nueva información, los otros campos no son editables. Al

enviar dicho formulario se ejecuta un servicio para actualizar los atributos del documento al que pertenece el usuario, esto se puede observar en la Fig. 25.

```
const handleSubmit = async () => {
  await updateDoc(doc(db, "Usuarios/" + window.cedulaEmpleado), {
    secondName: secondName,
    secondLastName: secondLastName,
    birthday: date,
    address: address,
    civilStatus: civilstatus,
    gender: gender,
    phoneHouse: phoneHouse,
    phoneNumber: phoneNumber,
    imageUser:data
  });
  clicked()
};
```

Fig. 25. Función del formulario para actualizar información del usuario

En el componente móvil, la función para modificar la información del usuario es la que se visualiza en Fig. 26. Este servicio recibe la información que se va actualizar (data) y llama al método de Firebas updateDoc() para modificar el documento correspondiente al usuario. Se debe mencionar que “global.dbCon” hace referencia a la conexión con la base de datos de Firebase y “Usuarios/global.id” es el acceso al documento respectivo.

```
export const savePersonalInformation = async (data) => {
  await updateDoc(doc(global.dbCon, "/Usuarios", global.id), data);
};
```

Fig. 26. Guardar información del usuario

Tarea 3. Cargar imagen de perfil del usuario

Existe un campo para que cargue una foto de perfil y al cargar la imagen se ejecuta un servicio para almacenar la imagen en Firebase Storage. Esta función se ejecuta dentro del useEffect y crea un directorio denominado “Profile/” y carga la imagen en Firebase con el nombre de la imagen por defecto, este proceso se observa en la Fig. 27. Valida el estado de la carga y general el url de la imagen que se genera en Firebase para tener acceso dentro del sistema. El url mencionado es el que se almacena dentro del atributo imageUser del usuario.

```
React.useEffect(() => {
  informationFunction(window.cedulaEmpleado);

  const uploadFile = () => {
    const name = file.name;
    const storageRef = ref(storage, "Profile/" + file.name);
    const uploadTask = uploadBytesResumable(storageRef, file);

    uploadTask.on(
      "state_changed",
      (snapshot) => {
        const progress =
          (snapshot.bytesTransferred / snapshot.totalBytes) * 100;
        setProgress(progress);
        switch (snapshot.state) {
          case "paused":
            console.log("La carga está pausada");
            break;
          case "running":
            console.log("La carga esta en proceso");
            break;
          default:
            break;
        }
        console.log("carga compleatada");
      },
      (error) => {
        console.log("error: " + error);
      },
      () => {
        getDownloadURL(uploadTask.snapshot.ref).then((downloadURL) => {
          console.log("URL: " + downloadURL);
          setData(downloadURL);
        });
      }
    );
  };
  file && uploadFile();

}, [file]);
```

Fig. 27. Función que carga la imagen en Firebase Storage y genera un url

En caso del componente móvil, esta tarea es complemento de la información que recibe la función que se detalla en la Fig. 26.

Tarea 4. Cargar documentos importantes.

En una vista del usuario empleado existe un apartado para subir los documentos, tales como cédula, papeleta de votación y hoja de vida.

Para realizar este proceso se define un arreglo vacío para poder cargar los tres documentos a través de inputs y al dar clic en “Guardar” se llama a la función asincrónica “SaveDocuments” que se muestra en la Fig. 28. En este paso se crea un nuevo documento dentro de la colección DATA del usuario

```
const SaveDocuments = async() =>{
  await setDoc(doc(db, "Usuarios/" + window.cedulaEmpleado  + "/DOCUMENTS/DATA"), {
    urlDocumentHojaVida: data[0],
    urlDocumentCedula: data[1],
    urlDocumentPapeleta: data[2]
  });
  console.log('Datos guardados')
  clicked();
}
```

Fig. 28. Función que almacena los documentos dentro de la base de datos

Para cargar los documentos a Firebase Storage se vuelve a usar la función descrita en la Tarea 3.

El componente móvil se almacena en la base de datos los documentos importantes del empleado, la función recibe la información y valida con un try y catch, si existe el documento dentro de la colección del usuario en la base de datos, la función se encarga de actualizar los datos, caso contrario crea el nuevo documento e ingresa la información. En la Fig. 29, se muestra el código utilizado para esta funcionalidad. Es decir, que el documento se registra en Firestore con la siguiente estructura: “Usuarios/ (cédula del empleado) /DOCUMENTS/DATA”.

```
export const saveDocumentPersonal = async (data) => {
    try {

        await updateDoc(doc(global.dbCon, "/Usuarios/"+global.id+"/DOCUMENTS","DATA"), data);
        global.documents=true;
    } catch (error) {

        await setDoc(doc(global.dbCon, "/Usuarios/"+global.id+"/DOCUMENTS","DATA"), data);
        global.documents=true;
    }

  };
```

Fig. 29. Cargar los documentos del empleado

Si es necesario traer la documentación del usuario empleado en el componente móvil es importante la función getDocumentsData() que se detalla en la Fig. 30, en donde se accede al documento correspondiente y se almacenan los datos en un arreglo temporal.

```
export const getDocumentsData = async (getDocuments) => {
  const q = doc(global.dbCon, "/Usuarios/" + global.id +"/DOCUMENTS/"+"DATA");
  const docSnap = await getDoc(q);
  let documentInformation = [];
  documentInformation.push(docSnap.data());
  getDocuments(documentInformation)
};
```

Fig. 30. Traer los documentos importantes

Tarea 5. Registrar horas de trabajo del usuario.

Antes de proceder con el registro de horas se debe validar el estado en el que se encuentra el usuario empleado, por defecto se crea el usuario con el estado NOTWORKING, los otros estados son WORKING, BREAK, BACKWORKING y FINISHED. Por ende, se accede a la base de datos y se setea el estado del usuario en la variable de estado correspondiente y otra información en caso de que sea necesaria, tal como muestra la Fig. 31, workingState es el dato de importancia según este estado se muestran los botones a los que tiene acceso el usuario.

```
const informationFunction = async (cedula) => {
  const docRef = doc(db, "Usuarios/" + cedula);
  const docSnap = await getDoc(docRef);

  if (docSnap.exists()) {
    setRol(docSnap.data().rol);
    setWorkingState(docSnap.data().workingState)
    setFirstName(docSnap.data().firstName);
    setLastName(docSnap.data().lastName);

  } else {
    // doc.data() will be undefined in this case
    console.log("No se pudo leer el documento");
  }
};
```

Fig. 31. Función para traer información de la base de datos

Al dar clic en “EMPEZAR”, y confirmar el inicio de la jornada se ejecuta la siguiente función asincrónica. Aquí se muestra la hora exacta al dar clic en la confirmación con la función GetHoraInicio. Por ende, se crea un nuevo documento dentro de la base de datos, según el id de la colección que le corresponda. En el atributo startWork se almacena dicha hora, así como se visualiza en la Fig. 32.

```
const handleInicio = async() => {
  console.log('Hora de entrada', GetHoraInicio());
  const hora = GetHoraInicio();
  console.log('horas, mont', month_year, day_day, hora)
  window.horaInicio= hora;
  //horasTrabajo.push(hora);
  const completeName= firstName + " " + lastName
  await setDoc(doc(db, "Usuarios/" + window.cedulaEmpleado  + "/" + month_year + "/" + day_day), {
      startWork: hora,
      id:day_day,
      completeName:completeName
  });
```

Fig. 32. Función que almacena la hora de entrada del empleado

La función de GetHoraInicio, hace uso de getHours, getMinutes y getSecons que son parte de las constantes de fecha para obtener la hora,minutos y segundos actuales. Para que se almacene en la base de datos con el siguiente formato hh:mm:ss se hicieron varias sentencias if para colocar un cero adelante en caso de que la hora, minuto o segundo sea menor que 10 y se concatenan las variables para que sea un string, tal como muestra la Fig. 33.

```
let newh, newm, news;
function GetHoraInicio(){
  const h = dateobj.getHours();
  const m = dateobj.getMinutes();
  const s = dateobj.getSeconds();
  if(h < 10){
      newh = "0" + h
  }else{
      newh = h
  }

  if(m < 10 ){
      newm = "0" + m
  } else {
      newm = m
  }

  if(s < 10 ){
      news = "0"+s
  }else{
      news = s
  }
  const horaactual = newh + ":" + newm + ":" + news
  return horaactual
}
```

Fig. 33. Establecer formato de la hora de entrada

El almacenamiento correcto de la información y filtrado de registros por mes y día se establecen mediante las variables de month_year y day_day, en la Fig. 34 se muestra el proceso.

```
const monthNames = ["ENERO", "FEBRERO", "MARZO", "ABRIL", "MAYO", "JUNIO", "JULIO", "AGOSTO", "SEPTIEMBRE", "OCTUBRE", "NOVIEMBRE", "DICIEMBRE"]
const dateMonth = new Date().getMonth()
const nombreMes = monthNames[dateMonth]
const month_year = nombreMes + "_" + dateobj.getFullYear()


const dayNames = ["LUNES", "MARTES", "MIERCOLES", "JUEVES", "VIERNES", "SABADO", "DOMINGO"]
const dateDays = new Date().getDay()
const nombreDias = dayNames[dateDays - 1]
const dayNumber = dateobj.getDate();
let dayNumberFinal;
if(dayNumber<10){
  dayNumberFinal = "0" + dayNumber;
}else {
  dayNumberFinal = dayNumber;
}

const day_day = nombreDias + "_" + dayNumberFinal
```

Fig. 34. Formato de mes y día

De esta manera el valor de startWork se almacena en la base de datos tal como muestra la Fig. 35. Y con el mismo procedimiento se registran los atributos de startBreak, al momento de ir al receso, startBack, cuando vuelve del receso y finishTime, cuando finaliza la jornada.

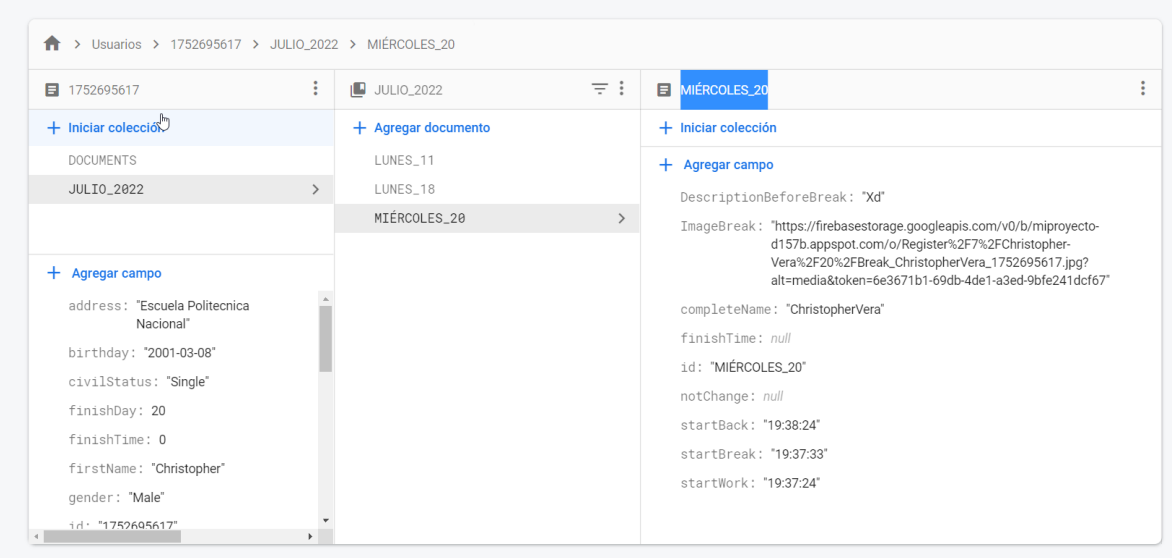


Fig. 35. Estructura de la colección por mes que genera el reporte de horas

Lo anteriormente detallado corresponde a las funciones para el apartado web, por lo tanto, para el componente móvil primero es necesario establecer una función para actualizar el estado de un empleado según como vaya registrando sus horas de trabajo, tal como se muestra en la Fig. 36. Esta recibe un estado a conveniencia, accede al documento dentro del Firestore y actualiza el atributo correspondiente.

```
export const updateStateWork = async (state) => {
  const docRefUser = doc(global.dbCon, "/Usuarios", global.id);
  await updateDoc(docRefUser, {
    workingState: state,
  });
  global.workState = state;
};
```

Fig. 36. Actualizar estado de trabajo del empleado

A continuación, se crea un documento o se actualiza según corresponda con la misma estructura que se menciona en la Fig. 34, el cambio se genera en la variable “saveMonth” que corresponde a “month_year” y “saveDay” se refiere a “day_day”. La función de la Fig. 37, crea el documento y subcolección dentro del usuario que inicio sesión con los atributos de las horas laborales nulos.

```
export const createTask = async () => {
  let date = new Date();
  const dias = [
    "domingo", // 0
    "lunes", // 1
    "martes",
    "miercoles",
    "jueves",
    "viernes",
    "sábado",
  ];
  const numeroDia = new Date().getDay();
  const nombreDia = dias[numeroDia].toUpperCase();
  let saveMonth = getMonth(date.getMonth() + 1) + "_" + date.getFullYear();
  let saveDay;
  if (date.getDate() >= 1 && date.getDate() <= 9) {
    saveDay = nombreDia + "_" + "0" + date.getDate();
  } else {
    saveDay = nombreDia + "_" + date.getDate();
  }
  const setTime = {
    startWork: null,
    startBreak: null,
    startBack: null,
    finishTime: null,
  };
  const setTimer = {
    notChange: null,
  };

  try {
    await updateDoc(
      doc(
        global.dbCon,
        "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDay
      ),
      setTimer
    );
  } catch (error) {
    await setDoc(
      doc(
        global.dbCon,
        "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDay
      ),
      setTime
    );
  }
};
```

Fig. 37. Crear documento de horas laborales con valores nulos

El almacenamiento de las horas correspondientes al inicio de jornada (startWork), inicio del receso (startBreak), final del receso (startBack) y final de la jornada (finishTime) se debe ocupar la función de la Fig. 38, Fig. 39 y Fig. 40, esta recibe una hora que se registra (time), un estado (state) para validar en que atributo se debe colocar la hora antes mencionada, una descripción (description) en caso de salir al receso o finalizar la jornada laboral y una

imagen (Image) para la evidencia correspondiente. A través, de estos parámetros se accede al documento correspondiente y se guardan las horas necesarias en la base de datos.

