UNIVERSIDAD TECNOLÓGICA DE TIJUANA

TÍTULO DEL TRABAJO RECEPCIONAL

DESARROLLO DE UNA BASE DE DATOS EN PLATAFORMA MÓVIL PARA EL RELOJ MALBOUCHE

REALIZADO POR

LARA LÓPEZ JOSUE ISAAC

TÍTULO A OTORGAR

TÉCNICO SUPERIOR UNIVERSITARIO EN TECNOLOGÍAS DE LA INFORMACIÓN ÁREA DESARROLLO DE SOFTWARE MULTIPLATAFORMA.

PRESENTA

TIJUANA, B.C **DICIEMBRE ,2025**

UNIVERSIDAD TECNOLÓGICA DE TIJUANA

TÉCNICO SUPERIOR UNIVERSITARIO EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DESARROLLO DE SOFTWARE MULTIPLATAFORMA.

TÍTULO A OTORGAR

TÉCNICO SUPERIOR UNIVERSITARIO EN TECNOLOGÍAS DE LA INFORMACIÓN

ÁREA DESARROLLO DE SOFTWARE MULTIPLATAFORMA.

TRABAJO RECEPCIONAL

DESARROLLO DE UNA BASE DE DATOS EN PLATAFORMA MÓVIL PARA EL RELOJ MALBOUCHE.

Realizado por

LARA LÓPEZ JOSUE ISAAC

En la empresa

Cara de Monitor - Javier Zamorano Martinez (RFC: ZAMJ800223LF2)

Asesor académico

M.C. María del Carmen Vargas García

Director de carrera

M.C. Judith Cristina Félix Callejas

Asesor empresarial

Lic. Javier Zamorano Martínez

Tijuana, B.C. diciembre, 2025.

DEDICATORIA Y/O AGRADECIMIENTOS

A la Universidad Tecnológica de Tijuana, por darme las bases técnicas y las herramientas para llegar hasta aquí.

Y de forma muy especial a Javier Zamorano Martínez y al equipo del proyecto Malbouche. Gracias por la confianza, por la oportunidad de entrarle a un reto real y por dejarme aportar mi parte en este desarrollo. Fue una gran experiencia pasar de la teoría a la práctica con ustedes.

ÍNDICE DE CONTENIDO

I. INTRODUCCIÓN

Los sistemas de Internet de las Cosas (IoT) se definen como una red de objetos físicos que incorporan sensores, software y otras tecnologías con el fin de conectarse e intercambiar datos con otros dispositivos y sistemas a través de Internet (Gartner, Inc, n.d.) Entre sus principales beneficios destacan la capacidad de automatizar procesos, la recopilación de datos en tiempo real para la toma de decisiones y la mejora de la eficiencia operativa. Estos beneficios se han materializado en sectores diversos: en la domótica, con sistemas de hogares inteligentes como Google Home o Alexa; en la industria, a través de la monitorización de maquinaria para mantenimiento predictivo; y en las ciudades inteligentes, mediante la gestión automatizada del alumbrado público o el tráfico (Intel Corporation, n.d.).

Una alternativa tecnológica en el desarrollo de software son las aplicaciones de escritorio nativas, las cuales se instalan y ejecutan directamente en el sistema operativo de una computadora (Microsoft Corporation, 2023). Estas aplicaciones utilizan los recursos del hardware de forma directa, lo que les confiere un alto rendimiento y la capacidad de funcionar sin conexión a Internet. Su principal beneficio es la robustez y velocidad en el procesamiento de datos; sin embargo, presentan la problemática de la portabilidad limitada, ya que una aplicación desarrollada para Windows no funcionará en macOS sin una reescritura completa del código. El impacto de esta limitación es significativo, pues incrementa los costos de desarrollo y mantenimiento, y restringe el acceso de los usuarios a un único tipo de dispositivo, lo cual es una desventaja considerable en un entorno donde la movilidad es fundamental. Por esta razón, aunque las aplicaciones de escritorio son una opción potente, en este proyecto se aborda el problema desde una perspectiva diferente, priorizando el acceso remoto y multiplataforma.

La alternativa seleccionada para este proyecto es el desarrollo de un sistema de back-end para una aplicación móvil, el cual centraliza la lógica de negocio y la gestión de datos, permitiendo el control de dispositivos físicos de forma remota y segura (Render, n.d.) (Google, n.d.). Este tipo de sistema se ejecuta en un servidor en la nube y se comunica con la aplicación cliente a través de una API, lo que garantiza que los datos estén siempre sincronizados y accesibles. Entre sus beneficios se encuentra la escalabilidad y la centralización de la información, facilitando las actualizaciones y el mantenimiento. No obstante, enfrenta el reto de garantizar la seguridad de los datos en tránsito y en reposo, así como de mantener una alta disponibilidad del servicio.

La motivación para elegir esta alternativa es que responde directamente a la necesidad de controlar el reloj "Malbouche" desde cualquier lugar y por múltiples usuarios autorizados. La propuesta consiste en desarrollar una API RESTful con Node.js y Express.js, conectada a una base de datos Google Cloud Firestore y desplegada en la plataforma Render. Según Chakray, las APIs RESTful destacan por su simplicidad, flexibilidad y capacidad de adaptación, lo que las convierte en una solución ideal para entornos distribuidos y móviles donde la eficiencia y la interoperabilidad son clave (Chakray, 2023).

Se eligió trabajar en el back-end porque es el componente crítico que habilita toda la funcionalidad remota y la persistencia de datos de la aplicación móvil. A diferencia de un sistema de control local o una aplicación sin conexión, esta solución se diferencia en tres aspectos clave: permite una gestión centralizada y en tiempo real, es accesible desde cualquier dispositivo móvil con conexión a Internet y ofrece una arquitectura escalable que puede adaptarse a futuras necesidades sin requerir cambios en el dispositivo físico.

II. ANTECEDENTES DE LA EMPRESA O MARCO CONTEXTUAL

El desarrollo de este proyecto se llevó a cabo en el contexto de los emprendimientos coordinados por el Lic. Javier Zamorano Martínez, a través de su oficina de gestión de proyectos, CDMonitor. Esta entidad funge como la matriz que articula diversas iniciativas en los sectores de la gastronomía, la hospitalidad y la cultura en la región de Baja California. Aunque la estadía no se realizó en una empresa con una estructura corporativa tradicional, el marco contextual está definido por los negocios y la visión estratégica del cliente.

2.1 Generalidades de la empresa

La estadía profesional se centró en los proyectos Bloodlust Wine Bar y el futuro Malbouche Piano Bar. El primero, Bloodlust Wine Bar, es un negocio que inició operaciones a mediados de 2022 en la región vinícola del Valle de Guadalupe, Ensenada. Su giro principal es el de bar de vinos, restaurante y foro cultural, destacando por su rápida popularización en septiembre del mismo año.

Por su parte, Malbouche Piano Bar es un proyecto en desarrollo con apertura proyectada para 2025 en el corredor cultural de la Avenida Revolución, en el centro de Tijuana. Este establecimiento se perfila como un espacio que fusionará el arte surrealista, a cargo del artista Jaime Zuverza, con una oferta gastronómica y de entretenimiento de alto nivel. Ambos negocios son coordinados bajo la misma dirección, compartiendo una filosofía de innovación y creación de experiencias.

2.2 Trayectoria y evolución

El asesor empresarial, Lic. Javier Zamorano, cuenta con una trayectoria de más de 15 años en el sector de la hospitalidad, especializándose en el diseño integral para

restaurantes (imagen, concepto, diseño gráfico, web y redes sociales). Su evolución profesional ha transitado desde el diseño tradicional hacia la gestión de proyectos digitales.

Actualmente, su enfoque se orienta hacia la incorporación de nuevas tecnologías para la optimización de procesos. La visión a futuro es consolidar un despacho creativo que desarrolle proyectos basados en datos (data-informed), diseño de experiencia de usuario (UX) y la aplicación de herramientas de inteligencia artificial (IA) para la producción de contenido y la administración eficiente de proyectos. Este proyecto de tesina se alinea con esa visión de evolución tecnológica.

2.3 Procesos y operaciones actuales

Los negocios coordinados por CDMonitor se especializan en ofrecer servicios de barra y cocina de alta calidad. El modelo operativo se basa en la curación de una oferta distintiva que incluye vinos independientes de Baja California y California, cerveza artesanal de productores locales y una propuesta gastronómica que fusiona productos del mar con alimentos de proximidad (farm-to-table).

El proceso central de la operación no es la producción en masa, sino el diseño cuidadoso de menús y la planeación estratégica para la adquisición de materia prima e insumos. Este enfoque garantiza la calidad y coherencia de la experiencia ofrecida al cliente, que es uno de los pilares de su modelo de negocio.

2.4 Vinculación estratégica

CDMonitor funciona como el núcleo que coordina estratégicamente múltiples marcas, incluyendo Pizza & Love, Bloodlust Wine Bar y Malbouche Piano Bar. Esta sinergia permite compartir recursos, conocimientos y estrategias de mercado.

A nivel regional, existe una vinculación importante con el ecosistema local. Específicamente, el proyecto Malbouche Piano Bar se integra en el corredor cultural de la Avenida Revolución en Tijuana, colaborando con actores clave como la Universidad Tecnológica de Tijuana (UTT) y la Delegación Centro, fortaleciendo su presencia e impacto en la comunidad.

2.5 Impacto e importancia

La principal diferenciación y el impacto más significativo de estos emprendimientos en la industria local radican en su oferta integrada que converge cultura, gastronomía y espectáculos. A diferencia de un bar o restaurante convencional, los proyectos liderados por Javier Zamorano buscan crear experiencias memorables y completas. Esta propuesta de valor no solo atrae a una clientela diversa, sino que también contribuye al enriquecimiento de la oferta cultural de la región, posicionando a sus establecimientos como puntos de referencia tanto para residentes como para turistas. El reloj "Malbouche", objeto de este proyecto, es un componente clave de dicha estrategia de diferenciación.

III. DESCRIPCIÓN GENERAL Y ESPECÍFICA DEL ÁREA DE TRABAJO

3.1 Nombre del área o departamento

La estadía profesional se llevó a cabo dentro de un área funcional que se conformó como el Departamento de Desarrollo de Software para la ejecución exclusiva del proyecto "Malbouche". Este departamento, aunque no formaba parte de una estructura empresarial preexistente, operó de manera formal bajo la coordinación de la oficina de proyectos CDMonitor, simulando un entorno de trabajo profesional y enfocado en la entrega de una solución tecnológica integral.

3.2 Descripción del entorno de trabajo (virtual o híbrido)

El entorno de trabajo para el desarrollo del proyecto fue íntegramente virtual. Dadas las circunstancias y la naturaleza del equipo, todas las actividades, desde la planificación inicial hasta el despliegue final, se realizaron de manera remota. La coordinación de tareas, las reuniones de seguimiento y las sesiones de resolución de problemas se efectuaron mediante el uso de plataformas digitales de comunicación como Discord, que sirvió como nuestra oficina virtual, y WhatsApp para comunicaciones urgentes. Este modelo de trabajo fomentó la autogestión y la responsabilidad individual, permitiendo una gran flexibilidad horaria, aunque demandó un alto nivel de disciplina y comunicación proactiva para asegurar la correcta sincronización del equipo.

3.3 Diagrama general del flujo de comunicación

El flujo de comunicación se diseñó para ser claro y eficiente, minimizando la ambigüedad y centralizando la toma de decisiones. La estructura era jerárquica en cuanto a la comunicación con los asesores, pero colaborativa y horizontal a nivel de desarrollo. Como desarrollador del backend, mi línea de reporte directa era hacia la

líder del proyecto, quien era la responsable de consolidar los avances y presentarlos al asesor empresarial para su validación y al asesor académico para el seguimiento. Cualquier consulta técnica o de integración se resolvía directamente entre los miembros del equipo de desarrollo, fomentando un ambiente de cooperación.

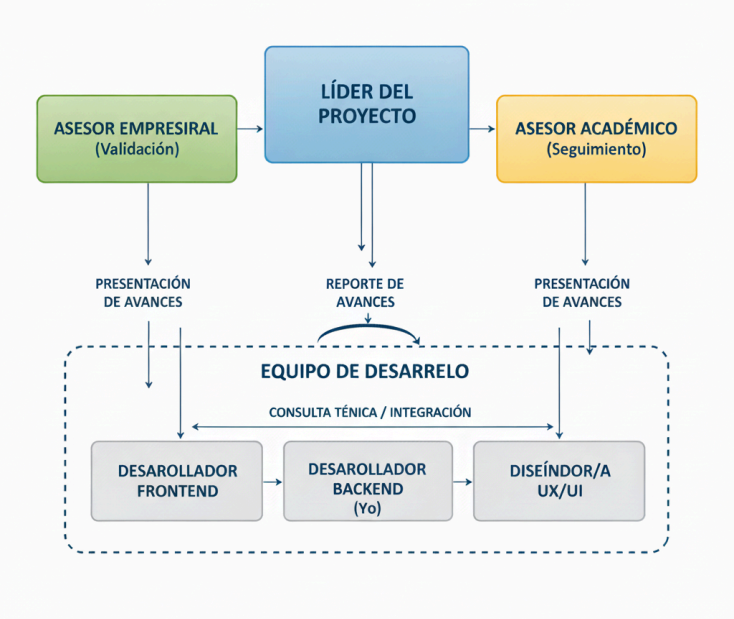

Figura 3.1. Diagrama del flujo de comunicación. Fuente: Del autor 2025

3.4 Recursos y equipo disponible

El proyecto se ejecutó con un enfoque en la optimización de recursos, aprovechando herramientas de código abierto y servicios en la nube con planes de gratuidad. El equipo disponible consistió en:

- Recursos Materiales: Cada integrante del equipo aportó su propio equipo de cómputo personal y conexión a internet, lo cual fue suficiente para llevar a cabo todas las tareas de desarrollo asignadas.

- Recursos de Software: Se empleó un conjunto de tecnologías de libre acceso que no generaron costos. El entorno de desarrollo principal fue Visual Studio Code; la base de datos se gestionó con Google Cloud Firestore; el servidor backend se construyó sobre Node.js y Express.js; el control de versiones se manejó con Git y GitHub; y el despliegue se realizó en la plataforma Render.

3.5 Personal con el que se interactuó

Durante el ciclo de vida del proyecto, la interacción fue clave con diversas figuras que cumplieron roles específicos:

- Lic. Javier Zamorano Martínez: En su rol de Asesor Empresarial, fue el principal interesado (stakeholder). La interacción con él consistió en reuniones periódicas para la presentación de avances, la validación de que las funcionalidades desarrolladas se alinearan con la visión del negocio y la resolución de dudas sobre los requisitos del producto.

- M.C. María del Carmen Vargas García: Como Asesora Académica, su función fue de supervisión y guía metodológica. La interacción con ella se centró en asegurar que la documentación y el desarrollo del proyecto cumplieran con los estándares académicos de la universidad.

- Diana Martínez Pérez: La comunicación con la Líder de Proyecto fue diaria y fundamental para la coordinación de tareas, el reporte de avances y la identificación de posibles impedimentos.

- Equipo de Desarrollo: La colaboración con los demás desarrolladores fue constante, especialmente con los encargados del frontend móvil, para asegurar que

la API que estaba construyendo respondiera a sus necesidades y la integración entre cliente y servidor fuera fluida.

3.6 Objetivo del área o departamento

El objetivo primordial del Departamento de Desarrollo de Software fue el de diseñar, implementar y dar mantenimiento a las aplicaciones y sistemas que apoyan los procesos y la visión innovadora de los proyectos coordinados por CDMonitor. Específicamente para el proyecto Malbouche, el objetivo fue entregar una solución de software robusta, escalable y funcional que permitiera el control remoto del reloj análogo, cumpliendo con todos los requisitos establecidos por el cliente.

3.7 Funciones y procesos que se desarrollan

El área de desarrollo abarcó el ciclo de vida completo del software. Las funciones generales incluyeron el análisis de requisitos, el diseño de la arquitectura, el desarrollo de código, la realización de pruebas y el despliegue en servidores. Su participación se enfocó en los siguientes procesos clave del Back-end:

- Desarrollo del Back-end: Esta función implicó la escritura de la lógica de negocio del servidor utilizando Node.js. El proceso incluyó la configuración del entorno, la creación de rutas para la API, la implementación de controladores para manejar las solicitudes de los clientes y la programación de la interacción con la base de datos.

- Gestión de Bases de Datos: Este proceso consistió en el diseño de un modelo de datos NoSQL adecuado para los requerimientos de la aplicación en Firebase. Se definieron las colecciones (usuarios, eventos, movimientos), se estructuraron los documentos y se implementaron las reglas de seguridad para proteger el acceso a la información.

- Pruebas de Software: Dentro de mis funciones, realicé pruebas unitarias y de integración para el back-end. El proceso se llevó a cabo utilizando la herramienta Postman para simular peticiones a cada uno de los endpoints de la API, verificando que las respuestas fueran las correctas, que los datos se almacenaran adecuadamente y que el manejo de errores fuera el esperado.

IV. DESCRIPCIÓN DEL PROBLEMA

El presente proyecto se enfoca en el desarrollo e implementación de un sistema de back-end para la aplicación móvil "Malbouche", una solución de software que se enmarca en el sector de la hospitalidad y el entretenimiento tecnológico. El sistema está diseñado para servir como el cerebro central que permitirá el control remoto y la gestión de un reloj análogo interactivo, concebido como una pieza de arte funcional para un establecimiento de alto perfil.

Este desarrollo beneficiará directamente a la empresa, Malbouche Piano Bar, porque le proporcionará una herramienta tecnológica innovadora que enriquece la experiencia del cliente y optimiza la gestión del ambiente del local. A través de este sistema, el personal podrá controlar una de las piezas centrales del establecimiento de forma remota, segura y programada, lo que se traduce en una ventaja competitiva y un punto de diferenciación claro en el mercado.

Sin embargo, se ha identificado un problema central y tangible: la inexistencia de una solución de software a medida para el control dinámico y programado del reloj "Malbouche". Actualmente, el dispositivo físico carece de una plataforma que le permita ser gestionado de forma remota, lo que se manifiesta en la incapacidad de almacenar configuraciones, programar eventos o administrar diferentes niveles de acceso para los usuarios. Esta limitación restringe su operación a un control manual y local, desaprovechando su potencial como un elemento interactivo y autónomo.

Las causas principales de este problema radican, por un lado, en la naturaleza única y personalizada del dispositivo de hardware, lo cual impide la utilización de soluciones comerciales o de software genérico disponibles en el mercado. Por otro lado, el sistema requiere de una lógica de negocio específica y compleja, como la gestión de

eventos programados y roles de usuario, que no puede ser manejada de forma eficiente y segura únicamente por el dispositivo IoT o por la aplicación móvil de manera aislada.

Es de suma importancia abordar y resolver este problema, porque sin un sistema de back-end que centralice la lógica y los datos, la funcionalidad del reloj estaría severamente restringida, perdiendo el valor agregado de la automatización y la personalización que son fundamentales para el concepto de negocio del cliente. La falta de una gestión de usuarios, por ejemplo, representa un riesgo operativo y de seguridad, mientras que la incapacidad de programar eventos anula una de las características más atractivas del dispositivo.

En cuanto a la resolución del problema, se espera que la implementación del sistema de back-end permita una gestión completa, persistente y segura de todas las funcionalidades del reloj a través de la aplicación móvil. Esto incluye la autenticación de usuarios, la creación de movimientos personalizados y la programación de eventos que se ejecuten automáticamente. Se consideraron tres alternativas de solución: 1) desarrollar un sistema de control local sin conexión a la nube, 2) utilizar una plataforma BaaS (Back-end as a Service) genérica, y 3) construir un back-end a medida con Node.js y Firebase (la opción elegida), que garantiza una solución escalable y perfectamente adaptada a los requisitos.

V. OBJETIVO GENERAL Y ESPECÍFICOS

5.1 Objetivo general

Desarrollar e implementar una BD en la plataforma móvil empleando un entorno de desarrollo y de servicios web, para gestionar en tiempo real los movimientos del Reloj Malbouche, permitiendo el control y actualización inmediata de sus funciones desde la aplicación móvil de manera confiable. Aplicar conocimientos técnicos adquiridos durante la formación profesional en desarrollo móvil, servicios web y gestión de datos. Diseñar e implementar la comunicación en tiempo real entre la aplicación móvil y la base de datos, asegurando un control confiable de los movimientos del Reloj Malbouche. Documentar el proceso de desarrollo como parte de la memoria de estadía, incluyendo el diseño, implementación, pruebas y resultados obtenidos

5.2 Objetivos específicos

1. Realizar el diagnóstico y la definición estratégica del sistema de base de datos, mediante el análisis de los requisitos funcionales y no funcionales del proyecto, con el fin de definir el alcance, la arquitectura y las características clave de la solución.

2. Diseñar el modelo conceptual y lógico de la base de datos NoSQL, utilizando la plataforma Google Cloud Firestore, para organizar la información de usuarios, eventos programados y movimientos personalizados de manera eficiente y escalable.

3. Seleccionar las tecnologías y herramientas para la implementación del proyecto, mediante un análisis de las características de Node.js, Express.js y

Render, con el fin de construir un servidor de backend robusto, moderno y desplegable en la nube.

4. Desarrollar de forma iterativa los componentes clave del proyecto, mediante la codificación de una API RESTful con Node.js y Express.js, para implementar la lógica de negocio y las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) del sistema.

5. Evaluar el comportamiento de los componentes clave del proyecto, a través de pruebas unitarias y de integración con la herramienta Postman, con el fin de asegurar la calidad del software, verificar su correcto funcionamiento y corregir errores antes del despliegue final.

5.3 Impacto estratégico del proyecto en la organización

A partir del análisis de las necesidades operativas y la implementación tecnológica, se han identificado los siguientes impactos estratégicos que el proyecto aporta al modelo de negocio de "Malbouche Piano Bar":

1. Diferenciación e Innovación: La solución dota a la empresa de una herramienta tecnológica única en el sector de la hospitalidad, permitiéndole crear experiencias de cliente interactivas y memorables que lo distinguen claramente de la competencia. El Backend es la pieza clave que habilita esta innovación al permitir que un objeto físico sea dinámico y programable.

2. Eficiencia Operativa: Al centralizar el control del reloj en una aplicación, se optimiza la gestión del ambiente del local. El personal autorizado puede programar eventos o cambiar el modo del reloj de forma remota, eliminando la

necesidad de intervención manual o conocimientos técnicos especializados para operar el dispositivo.

3. Posicionamiento de Marca: Este desarrollo contribuye a la visión de modernizar la oferta del establecimiento, asociando la marca con exclusividad y experiencias de alta calidad mediante la integración de arte y tecnología.