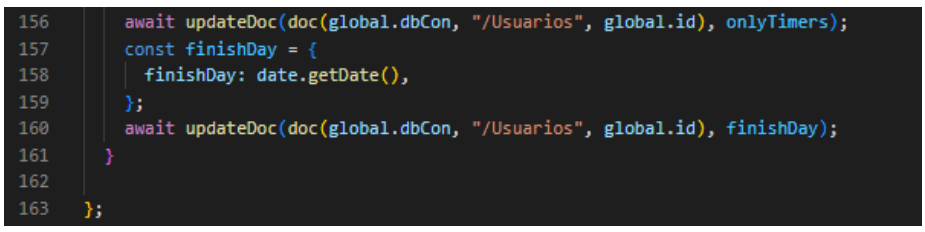
Se maneja el tributo finishDay, que corresponde al día actual que el usuario empleado termina su jornada laboral y se establece solo cuando el usuario da clic en “Finalizar”.

```
export const saveTimeUser = async (time, state, Description, Image) => {

  let date = new Date();

  const dias = [
    "domingo", // 0
    "lunes", // 1
    "martes",
    "miercoles",
    "jueves",
    "viernes",
    "sábado",
  ];
  const numeroDia = new Date().getDay();
  const nombreDia = dias[numeroDia].toUpperCase();
  let saveMonth = getMonth(date.getMonth() + 1) + "_" + date.getFullYear();
  let saveDayControl;
  let saveDay;
  if (date.getDate() >= 1 && date.getDate() <= 9) {
    saveDay = "0" + date.getDate();
    saveDayControl = nombreDia + "_" + "0" + date.getDate();
  } else {
    saveDay = date.getDate();
    saveDayControl = nombreDia + "_" + date.getDate();
  }

  if (state == "startWork") {
    const timer = {
      startWork: time,
    };
    const timerSet = {
      completeName: global.name + " " + global.lastName,
      id: saveDay,
    };
    await updateDoc(
      doc(
        global.dbCon,
        "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDayControl
      ),
      timer
    );
    await updateDoc(
      doc(
        global.dbCon,
        "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDayControl
      ),
      timerSet
    );
    await updateDoc(doc(global.dbCon, "/Usuarios", global.id), timer);
```

Fig. 38. Función que almacena actividad horaria del usuario empleado (parte 1)

```
109   } else if (state == "startBreak") {
110     const timers = {
111       startBreak: time,
112       DescriptionBeforeBreak: Description,
113       ImageBreak: Image,
114     };
115     await updateDoc(
116       doc(
117         global.dbCon,
118         "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDayControl
119       ),
120       timers
121     );
122     const onlyTimers = {
123       startBreak: time,
124       stateBreak: true,
125     };
126     await updateDoc(doc(global.dbCon, "/Usuarios", global.id), onlyTimers);
127     global.stateBreak = true;
128   } else if (state == "startBack") {
129     const timers = {
130       startBack: time,
131     };
132     await updateDoc(
133       doc(
134         global.dbCon,
135         "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDayControl
136       ),
137       timers
138     );
139     await updateDoc(doc(global.dbCon, "/Usuarios", global.id), timers);
140   } else {
141     const timers = {
142       finishTime: time,
143       DescriptionFinishDay: Description,
144       ImageFinish: Image,
145     };
146     const onlyTimers = {
147       finishTime: time,
148     };
149     await updateDoc(
150       doc(
151         global.dbCon,
152         "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDayControl
153       ),
154       timers
155     );
```

Fig. 39. Función que almacena actividad horaria del usuario empleado (parte 2)



```
156     await updateDoc(doc(global.dbCon, "/Usuarios", global.id), onlyTimers);
157     const finishDay = {
158       finishDay: date.getDate(),
159     };
160     await updateDoc(doc(global.dbCon, "/Usuarios", global.id), finishDay);
161   }
162
163 };
```

Fig. 40. Función que almacena actividad horaria del usuario empleado (parte 3)

Tarea 6. Calcular las horas de trabajo de un día del usuario empleado.

Al momento que el usuario da clic en “FINALIZAR”, se ejecuta la función para calcular el tiempo total de trabajo y se almacena en el atributo totalDay.

La lógica que se usa en esta función es transformar cada hora registrada en segundos, sumarlas y luego volver a transformarlas al formato hh:mm:ss para registrar en la base de datos. La Fig. 41 y Fig. 42, describen el proceso matemático de las horas.

```
199   const calculateTotalHours = async () => {
200     let horaFinalRegistro;
201     if (timetowork === "08:00:00") {
202       let horaInicioArray = startWork.split(":");
203       let horaBreakArray = startBreak.split(":");
204       let horaBackArray = startBack.split(":");
205       let horaFinishArray = window.horaFinish.split(":");
206
207       let hour1 = Number(horaInicioArray[0]) * 3600;
208       let min1 = Number(horaInicioArray[1]) * 60;
209       let sec1 = Number(horaInicioArray[2]);
210
211       let horaIncioaSegundos = Number(hour1 + min1 + sec1);
212       //console.log("HoraInicioSegundos: " + horaIncioaSegundos);
213
214       let hour2 = Number(horaBreakArray[0]) * 3600;
215       let min2 = Number(horaBreakArray[1]) * 60;
216       let sec2 = Number(horaBreakArray[2]);
217
218       let horaBreakaSegundos = Number(hour2 + min2 + sec2);
219       //console.log("horaBreakaSegundos", horaBreakaSegundos);
220
221       let hour3 = Number(horaBackArray[0]) * 3600;
222       let min3 = Number(horaBackArray[1]) * 60;
223       let sec3 = Number(horaBackArray[2]);
224
225       let horaBackaSegundos = Number(hour3 + min3 + sec3);
226       //console.log("horaBackaSegundos", horaBackaSegundos);
227
228       let hour4 = Number(horaFinishArray[0]) * 3600;
229       let min4 = Number(horaFinishArray[1]) * 60;
230       let sec4 = Number(horaFinishArray[2]);
231
232       let horaFinishaSegundos = Number(hour4 + min4 + sec4);
233
234       //console.log("horaFinishaSegundos", horaFinishaSegundos);
235
236       let restaHoraBreakInicio = horaBreakaSegundos - horaIncioaSegundos;
237       let restaHorsFinishBack = horaFinishaSegundos - horaBackaSegundos;
238
239       let hoursTotalDay = restaHoraBreakInicio + restaHorsFinishBack;
240
```

Fig. 41. Guardar horas diarias y mensuales (parte 1)

```
240
241         let horaFinal = Math.trunc(hoursTotalDay / 3600);
242         let horaFinalView;
243         if (horaFinal < 10) {
244           horaFinalView = "0" + horaFinal;
245         } else {
246           horaFinalView = horaFinal;
247         }
248
249         let restoHoraFinal = hoursTotalDay % 3600;
250         let minFinal = Math.trunc(restoHoraFinal / 60);
251         let minFinalView;
252         if (minFinal < 10) {
253           minFinalView = "0" + minFinal;
254         } else {
255           minFinalView = minFinal;
256         }
257         let secFinal = restoHoraFinal % 60;
258         let secFinalView;
259         if (secFinal < 10) {
260           secFinalView = "0" + secFinal;
261         } else {
262           secFinalView = secFinal;
263         }
264
265         horaFinalRegistro =
266           horaFinalView + ":" + minFinalView + ":" + secFinalView;
267
268         //console.log("Tiempo trabajado", horaFinalRegistro);
269       } else {
270         let horaInicioArray = startWork.split(":");
271         let horaFinishArray = window.horaFinish.split(":");
272
273         let hour1 = Number(horaInicioArray[0]) * 3600;
274         let min1 = Number(horaInicioArray[1]) * 60;
275         let sec1 = Number(horaInicioArray[2]);
276
277         let horaIncioaSegundos = Number(hour1 + min1 + sec1);
278         //console.log("HoraInicioSegundos: " + horaIncioaSegundos);
279
280         let hour4 = Number(horaFinishArray[0]) * 3600;
281         let min4 = Number(horaFinishArray[1]) * 60;
282         let sec4 = Number(horaFinishArray[2]);
283
284         let horaFinishaSegundos = Number(hour4 + min4 + sec4);
```

Fig. 42. Guardar horas diarias y mensuales (parte 2)

En la Fig. 43, se concluye con el valor de horaFinalRegistro que ya tiene un formato transformado en horas y se guarda con el formato hh:mm:ss detro del documento del usuario empleado que corresponde. Se debe mencionar otro atributo el permission que es un atributo booleano que se coloca en true en caso de que el horario de trabajo sea normal (8 horas laborales), caso contrario se coloca en false porque el horario posee algún tipo de

permiso o es horario materno. Esta validación es a través de la variable timetowork que se trae desde la base de datos.

```
285
286       let RestoHoraInicioFin = horaFinishaSegundos - horaIncioaSegundos;
287
288       let horaFinal = Math.trunc(RestoHoraInicioFin / 3600);
289       let horaFinalView;
290       if (horaFinal < 10) {
291         horaFinalView = "0" + horaFinal;
292       } else {
293         horaFinalView = horaFinal;
294       }
295
296       let restoHoraFinal = RestoHoraInicioFin % 3600;
297       let minFinal = Math.trunc(restoHoraFinal / 60);
298       let minFinalView;
299       if (minFinal < 10) {
300         minFinalView = "0" + minFinal;
301       } else {
302         minFinalView = minFinal;
303       }
304       let secFinal = restoHoraFinal % 60;
305       let secFinalView;
306       if (secFinal < 10) {
307         secFinalView = "0" + secFinal;
308       } else {
309         secFinalView = secFinal;
310       }
311
312       horaFinalRegistro =
313         horaFinalView + ":" + minFinalView + ":" + secFinalView;
314     }
315     let permission;
316     if(timetowork==="08:00:00"){
317       permission=false;
318     } else{
319       permission=true;
320     }
321
322     await updateDoc(
323       doc(
324         db,
325         "Usuarios/" + window.cedulaEmpleado + "/" + month_year + "/" + day_day
326       ),
```

Fig. 43. Guardar horas diarias y mensuales (parte 3)

Desde este punto la función empieza el cálculo de horas mensuales, Fig. 44. A partir de la sentencia try, se accede a la colección "/MONTHLY_REGISTER/" del empleado y se trae el valor de totalHours, que corresponde a las horas mensuales existentes hasta el registro, una vez que se almacena el valor, se procede a recalcular la hora en segundos y se le suma el nuevo valor de hora diaria que se registró anteriormente.

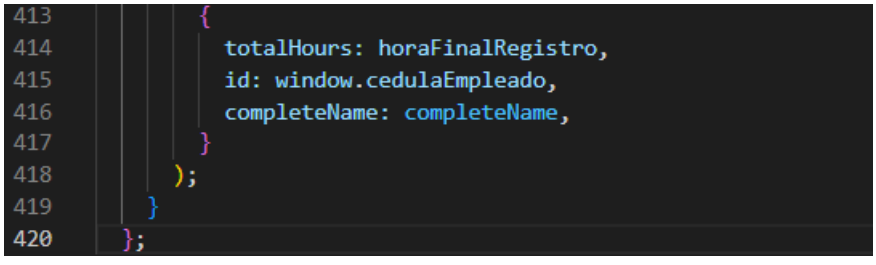
```
      {
        totalDay: horaFinalRegistro,
        permissionType:permission
      }
    );

    try {
      let valorAnterior;
      const docRef2 = doc(
        db,
        "Usuarios/" + window.cedulaEmpleado + "/MONTHLY_REGISTER/" + month_year
      );
      const docSnap2 = await getDoc(docRef2);
      if (docSnap2.exists()) {
        valorAnterior = docSnap2.data().totalHours;
      }

      let horaFinishRegister = valorAnterior.split(":");

      let hour1 = Number(horaFinishRegister[0]) * 3600;
      let min1 = Number(horaFinishRegister[1]) * 60;
      let sec1 = Number(horaFinishRegister[2]);

      let horaEnSegundos = Number(hour1 + min1 + sec1);
      //console.log("HoraInicioSegundos: " + horaEnSegundos);

      let newHoraFinishRegister = horaFinalRegistro.split(":");

      let hour2 = Number(newHoraFinishRegister[0]) * 3600;
      let min2 = Number(newHoraFinishRegister[1]) * 60;
      let sec2 = Number(newHoraFinishRegister[2]);

      let horaEnSegundos2 = Number(hour2 + min2 + sec2);
      //console.log("HoraInicioSegundos: " + horaEnSegundos2);

      let sumaHoras = horaEnSegundos + horaEnSegundos2;

      let horaFinal = Math.trunc(sumaHoras / 3600);
      let horaFinalView;
      if (horaFinal < 10) {
        horaFinalView = "0" + horaFinal;
      } else {
        horaFinalView = horaFinal;
      }
```

Fig. 44. Guardar horas diarias y mensuales (parte 4)

En la Fig. 45 y Fig. 46, ya se obtiene el nuevo cálculo de las horas mensuales y se lo almacena en la base de datos, pero también se controla la posibilidad de que exista un error, es decir si la colección no existe dentro documento del empleado la función se encarga de crearla con el valor de horaFinalRegistro de la Fig. 43. Y a parir de este valor se irá incrementando el conteo de horas mensuales.

```
371
372     let restoHoraFinal = sumaHoras % 3600;
373     let minFinal = Math.trunc(restoHoraFinal / 60);
374     let minFinalView;
375     if (minFinal < 10) {
376       minFinalView = "0" + minFinal;
377     } else {
378       minFinalView = minFinal;
379     }
380     let secFinal = restoHoraFinal % 60;
381     let secFinalView;
382     if (secFinal < 10) {
383       secFinalView = "0" + secFinal;
384     } else {
385       secFinalView = secFinal;
386     }
387
388     const horaMes = horaFinalView + ":" + minFinalView + ":" + secFinalView;
389
390     await updateDoc(
391       doc(
392         db,
393         "Usuarios/" +
394           window.cedulaEmpleado +
395           "/MONTHLY_REGISTER/" +
396           month_year
397       ),
398       {
399         totalHours: horaMes,
400       }
401     );
402   } catch (error) {
403     console.log(firstName  + "    "+lastName)
404     const completeName = firstName + " " + lastName;
405     await setDoc(
406       doc(
407         db,
408         "Usuarios/" +
409           window.cedulaEmpleado +
410           "/MONTHLY_REGISTER/" +
411           month_year
412       ),
```

Fig. 45. Guardar horas diarias y mensuales (parte 5)



```
413             {
414               totalHours: horaFinalRegistro,
415               id: window.cedulaEmpleado,
416               completeName: completeName,
417             }
418           );
419         }
420       };
```

Fig. 46. Guardar horas diarias y mensuales (parte 6)

La suma de horas para el componente móvil es el mismo que se mencionó anteriormente pero se usa la función sumarHoras() donde usa la misma estructura para el nombre del documento y realiza en mismo cálculo matemático para que la suma de horas de realice correctamente. En este caso se manejan los errores en caso de que existan para que no se bloquee el flujo de la aplicación. En la Fig. 47 y Fig. 48, se muestran las transformaciones de horas a minutos y viceversa para poder sumar todo el registro en segundos y almacenar en la base de datos nuevamente en formato hh:mm:ss.

```
export const sumarHoras = async () => {
  // TRAER DE LA BASE LOS DATOS
  let timerInformation = [];
  let date = new Date();
  const dias = [
    "domingo", // 0
    "lunes", // 1
    "martes",
    "miercoles",
    "jueves",
    "viernes",
    "sábado",
  ];
  const numeroDia = new Date().getDay();
  const nombreDia = dias[numeroDia].toUpperCase();
  let saveMonth = getMonth(date.getMonth() + 1) + "_" + date.getFullYear();
  let saveDay;
  if (date.getDate() >= 1 && date.getDate() <= 9) {
    saveDay = nombreDia + "_" + "0" + date.getDate();
  } else {
    saveDay = nombreDia + "_" + date.getDate();
  }

  let startWork;
  let startBreak;
  let startBack;
  let finishDay;
  try {
    const q = doc(
      global.dbCon,
      "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDay
    );
    const docSnap = await getDoc(q);
    timerInformation.push(docSnap.data());
  } catch (error) {

  }
  for (let i = 0; i < timerInformation.length; i++) {
    startWork = timerInformation[i].startWork;
    startBreak = timerInformation[i].startBreak;
    if (startBreak == null || startBreak == undefined) {
      startBreak = "00:00:00";
    }
```

Fig. 47. Calcular horas diarias y mensuales de trabajo (parte 1)