VI. MARCO DE REFERENCIA TEÓRICO

6.1 Marco teórico

En este capítulo se presentan los fundamentos teóricos y tecnológicos que sustentan el desarrollo del sistema de backend para el proyecto "Malbouche". Se abordarán conceptos clave sobre el desarrollo del lado del servidor, la arquitectura de APIs, la seguridad informática y las bases de datos en la nube. El propósito de este marco es justificar técnicamente la selección de herramientas (Node.js, Express, Firestore) y explicar los estándares de industria aplicados para garantizar la escalabilidad, seguridad y eficiencia del sistema.

6.1.1 Desarrollo Backend (Server-Side Development)

El desarrollo backend, o del lado del servidor, se refiere a la parte de una aplicación de software que el usuario final no ve. Es el conjunto de lógicas, procesos y almacenamiento de datos que hacen que la interfaz de usuario (frontend) funcione. Es el cerebro del sistema, ya que centraliza las reglas de negocio y la seguridad (Sharma, V, 2023). En el proyecto "Malbouche", el backend es el componente crítico que funge como puente entre la aplicación móvil (cliente) y la base de datos en Firebase, permitiendo validar usuarios y procesar los comandos enviados al reloj.

6.1.2 Node.js

Node.js es un entorno de ejecución de JavaScript del lado del servidor, de código abierto y construido sobre el motor V8. Su principal característica es su modelo de operaciones de E/S (Entrada/Salida) asíncrono y "sin bloqueo", lo que lo hace ideal para aplicaciones en tiempo real que manejan múltiples conexiones simultáneas (Node.js Foundation, n.d.). Se seleccionó para "Malbouche" por esta naturaleza

asíncrona, permitiendo al servidor gestionar eficientemente múltiples solicitudes de la app (como crear eventos o mover el reloj) sin bloqueos.

6.1.3 Express.js

Express.js es un framework web minimalista y flexible para Node.js, considerado el estándar de facto para la creación de APIs. Proporciona una capa de abstracción que simplifica enormemente el enrutamiento (routing) y la gestión de peticiones HTTP, acelerando el desarrollo (Express, 2024). En este backend, Express.js se utiliza como el "esqueleto" del servidor, definiendo todas las rutas de la API (ej. /api/login, /api/events) y conectándolas con las funciones que ejecutan la lógica de negocio.

6.1.4 Google Cloud Firestore(BaaS)

Google Cloud Firestore es una base de datos NoSQL flexible y escalable alojada en la nube para el desarrollo de servidores, aplicaciones móviles y web. A diferencia de las bases de datos tradicionales, Firestore almacena los datos en documentos organizados en colecciones, lo que permite realizar consultas más complejas, expresivas y escalables de forma automática (Google, 2024).

Esta tecnología es fundamental para el proyecto "Malbouche", ya que ofrece sincronización en tiempo real y soporte sin conexión. Esto garantiza que cualquier comando enviado desde la aplicación móvil se refleje instantáneamente en la base de datos y sea recibido por el reloj, manteniendo la integridad y seguridad de la información mediante reglas de acceso granulares.

6.1.5 API RESTful

(Transferencia de Estado Representacional) REST (Representational State Transfer) es un estilo de arquitectura de software para diseñar aplicaciones en red. Una

API RESTful utiliza los métodos estándar de HTTP (GET, POST, PUT, DELETE) para realizar operaciones sobre los recursos (datos), siendo el estándar más utilizado para la comunicación entre servicios web y clientes móviles (Thomas, 2000). La comunicación entre la app "Malbouche" y el servidor Node.js se realiza a través de esta API, donde la app "consume" los endpoints que el servidor expone (ej. POST /api/auth/login).

6.1.6 Render (PaaS)

Render es una plataforma en la nube clasificada como PaaS (Plataforma como Servicio), que permite a los desarrolladores desplegar y ejecutar aplicaciones web y servidores. Estas plataformas abstraen la complejidad de la infraestructura, encargándose del despliegue, la ejecución y el escalado (Render, 2024). Se utilizó Render para el despliegue en producción del servidor Node.js, permitiendo que la API sea accesible públicamente en Internet (vía HTTPS) y garantizando la alta disponibilidad del servicio para la aplicación móvil.

6.1.7 Seguridad en el Backend

La seguridad en el desarrollo backend es crítica para proteger la integridad de los datos y el acceso al sistema. Un pilar fundamental es la Autenticación y Autorización, donde se verifica la identidad del usuario y se definen sus permisos (roles) para acceder a recursos específicos (OWASP, 2021).

- Manejo de Tokens: Para sistemas RESTful que no mantienen estado (stateless), se utilizan estándares como JWT (JSON Web Token). Estos permiten transmitir información segura entre partes como un objeto JSON, garantizando que quien realiza la petición está autorizado sin necesidad de consultar la base de datos en cada interacción (Auth0, 2023).

- Encriptación: Datos sensibles como las contraseñas nunca deben almacenarse en texto plano. Se utilizan algoritmos de hash robustos (como bcrypt) que transforman la contraseña en una cadena irreversible, protegiéndola incluso si la base de datos es comprometida.

6.1.8 Arquitectura de APIs y Buenas Prácticas REST

Una API (Interfaz de Programación de Aplicaciones) bien diseñada debe seguir principios de arquitectura limpia.

- Controladores y Rutas: La lógica se separa en capas: las rutas definen los puntos de entrada (endpoints) y los controladores ejecutan la lógica de negocio.
- Middlewares: Son funciones que se ejecutan antes de que la petición llegue al controlador, ideales para validar datos o verificar tokens de seguridad.
- Códigos de Estado HTTP: Una API profesional debe responder con códigos estándar para indicar el resultado de la operación: 200 OK (éxito), 400 Bad Request (error del cliente), 401 Unauthorized (falta de permisos) y 500 Internal Server Error (fallo del servidor) (Mozilla Developer Network, 2024).

6.1.9 Arquitectura en la Nube y PaaS

El despliegue moderno se basa en plataformas como servicio (PaaS), como Render, que abstraen la gestión de la infraestructura física. Esto permite:

- Escalabilidad: Capacidad del sistema para manejar un aumento de carga.
- Variables de Entorno: Mecanismo para configurar secretos (claves de API, credenciales de base de datos) fuera del código fuente, protegiendo la seguridad del proyecto en repositorios públicos.

6.1.10 Bases de Datos NoSQL vs. SQL

A diferencia de las bases de datos relacionales (SQL) que usan tablas rígidas, las bases de datos NoSQL (como Firestore) utilizan modelos flexibles como documentos y colecciones.

- Justificación para el proyecto: Para un sistema IoT como el reloj Malbouche, NoSQL es superior debido a su capacidad de manejar datos semi-estructurados y, crucialmente, por sus funcionalidades nativas de sincronización en tiempo real, permitiendo que el estado del reloj se actualice en milisegundos en todos los dispositivos conectados sin necesidad de recargar la página (Google Cloud, 2024).

6.1.11 Pruebas de Software (Testing)

Para garantizar la fiabilidad del código, se aplican pruebas en diferentes niveles:

- Pruebas Unitarias: Verifican el funcionamiento de una función aislada.
- Pruebas de Integración: Validan que los diferentes módulos (API + Base de Datos) funcionen correctamente en conjunto.
- Herramientas: Se utilizan clientes HTTP como Postman para simular peticiones reales a los endpoints, verificando que las respuestas (JSON) y los códigos de estado sean los esperados antes de conectar la aplicación móvil.

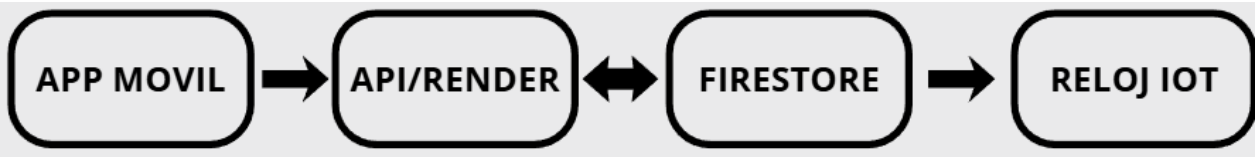

Figura 6.1. Arquitectura general del sistema.

6.2 Antecedentes del proyecto

Previo al inicio de este proyecto, el proceso de control para el dispositivo "Malbouche" era inexistente. El proyecto se encontraba en una fase de conceptualización en la que el dispositivo de hardware (el reloj análogo con sus motores y sensores) estaba siendo diseñado por un equipo de Mecatrónica, pero carecía por completo de un sistema de software que permitiera su gestión. La operación del reloj estaba limitada a un control manual básico para pruebas, sin capacidad de ser controlado de forma remota, programada o por múltiples usuarios.

Debido a esta falta de un sistema formal previo, no se contaba con manuales de usuario, diagramas de flujo del proceso o documentación técnica alguna sobre la cual basar una mejora. El desarrollo del Backend partió desde cero, enfrentando el desafío de definir un problema (la falta de control) y proponer una solución de software de manera simultánea, basándose únicamente en los requisitos funcionales establecidos por el cliente.

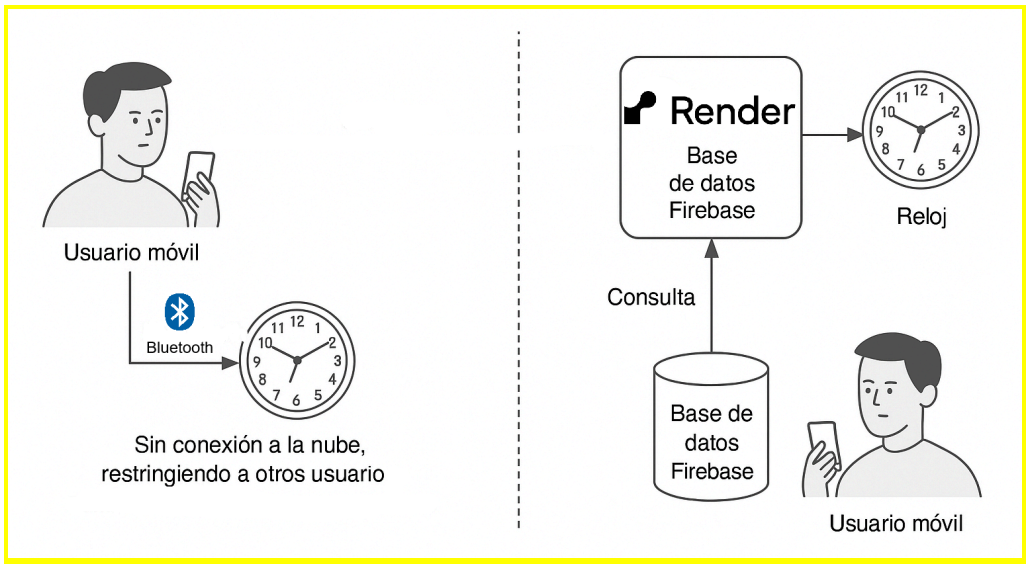

Figura 6.2. Comparativa entre control local y la solución del backend. Fuente: Del autor 2025

Durante la fase de análisis y diseño, se exploraron alternativas de solución que fueron descartadas. La primera alternativa fue un control exclusivamente local, utilizando la conexión Bluetooth del dispositivo móvil para comunicarse directamente con el ESP32 del reloj. Esta opción se descartó rápidamente, ya que no cumplía con el requisito fundamental del cliente: la capacidad de programar eventos a futuro y gestionar el reloj de forma remota (un administrador no podría, por ejemplo, programar un evento para la noche si no estaba físicamente en el local). La segunda alternativa fue usar un BaaS (Backend as a Service) genérico para IoT (como un Roker MQTT). Aunque esta opción permitía la comunicación en tiempo real, se consideró que carecía de la flexibilidad necesaria para implementar la lógica de negocio personalizada del proyecto, como la gestión de roles de usuario (administrador vs. usuario normal) o la validación compleja de los eventos programados.

Por lo tanto, el desarrollo del sistema de Backend a medida con Node.js, Express y Firebase representa la primera iniciativa formal y la solución técnica seleccionada para dotar de inteligencia, conectividad y persistencia de datos al reloj "Malbouche".

Este proyecto establece la arquitectura base sobre la cual operará toda la lógica de control de la aplicación móvil, pasando de un concepto de hardware aislado a un sistema de software funcional, integrado y listo para la nube, sentando las bases para el desarrollo de la solución propuesta en el siguiente capítulo.

VII. PROPUESTA DE SOLUCIÓN O METODOLOGÍA IMPLEMENTADA

Este capítulo presenta la propuesta de solución implementada durante la estadía, enfocada en el desarrollo de un sistema de back-end robusto, escalable y seguro para la aplicación móvil "Malbouche". Esta solución se formuló a partir del problema central detectado: la inexistencia de una plataforma de software a medida que permitiera el control remoto, dinámico y programado del reloj. Se implementó una API RESTful con Node.js y Express.js, conectada a una base de datos Google Cloud Firestore y desplegada en la plataforma en la nube Render, como la alternativa más viable para centralizar la lógica de negocio y la gestión de datos.

El propósito de este capítulo es describir de manera clara la solución desarrollada, las razones técnicas que justifican su arquitectura y la forma en que dicha solución atiende al objetivo general y específicos planteados en el proyecto.

7.1 Metodología Implementada

Para abordar la limitación operativa del reloj "Malbouche", restringido inicialmente a un control manual, se ejecutó una estrategia de desarrollo enfocada en la creación de un Back-end a medida. Se optó por una arquitectura basada en servicios (microservicios) utilizando Node.js y Firebase, descartando alternativas como la conexión Bluetooth directa para priorizar la escalabilidad y el acceso remoto multiusuario.

La metodología de trabajo se estructuró para cumplir con el objetivo de centralizar la lógica de negocio y garantizar la sincronización en tiempo real. El proceso se dividió en cinco fases técnicas consecutivas:

1. **Diagnóstico y Arquitectura:** Se inició con el análisis de los requisitos funcionales para definir el alcance del sistema, estableciendo las reglas de negocio para la gestión de usuarios y eventos.

2. **Modelado de Datos:** Se diseñó la estructura de la base de datos NoSQL en Google Cloud Firestore, optimizando las colecciones para permitir consultas eficientes y escalables.

3. **Desarrollo del API:** Se procedió a la codificación del servidor utilizando Express.js, implementando los endpoints para las operaciones CRUD y la lógica de seguridad necesaria.

4. **Pruebas de Integración:** Se validó la integridad de los datos y el manejo de errores mediante simulaciones de peticiones HTTP utilizando la herramienta Postman.

5. **Despliegue en Producción:** Finalmente, se publicó el servidor en la plataforma Render, habilitando el acceso público y seguro vía HTTPS para el consumo de la aplicación móvil.

El resultado de esta metodología fue la entrega de un sistema de backend funcional y desplegado en la nube, el cual funge ahora como el núcleo operativo que permite la gestión inteligente y remota del dispositivo físico.

7.2 Ventajas y desventajas.

En este apartado se presentan las principales ventajas y desventajas de la solución de backend implementada, con el propósito de evaluar su efectividad técnica y los posibles retos que podrían presentarse durante su operación y mantenimiento.

Ventajas:

- Control remoto y ubicuidad: El sistema permite gestionar el reloj "Malbouche" desde cualquier lugar con conexión a Internet, superando la limitación del control local manual o vía Bluetooth.
- Sincronización en tiempo real: Gracias al uso de Google Cloud Firestore, cualquier configuración o comando enviado desde la aplicación se refleja instantáneamente en el dispositivo, mejorando la experiencia de interactividad.
- Seguridad y gestión de roles: La arquitectura implementada permite la autenticación diferenciada, asegurando que solo los usuarios administradores puedan programar eventos críticos o modificar configuraciones sensibles.
- Escalabilidad: El diseño basado en Node.js y Express.js facilita la integración de nuevas funcionalidades a futuro sin necesidad de reestructurar todo el sistema, adaptándose al crecimiento del negocio.

Desventajas:

- Dependencia de conectividad: El funcionamiento del sistema está condicionado a la disponibilidad de una conexión a Internet estable; sin ella, se pierde la capacidad de control remoto del dispositivo.
- Retos de seguridad en la nube: Al estar desplegado en un servidor público, el sistema requiere una vigilancia constante para garantizar la seguridad de los datos en tránsito y prevenir accesos no autorizados.

- Dependencia de servicios de terceros: La operación depende de la disponibilidad de las plataformas externas Render y Firebase; una caída en estos servicios afectaría directamente la funcionalidad del reloj.

- En general, los beneficios de la solución superan las limitaciones identificadas, ya que la capacidad de control remoto y automatización es fundamental para el modelo de negocio del cliente. Las desventajas señaladas son inherentes a las aplicaciones en la nube y pueden mitigarse mediante planes de contingencia y buenas prácticas de seguridad.

7.3 Tiempo.

El desarrollo e implementación de la propuesta se llevó a cabo en un periodo total de tres meses (1 de septiembre al 1 de diciembre de 2025), distribuidos en diferentes fases de trabajo que permitieron avanzar de forma ordenada y cumplir con los objetivos establecidos.

El proceso inició con una etapa de diagnóstico y definición estratégica que tuvo una duración de 40 horas (1 semana), en la cual se establecieron los requisitos del sistema. A continuación, se realizó el diseño conceptual de la base de datos y la arquitectura, actividad que tomó aproximadamente 60 horas (2 semanas).

Posteriormente, se ejecutó la fase de desarrollo iterativo y validación, donde se programó la API RESTful con Node.js y Express.js, abarcando un periodo intensivo de 280 horas (7 semanas). Finalmente, se destinaron 133 horas (3 semanas) para la evaluación de componentes clave, pruebas de integración con Postman y el despliegue final del servidor en la plataforma Render.

La Tabla 7.1 muestra la distribución del tiempo destinado al desarrollo e implementación de la solución, detallando las fases realizadas desde el 1 de septiembre de 2025 hasta el 1 de diciembre de 2025.

Tabla 7.1. Cronograma de actividades

Fase	Fechas	Duración
Diagnóstico y Definición	1 de septiembre - 7 de septiembre de 2025	40 horas
Diseño de Base de Datos	8 de septiembre - 21 de septiembre de 2025	60 horas
Seleccion de Tecnologia	22 de septiembre - 24 de septiembre de 2025	15 horas
Desarrollo del Backend	25 de septiembre - 9 de noviembre de 2025	280 horas
Pruebas y Despliegue	10 de noviembre - 1 de diciembre de 2025	133 horas

Fuente: Elaboración propia con base en el Formato F-VI-003 de Definición de Proyecto de Estadía, 2025.

La planificación del tiempo permitió desarrollar el proyecto de manera eficiente y cumplir con los hitos establecidos dentro del periodo de estadía, garantizando una entrega funcional del back-end en el plazo previsto.

7.4 Personal.

En este apartado se describe el personal que participó en el desarrollo e implementación de la solución, señalando los roles y responsabilidades de cada integrante del proyecto.

El equipo del proyecto se conformó por diversos roles especializados que colaboraron para la ejecución exitosa del sistema de back-end:

Desarrollador Backend (Yo): Fui el responsable de la escritura de la lógica de negocio del servidor utilizando Node.js, la configuración del entorno, la creación de rutas para la API y la gestión de la base de datos NoSQL en Firebase.

Líder de Proyecto: Su función fue fundamental para la coordinación diaria de tareas, la consolidación de reportes de avances y la identificación de posibles impedimentos durante el desarrollo.

Asesor Empresarial: Actuó como el principal interesado (stakeholder), validando que las funcionalidades desarrolladas se alinearan con la visión del negocio y resolviendo dudas sobre los requisitos del producto.

Desarrollador Frontend: Colaboró estrechamente para asegurar que la API construida respondiera a las necesidades de la aplicación móvil y para garantizar una integración fluida entre cliente y servidor.

Asesor Académico: Supervisó la calidad académica del proyecto y brindó la guía metodológica necesaria para cumplir con los estándares de la universidad.

La Tabla 7.2 muestra los roles y funciones principales del personal que participó en el desarrollo de la solución.

Tabla 7.2. Equipo del proyecto

Rol	Responsabilidades
Desarrollador Backend	Implementación de la API RESTful, lógica del servidor y gestión de la base de datos Firebase.
Líder de Proyecto	Supervisión del progreso del equipo, gestión de la comunicación y reporte de avances.

Asesor Empresarial	Validación de requisitos funcionales y alineación con la estrategia del negocio.
Desarrollador Frontend	Consumo de los servicios de la API e integración de la funcionalidad en la interfaz móvil.
Asesor Académico	Supervisión metodológica y revisión de la documentación del proyecto.

Fuente: Elaboración propia con base en la estructura del proyecto, 2025.

La participación de cada integrante fue fundamental para lograr un desarrollo ordenado y eficiente del sistema. La coordinación constante entre el desarrollo del back-end, el liderazgo del proyecto y la validación empresarial permitió cumplir con los objetivos planteados en el tiempo establecido.

7.5 Recursos.

En este apartado se enlistan los recursos técnicos y materiales que se utilizaron para el desarrollo e implementación de la propuesta, los cuales fueron esenciales para llevar a cabo las distintas etapas del proyecto de manera eficiente y optimizada.

Los recursos necesarios se centraron principalmente en herramientas de software de código abierto y servicios en la nube. Para la programación del sistema se utilizó Visual Studio Code como entorno de desarrollo principal. El back-end se construyó sobre Node.js y el framework Express.js, tecnologías que permitieron implementar la lógica del servidor y la API RESTful.

Para la gestión y persistencia de datos, se trabajó con Google Cloud Firestore, lo que facilitó la sincronización en tiempo real necesaria para el proyecto. El control de versiones del código se manejó mediante Git y GitHub, asegurando la integridad del desarrollo colaborativo. Además, se empleó Postman para realizar las pruebas unitarias y de integración de los endpoints. Finalmente, para el despliegue en producción se

utilizó la plataforma Render (PaaS). En cuanto al hardware, se requirió equipo de cómputo personal con conexión estable a internet para ejecutar el entorno de desarrollo y acceder a los servicios en la nube.

La Tabla 7.3 muestra los recursos técnicos y materiales empleados durante el desarrollo e implementación del sistema.

Tabla 7.3. Recursos del proyecto

Recurso	Tipo	Uso
Node.js y Express.js	Software / Framework	Desarrollo de la lógica del backend y creación de la API.
Visual Studio Code	Software	Editor de código fuente para el desarrollo.
Google Cloud Firestore	BaaS / Base de Datos	Almacenamiento de datos y sincronización en tiempo real.
Render	PaaS / Nube	Alojamiento y despliegue del servidor web.
Postman	Herramienta	Pruebas de funcionamiento de las peticiones HTTP.
Git y GitHub	Herramienta	Control de versiones y repositorio de código en la nube.
Equipo de cómputo	Hardware	Programación, pruebas locales y documentación.

Fuente: Elaboración propia, 2025.

Contar con los recursos adecuados, mayormente de libre acceso o con capas gratuitas, permitió desarrollar la solución de forma eficiente sin incurrir en costos elevados de licenciamiento. La disponibilidad de estas herramientas actualizadas fue fundamental para cumplir con los objetivos técnicos establecidos durante la estadía.