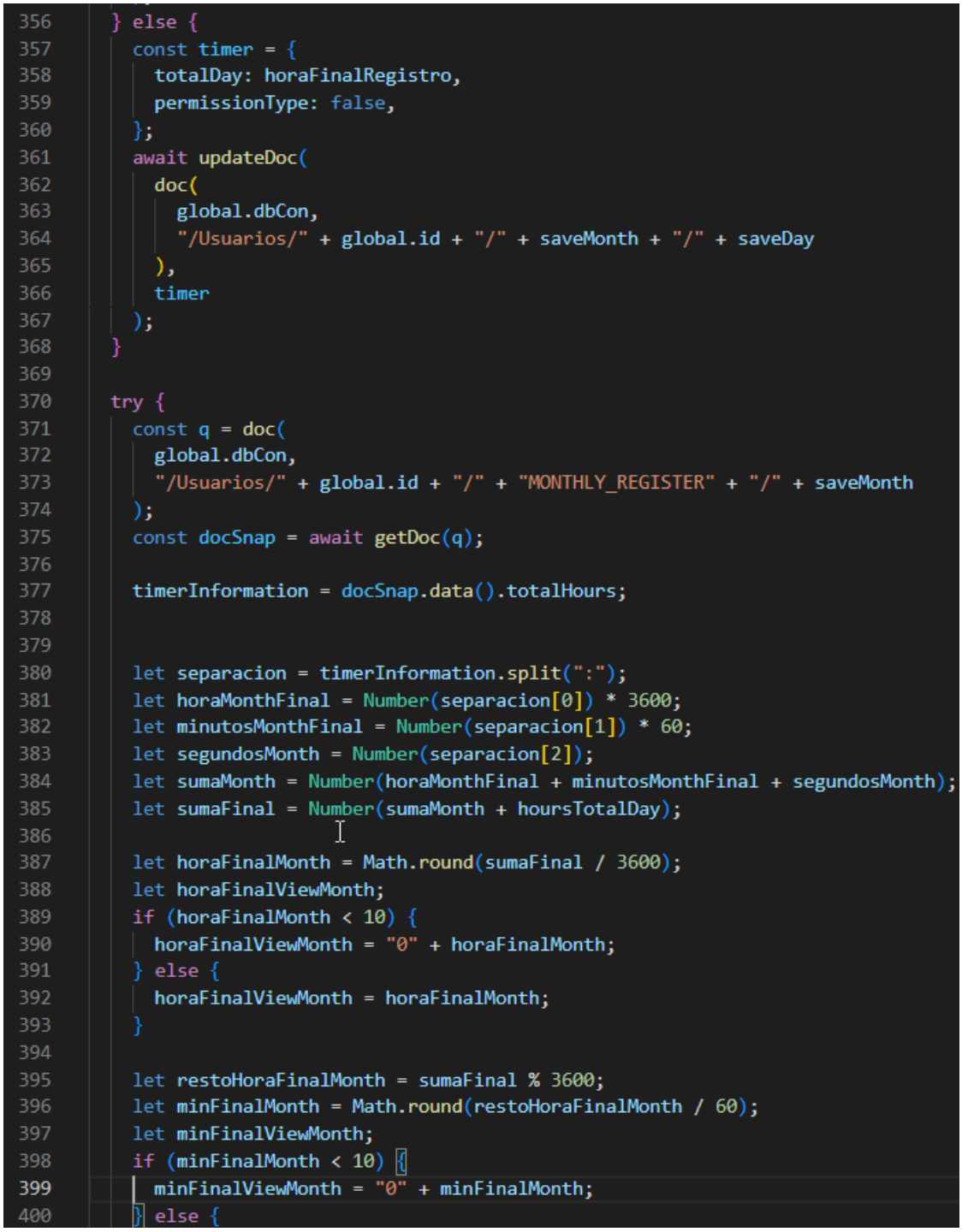
```
272       startBack = timerInformation[i].startBack;
273       if (startBack == null || startBack == undefined) {
274         startBack = "00:00:00";
275       }
276       finishDay = timerInformation[i].finishTime;
277     }
278     let horaInicioArray = startWork.split(":");
279     let horaBreakArray = startBreak.split(":");
280     let horaBackArray = startBack.split(":");
281     let horaFinishArray = finishDay.split(":");
282
283     let hour1 = Number(horaInicioArray[0]) * 3600;
284     let min1 = Number(horaInicioArray[1]) * 60;
285     let sec1 = Number(horaInicioArray[2]);
286
287     let horaIncioaSegundos = Number(hour1 + min1 + sec1);
288
289
290     let hour2 = Number(horaBreakArray[0]) * 3600;
291     let min2 = Number(horaBreakArray[1]) * 60;
292     let sec2 = Number(horaBreakArray[2]);
293
294     let horaBreakaSegundos = Number(hour2 + min2 + sec2);
295
296
297     let hour3 = Number(horaBackArray[0]) * 3600;
298     let min3 = Number(horaBackArray[1]) * 60;
299     let sec3 = Number(horaBackArray[2]);
300
301     let horaBackaSegundos = Number(hour3 + min3 + sec3);
302
303
304     let hour4 = Number(horaFinishArray[0]) * 3600;
305     let min4 = Number(horaFinishArray[1]) * 60;
306     let sec4 = Number(horaFinishArray[2]);
307
308     let horaFinishaSegundos = Number(hour4 + min4 + sec4);
309
310
311
312     let restaHoraBreakInicio = horaBreakaSegundos - horaIncioaSegundos;
313     let restaHorsFinishBack = horaFinishaSegundos - horaBackaSegundos;
314
315     let hoursTotalDay = restaHoraBreakInicio + restaHorsFinishBack;
```

Fig. 48. Calcular horas diarias y mensuales de trabajo (parte 2)

En Fig. 49 y Fig. 50, se muestra el acceso al documento y actualización del campo “totalDay” que representa la suma total diaria de la jornada laboral. En este punto también se debe mencionar que existe la validación del horario, en caso de que sean 4 horas o 6 se coloque el atributo permissionType en true y caso contrario si el horario son 8 horas se colocal en false para poder manejar distintos componentes de la interfaz móvil.

```
316
317   let horaFinal = Math.round(hoursTotalDay / 3600);
318   let horaFinalView;
319   if (horaFinal < 10) {
320     horaFinalView = "0" + horaFinal;
321   } else {
322     horaFinalView = horaFinal;
323   }
324
325   let restoHoraFinal = hoursTotalDay % 3600;
326   let minFinal = Math.round(restoHoraFinal / 60);
327   let minFinalView;
328   if (minFinal < 10) {
329     minFinalView = "0" + minFinal;
330   } else {
331     minFinalView = minFinal;
332   }
333   let secFinal = restoHoraFinal % 60;
334   let secFinalView;
335   if (secFinal < 10) {
336     secFinalView = "0" + secFinal;
337   } else {
338     secFinalView = secFinal;
339   }
340
341   let horaFinalRegistro =
342     horaFinalView + ":" + minFinalView + ":" + secFinalView;
343
344   if (global.timeToWork == "04" || global.timeToWork == "06") {
345     const timer = {
346       totalDay: horaFinalRegistro,
347       permissionType: true,
348     };
349     await updateDoc(
350       doc(
351         global.dbCon,
352         "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDay
353       ),
354       timer
355     );
```

Fig. 49. Calcular horas diarias y mensuales de trabajo (parte 3)



```
356   } else {
357     const timer = {
358       totalDay: horaFinalRegistro,
359       permissionType: false,
360     };
361     await updateDoc(
362       doc(
363         global.dbCon,
364         "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDay
365       ),
366       timer
367     );
368   }
369
370   try {
371     const q = doc(
372       global.dbCon,
373       "/Usuarios/" + global.id + "/" + "MONTHLY_REGISTER" + "/" + saveMonth
374     );
375     const docSnap = await getDoc(q);
376
377     timerInformation = docSnap.data().totalHours;
378
379
380     let separacion = timerInformation.split(":");
381     let horaMonthFinal = Number(separacion[0]) * 3600;
382     let minutosMonthFinal = Number(separacion[1]) * 60;
383     let segundosMonth = Number(separacion[2]);
384     let sumaMonth = Number(horaMonthFinal + minutosMonthFinal + segundosMonth);
385     let sumaFinal = Number(sumaMonth + hoursTotalDay);
386
387     let horaFinalMonth = Math.round(sumaFinal / 3600);
388     let horaFinalViewMonth;
389     if (horaFinalMonth < 10) {
390       horaFinalViewMonth = "0" + horaFinalMonth;
391     } else {
392       horaFinalViewMonth = horaFinalMonth;
393     }
394
395     let restoHoraFinalMonth = sumaFinal % 3600;
396     let minFinalMonth = Math.round(restoHoraFinalMonth / 60);
397     let minFinalViewMonth;
398     if (minFinalMonth < 10) {
399       minFinalViewMonth = "0" + minFinalMonth;
400     } else {
```

Fig. 50. Calcular horas diarias y mensuales de trabajo (parte 4)

En este caso para registrar la hora total mensual, también se valida, en caso de que no exista el documento de registro mensual se crea con el primer valor del registro diario y caso contrario si ya existe el documento la función debe ir sumando las horas que se van registrado de forma diaria tal como se muestra en Fig. 51.

```
401         minFinalViewMonth = minFinalMonth;
402       }
403       let secFinalMonth = restoHoraFinalMonth % 60;
404       let secFinalViewMonth;
405       if (secFinalMonth < 10) {
406         secFinalViewMonth = "0" + secFinalMonth;
407       } else {
408         secFinalViewMonth = secFinalMonth;
409       }
410       let month =
411         horaFinalViewMonth + ":" + minFinalViewMonth + ":" + secFinalViewMonth;
412
413       const monthTime = {
414         totalHours: month,
415         id: global.id,
416         completeName: global.name + " " + global.lastName,
417       };
418       await updateDoc(
419         doc(
420           global.dbCon,
421           "/Usuarios/" + global.id + "/" + "MONTHLY_REGISTER" + "/" + saveMonth
422         ),
423         monthTime
424       );
425     } catch (error) {
426       const monthTime = {
427         totalHours: horaFinalRegistro,
428         id: global.id,
429         completeName: global.name + " " + global.lastName,
430       };
431       await setDoc(
432         doc(
433           global.dbCon,
434           "/Usuarios/" + global.id + "/" + "MONTHLY_REGISTER" + "/" + saveMonth
435         ),
436         monthTime
437       );
438     }
439   };
```

Fig. 51. Calcular horas diarias y mensuales de trabajo (parte 5)

Sprint 3. Funciones principales del usuario Administrador

En este punto, se colocan las funciones del administrador dentro del sistema web, es decir un usuario con este privilegio puede revisar, modificar su información, gestionar a los usuarios empleados y consultar actividad laboral de forma diaria o mensual, según corresponda.

Lista de tareas:

- Tarea 1. Mostrar la información de todos los usuarios.
- Tarea 2. Eliminar un usuario de la base de datos

- Tarea 3. Mostrar el historial mensual de trabajo de los usuarios.

Tarea 1. Mostrar la información de todos los usuarios.

Para obtener la información de todos los usuarios registrados en la base de datos es necesario traer la colección completa de “Usuarios” y guardarla en una variable de estado tipo array. El siguiente servicio retorna la información de los usuarios con rol “Empleado”, se observa en la Fig. 52.

```
const getPersonalInformation = async () => {
  let usuarios = [];
  const docRefCargos = query(collection(db, "Usuarios/"));
  const docSnap = await getDocs(docRefCargos);

  docSnap.forEach((doc) => {
    if (doc.data().rol === "Empleado") {
      usuarios.push(doc.data());
    }
  });

  return usuarios;
};

export default getPersonalInformation;
```

Fig. 52. Traer la información de los usuarios empleados

La función se debe llamar dentro del useEffect, de esta manera la colección de los usuarios se almacena en “usuarios”, como se visualiza en la Fig. 53 y de esta manera se puede usar para mostrar toda la información en la interfaz, realizar búsquedas específicas o realizar paginación en caso de que sean muchos usuarios registrados.

Línea de código: const [usuario, setUsuarios] =useState([])

```
getPersonalInformation().then((usuarios) => {
  setUsuarios(usuarios);
  setItems([...usuarios].splice(0, ITEMS_PER_PAGE));
});
```

Fig. 53. Setear la información en la variable de estado

Tarea 2. Eliminar un usuario de la base de datos

Al obtener toda la información de todos los usuarios, se puede acceder a los datos de cada usuario y el administrador puede actualizar la información después de capturar los campos de un formulario con la función de la Fig. 54.

```
const updateUser = async () => {
  await updateDoc(doc(db, "Usuarios/" + cedula), {
    secondName: secondName,
    secondLastName: secondLastName,
    birthday: date,
    address: address,
    civilStatus: civilstatus,
    gender: gender,
    phoneHouse: phoneHouse,
    phoneNumber: phoneNumber,
    lastName: lastName,
    email: email,
    firstName: firstName,
    id:cedula,
    timetowork: timetowork
  });
  clickedupdate();

};
```

Fig. 54. Actualizar la información del usuario

De la misma manera, el administrador tiene el permiso para eliminar a un usuario con la siguiente función asincrónica de la Fig. 55.

```
const deleteUser = async () => {
  await deleteDoc(doc(db, "Usuarios/" + cedula));
  await deleteDoc(doc(db, "Roles/" + email));
  console.log("Usuario" + cedula + " eliminado");
  clickedDelete();
};
```

Fig. 55. Eliminar usuario

Tarea 3. Mostrar el historial mensual de trabajo de los usuarios.

En este punto, es necesario traer toda la colección de usuarios para posterior a eso traer la colección del mes que desea observar el administrador, tal como indica la Fig. 56.

```
getPersonalInformation().then((usuarios) => {
  setUsuarios(usuarios);
});
```

Fig. 56. Setear la información en una variable de estado

A continuación, en la Fig. 57, se realiza la consulta según el mes que ingrese el administrador o gerente a través de un input de tipo month. El retorno son los usuarios que trabajaron en dicho mes junto con sus horas totales con una opción en donde se detalla la consulta diaria de cada usuario. Es decir, si se elige el mes Abril, se le muestran 3 usuarios con el registro de ese mes y sus horas totales y en cada usuario se setea la cedula para que se muestra un detalle individual y diario del mes consultado.

```
122  const consulta = async () => {
123    empleados.splice(0, empleados.length);
124    setEmpleados([]);
125    setIsLoading(true);
126    if (date == null) {
127      setIsLoading(false);
128      swal({
129        text: "Por favor, ingrese un mes para realizar la consulta.",
130        icon: "info",
131        button: "Aceptar",
132      });
133    }
134    const horasTrabajoTmp = date.split("-");
135    //console.log("date", horasTrabajoTmp);
136    if (horasTrabajoTmp[1] < 10) {
137      const dateMonth = horasTrabajoTmp[1].split("0");
138      monthName = monthNames[dateMonth[1] - 1];
139    } else {
140      const dateMonth = horasTrabajoTmp[1];
141      monthName = monthNames[dateMonth - 1];
142    }
143
144    month_year = monthName + "_" + horasTrabajoTmp[0];
145
146    //console.log("date", month_year);
147    let i;
148    for (i = 0; i < usuarios.length; i++) {
149      const docRef = doc(
150        db,
151        "Usuarios/" + usuarios[i].id + "/MONTHLY_REGISTER/" + month_year
152      );
153      const docSnap = await getDoc(docRef);
154
155      if (docSnap.exists()) {
156        empleados.push(docSnap.data());
157      }
158    }
159    //console.log("Empleados que trabajaron en este mes", empleados)
160    setEmpleados(empleados);
161    setIsLoading(false);
162
163    if (empleados.length === 0) {
164      swal({
165        text: "No existen registros en este mes.",
166        icon: "info",
167        button: "Aceptar",
168      });
169    }
170  };
```

Fig. 57. Consultar en la base el mes y las horas de trabajo

Cada vez que se inicia una búsqueda es necesario eliminar el arreglo anterior para que no se vaya agregando datos innecesarios. Se obtiene el mes con la función split para transformar el string en un arreglo que almacena mes y año en números. Luego se establece el valor de month_year para saber en qué colección se debe hacer la consulta. Finalmente, se accede a la colección "/MONTHLY_REGISTER/" que corresponde y se setean los usuarios del mes de consulta.