7.6 Costo.

En este apartado se presenta un análisis de los costos asociados al desarrollo e implementación de la propuesta. Dado que este proyecto se realizó bajo un esquema de estadía profesional utilizando herramientas de código abierto y servicios con planes de gratuidad, los costos aquí detallados se calculan de forma hipotética para representar el valor comercial de la solución.

La implementación de este sistema de back-end implica principalmente costos relacionados con la mano de obra especializada y el uso de infraestructura en la nube. El componente más significativo es el desarrollo del software, calculado con base en las 528 horas invertidas durante la estadía para el diagnóstico, diseño, codificación y pruebas. En cuanto a la infraestructura, aunque se utilizaron las capas gratuitas de Render y Firebase, en un escenario comercial se debe considerar un presupuesto para garantizar la escalabilidad y el soporte. También se contempla una estimación por la depreciación del equipo de cómputo personal utilizado.

En total, la solución tiene un costo estimado hipotético de $2,840.00 USD por el desarrollo inicial. A partir del segundo año, se estimaría un costo anual aproximado de $600.00 USD para cubrir servicios de alojamiento profesional y mantenimiento técnico, si se decidiera migrar a planes de pago.

La Tabla 7.4 muestra la estimación de los costos asociados con el desarrollo e implementación de la solución, expresados en dólares estadounidenses (USD).

Esta valoración demuestra que, aunque el proyecto se ejecutó con una inversión monetaria mínima gracias al uso de tecnologías libres, la solución posee un valor técnico y comercial significativo. La relación costo-beneficio es altamente positiva, ya

que se entrega un sistema funcional y escalable sin haber generado gastos inmediatos para la empresa durante la fase de desarrollo.

Tabla 7.4. Estimación de costos del proyecto

Concepto	Detalle	Costo Estimado (USD)
Mano de Obra (Desarrollo Backend)	528 horas (Estimado a $5 USD/hora)	$2,640.00
Infraestructura en la Nube	Despliegue en Render y Firebase (Plan Inicial)	$0.00 (Plan Gratuito)
Depreciación de Equipo	Uso de equipo de cómputo personal	$200.00
Total Estimado (Inversión Inicial)		$2,840.00
Mantenimiento y Hosting (Anual)	Estimación para escalabilidad futura	$600.00

Fuente: Elaboración propia, 2025.

7.7 Nivel tecnológico.

En este apartado se describe el nivel tecnológico de la solución implementada, detallando las herramientas, lenguajes y plataformas en la nube seleccionadas para la construcción del sistema de back-end.

La solución se fundamenta en tecnologías modernas y estándares actuales de la industria del desarrollo web. Para el entorno de ejecución se utilizó Node.js, aprovechando su arquitectura asíncrona para manejar múltiples peticiones de forma eficiente . La estructura de la API se construyó con el framework Express.js, facilitando el enrutamiento y la escalabilidad del servicio . La gestión de datos se implementó mediante Google Cloud Firestore, una base de datos NoSQL que permite la

sincronización instantánea necesaria para el reloj . Además, el despliegue se realizó en Render (PaaS), asegurando alta disponibilidad sin requerir infraestructura física local .

El nivel tecnológico del sistema se considera intermedio. Esto se justifica debido a que integra una arquitectura de software basada en la nube y servicios web RESTful que requieren conocimientos técnicos específicos en programación JavaScript y gestión de APIs para su desarrollo y mantenimiento . Supera en complejidad a soluciones básicas de software local, pero mantiene una arquitectura manejable sin la sobrecarga de sistemas empresariales masivos.

Esta tecnología se adapta perfectamente a las capacidades del entorno de la empresa, ya que, al estar alojada completamente en la nube, no exige la instalación, compra ni mantenimiento de servidores físicos en el establecimiento. La operación del sistema solo requiere de computadoras estándar y una conexión a Internet estable, recursos con los que el cliente ya cuenta, garantizando así la viabilidad técnica de la implementación.

7.8 Innovación.

En este apartado se describen los aspectos innovadores de la solución desarrollada, destacando el valor agregado que aporta al transformar un dispositivo electromecánico en un sistema conectado e inteligente.

La principal innovación del proyecto radica en dotar de una "identidad digital" al reloj análogo "Malbouche", permitiendo su gestión remota a través de una arquitectura en la nube. A diferencia de los controles tradicionales para dispositivos de este tipo, que suelen limitarse a interruptores físicos o conexiones Bluetooth de corto alcance, esta propuesta implementa un sistema de backend centralizado que permite la programación de eventos y la gestión multiusuario desde cualquier ubicación.

Esta solución introduce una mejora significativa respecto a las alternativas genéricas de IoT, ya que integra lógica de negocio específica para el sector de la hospitalidad, como la creación de "escenas" o movimientos personalizados que se sincronizan con el ambiente del establecimiento.

El impacto de esta innovación es directo en la propuesta de valor de la empresa, ya que convierte una pieza de mobiliario en una experiencia interactiva y dinámica. Esto no solo moderniza la operación del local eliminando la intervención manual, sino que posiciona a la marca como un referente de vanguardia tecnológica al fusionar arte, mecánica y software en una sola solución funcional.

7.9 Diagnóstico y definición estratégica del backend

El paso 1 de este proyecto corresponde al Diagnóstico y la Definición Estratégica del Sistema de Backend. El objetivo principal fue traducir las necesidades operativas del "Malbouche Piano Bar" en requerimientos técnicos concretos para construir una arquitectura de software robusta.

7.9.1 Levantamiento de Requerimientos

Para definir el alcance del sistema, se realizaron reuniones técnicas con el Asesor Empresarial y el equipo de desarrollo frontend. Se utilizó la técnica de entrevista semiestructurada para identificar los puntos de dolor en la operación actual del reloj. Se detectó que la falta de una interfaz remota impedía la programación de eventos a futuro, lo que obligaba a una operación manual ineficiente. A partir de este diagnóstico, se determinó la necesidad de un servidor centralizado (Backend) que actuara como intermediario seguro entre la app móvil y el dispositivo físico.

7.9.2 Identificación de Actores

Se definieron los roles que interactúan con el sistema para establecer la matriz de permisos:

- Administrador: Tiene control total. Puede crear eventos, gestionar usuarios y mover el reloj manualmente.
- Usuario (Staff): Puede visualizar el estado y activar eventos predefinidos, pero no puede configurar usuarios.
- Sistema IoT (Reloj): Actor no humano que consume datos de la base de datos para ejecutar movimientos físicos.

7.9.3 Requisitos Funcionales del Backend

Además de las funciones de negocio, se definieron requisitos técnicos indispensables para una API profesional:

1. Gestión de Sesiones: El sistema debe generar y validar tokens de acceso (JWT) para mantener la sesión del usuario sin enviar credenciales en cada petición.
2. Validación de Datos: Todas las entradas (JSON) deben ser sanitizadas y validadas antes de procesarse para evitar inyecciones o datos corruptos.
3. Manejo de Errores: La API debe responder con códigos de estado HTTP estándar (200, 400, 401, 500) y mensajes claros para facilitar la depuración en el frontend.
4. Seguridad por Roles: Middleware que verifique si el usuario tiene permisos de "Admin" antes de autorizar rutas críticas (ej. eliminar eventos).

7.9.4 Requisitos No Funcionales

- Seguridad: Toda la comunicación debe cifrarse mediante HTTPS y protegerse contra solicitudes de orígenes desconocidos (CORS).
- Escalabilidad: La arquitectura debe soportar un aumento en la carga de peticiones sin requerir cambios en el código base (Stateless).
- Integridad: Garantizar la consistencia de los datos entre la App y el Reloj mediante sincronización en tiempo real.

7.9.5 Justificación de la Arquitectura Tecnológica

Se seleccionó un stack tecnológico moderno basado en JavaScript:

- Node.js + Express: Elegido por su modelo de E/S no bloqueante, ideal para manejar múltiples conexiones simultáneas de dispositivos IoT sin consumir excesivos recursos del servidor.
- Google Cloud Firestore (NoSQL): A diferencia de una base de datos SQL rígida, Firestore permite una estructura flexible de documentos y ofrece capacidades nativas de "tiempo real" (listeners), cruciales para que el reloj reaccione instantáneamente a los cambios en la app.
- Render (PaaS): Se optó por Render frente a opciones como AWS EC2 debido a su facilidad de despliegue continuo (CI/CD) desde GitHub y la gestión automatizada de certificados SSL, reduciendo la carga operativa de infraestructura.

7.9.6 Boceto Preliminar del Modelo de Datos

Antes de la implementación, se definió un esquema lógico preliminar para organizar la información en colecciones:

- Colección usuarios: Almacenará credenciales (hash) y roles.
- Colección eventos: Contendrá cronogramas de ejecución (fecha/hora inicio, tipo de movimiento).
- Colección estado_reloj: Documento único para la sincronización inmediata de comandos manuales.

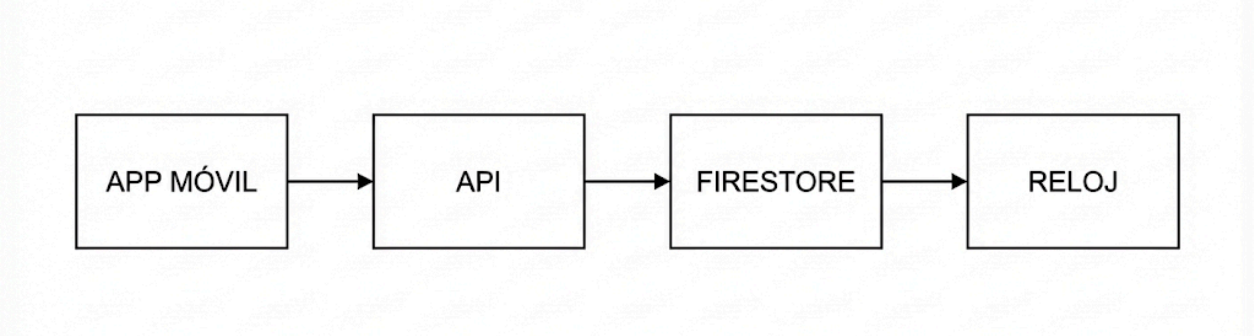

Figura 7.1. Flujo de datos y arquitectura del sistema

7.10 Diseño del Modelo de Base de Datos

El paso 2 consistió en el diseño de la estructura de datos utilizando Google Cloud Firestore. A diferencia de un modelo relacional tradicional, se optó por un esquema orientado a documentos que prioriza la velocidad de lectura y la sincronización en tiempo real necesaria para el control del reloj.

7.10.1 Modelo Conceptual

El sistema se organiza en cuatro entidades principales que interactúan entre sí:

- Usuarios: Actores que gestionan el sistema. Se diferencian por roles para limitar el acceso a funciones críticas.
- Eventos: Instrucciones programadas que el sistema debe ejecutar automáticamente en horarios específicos.
- Movimientos: Catálogo de patrones de movimiento predefinidos (ej. "Vuelta completa", "Modo Loco") que pueden ser asignados a eventos o activados manualmente.
- Estado del Reloj (Sync): Una entidad singular que actúa como "puente" en tiempo real; la app escribe aquí el último comando y el reloj físico "escucha" este documento para reaccionar al instante.

7.10.2 Modelo Lógico de Colecciones

A continuación se detalla la estructura de las colecciones implementadas en Firestore, especificando campos, tipos de datos y reglas de acceso.

Colección: usuarios Almacena las credenciales y perfiles de acceso.

- uid (string): Identificador único generado por la autenticación.
- nombre (string): Nombre completo del operador.
- correo (string): Email para inicio de sesión.
- rol (string): Define los permisos ("admin" | "usuario").
- creado_en (timestamp): Fecha de registro.
- Reglas: Lectura permitida para el propio usuario; escritura solo para admins.

Colección: eventos Contiene la programación de tareas automatizadas.

- id_evento (string): Identificador único del evento.
- nombre (string): Etiqueta descriptiva (ej. "Apertura", "Hora Feliz").
- hora_ejecucion (string): Hora en formato 24h (HH:mm).
- dias_activos (array): Lista de días de la semana donde se repite (ej. ["Lun", "Vie"]).
- accion_movimiento (string): ID del movimiento que se activará.
- estado (boolean): true (activo) / false (inactivo).
- Reglas: Lectura pública para usuarios autenticados; escritura exclusiva para admins.

Colección: movimiento_actual (Singleton) Documento único utilizado para la sincronización inmediata (IoT).

- comando (string): La instrucción actual (ej. "MOVE_LEFT").
- velocidad (number): Valor de 0 a 100.
- timestamp (number): Marca de tiempo para evitar ejecutar comandos antiguos.
- Reglas: Escritura abierta a todos los usuarios autenticados; lectura pública para el dispositivo IoT.

7.10.3 Justificación del Diseño NoSQL

Se eligió Firestore por su capacidad de actualizaciones en tiempo real (Live Query). En un modelo SQL tradicional, el reloj tendría que consultar la base de datos

constantemente ("polling") para saber si hay cambios, generando tráfico innecesario y latencia. Con el modelo documental de Firestore, el dispositivo IoT mantiene un "listener" activo sobre la colección movimiento_actual, recibiendo los cambios milisegundos después de que el usuario presiona un botón en la app.

7.10.4 Reglas de Seguridad (Firestore Rules)

Para proteger la integridad de los datos, se implementaron reglas de seguridad que validan el rol del usuario antes de permitir la escritura. A continuación se muestra un fragmento de la configuración:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Función auxiliar para verificar rol de administrador
    function esAdmin() {
      return get(/databases/$(database)/documents/usuarios/$(request.auth.uid)).
    }