Para la consulta diaria de un mes especifico, se usa la consulta de la Fig. 58 en donde se accede al mes específico dentro del documento del empleado y le detalla la actividad diaria de un solo empleado.

```
const consulta = async () => {
  empleados.splice(0, empleados.length);
  setEmpleados([]);

  const horasTrabajoTmp = mes.split("-");
  //console.log("date", horasTrabajoTmp);
  if (horasTrabajoTmp[1] < 10) {
    const dateMonth = horasTrabajoTmp[1].split("0");
    monthName = monthNames[dateMonth[1] - 1];
  } else {
    const dateMonth = horasTrabajoTmp[1];
    monthName = monthNames[dateMonth - 1];
  }

  month_year = monthName + "_" + horasTrabajoTmp[0];

  //console.log("date", month_year);

  const docRef = query(
    collection(db, "Usuarios/" + cedula + "/" + month_year)
  );
  const docSnap = await getDocs(docRef);

  docSnap.forEach((doc) => {
    empleados.push(doc.data());
  });

  setEmpleados(empleados);
  //console.log("arreglo de empleados", empleados);
};
```

Fig. 58. Consulta diaria en un mes especifico de un empleado

El usuario empleado también puede realizar consultas diarias o mensuales, usando la aplicación móvil y lo hace a través de la función getTimer() y getTimerMonth() respectivamente. La primera se refleja en la Fig. 59 y la segunda en la Fig. 60. La función 1 (getTimer()) recibe otra función (refreshScreen) que de tipo set y el otro parámetro es en sí el día de consulta (dayNumer). Se establece la misma estructura para consultar en el documento correcto, entonces se accede a la base de datos y se guarda la información en un arreglo temporal que luego se almacena en refreshScreen, en caso de que no haya información del día de consulta se seta en null el arreglo.

```
export const getTimers = async (refreshScreen, dayNumber) => {

  let date = new Date();
  const dias = [
    "domingo", // 0
    "lunes", // 1
    "martes",
    "miercoles",
    "jueves",
    "viernes",
    "sábado",
  ];
  const numeroDia = dayNumber.getDay();
  const nombreDia = dias[numeroDia].toUpperCase();
  let saveMonth =
    getMonth(dayNumber.getMonth() + 1) + "_" + dayNumber.getFullYear();
  let saveDay;
  if (dayNumber.getDate() >= 1 && dayNumber.getDate() <= 9) {
    saveDay = nombreDia + "_" + "0" + dayNumber.getDate();
  } else {
    saveDay = nombreDia + "_" + dayNumber.getDate();
  }

  const q = doc(
    global.dbCon,
    "/Usuarios/" + global.id + "/" + saveMonth + "/" + saveDay
  );
  const docSnap = await getDoc(q);
  let timerInformation = [];
  timerInformation.push(docSnap.data());
  if (timerInformation[0] == undefined) {
    refreshScreen(null);
  } else {
    refreshScreen(timerInformation[0]);
  }

};
```

Fig. 59. Consulta diaria de las horas de trabajo

La función 2 (getTimersMoth), de igual manera recibe otra función y el día de consulta. Primero se encarga de establecer el mes de búsqueda a través del día y con esta variable accede al documento “MONTHLY_REGISTER” del usuario y guarda los datos del documento en un arreglo temporal el cual solo necesita el atributo totalHours que hace referencia a la suma mensual de trabajo de un usuario.

```
export const getTimersMonth = async (refreshScreen, dayNumber) => {

  const monthDate =
    getMonth(dayNumber.getMonth() + 1) + "_" + dayNumber.getFullYear();

  const q = doc(
    global.dbCon,
    "/Usuarios/" + global.id + "/" + "MONTHLY_REGISTER" + "/" + monthDate
  );
  const docSnap = await getDoc(q);
  let timerInformation = [];
  try {
    timerInformation = docSnap.data().totalHours;
  }catch (error) {
    timerInformation=0;
  }


  if (timerInformation == undefined) {
    refreshScreen(null);
  } else {
    refreshScreen(timerInformation);
  }
};
```

Fig. 60. Consulta mensual de horas de trabajo

Sprint 4. Funciones adicionales para los distintos roles

El objetivo de este Sprint corresponde a las funciones más específicas del usuario, lo que quiere decir que todos los usuarios deben verificar sus correos electrónicos, pueden cambiar sus claves y en caso de ser usuario empleado se ve en la obligación de cargar documentos para poder registrar su actividad y debe cumplir con un horario de 4, 6 u 8 horas según corresponda. Por otra parte, el usuario administrador es quien gestiona las opciones de cargos disponibles.

Lista de tareas:

- Tarea 1. Validar un correo válido para iniciar sesión.
- Tarea 2. Permitir al usuario cambiar su clave de acceso.
- Tarea 3. Establecer horario laboral
- Tarea 4. Validar que el empleado cumpla con el horario laboral.
- Tarea 5. Obligar subir la documentación al usuario.
- Tarea 6. CRUD para manejo de cargos que posee un empleado o un gerente.

Tarea 1. Validar un correo válido para iniciar sesión.

Cuando un empleado o gerente es registrado en el sistema, necesita que su correo exista y sea válido para que le llegue un correo de verificación. En la Fig. 61, se muestra la función de Firebase que nos permite crear un usuario con email y contraseña y después de esto procede a enviar una verificación.

```
createUserWithEmailAndPassword(auth2, emailEmp, password).then(() => {
  sendEmailVerification(auth2.currentUser)
    .then(() => {})
    .catch((err) =>
      swal({
        text: err.message,
        icon: "success",
        button: "Aceptar",
      })
    );
});
```

Fig. 61. Función crear usuario y enviar correo de verificación

Este paso es importante para que el usuario inicie sesión si y solo si cumple con haber confirmado su correo electrónico a través de una notificación que envía Firebase. Esta validación inicia con el seteo de la variable window.emailVerified que está en el archivo UserProvider.js como un observable, como se muestra en la Fig. 62.

```
React.useEffect(() => {
  //se ejecuta pero luego se destruye, esto es un observable
  const unsusbribe = onAuthStateChanged(auth, (user) => {
    console.log("User", user);

    if (user) {
      const { email, photoURL, displayName, uid, emailVerified } = user;
      setUser({ email, photoURL, displayName, uid, emailVerified });
      window.emailVerified = emailVerified;
    } else {
      setUser(null);
    }
  });

  return () => unsusbribe();
}, []);
```

Fig. 62. Definir el valor de la variable window.emailVerified a través de UserProvider

Una vez que se obtiene el valor de emailVerified, true si el correo está verificado y false en caso contrario, se procede con la validación al momento de iniciar sesión, se puede observar la línea de código en la Fig. 63.

```
if (window.emailVerified === true) {
  if (rol[0] === "admin") {
    navigate("/admin");
  } else if (rol[0] === "Empleado") {
    navigate("/empleado");
  } else if (rol[0] === "Gerente") {
    navigate("/manager");
  }
} else {
  swal({
    text: "Correo no verificado.",
    icon: "warning",
    button: "Aceptar",
  });

  signOutUser(email, password);
  navigate("/login");
}
```

Fig. 63. Validación según el valor de emailVerified

Si el resultado es true, logra acceder a las rutas según el rol que corresponda, pero si el resultado es false salta una alerta de advertencia y no permite que el usuario inicie sesión.

Para la sección móvil se ocupa la misma variable de emailVerified para la validación, se la llama al iniciar sesión, tal como muestra la Fig. 64.

```
onPress={() => {
  signInWithEmailAndPassword(auth, emailUser, password)
    .then(async (userCredential) => {
      if(userCredential.user.emailVerified==false){
        setEmail(true);
        setActive(true);
      }else{
        setStateModal(true);
        setEmail(false);
        global.email = emailUser;
        await getPersonalRol();
        if (global.rol == "Empleado") {
          await getDocumentsData();
          await createTask();
          navigation.navigate("TIMER");
          setStateModal(false);
        } else {
          navigation.navigate("LOGINS");
          setStateModal(false);
        }
      }
    })
    .catch((error) => {
      setEmail(false);

      setActive(true);
      setStateModal(false);
    });
}}
```

Fig. 64. Validar inicio de sesión con verificación de correo

La continuación del flujo de inicio de sesión del usuario es necesario tener el rol de “Empleado”, y eso se da a través de la función getPersonalRol() que primero accede al documento “Roles/ejemplo@mail.com”" donde se trae un usuario específico según el usuario haya iniciado sesión y se setea el valor del rol que tiene dentro del sistema.

```
// EXTRAER DATOS DEl ROL DEL USUARIO
export const getPersonalRol = async (Show) => {
  const personalRol = doc(global.dbCon, "/Roles/" + global.email);
  const docSnapRol = await getDoc(personalRol);
  let Rol = docSnapRol.data().id;
  let Trabalo = docSnapRol.data().rol;
  // TRAER INFORMATION
  global.rol = Trabalo;
  if (Trabalo == "Empleado") {
    const q = doc(global.dbCon, "/Usuarios/" + Rol);
    const docSnap = await getDoc(q);
    let personalInformation = [];

    personalInformation.push(docSnap.data());
    for (let i = 0; i < personalInformation.length; i++) {
      global.id = personalInformation[i].id;
      global.finishDay = personalInformation[i].finishDay;
      global.workState = personalInformation[i].workingState;
      global.name = personalInformation[i].firstName;
      global.lastName = personalInformation[i].lastName;
      global.picture = personalInformation[i].imageUser;
      global.totalMonth = personalInformation[i].totalMonth;
      global.stateBreak = personalInformation[i].stateBreak;
      global.workStation = personalInformation[i].workstation;
      global.timetoWork = personalInformation[i].timetowork;
    }

    let time=global.timetoWork.split(':')
    global.timeToWork=time[0]
    let date = new Date();
    // IGUALA SI EL DIA ACTUAL ES IGUAL AL DIA ALMACENADO EN LA BASE
    if (global.finishDay == null || global.workState == null) {
      await updateStateWork("NOTWORKING");
      const finishDay = {
        finishDay: date.getDate(),
        totalMonth:0,
      };
      await updateDoc(doc(global.dbCon, "/Usuarios", global.id), finishDay);
    } else {
      if (date.getDate() == global.finishDay) {
      } else {
        await updateStateWork("NOTWORKING");
        global.workState = "NOTWORKING";
        const finishDay = {
          startWork: 0,
          startBreak: 0,
          startBack: 0,
          finishTime:0,
          finishDay: date.getDate(),
          stateBreak: false,
        };
        global.stateBreak=false,
        await updateDoc(doc(global.dbCon, "/Usuarios", global.id), finishDay);
      }
    }
  }
  if(global.totalMonth==null){
    global.totalMonth =0

  }

};
```

Fig. 65. Información del usuario según el rol empleado

Tarea 2. Permitir al usuario cambiar su clave de acceso.

Para cambiar la contraseña existen dos panoramas, el primer se da cuando el usuario ha olvidado su clave y necesita recuperarla y es necesario que el usuario solo proporcione su

correo electrónico para que se le envíe un enlace dónde puede reestablecer su clave, tal como se muestra en la Fig. 66.

```
sendPasswordResetEmail(auth, email)
  .then(() => {
    swal({
      text: "Se ha enviado un enlace a su correo electrónico para reestablecer su contraseña.",
      icon: "success",
      button: "Aceptar",
    });
    navigate("/login");
  })
  .catch((error) => {
    const errorCode = error.code;
    const errorMessage = error.message;
  });
```

Fig. 66. Función para resetear contraseña

La notificación que llega es de este tipo como se muestra en la Fig. 67.

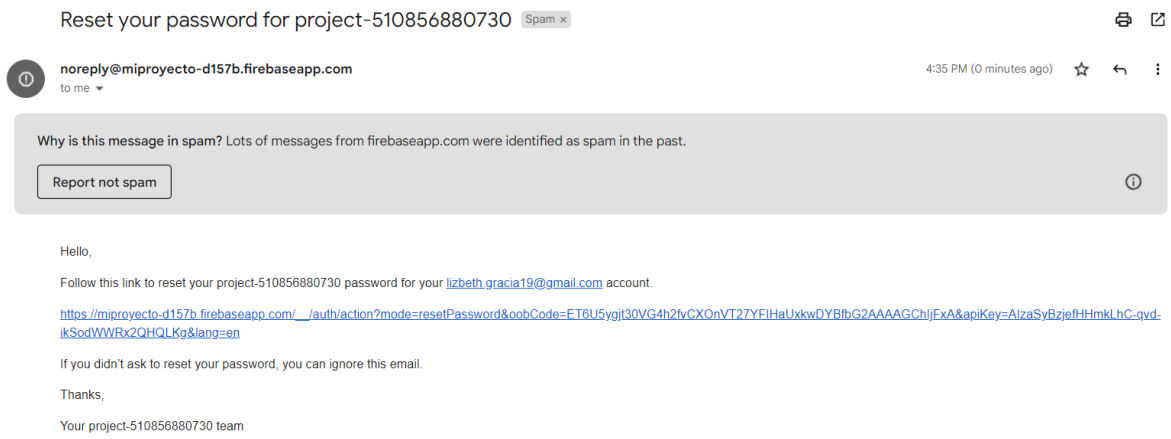


Fig. 67. Correo de restablecimiento de contraseña

Otro panorama es que el usuario si conoce su clave personal, pero quiere cambiarla después de que haya iniciado sesión. Por ende, en la interfaz de usuario se le proporciona una opción para realizar este cambio a través de un formulario que recibe 3 parámetros, la clave actual del usuario que inició sesión, la nueva clave y la confirmación de la misma.

Por lo mencionado, el método que se describe en la Fig. 68, muestra el proceso para cambiar la clave de un usuario. Necesita dos parámetros, la contraseña actual y la nueva contraseña, entonces el método obtiene el usuario que tiene la sesión activa y a través de reauthenticatedWidhCredential logra validar si es la clave correcta o no, además el método posee una validación extra, es decir, en caso de que la nueva contraseña y la confirmación de esta no coincidan no permita ejecutar el método de Firebase updatePassword. Si cumple con lo mencionado anteriormente mostrará una alerta de operación exitosa.

```
const resetUserPassword = async (password, newPassword) => {
  const user = auth.currentUser;
  const cred = EmailAuthProvider.credential(user.email, password);
  try {
    await reauthenticateWithCredential(user, cred);
    if (newPassword === newPassword2) {
      await updatePassword(auth.currentUser, newPassword);
      swal({
        text: "Contraseña actualizada correctamente.",
        icon: "success",
        button: "Aceptar",
      });
      navigate("/empleado");
    } else {
      swal({
        text: "La clave no se actualizó correctamente porque las nuevas contraseñas no coinciden.",
        icon: "error",
        button: "Aceptar",
      });
    }


  } catch (e) {
    if (e.code == "auth/wrong-password") {
      swal({
        text: "Contraseña actual inválida.",
        icon: "error",
        button: "Aceptar",
      });
    }
  }
};
```

Fig. 68. Función de cambio de contraseña con la sesión iniciada

Tarea 3. Establecer horario laboral

Al crear un usuario con rol empleado, este se crea por defecto con el horario normal de trabajo, es decir, tiene la obligación de cumplir con 8 horas laborales, pero el usuario administrador o gerente pueden cambiar este horario, las opciones disponibles son: horario normal, horario materno con 6 horas de trabajo y horario con permisos solo con 4 horas de trabajo. Al momento de actualizar la información de perfil de un empleado desde la vista de administrador o gerente se agrega un nuevo atributo “timetowork” para que se guarde en la base de datos y luego se pueda acceder al mismo y usarlo para validar otras funcionalidades. El nuevo atributo se observa en la Fig. 69.

```
const updateUser = async () => {
  await updateDoc(doc(db, "Usuarios/" + cedula), {
    secondName: secondName,
    secondLastName: secondLastName,
    birthday: date,
    address: address,
    civilStatus: civilstatus,
    gender: gender,
    phoneHouse: phoneHouse,
    phoneNumber: phoneNumber,
    lastName: lastName,
    email: email,
    firstName: firstName,
    id: cedula,
    timetowork: timetowork,
  });
  setModalOnInformation(false);
  setChoiceInformation(false);
  setChoiceUsers(false);
  setModalOnUsers(false);
  swal({
    text: "Usuario actualizado correctamente.",
    icon: "success",
    button: "Aceptar",
  });
};
```

Fig. 69. Nuevo atributo asignado “timetowork”

Tarea 4. Validar que el empleado cumpla con el horario laboral.

A través de la variable de estado timetowork se le atribuye el valor de horario de trabajo que le corresponde al usuario según lo que este registrado en la base de datos usando el método de la Fig. 70.

```
const informationFunction = async (cedula) => {
  const docRef = doc(db, "Usuarios/" + cedula);
  const docSnap = await getDoc(docRef);

  if (docSnap.exists()) {
    setRol(docSnap.data().rol);
    setWorkingState(docSnap.data().workingState);
    setFirstName(docSnap.data().firstName);
    setLastName(docSnap.data().lastName);
    setTimeToWork(docSnap.data().timetowork);
  } else {
    // doc.data() will be undefined in this case
    console.log("No se pudo leer el documento");
  }
};
```

Fig. 70. Traer el nuevo valor desde Firebase y setear en la variable de estado

Por lo tanto, con este valor se activa o se desactiva el botón de finalizar según corresponda, tal como muestra la Fig. 71. Cabe resaltar que el inicio de la jornada se considera desde las 07:00 am.

```
const validateButtonOut = () => {
  if (timetoWork === "08:00:00" && time > "15:50:00") {
    return false;
  } else if (timetoWork === "06:00:00" && time > "12:50:00") {
    return false;
  } else if (timetoWork === "04:00:00" && time > "10:50:00") {
    return false;
  } else {
    return true;
  }
};
```

Fig. 71. Definir horario para que el botón se active

Tarea 5. Obligar subir la documentación al usuario.

Después que el usuario empleado verifica su correo, tiene acceso al sistema y se le da la posibilidad registrar sus horas de trabajo, pero para empezar es requisito cargar los documentos obligatorios. Para este proceso se verifica la existencia del documento en la base de datos. En la Fig. 72, se observa a un usuario que ya ha subido los documentos y por ende se ha creado la colección.

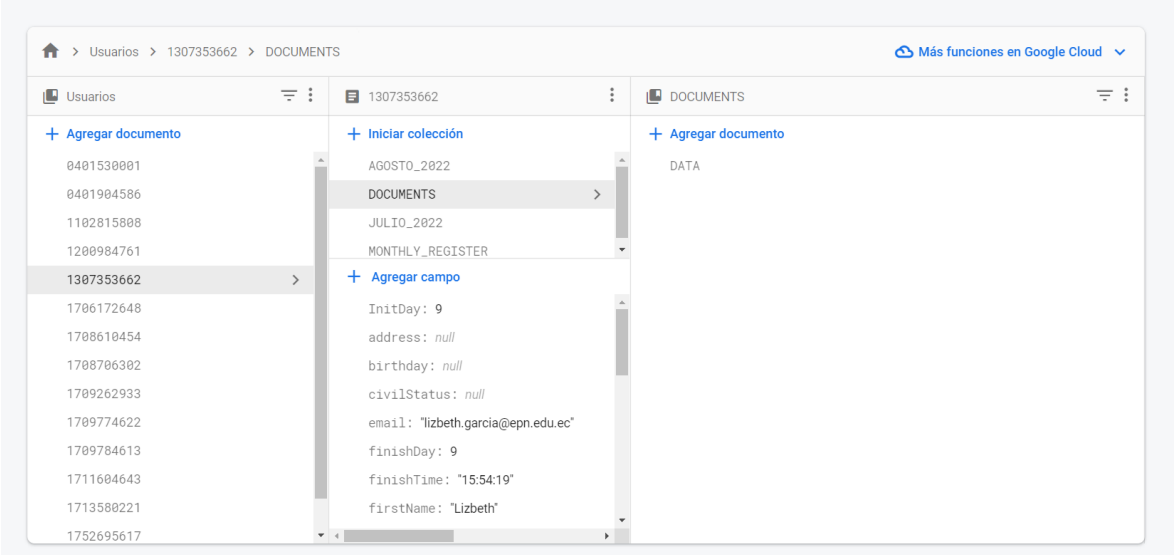


Fig. 72. Estructura de la base de datos cuando sube los documentos importantes

Si la colección ya existe el botón para empezar con el registro de horas se activa, caso contrario permanece inactivo. En la Fig. 73, se visualiza la verificación del documento dentro de la colección de Firestore.

```
const existsDocuments = async (cedula) => {
  const docRef = doc(db, "Usuarios/" + cedula + "/DOCUMENTS/DATA");
  const docSnap = await getDoc(docRef);

  if (docSnap.exists()) {
    if (data.length == 0) {
      swal({
        text: "Los documentos ya han sido registrados, pero puede actualizarlos en cualquier momento. Tome en cuenta que si desea actualizar debe registra
        icon: "info",
        button: "Aceptar",
      });
    }

    setEstado(true);
    setMensaje("Documento ya registrado.");
  } else {
    swal({
      text: "Los documentos son un requisito para poder empezar a registrar sus horas de trabajo.",
      icon: "info",
      button: "Aceptar",
    });
  }
};
```

Fig. 73. Validar la existencia del documento “DATA” de la Fig. 72

Tarea 6. CRUD para manejo de cargos que posee un empleado o un gerente.

El usuario administrador es quien tiene los permisos necesarios para gestionar los cargos disponibles dentro del sistema. Por lo tanto, es capaz de registrar un nuevo cargo, eliminarlo o actualizarlo para que pueda ser accedido desde el formulario “Registro de usuario” y según los cargos disponibles el gerente puede seleccionar dicho atributo.

En la Fig. 74, se describe el método para agregar un nuevo cargo a la base de datos. Se debe mencionar que el id del documento se va incrementando según el último id registrado. Es decir, si el último cargo tiene id “Cargo_1”, el método se encarga de separar la cadena de texto en “Cargo” y “1”, se toma el segundo valor y se incrementa en uno para que el nuevo cargo se registre como “Cargo_2” y en caso de que no haya ningún cargo registrado por defecto se coloca el id “Cargo_1”.

```
const agregarCargos = async (valorCargo) => {
  let arregloTmp = [];
  for (let i = 0; i < todosCargos.length; i++) {
    let separarCargo = todosCargos[i].referencia.split("_");
    arregloTmp.push(Number(separarCargo[1]));
  }

  let num = Math.max(...arregloTmp);

  let numeroCargo = Number(num) + 1;
  let idCargo;
  if (numeroCargo > 0) {
    idCargo = "Cargo_" + numeroCargo;
  } else {
    idCargo = "Cargo_1";
  }

  await setDoc(doc(db, "Cargos/" + idCargo), {
    referencia: idCargo,
    id: valorCargo,
  });

  swal({
    text: "Cargo agregado correctamente.",
    icon: "success",
    button: "Aceptar",
  });

  setCargos("");
  getCargos();
};
```

Fig. 74. Agregar nuevo cargo

De tal manera que en la base de datos se registra con la estructura de la Fig. 75.

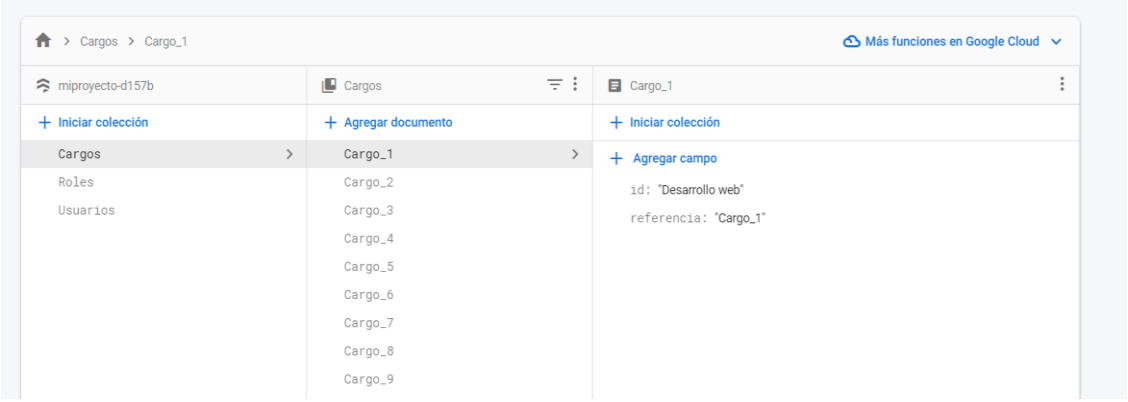


Fig. 75. Estructura de la base de datos de “Cargos”

En la Fig. 76, se muestra el método de eliminar cargo a través del id que le corresponde.

```
const eliminarCargos = async (variableCargo) => {
  await deleteDoc(doc(db, "Cargos/" + variableCargo));
  getCargos();
  swal({
    text: "Cargo eliminado correctamente.",
    icon: "info",
    button: "Aceptar",
  });
};
```

Fig. 76. Eliminar cargo

En la Fig. 77, se observa el método para actualizar un cargo que recibe el id del cargo y el nuevo valor para que se actualice en la base de datos y muestra alerta exitosa.

```
const actualizarCargo = async (id, nuevoCargo) => {
  await updateDoc(doc(db, "Cargos/" + id), {
    id: nuevoCargo,
  });

  swal({
    text: "Cargo actualizado.",
    icon: "success",
    button: "Aceptar",
  });

  getCargos();
};
```

Fig. 77. Actualizar cargo

Finalmente, en la Fig. 78, se observa el método que trae todos los cargos registrados en la base de datos y los mismos se setean en un arreglo para que puedan mostrarse en la interfaz del usuario.

```
const getCargos = async () => {
  let cargosTmp = [];
  const docRefCargos = query(collection(db, "Cargos/"));
  const docSnap = await getDocs(docRefCargos);

  docSnap.forEach((doc) => {
    cargosTmp.push(doc.data());
  });

  setTodosCargos(cargosTmp);
};
```

Fig. 78. Leer todos los cargos registrados

Sprint 5. Pruebas

Finalmente, este Sprint tiene como objetivo validar el correcto funcionamiento del componente Backend, es decir que describe las pruebas unitarias, de seguridad y de carga de la aplicación para verificar que sea optima y cumpla con los requerimientos establecidos al inicio de planteamiento del presente proyecto.

Lista de tareas:

- Tarea 1. Pruebas unitarias
- Tarea 2. Pruebas de seguridad
- Tarea 3. Pruebas de carga

Tarea 1. Pruebas unitarias

Las pruebas unitarias se iban ejecutando conforme se iba cumpliendo un requerimiento y se validan conforme iba fluyendo el componente Frontend y móvil junto con la lógica del sistema, es decir, el Backend.

Todos los métodos de Firebase fueron aplicados conforme la lógica del negocio lo iba necesitando. Cada consulta y escritura sobre la base de datos se realiza exitosamente. Evidencia de esto es la Fig. 79, donde se muestran todas las operaciones exitosas a la base de datos.

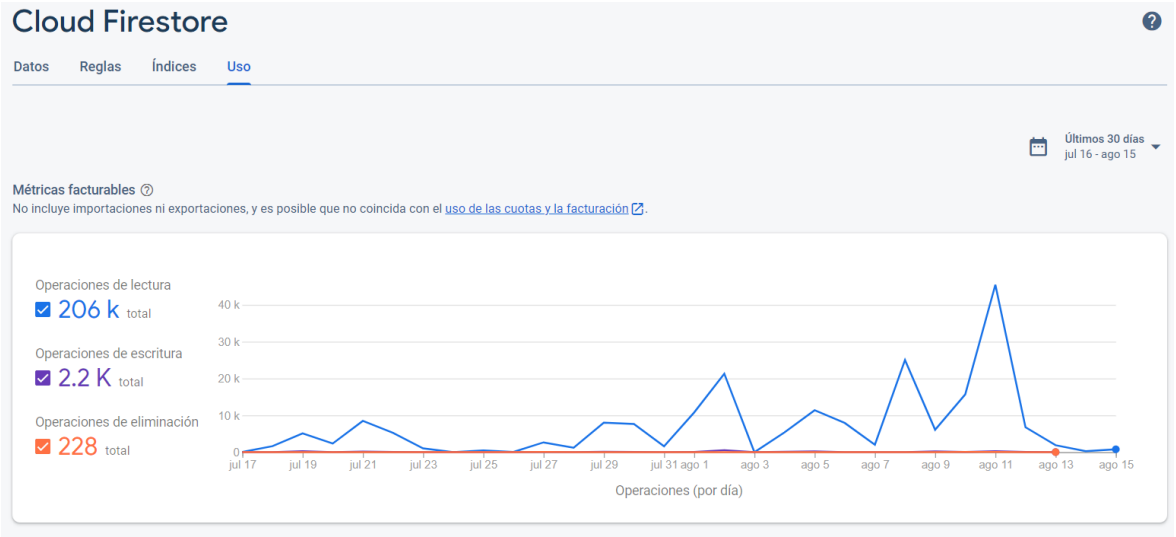


Fig. 79. Operaciones en la base de datos

Tarea 2. Pruebas de seguridad

Para este apartado, se debe tomar en cuenta que el sistema web cuenta con 3 perfiles de usuario por lo tanto es importante manejar la seguridad en la navegación del usuario según el rol que se le haya asignado, es decir. Si un usuario se registró como empleado no tienen acceso a ningún componente o función del usuario administrador o gerente. En caso de la aplicación móvil solo lo usuarios con el rol de empleado pueden acceder al sistema. Para esto, se establece una colección principal en la base de datos denominada “Roles”, donde los documentos se manejan según un id único, el correo electrónico, y se lee la base de datos, al iniciar sesión se valida el correo y el rol que se ha registrado y a través de esto se le redirecciona a una interfaz específica para cumplir con el objetivo de cada usuario.

En la Fig. 80, se muestra el servicio que devuelve la información del documento que se encuentra en “/Roles/” correspondiente al id de correo electrónico del usuario que desea iniciar sesión.

```
async function actualizarRol(email) {
  const docRef2 = doc(db, "Roles/" + email);
  const docSnap2 = await getDoc(docRef2);
  rol.push(docSnap2.data().rol);
}
```

Fig. 80. Consulta el documento según el correo que vaya a iniciar sesión.

Los atributos que se almacenan en rol de la función actualizarRol(), son los que se detallan en la Fig. 81. Y se observa que el campo rol es el define el perfil del usuario.

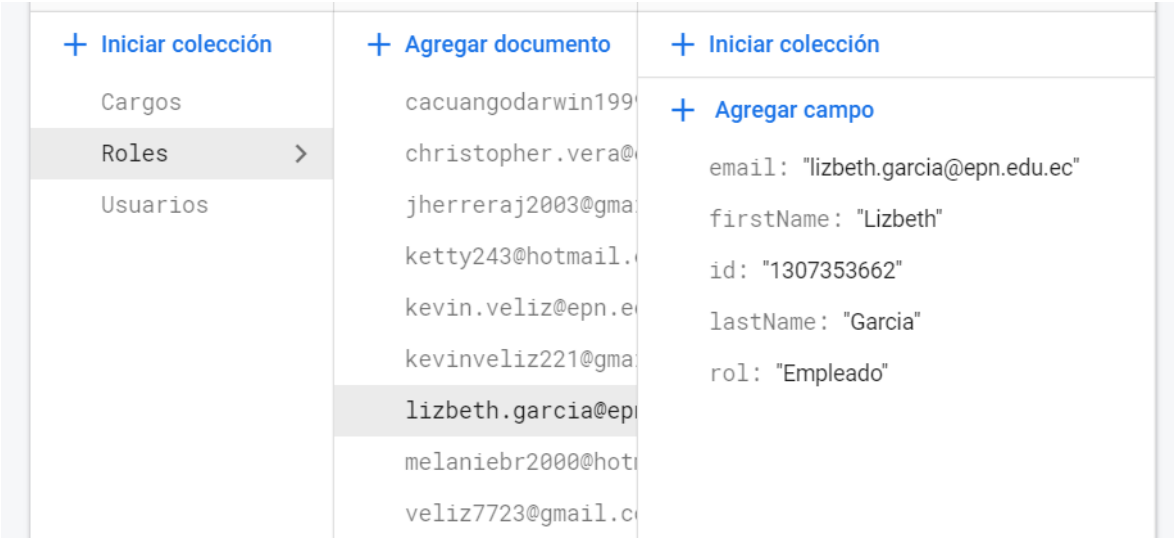


Fig. 81. Colección roles con acceso a un documento específico

Por lo tanto, se usa el campo rol para la navegación correspondiente al iniciar sesión, tal como se muestra en la Fig. 82.

```
await actualizarRol(email);
await loginUser(email, password);
setIsLoading(false);
//console.log("Usuario activo", window.emailVerified);

if (window.emailVerified === true) {
  if (rol[0] === "admin") {
    navigate("/admin");
  } else if (rol[0] === "Empleado") {
    navigate("/empleado");
  } else if (rol[0] === "Gerente") {
    navigate("/manager");
  }
} else {
  swal({
    text: "Correo no verificado.",
    icon: "warning",
    button: "Aceptar",
  });

  signOutUser(email, password);
  navigate("/login");
}
```

Fig. 82. Validar rol y redirección a una ruta especifica

Y el mismo procedimiento de leer el rol se realiza en los componentes de vista de cada usuario, es decir si el usuario es administrador se le muestra la interfaz del componente,

caso contrario se muestra un mensaje “Componente no disponible para su usuario”. En la Fig. 83, se muestra un ejemplo de la validación del roles, cabe mencionar que se validan los tres roles empleados en el sistema (admin, empleado y gerente).

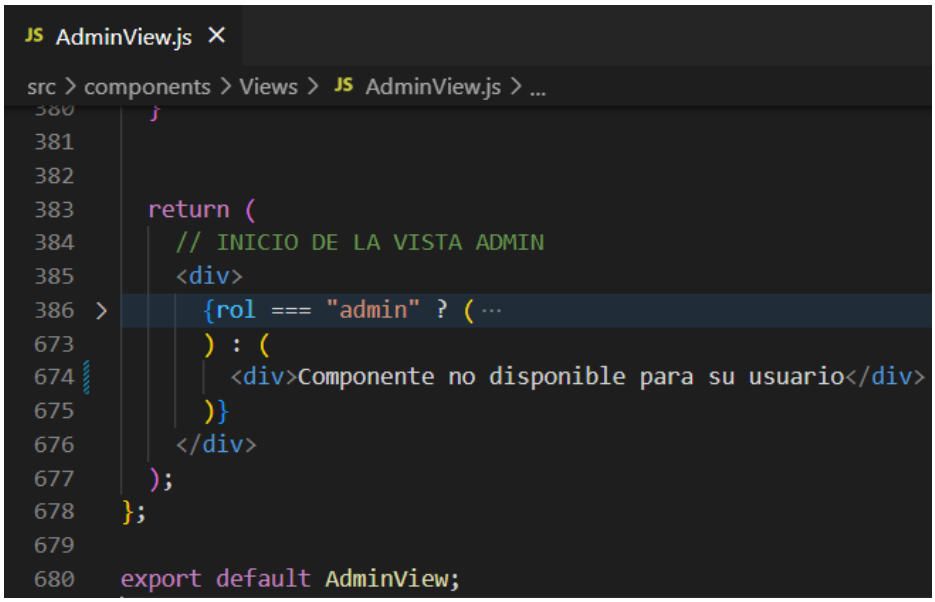


Fig. 83. Componente según el rol “admin”

Tarea 3. Pruebas de carga

Las acciones de lectura y escritura sobre la base de datos de Firebase se realizan en un intervalo de tiempo no muy largo, tal como se evidencia a continuación.

El primer caso es el inicio de sesión y la consulta se demora 13 milisegundos aproximadamente en validar el rol e iniciar sesión, tal como se muestra en la consola de la Fig. 84.

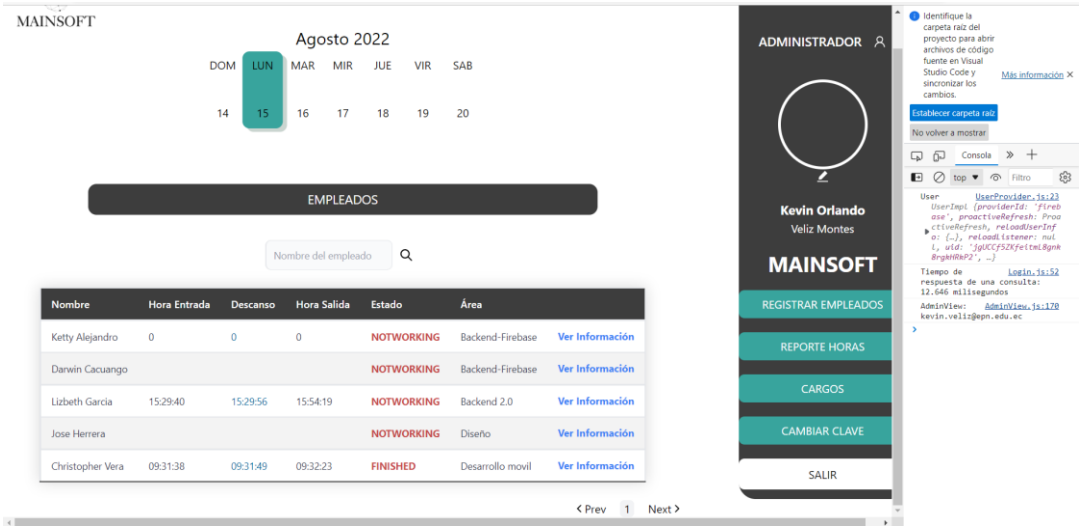


Fig. 84. Tiempo de carga al iniciar sesión.

La segunda prueba se la realiza en la consulta de horas mensuales y se reporta aproximadamente 1 milisegundo tal como se visualiza en la Fig. 85.

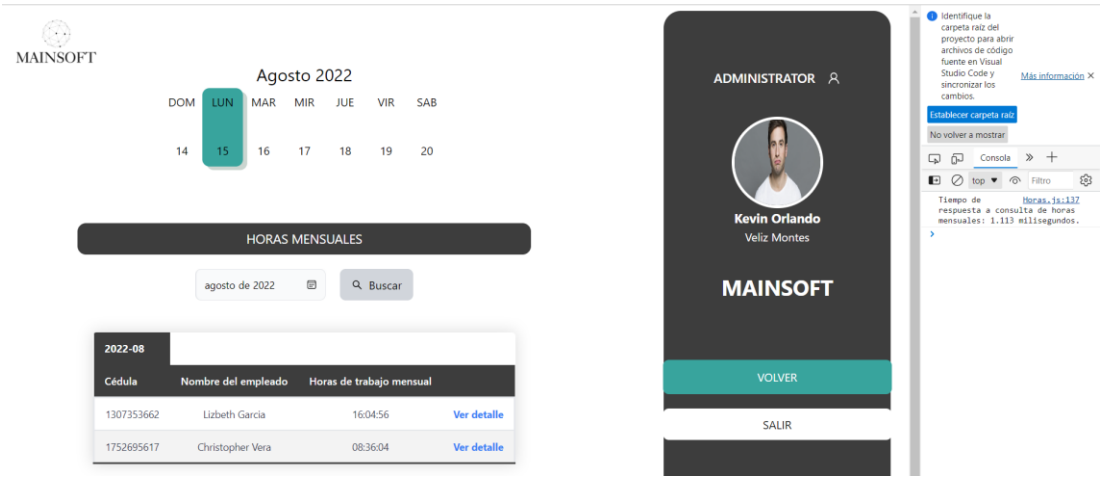


Fig. 85. Tiempo de respuesta ante consulta mensual

En la consulta diaria de horas de trabajo se tarda un milisegundo en completar la tarea, tal como muestra la Fig. 86.

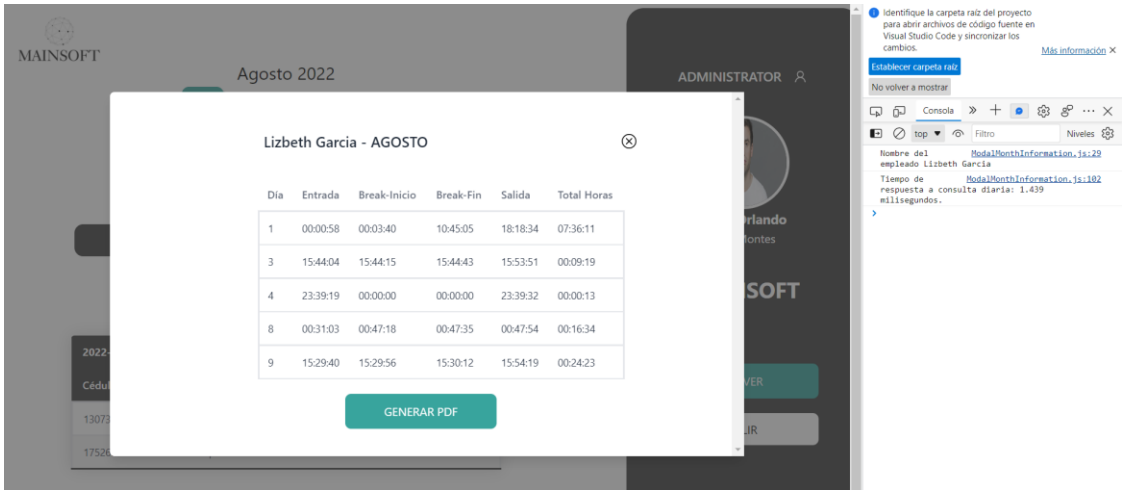


Fig. 86. Tiempo de respuesta ante la consulta diaria

Al traer la información de usuario solo la carga no se demora ni un milisegundo, como se evidencia en la Fig. 87. Este tipo de consulta se realiza en todos los usuarios para poder traer la información y editarla.

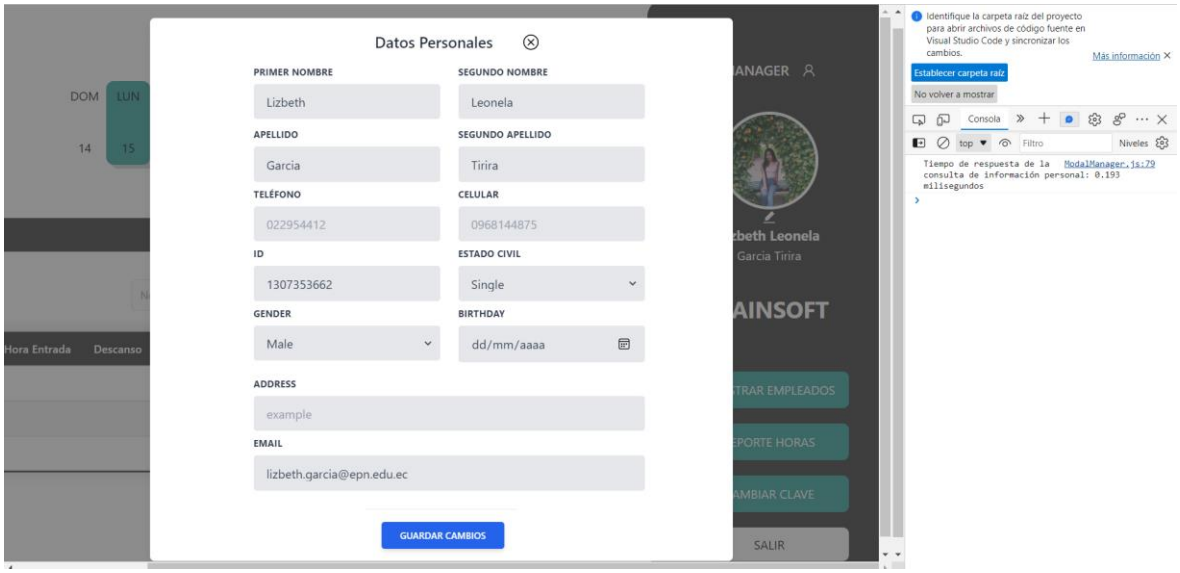


Fig. 87. Tiempo de respuesta ante la consulta de información personal

Por lo antes revisado, se puede decir que la consulta a los diferentes métodos de Firebase es bastante rápido, por lo que demuestra una carga óptima y útil para que el usuario pueda navegar sin interrupciones o deba esperar mucho tiempo ante el flujo del sistema.

4 CONCLUSIONES

A continuación, se detallan las conclusiones que se generaron tras la culminación del presente proyecto.

- El componente Backend para el control de empleados de empresas en crecimiento se concluyó exitosamente debido a que cada uno de los requerimientos mencionados fueron implementados en el sistema web y móvil, donde los datos se almacenan en una base de datos estructurada acorde a la lógica del proyecto.

- La base de datos utilizada el sistema web y móvil es no relacional. Por tanto, se implementan estructuras tipo JSON y manejo de claves asociadas a las colecciones donde se almacena la información, es decir que el sistema usa Cloud Firestore.

- El inicio de sesión de usuarios se gestiona a través de Firebase Authentication con el uso de un correo válido y una clave proporcionada por el administrador, la misma puede ser cambiada en cualquier momento.

- Las consultas, ingresos, actualizaciones y eliminación de registros sobre la base de datos se implementaron a través de los diferentes métodos de Firebase.

- La prueba de seguridad de navegación fue validada, esto se refiere a que según el rol del usuario se le presenta una interfaz gráfica y en caso del componente web un usuario no puede navegar a rutas si no tiene el permiso correspondiente.

5 RECOMENDACIONES

A continuación, se detallan las recomendaciones que se establecen al culminar con este proyecto de titulación.

- Para que el sistema siga con el flujo correspondiente, es recomendable mantener las dependencias de Firebase en su última actualización para que no existan errores en la lectura o escritura de la base de datos.
- Al momento de eliminar un usuario, el usuario se elimina de Cloud Firestore pero no se elimina del Firebase Authentication, esto porque no existe un método como tal para realizar dicha eliminación, por lo tanto, se recomienda no registrar nuevos usuarios con correos existentes en la autenticación, pero en caso de que se lo realice tomar en cuenta que la información procede a sobrescribirse.
- El servicio gratuito de la base de datos logra soportar un numero límite de usuarios, lectura y escritura de la base de datos, esto quiere decir que, si hay muchos usuarios conectados, más allá de 100, la base empieza a presentar desequilibrios. En caso de que el sistema escale y necesite manejar más usuarios se recomienda optar por el servicio de Firebase con la versión de pago según la necesidad.

6 REFERENCIAS BIBLIOGRÁFICAS

[1] TIC Portal, «ERP para pequeñas empresas: ¿hay sistemas que se adaptan y crecen con las empresas?,» European Knowledge Center for Information Technology, 1 11 2021. [En línea]. Available: https://www.ticportal.es/temas/enterprise-resource-planning/tamano-empresarial/erp-pequenas-empresas. [Último acceso: 15 05 2022].

[2] J. Castro, «¿Cómo ayuda la tecnología a las empresas para crecer y competir?,» Corponet, 21 02 2017. [En línea]. Available: https://blog.corponet.com/como-ayuda-la-tecnologia-a-las-empresas-para-crecer-y-competir. [Último acceso: 15 05 2022].

[3] TIC Portal, «¿Qué hace el software de recursos humanos?,» European Knowledge Center for Information Technology, 3 11 2015. [En línea]. Available: https://www.ticportal.es/temas/software-gestion-recursos-humanos/software-recursos-humanos. [Último acceso: 15 05 2022].

[4] APSER, «El software ERP: ejemplos, tipos y uso en la empresa,» Equipo Redacción apser, 26 04 2015. [En línea]. Available: https://apser.es/el-software-erp-ejemplos-tipos-y-uso-en-la-empresa/. [Último acceso: 18 05 2022].

[5] S. Izquierdo, «OpenERPweb Gestión Empresarial para Empresas y Pymes,» [En línea]. Available: http://www.openerpweb.es/que-es-openerp/. [Último acceso: 18 05 2022].

[6] M. Presta, «Los 10 mejores servicios de Backend en la nube,» [En línea]. Available: https://blog.back4app.com/es/los-10-mejores-servicios-de-Backend-en-la-nube/#Que_es_un_Backend_como_servicio. [Último acceso: 15 05 2022].

[7] Google, «Firebase Realtime Database,» Google, [En línea]. Available: https://Firebase.google.com/products/realtime-database?hl=es-419#:~:text=Firebase%20Realtime%20Database%20es%20una,de%20app%20a%20escala%20global.. [Último acceso: 18 05 2022].

[8] Santander Universidades, «Metodologías de desarrollo de software: ¿qué son?,» Santander, 21 12 2020. [En línea]. Available: https://www.becas-santander.com/es/blog/metodologias-desarrollo-software.html. [Último acceso: 16 05 22].

[9] E. G. Maida y J. Pacienzia, «Metodologías de desarrollo de software,» 12 2015. [En línea]. Available: https://repositorio.uca.edu.ar/bitstream/123456789/522/1/metodologias-desarrollo-software.pdf. [Último acceso: 16 05 2022].

[10] J. L. Enríquez Ruiz, E. Farías Palacín, E. Flores Flores, C. Honores Solano, R. Llanos Muñoz, W. López Cordero, V. O. Medina Luna, C. Olivos Colchado, C. Torres Quito, G. Velásquez Soto y A. Zúñiga Ángeles, «METODOLOGÍA DE DESARROLLO,» 2017. [En línea]. Available: uladech.edu.pe/images/stories/universidad/documentos/2018/metodologia-desarrollo-software-v001.pdf. [Último acceso: 16 05 2022].

[11] EBF, «Ventajas y desventajas de las metodologías Agile (ágiles),» EBF, 11 09 2019. [En línea]. Available: https://ebf.com.es/blog/ventajas-y-desventajas-de-las-metodologias-agiles-y-su-aplicacion-en-el-trabajo/. [Último acceso: 16 05 2022].

[12] A. Junquera, «Metodologías ágiles: ¿qué diferencia hay entre Scrum, Kanban y XP?,» Grupo digital, 24 10 2019. [En línea]. Available: https://www.grupodigital.eu/blog/metodologias-agiles/. [Último acceso: 26 05 2022].

[13] Arimetrics , «Qué es Backend,» Arimetrics , 2022. [En línea]. Available: https://www.arimetrics.com/glosario-digital/Backend#Para_que_sirve_un_Backend. [Último acceso: 16 05 2022].

[14] Escuela WOW, «Qué es Backend,» Escuela WOW, 23 09 2021. [En línea]. Available: https://escuelawow.com/que-es-Backend/. [Último acceso: 16 05 2022].

[15] M. P. CARDONA, «Firebase, qué es y para qué sirve la plataforma de Google,» IEBS, 14 10 2016. [En línea]. Available: https://www.iebschool.com/blog/Firebase-que-es-para-que-sirve-la-plataforma-desarroladores-google-seo-sem/. [Último acceso: 18 05 2022].

[16] QuestionPro, «Métodos de investigación: Qué son y cómo elegirlos,» QuestionPro, [En línea]. Available: https://www.questionpro.com/blog/es/metodos-de-investigacion/. [Último acceso: 29 05 2022].

[17] E. R. Chagoya, «Métodos y técnicas de investigación,» Gestiopolis, 12 05 2015. [En línea]. Available: https://www.gestiopolis.com/metodos-y-tecnicas-de-investigacion/. [Último acceso: 29 05 2022].

[18] M. A. d. Dios, «Scrum: qué es y cómo funciona este marco de trabajo,» SALES ENABLEMENT, 09 05 2022. [En línea]. Available: https://www.wearemarketing.com/es/blog/metodologia-scrum-que-es-y-como-funciona.html. [Último acceso: 29 05 2022].

[19] SOFTENG, «Metodología Scrum para desarrollo de software - aplicaciones complejas,» SOFTENG, [En línea]. Available: https://www.softeng.es/es-es/empresa/metodologias-de-trabajo/metodologia-scrum.html. [Último acceso: 29 05 2022].

[20] S. Monroy, «¿Cuáles son los roles de la metodología Scrum?,» APA, 14 12 2021. [En línea]. Available: https://www.apd.es/roles-metodologia-scrum/. [Último acceso: 29 05 2022].

[21] P. Cordero, «¿Qué es un Scrum Master y cuáles son sus funciones?,» Crehana, 22 10 2020. [En línea]. Available: https://www.crehana.com/blog/empresas/que-es-un-scrum-master-y-cuales-son-sus-funciones/. [Último acceso: 05 29 2022].

[22] Edicom Group, «Metodología Scrum: claves de nuestros Scrum Másters, roles, herramientas y eventos,» 3 08 2021. [En línea]. Available: https://careers.edicomgroup.com/metodologia-scrum/. [Último acceso: 29 05 2022].

[23] M. Bara, «Roles, eventos y artefactos en la metodología Scrum,» OBS, 05 09 2017. [En línea]. Available: https://www.obsbusiness.school/blog/roles-eventos-y-artefactos-en-la-metodologia-scrum. [Último acceso: 29 05 2022].

[24] BIOS Software, «Levantamiento de Requerimientos y Desarrollo de Conceptos,» [En línea]. Available: https://biossoft.net/servicio-levantamiento-conceptos.html#:~:text=El%20Levantamiento%20de%20requerimientos%20se,impuestas%20por%20las%20distintas%20partes.. [Último acceso: 29 05 2022].

[25] Scrum México, «Escribiendo Historias de Usuario,» 2 08 2018. [En línea]. Available: https://scrum.mx/informate/historias-de-usuario. [Último acceso: 05 29 2022].

[26] Integra IT, «Sprint y Sprint Backlog: puntos esenciales de SCRUM,» 28 07 2018. [En línea]. Available: https://integrait.com.mx/blog/Sprint-y-Sprint-backlog/. [Último acceso: 29 05 2022].

[27] Instituto Politécnico nacional -UPIICSA, «Diseño Arquitectónico,» [En línea]. Available: http://upiicsa.tecnologia-educativa.com.mx/docs/u2/s3/DISENO%20ARQUITECTONICO.pdf. [Último acceso: 05 06 2022].

[28] E. Novoseltseva, «Características Y Principios De La Arquitectura De Datos,» Apiumhub, 09 03 2021. [En línea]. Available: https://apiumhub.com/es/tech-blog-barcelona/caracteristicas-principios-arquitectura-de-datos/. [Último acceso: 05 06 2022].

[29] Software web & apps, «PATRONES DE ARQUITECTURA Y DISEÑO DE SOFTWARE,» Software web & apps, 14 08 2018. [En línea]. Available: https://www.desarrollodepaginasweb.com.mx/patrones-de-arquitectura-de-software/. [Último acceso: 06 06 2022].

[30] Desarrolloweb, «Qué es MVC,» Desarrolloweb, 28 07 2020. [En línea]. Available: https://desarrolloweb.com/articulos/que-es-mvc.html. [Último acceso: 06 06 2022].

[31] R. Hernandez, «El patrón modelo-vista-controlador: Arquitectura y frameworks explicados,» FreeCodeCamp, 28 06 2021. [En línea]. Available: https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/. [Último acceso: 06 06 2022].

[32] E. A. MERO CAICEDO y E. A. CEDEÑO FLORES, «DESARROLLO E IMPLEMENTACIÓN DE UN APLICATIVO MÓVIL ORIENTADA PARA GESTIONAR EL CONTROL DE INCIDENCIAS Y EVIDENCIAS DE LOS SERVICIOS TÉCNICOS DE LA EMPRESA CEMZ DE LA CIUDAD DE MANTA,» 2018. [En línea]. Available: https://1library.co/article/herramientas-de-desarrollo-definiciones-conceptuales-contexto-te%C3%B3rico.lzge296y. [Último acceso: 06 06 2022].

[33] D. d. Luca, «Visual Studio Code: características principales,» [En línea]. Available: https://damiandeluca.com.ar/visual-studio-code-caracteristicas-principales#:~:text=Ventajas%20de%20Visual%20Studio%20Code&text=Visual%20Studio%20Code%20es%20una,la%20vista%20a%20nuestro%20gusto.. [Último acceso: 06 06 2022].

7 ANEXOS

En esta sección del documento se mencionan los 4 anexos que se han añadido para revisar a detalle el proceso previo al desarrollo del componente.

- ANEXO I. Resultado de Turnitin.
- ANEXO II. Manual técnico
- ANEXO III. Manual de usuario
- ANEXO IV. Manual de instalación

Anexo I

A continuación, se presenta una captura de pantalla en donde se verifica la autenticidad del presente proyecto y el resultado obtenido tras someter al documento en la herramienta de Turnitin.

ESCUELA POLITÉCNICA NACIONAL
ESCUELA DE FORMACIÓN DE TECNÓLOGOS
CAMPUS POLITÉCNICO "ING. JOSÉ RUBÉN ORELLANA"

CERTIFICADO DE ORIGINALIDAD

Quito, D.M. 08 de 09 de 2022

De mi consideración:

Yo, MAYRA ISABEL ALVAREZ JIMÉNEZ, en calidad de Director del Trabajo de Integración Curricular titulado "DESARROLLO DEL BACKEND PARA EL CONTROL DE EMPLEADOS DE EMPRESAS EN CRECIMIENTO" elaborado por la estudiante LIZBETH LEONELA GARCIA TIRIRA de la carrera en DESARROLLO DE SOFTWARE, certifico que he empleado la herramienta Turnitin para la revisión de originalidad del documento escrito completo, producto del Trabajo de Integración Curricular indicado.

El documento escrito tiene un índice de similitud del 9%.

Es todo cuanto puedo certificar en honor a la verdad, pudiendo el interesado hacer uso del presente documento para los trámites de titulación.

NOTA: Se adjunta el informe generado por la herramienta Turnitin.

Atentamente,

Ing. Mayra Isabel Alvarez Jiménez
Técnico Docente
ESFOT

Anexo II

En este anexo, se detallan los 4 artefactos usados en el proyecto que son característicos de la metodología SCRUM. El primero corresponde a la recopilación de requerimientos donde se establecen las funciones que debe poseer sistema, el segundo hace referencia a las historias de usuario con su respectiva estructura y descripción, el tercero es el Product Backlog que representan los requerimientos hechos historias de usuario y su respectiva prioridad en el desarrollo y finalmente, el ultimo artefacto es el Sprint Backlog representa el plan establecido para el desarrollo.

Recopilación de requerimientos

La Tabla XI, se mencionan los requerimientos que se han considerado para el desarrollo del sistema de control de empleados para empresas en crecimiento. El mismo que gestiona las horas de trabajo y actividades que realiza el empleado.

Tabla XI. Recopilación de requerimientos

ID-RR	Enunciado del ítem
RR-01	Como usuario administrador, necesita una autenticación de rol para iniciar sesión.
RR-02	Como usuario administrador, necesita registrar usuarios con roles empleado o gerente para llevar un registro de los mismos.
RR-03	Como usuario administrador, necesita disponer de la información de los usuarios del sistema para poder editar, eliminar o actualizar los datos en caso de que sea necesario.
RR-04	Como usuario administrador, necesita gestionar el contenido de todo el sistema.
RR-05	Como usuario gerente, necesito conocer el estado de mis empleados, es decir, si está trabajando, en receso o fuera de trabajo para supervisar su desempeño.
RR-06	Como usuario gerente, necesito una autenticación de rol para iniciar sesión.
RR-07	Como usuario gerente, necesito registrar a un usuario empleado.

RR-08	Como usuario gerente, necesito visualizar la información del empleado para conocer el estado de cada uno, pero sin necesidad de actualizar o editar la información.
RR-09	Como usuario gerente, necesito visualizar las horas de trabajo que posee cada empleado junto con su estado.
RR-10	Como usuario empleado, necesito una autenticación de rol para iniciar sesión.
RR-11	Como usuario empleado, necesito ingresar la hora de entrada al trabajo, de salida y de receso para evidenciar mi labor, este aspecto se define con un cambio de estado para que el administrador y gerente pueda observar.
RR-12	Como usuario empleado, necesito actualizar mi información personal específica excepto información ya definida.
RR-13	Como usuario empleado, necesito cargar mis documentos importantes como cédula, hoja de vida y papeleta de votación para que mi información esté al día.
RR-14	Como usuario empleado, necesito ver mi historial de horas trabajadas para confirmar mi horario de trabajo.
RR-15	Como usuario administrador, necesito manejar un crud de cargos para que existan opciones de cargos al momento de registrar un nuevo usuario.

Historias de Usuario

A partir de la Tabla XII hasta la

Tabla XXV, se muestra las historias de usuario realizadas como guía en el desarrollo del proyecto, en la sección **Artefactos** se describe la historia de usuario Nro.1.

Tabla XII. Historia de usuario Nro. 2

HISTORIA DE USUARIO

Identificador (ID): HU02	**Usuario:** Administrador y Gerente
Nombre de Historia: Almacenar en la base de datos un usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Iteración asignada: 1	
Responsable: Lizbeth García	
Descripción: El administrador o gerente es el encargado de registrar a un usuario con el rol empleado para que pueda acceder al sistema.	
Observación: El registro es un formulario que debe especificar nombre, apellido, cédula, email, rol y un cargo.	

Tabla XIII. Historia de usuario Nro. 3

HISTORIA DE USUARIO	
Identificador (ID): HU03	**Usuario:** Administrador
Nombre de Historia: Consultar en la base de datos la información del empleado	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Iteración asignada: 2	
Responsable: Lizbeth García	
Descripción: El administrador cuenta con un panel principal donde puede visualizar la información actualizada de cada empleado.	
Observación: En el panel principal solo se muestra nombre y apellido del empleado, y al dar clic en la persona se detalla la demás información como dirección, número de teléfono, número de teléfono alterno, cédula, email, etc.	

Tabla XIV. Historia de usuario Nro. 4

HISTORIA DE USUARIO	
Identificador (ID): HU04	**Usuario:** Administrador
Nombre de Historia: Consultar en la base de datos las horas de trabajo de los empleados.	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media

Iteración asignada: 3
Responsable: Lizbeth García
Descripción: El usuario administrador tiene el panel principal que se subdivide por día y mes, donde se muestran las horas de trabajo de cada empleado.
Observación: Las consultas se actualizan en tiempo real sin necesidad de recargar la página.

Tabla XV. Historia de usuario Nro. 5

HISTORIA DE USUARIO	
Identificador (ID): HU05	**Usuario:** Empleado
Nombre de Historia: Consultar en la base de datos el estado del empleado	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Bajo
Iteración asignada: 2	
Responsable: Lizbeth García	
Descripción: El usuario empleado registra sus horas de trabajo donde se almacena la hora como tal y el estado: working, notworkig, break,backworking o workingpermissions.	
Observación: Las consultas se actualizan automáticamente para que cambie la opción de los botones.	

Tabla XVI. Historia de usuario Nro. 6

HISTORIA DE USUARIO	
Identificador (ID): HU06	**Usuario:** Gerente, Administrador y Empleado
Nombre de Historia: Verificar correo electrónico	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Baja
Iteración asignada: 4	
Responsable: Lizbeth García	

Descripción: Todos los usuarios tienen la obligación de verificar su correo para poder iniciar sesión.
Observación: Cuando se registra a un nuevo usuario se le debe enviar un correo con un enlace de verificación para que valide que es un correo electrónico correcto.

Tabla XVII. Historia de usuario Nro. 7

HISTORIA DE USUARIO	
Identificador (ID): HU07	**Usuario:** Gerente y Administrador
Nombre de Historia: Registrar a un usuario	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Iteración asignada: 1	
Responsable: Lizbeth García	
Descripción: El usuario gerente o administrador se encargan de crear cuenta a sus empleados.	
Observación: El gerente y empleado tiene un formulario de registro para sus empleados, el cual consta de nombre, apellido, cédula, correo, contraseña, rol y cargo.	

Tabla XVIII. Historia de usuario Nro. 8

HISTORIA DE USUARIO	
Identificador (ID): HU08	**Usuario:** Gerente y Administrador
Nombre de Historia: Consulta en la base de datos de la información del empleado	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Iteración asignada: 3	
Responsable: Lizbeth García	
Descripción: El usuario gerente y administrador puede visualizar la información del empleado y también puede editar y actualizar campos.	
Observación: El gerente o administrador pueden eliminar a un usuario empleado.	

Tabla XIX. Historia de usuario Nro. 9

HISTORIA DE USUARIO	
Identificador (ID): HU09	**Usuario:** Gerente
Nombre de Historia: Consulta a la base de datos las horas de trabajo de los empleados	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Baja
Iteración asignada: 3	
Responsable: Lizbeth García	
Descripción: El usuario gerente puede poseer un panel principal con el nombre y apellido de sus empleados y las horas de trabajo.	
Observación: La información se actualiza automáticamente.	

Tabla XX. Historia de usuario Nro. 10

HISTORIA DE USUARIO	
Identificador (ID): HU10	**Usuario:** Administrador, Gerente y Empleado
Nombre de Historia: Reestablecer contraseña	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Iteración asignada: 4	
Responsable: Lizbeth García	
Descripción: Un usuario puede cambiar su contraseña para establecer un mejor nivel de seguridad.	
Observación: El usuario es capaz de reestablecer su contraseña desde un correo para cambiar su clave o a través de la interfaz una vez que haya iniciado sesión.	

Tabla XXI. Historia de usuario Nro. 11

HISTORIA DE USUARIO	
Identificador (ID): HU11	**Usuario:** Empleado
Nombre de Historia: Registro en la base de datos de la actividad del empleado	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Alta
Iteración asignada: 2	
Responsable: Lizbeth García	
Descripción: El usuario empleado tiene la función principal de registrar su hora de entrada, salida y recesos.	
Observación: El registro de horas viene acompañado de una pequeña descripción de lo ejecutado.	

Tabla XXII. Historia de usuario Nro. 12

HISTORIA DE USUARIO	
Identificador (ID): HU12	**Usuario:** Empleado
Nombre de Historia: Actualizar la base de datos según la información del empleado	
Prioridad en negocio: Alta	**Riesgo en desarrollo:** Media
Iteración asignada: 2	
Responsable: Lizbeth García	
Descripción: El usuario empleado, después de haber sido registrado, puede actualizar su perfil personal con el resto de información.	
Observación: Los campos extras son: dirección, teléfono, estado civil u otros campos que exija la empresa.	

Tabla XXIII. Historia de usuario Nro. 13

HISTORIA DE USUARIO	
Identificador (ID): HU13	**Usuario:** Empleado
Nombre de Historia: Registrar en la base de datos los documentos importantes	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Alta
Iteración asignada: 2	
Responsable: Lizbeth García	
Descripción: El usuario empleado tiene un apartado para subir los documentos importantes, tales como hoja de vida y copia de cédula.	
Observación: Campo obligatorio para que pueda registrar sus horas de trabajo.	

Tabla XXIV. Historia de usuario Nro. 14

HISTORIA DE USUARIO	
Identificador (ID): HU14	**Usuario:** Empleado
Nombre de Historia: Consulta en la base de datos sobre la jornada laboral.	
Prioridad en negocio: Baja	**Riesgo en desarrollo:** Media
Iteración asignada: 4	
Responsable: Lizbeth García	
Descripción: El usuario empleado puede revisar el historial de horas trabajadas para verificar si concuerda con la realidad.	
Observación: El usuario empleado solo puede visualizar, pero no editar esta información.	

Tabla XXV. Historia de usuario Nro. 15

HISTORIA DE USUARIO	
Identificador (ID): HU15	**Usuario:** Administrador
Nombre de Historia: Manejar los cargos de un empleado	
Prioridad en negocio: Media	**Riesgo en desarrollo:** Media
Iteración asignada: 4	
Responsable: Lizbeth García	
Descripción: El usuario administrador se encarga de establecer los cargos disponibles de la empresa para que se muestren al momento de registrar un nuevo usuario.	
Observación: El administrador puede crear un nuevo cargo, eliminarlo si es necesario o actualizar si hubo algún error al ingresarlo en la base de datos.	

Product Backlog

En la Tabla XXVI, se muestra la lista de requerimientos del sistema además de la información del número de iteración, el estado y la prioridad del requisito, como complemento de la información de las historias de usuario.

Tabla XXVI. Product Backlog

ID-HU	Historia de usuario	Iteración	Estado	Prioridad
HU01	Servicio para inicio de sesión según el rol de administrador	1	Completa	Alta
HU02	Almacenar en la base de datos un usuario con rol empleado	1	Completa	Alta
HU03	Consultar en la base de datos la información del empleado	2	Completa	Media
HU04	Consultar en la base de datos las horas de trabajo de los empleados	3	Completa	Alta
HU05	Consultar en la base de datos el estado del empleado	2	Completa	Media

HU06	Servicio para inicio de sesión según el rol de gerente	4	Completa	Alta
HU07	Registrar a un usuario con el rol de empleado	1	Completa	Alta
HU08	Consulta en la base de datos de la información del empleado	3	Completa	Media
HU09	Consulta a la base de datos las horas de trabajo de los empleados	3	Completa	Media
HU10	Servicio de inicio de sesión según el rol empleado	4	Completa	Alta
HU11	Registro en la base de datos de la actividad del empleado	2	Completa	Alta
HU12	Actualizar la base de datos según la información del empleado	2	Completa	Alta
HU13	Registrar en la base de datos los documentos importantes	2	Completa	Media
HU14	Consulta en la base de datos sobre la jornada laboral.	4	Completa	Baja
HU15	Registrar en la base de datos las horas extras	4	Completa	Media

Sprint Backlog

En la Tabla XXVII, se describe la división del proyecto en Sprints con información del nombre de la historia de usuario, su identificador, las tareas a realizar y la estimación de tiempo.

Tabla XXVII. Sprint Backlog

ID-SB	Nombre	ID-HU	HISTORIA DE USUARIO	TAREAS	TIEMPO ESTIMADO
SB-000	Configuración del entorno de desarrollo.	N/A	N/A	● Crear un proyecto dentro de Firebase para guardar la información necesaria. ● Configurar el entorno de desarrollo en React para validar los scripts de registro y consulta en el entorno web. ● Configurar el entorno de desarrollo en React Native para validar los scripts de registro y consulta en el entorno móvil.	10H
SB-001	Autenticación	HU07	Registrar a un usuario	● Registrar a un usuario, el registro se realiza con una sesión iniciada. Si el usuario es administrador puede agregar usuarios tipo administrador, gerente o usuario y si el usuario es gerente solo puede registrar a usuarios tipo gerente o empleado. El usuario empleado no tiene la posibilidad de crear nuevos usuarios. ● Estructurar la base de datos y establecer nombre de colecciones y documentos respectivos. ● Registrar una colección específica para manejar los roles del usuario.	50H

		HU02	Almacenar en la base de datos un usuario	● Colocar privilegios a usuarios tipo gerente y administrador. ● Estructurar la base de datos del usuario y colocar los atributos correspondientes para que el Frontend y móvil puedan ocupar las variables necesarias.	
		HU01	Servicio para inicio de sesión	● Establecer la función que permite iniciar sesión y manejar la interfaz del usuario según el rol que se le haya asignado.	
SB-002	Funciones principales del usuario Empleado	HU03	Consultar en la base de datos la información del empleado	● Traer de la base de datos la información del usuario empleado y almacenar en variables de estado para que el apartado web y móvil puedan tener acceso a dicha información.	60H
		HU12	Actualizar la base de datos según la información del empleado	● Traer la información del usuario y permitir la modificación de los datos personales del usuario empleado.	
		HU13	Registrar en la base de datos los documentos importantes	● Permitir cargar documentos importantes como la hoja de vida, cédula y papeleta de votación al usuario empleado.	

		HU11	Registro en la base de datos de la actividad del empleado	• Permitir el registro de horas de trabajo del usuario. En la base de datos se debe almacenar la hora de entrada, hora de receso, hora al regresar del receso y hora final de la jornada. Además, se actualiza el estado del empleado.	
		HU05	Consultar en la base de datos el estado del empleado	• Al finalizar la jornada de trabajo el sistema debe calcular las horas de trabajo de un día del usuario empleado.	
SB-003	Funciones principales del usuario administrador y gerente	HU08	Consulta en la base de datos de la información del empleado	• El usuario administrador y el usuario gerente deben visualizar la información de todos los empleados por lo tanto desde la base de datos se trae la información y se debe almacenar en un arreglo para que pueda mostrar la información. • El usuario administrador y gerente tienen la capacidad de eliminar un usuario de la base de datos • El usuario administrador y gerente tienen la capacidad de editar la información de cualquiera de sus empleados, por lo tanto, debe existir la función para editar en la base de datos y actualizar el documento que le corresponda a cada empleado.	40H

		HU04	Consultar en la base de datos las horas de trabajo de los empleados	• El usuario administrador puede visualizar el historial mensual de trabajo de los usuarios, por lo mencionado, se debe traer la suma total de las horas de la base de datos.	
		HU09	Consulta a la base de datos las horas de trabajo de los empleados	• El usuario gerente puede visualizar el historial mensual y diario del trabajo de los usuarios, por lo mencionado, se debe traer la suma total de las horas de la base de datos	
SB-004	Funciones adicionales para los distintos roles	HU06	Verificar correo electrónico	• Para que un usuario pueda iniciar sesión debe verificar su correo, a través de esta confirmación el sistema permite o no el flujo del programa.	40H
		HU10	Reestablecer contraseña	• Cualquiera de los usuarios puede cambiar su clave de acceso sea para reestablecer por haber olvidado su clave o solo porque desea cambiarla.	
		HU14	Consulta en la base de datos sobre la jornada laboral.	• Validar que el empleado cumpla con el horario laboral para esta tarea es necesario leer la base de datos y según el horario que se le haya establecido establece un horario para que el usuario pueda finalizar su jornada. • Al registrar un nuevo empleado este se crea por defecto con un	

				horario de 8 horas, pero el gerente o administrador puede editar en la base de datos y cambiar este horario a uno de 6 o 4 horas. • Obligar subir la documentación al usuario.	
		HU15	Manejar los cargos de un empleado	• CRUD para manejo de cargos que posee un empleado o un gerente.	
SB-005	Pruebas y despliegue a producción	N/A	N/A	• Pruebas unitarias • Pruebas de seguridad • Pruebas de carga	20H
Documentación					20H
TOTAL					240H

Nota: El documento individual del manual técnico se lo puede encontrar en el siguiente enlace: ManualTecnico_LizbethGarcia.pdf

Anexo III

En el siguiente enlace encontraran el manual de usuario a través de un video que muestra la funcionalidad del sistema en base a los tres perfiles (admin, gerente, empleado) que se maneja en el sistema web y un perfil (empleado) para la aplicación móvil de nuestro proyecto MainSoft.

URL: https://www.youtube.com/watch?v=EiG2pkAdN4Q&ab_channel=MateoVera

Anexo IV

A continuación, los enlaces y credenciales de acceso hacia el sistema web y aplicación móvil. Además, se adjunta el Código Fuente del sistema y manual de instalación.

Credenciales se acceso

Para acceder al sistema web del proyecto MainSoft, lo hace a través del siguiente enlace: https://mainsoft.vercel.app/login

- Perfil administrador

  Correo: kevin.veliz@epn.edu.ec

  Contraseña: MainSoft1308*

- Perfil gerente

  Correo: lizbeth.garcia@epn.edu.ecc

  Contraseña: MainSoft1307*

- Perfil empleado

  Correo: christopher.vera@epn.edu.ec

  Contraseña: MainSoft1752*

El acceso a la aplicación móvil se lo hace a través de un APK que se descarga mediante el siguiente enlace, con las credenciales del usuario empleado.

https://expo.dev/artifacts/91375a5a-26a2-49f6-b805-8012f97d72ae

Código fuente

El código fuente del presente proyecto lo pueden encontrar en dos repositorios, el primero donde se visualiza el código para el sistema web y el segundo que contiene el código de la aplicación móvil.

- Sistema web: https://github.com/KevinVeliz/MainSoft-WEB
- Aplicación móvil: https://github.com/Chriss78Vera/MainSoft

El manual de instalación se puede observar a detalle en el siguiente enlace:

- ManualInstalación_LizbethGarcia.pdf