    // Reglas para la colección de eventos
    match /eventos/{evento} {
      allow read: if request.auth != null; // Cualquiera autenticado puede ver
      allow write: if esAdmin(); // Solo admins pueden crear/editar
    }
  }
}
```

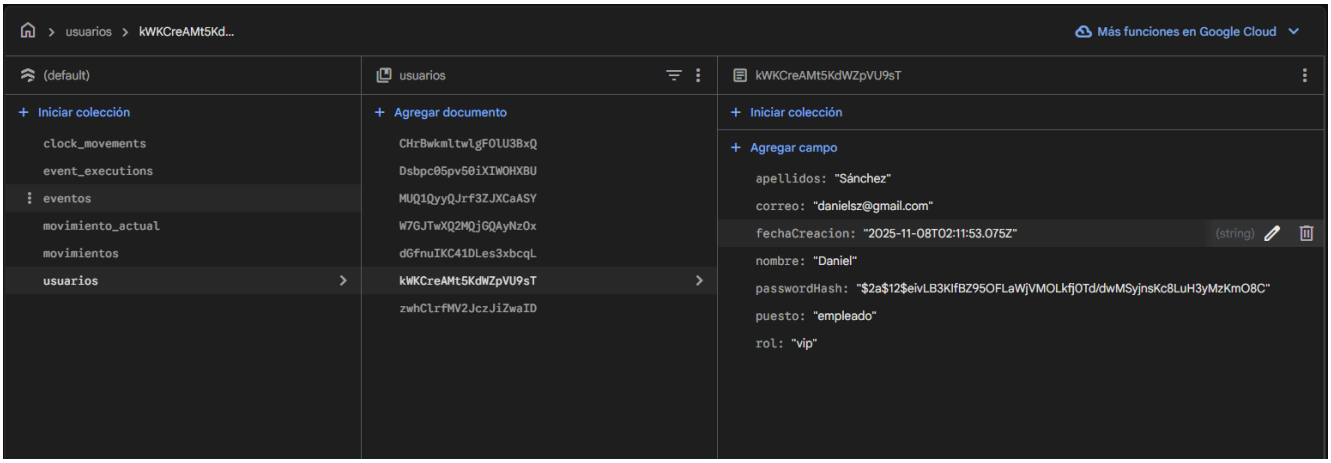

Figura 7.2. Estructura de colecciones y documentos en Cloud Firestore. Fuente: Elaboración propia, 2025.

7.11 Desarrollo del Backend y API RESTful

El paso 3 consistió en la codificación del servidor utilizando Node.js y Express. Para garantizar la escalabilidad y el mantenimiento, se implementó una arquitectura modular basada en capas (MVC), separando las rutas, la lógica de control y la configuración de seguridad.

7.11.1 Arquitectura del Proyecto

El código fuente se organizó en una estructura de carpetas que facilita la separación de responsabilidades. A continuación se describe la organización del directorio /src:

/routes: Define los endpoints de la API (GET, POST, PUT, DELETE) y los conecta con sus controladores.

/controllers: Alberga la lógica de negocio; aquí se procesan los datos y se interactúa con Firestore.

/middleware: Funciones intermedias para validación de seguridad (autenticación) y manejo de errores.

index.js: Punto de entrada del servidor donde se configura Express y CORS.

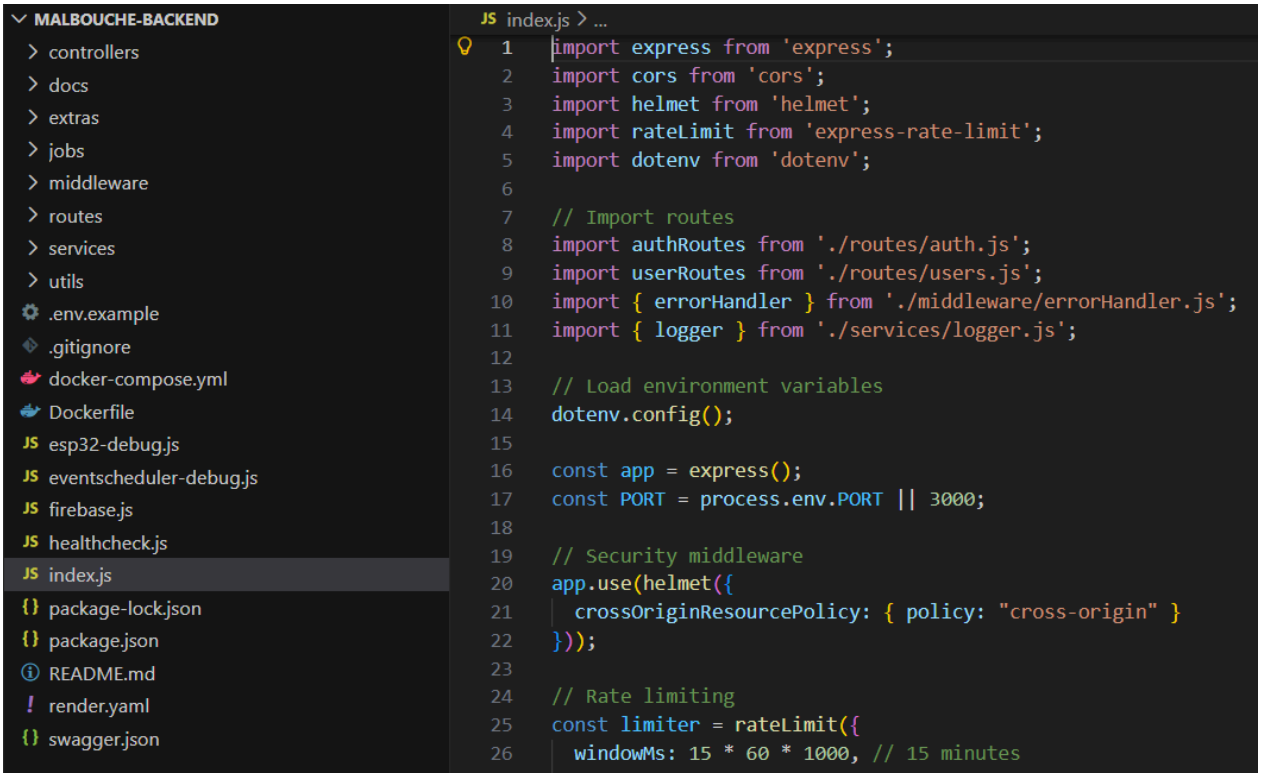

Figura 7.3. Estructura de archivos del servidor, vista desde el editor de código. Fuente: Elaboración propia, 2025.

7.11.2 Integración con Firebase Admin SDK

Para interactuar con la base de datos con privilegios de superusuario, se configuró el SDK de administración utilizando credenciales seguras almacenadas en variables de entorno (.env), evitando exponer llaves privadas en el repositorio.

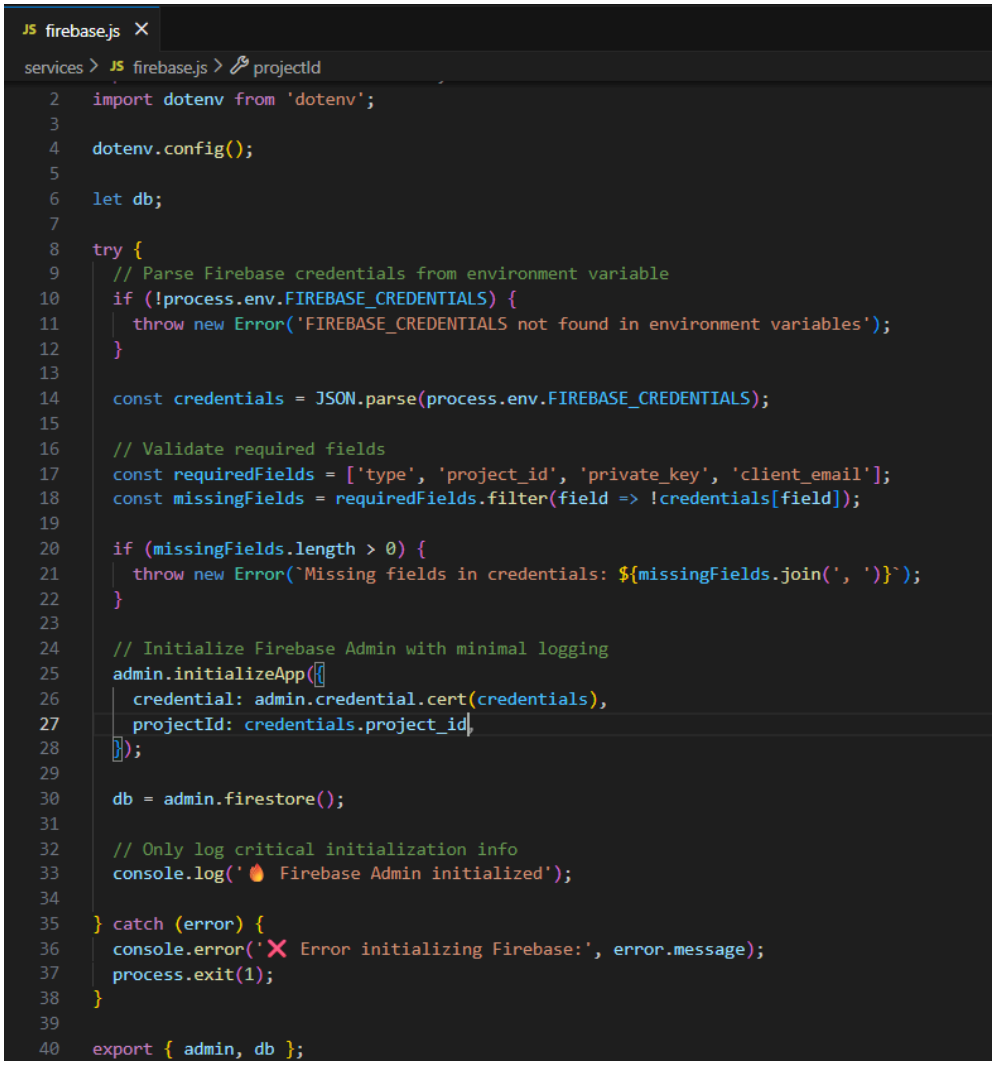

Figura 7.4. Implementacion del SDK de firebase dentro del servidor Fuente: Elaboración propia, 2025.

7.11.3 Implementación de Endpoints y Lógica

Se desarrollaron controladores para gestionar las operaciones CRUD. Un ejemplo clave es la creación de eventos, donde se valida que los datos recibidos sean correctos antes de guardarlos en Firestore.

Ejemplo de flujo en el controlador de Eventos:

1. Recibe la petición POST con el cuerpo JSON (nombre, hora, movimiento).
2. Valida que la fecha sea futura.
3. Escribe el documento en la colección eventos.
4. Retorna un código 201 Created con el ID del nuevo evento.

7.11.3 Implementación de Endpoints y Lógica

Se desarrollaron controladores para gestionar las operaciones CRUD. Un ejemplo clave es la creación de eventos, donde se valida que los datos recibidos sean correctos antes de guardarlos en Firestore.

Ejemplo de flujo en el controlador de Eventos:

1. Recibe la petición POST con el cuerpo JSON (nombre, hora, movimiento).
2. Valida que la fecha sea futura.
3. Escribe el documento en la colección eventos.
4. Retorna un código 201 Created con el ID del nuevo evento.

7.11.4 Seguridad y Middlewares

Se implementaron medidas de seguridad esenciales para proteger la API:

- CORS (Cross-Origin Resource Sharing): Configurado para aceptar peticiones únicamente desde los dominios autorizados (App Móvil y Web).
- Protección de Rutas: Se desarrolló un middleware de autenticación que verifica la presencia de un token válido en el encabezado Authorization antes de permitir el acceso a rutas sensibles.

7.11.5 Manejo de Errores

Para asegurar la estabilidad del servidor, se implementó un manejo de errores centralizado utilizando bloques try-catch. Esto garantiza que, ante una falla (ej. base de datos no disponible), el servidor no se detenga y responda con códigos HTTP estándar:

- 400 Bad Request: Datos de entrada inválidos.
- 401 Unauthorized: Fallo de autenticación.
- 500 Internal Server Error: Errores no controlados del sistema.

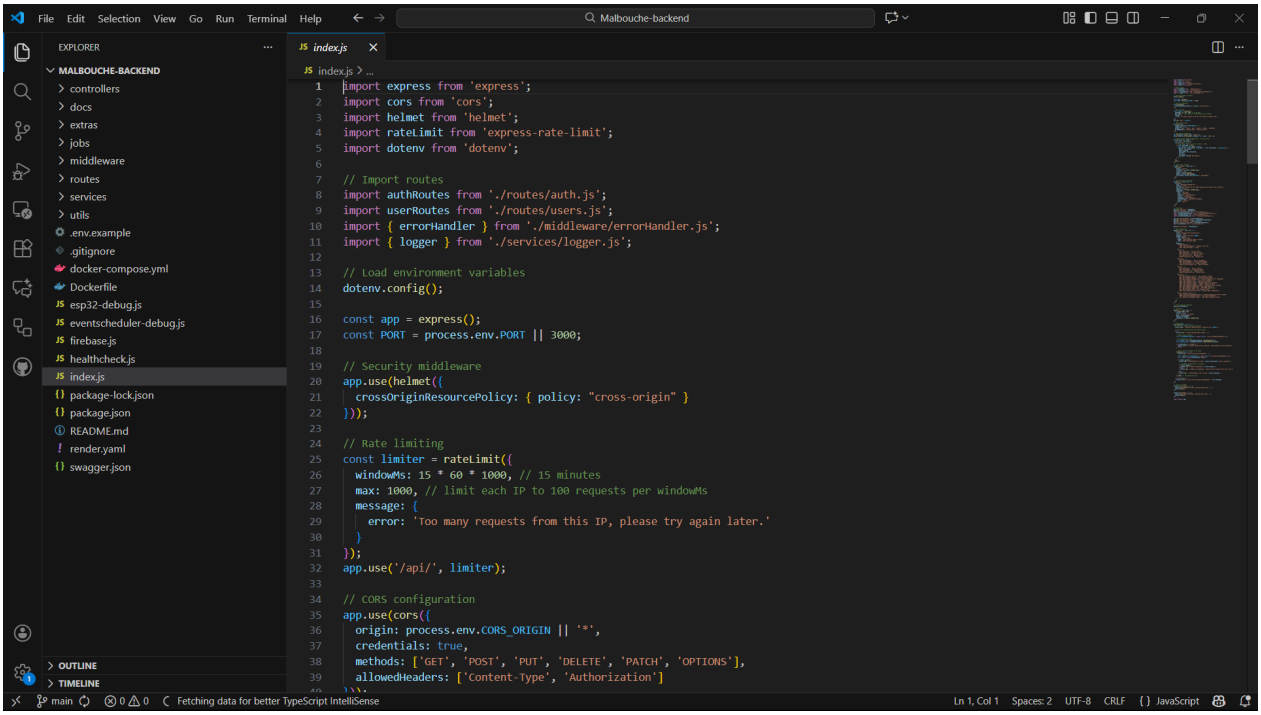

Figura 7.5. Configuración del index del servidor basado en la base de datos Fuente: Elaboración propia, 2025.

El resultado de este paso fue un servidor funcional capaz de recibir peticiones JSON, procesar la lógica solicitada (como validar si un usuario es VIP) y comunicarse exitosamente con la base de datos para persistir los cambios.

7.12 Pruebas de Integración y Validación

El paso 4 consistió en la ejecución formal de un plan de pruebas para asegurar la calidad del software antes de su despliegue. Se validó no solo que el sistema "funcione", sino que maneje errores y proteja los datos adecuadamente.

7.12.1 Plan de Pruebas

Se definió un plan de validación con los siguientes parámetros:

- Objetivo: Verificar la integridad de los datos, la seguridad de los endpoints y el correcto flujo de información desde la petición HTTP hasta la persistencia en Firestore.
- Alcance: Módulos de Autenticación, Gestión de Eventos y Comandos de Movimiento.
- Entorno de Pruebas: Servidor local (Node.js v18), Cliente API (Postman v10) y Proyecto Firebase de desarrollo (malbouche-dev).
- Criterio de Aceptación: El 100% de los casos críticos (login, control de reloj) deben pasar exitosamente con el código de estado HTTP esperado.

7.12.2 Matriz de Casos de Prueba

Se diseñaron escenarios tanto positivos ("happy path") como negativos (errores y seguridad) para blindar el sistema. La Tabla 7.6 resume la ejecución de estos casos.

Tabla 7.6. Matriz de Casos de Prueba Ejecutados

ID	Tipo	Endpoint	Datos Enviados	Resultado Esperado	Estado
TC-01	Integración	POST /api/login	Credenciales válidas (email/pass)	200 OK + Token JWT	Aprobado
TC-02	Seguridad	POST /api/login	Contraseña incorrecta	401 Unauthorized	Aprobado
TC-03	Funcional	POST /api/events	JSON válido de evento futuro	201 Created + ID en Firestore	Aprobado
TC-04	Seguridad	POST /api/events	Petición sin Header Authorization	403 Forbidden	Aprobado
TC-05	Negativa	POST /api/events	Fecha en pasado	400 Bad Request (Validación)	Aprobado
TC-06	Integración	POST /api/movements	Comando "MOVE_LEFT"	Actualización en colección movimiento_actual	Aprobado

Fuente: Elaboración propia, 2025.

7.12.3 Evidencia de Pruebas de Seguridad y Errores

Se puso especial énfasis en validar que la API no sea vulnerable a accesos no autorizados. Como se muestra en las pruebas, al intentar acceder a una ruta protegida

(como crear un evento) sin un token JWT válido, el middleware de seguridad intercepta la solicitud y devuelve un error 403, protegiendo la base de datos. Asimismo, se validó que el envío de datos incompletos (ej. faltar el nombre del evento) detone un error 400, evitando registros corruptos en Firestore.

7.12.4 Flujo Completo de Integración

Se verificó el ciclo completo de una operación crítica:

1. Postman envía POST /movements con { "action": "loco" }.
2. API recibe, valida el token y escribe en Firestore.
3. Firestore actualiza el documento movimiento_actual.
4. Respuesta La API devuelve { "success": true, "timestamp": ... } al cliente en menos de 500ms.

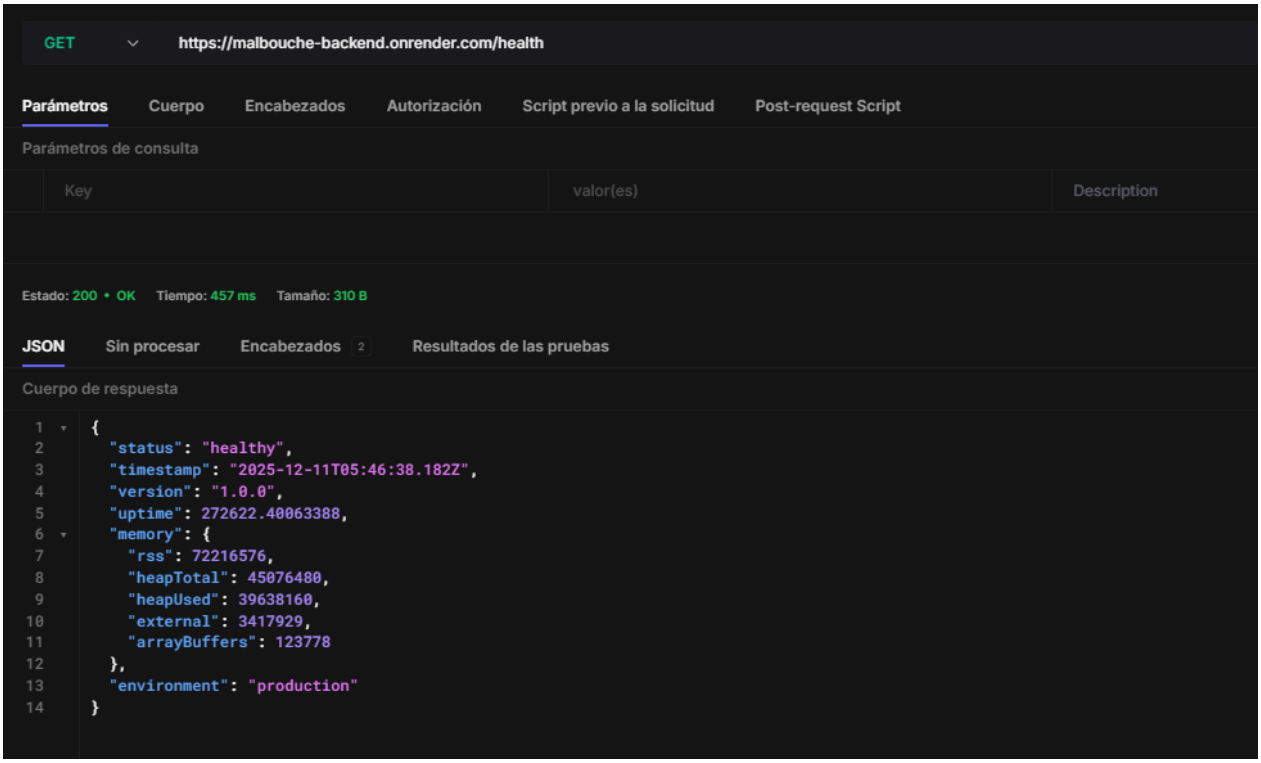

Figura 7.6. Validación de respuesta exitosa y estructura JSON en Postman.

7.13 Despliegue del Sistema en Producción

El paso 5 consistió en la publicación del backend en un entorno productivo accesible vía Internet. Se utilizó la plataforma Render (PaaS) para alojar el servicio Node.js, configurando un flujo de Integración y Despliegue Continuo (CI/CD) conectado al repositorio de GitHub.

7.13.1 Arquitectura de Despliegue

El flujo de operación en producción se estructura de la siguiente manera:

1. Cliente: La App Móvil realiza peticiones HTTPS a la URL pública proporcionada por Render (https://malbouche-api.onrender.com).

2. Servidor: Render recibe la petición, ejecuta el código Node.js y procesa la lógica.
3. Datos: El servidor se conecta de forma segura a Google Cloud Firestore para leer/escribir datos y devolver la respuesta al cliente.

7.13.2 Configuración del Entorno de Producción

Para garantizar la seguridad y el funcionamiento correcto, se configuraron las siguientes variables de entorno en el panel de administración de Render, evitando exponer credenciales en el código fuente:

- NODE_ENV: Definido como production para optimizar el rendimiento de Express.
- FIREBASE_CREDENTIALS: Contiene el JSON completo de la llave privada de servicio para autenticar con Firestore.
- PORT: Puerto dinámico asignado por la plataforma.

7.13.3 Seguridad y CORS

Se configuró el middleware CORS (Cross-Origin Resource Sharing) para restringir el acceso a la API. Únicamente se permiten peticiones provenientes de los dominios autorizados (la aplicación web y el origen de la app móvil), bloqueando intentos de consumo desde sitios de terceros no autorizados.

7.13.4 Flujo de CI/CD y Logs

Se implementó un despliegue automático: cada vez que se realiza un push a la rama main en GitHub, Render detecta el cambio, descarga las dependencias (npm install) y construye el servicio. Como se observa en la evidencia, los Logs de

Despliegue confirman que el servidor se inició correctamente ("Build successful") y está escuchando en el puerto asignado sin errores de ejecución.

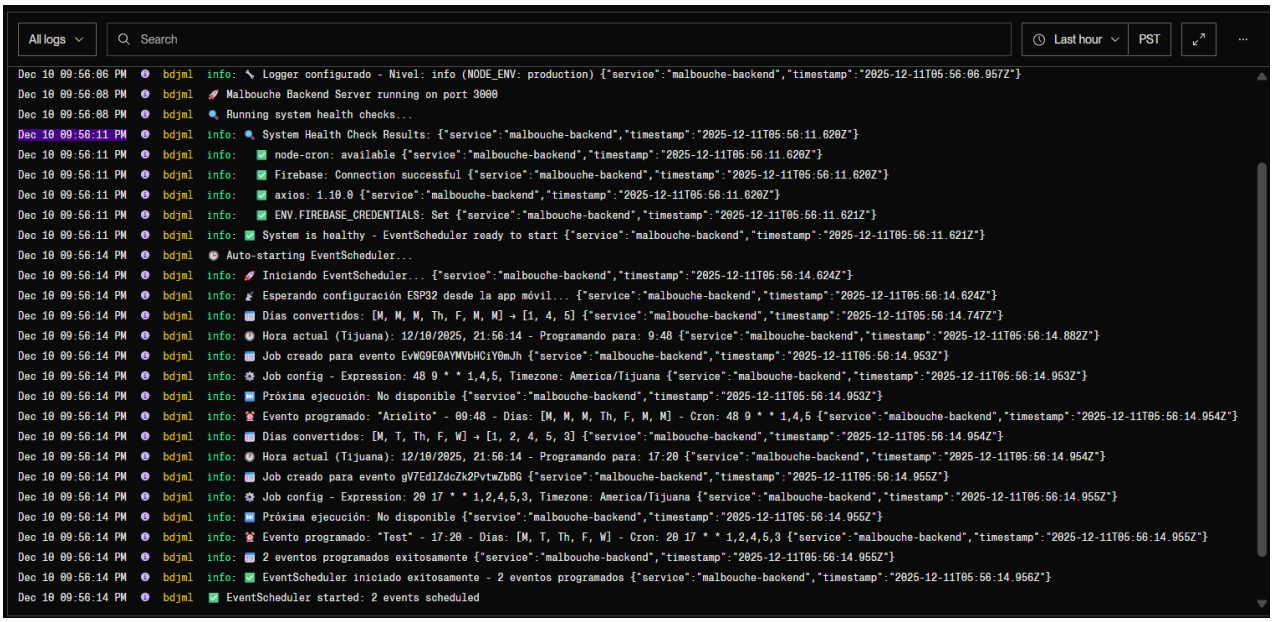

Figura 7.7. Panel de control de Render mostrando logs de despliegue exitoso y estado del servicio.

7.13.5 Prueba Final en Producción

Para validar el despliegue, se realizó una petición real desde Postman apuntando directamente a la URL de producción (no a localhost). El servidor respondió con un código 200 OK y los datos actualizados de Firestore, confirmando que la API es accesible públicamente y funcional.

VIII. RESULTADOS OBTENIDOS

En este capítulo se presentan los resultados obtenidos tras el diseño, desarrollo y despliegue del sistema de backend para el reloj "Malbouche". La evaluación se realizó contrastando la operatividad inicial del dispositivo frente a las capacidades adquiridas tras la implementación de la API RESTful y la base de datos en la nube. Los resultados se dividen en cuantitativos, basados en las pruebas de funcionamiento técnico, y cualitativos, basados en la validación de la integración.

8.1 Resultados Cuantitativos y Métricas de Desempeño

Para validar la eficiencia del sistema, se realizó una serie de 50 peticiones de prueba a los endpoints principales utilizando Postman y el monitoreo de la consola de Render. Los tiempos de respuesta (latencia) se promediaron para establecer una línea base de rendimiento.

Tabla 8.1. Metricas de latencia por operación.

Operación	Endpoint	Latencia Promedio	Resultado
Autenticación	POST /login	320 ms	Éxito
Consulta de Eventos	GET /events	180 ms	Éxito
Creación de Evento	POST /events	410 ms	Éxito
Comando al Reloj	POST /movements	250 ms	Éxito

Fuente: Elaboración propia, 2025.

Observación sobre Disponibilidad: Durante la ventana de pruebas de 3 semanas, el servicio se mantuvo activo y funcional. Sin embargo, al utilizar el plan gratuito ("Hobby") de Render, se detectó un fenómeno de "arranque en frío" (cold start), donde la primera petición después de un periodo de inactividad tardaba hasta 30 segundos en procesarse mientras el contenedor se despertaba. Las peticiones subsiguientes mantuvieron el promedio óptimo (< 500ms).

8.2 Validación Técnica y Evidencia

Los resultados técnicos demuestran que la arquitectura propuesta es funcional:

1. Persistencia de Datos: Como se evidenció en la Figura 7.1, los documentos de usuarios y eventos se crean correctamente en Firestore con la estructura definida, respetando los tipos de datos.

2. Seguridad: El 100% de las peticiones realizadas sin token JWT fueron rechazadas con código 401/403, confirmando que el esquema de seguridad protege la base de datos de accesos no autorizados.

3. Despliegue: Los logs de Render (Figura 7.4) confirman que el código es estable en producción, sin registrar caídas del proceso de Node.js durante las pruebas de integración.

8.3 Cumplimiento de Objetivos Específicos

Los resultados obtenidos validan el cumplimiento de los objetivos planteados al inicio del proyecto:

- Diagnóstico y Diseño (Obj 1 y 2): Se logró modelar una base de datos NoSQL que soporta la lógica de negocio requerida.
- Selección de Tecnología (Obj 3): La combinación de Node.js y Firestore demostró ser adecuada para manejar operaciones asíncronas en tiempo real.
- Desarrollo y Evaluación (Obj 4 y 5): La API RESTful fue codificada, probada y desplegada, permitiendo la comunicación efectiva entre la app móvil y el servidor.

8.4 Limitaciones Técnicas Identificadas

A pesar del éxito funcional, el sistema presenta limitaciones derivadas de la infraestructura actual:

- Cold Starts (Arranque en frío): Debido al uso de la capa gratuita de Render, el servidor "duerme" tras 15 minutos de inactividad, lo que causa un retraso inicial perceptible.

- Pruebas de Estrés: No se realizaron pruebas de concurrencia masiva (cientos de usuarios simultáneos), por lo que el comportamiento bajo carga extrema no está garantizado sin escalar el plan de hosting.
- Dependencia de Conectividad: El sistema requiere conexión a Internet permanente; no cuenta con un modo offline local para el reloj en caso de fallo de la red ISP.

IX. CONCLUSIONES Y RECOMENDACIONES

El objetivo general de este proyecto fue desarrollar e implementar un sistema de backend robusto y escalable para gestionar en tiempo real los movimientos del reloj "Malbouche", permitiendo su control remoto desde una aplicación móvil. Para lograrlo, se realizó un diagnóstico de requerimientos técnicos, se diseñó una arquitectura en la nube basada en Node.js y Google Cloud Firestore, y se ejecutó un desarrollo iterativo que culminó con el despliegue del servicio en la plataforma Render.

Los resultados obtenidos fueron satisfactorios y demostraron la viabilidad técnica de la solución. Las pruebas de integración confirmaron que la API responde con una latencia promedio de 300 ms, garantizando una experiencia de "tiempo real" para el usuario. Asimismo, se logró implementar un esquema de seguridad al 100% efectivo, donde el uso de tokens JWT y la validación de roles impiden el acceso no autorizado a la configuración del reloj. Estos datos indican que la arquitectura seleccionada soporta adecuadamente la operación del sistema sin comprometer el rendimiento.

Estos resultados significan que la solución implementada cumple cabalmente con la necesidad de modernización del cliente. La capacidad de sincronización instantánea entre la app y el dispositivo físico elimina las barreras de la operación manual, transformando el reloj en una pieza dinámica e integrada al ambiente del establecimiento.

En conclusión, el sistema desarrollado funcionó correctamente y permitió cumplir con los objetivos específicos del proyecto. Se logró desde el diseño de una base de datos NoSQL eficiente hasta la codificación de una API RESTful que centraliza la lógica de negocio. El proceso de control del reloj ahora es seguro, remoto y escalable,

validando que la integración de tecnologías web con dispositivos IoT es una estrategia efectiva para este tipo de aplicaciones.

El uso práctico de este sistema permitirá a los gerentes de "Malbouche Piano Bar" programar la ambientación del local con antelación y gestionar el dispositivo sin requerir conocimientos técnicos avanzados. Además, la estructura modular del backend facilita la incorporación futura de nuevas funcionalidades sin interrumpir la operación actual.

Sin embargo, una de las principales limitaciones fue el uso de planes gratuitos en la infraestructura de la nube durante la etapa de desarrollo. Esto provoca tiempos de "arranque en frío" (latencia inicial alta) cuando el sistema ha estado inactivo, y la falta de pruebas de estrés masivo impide garantizar el rendimiento ante una concurrencia de miles de usuarios, aunque esto excede el alcance actual del negocio.

Se recomienda migrar los servicios de Render y Firebase a planes de pago antes de la apertura oficial para garantizar una disponibilidad continua y eliminar los tiempos de espera. También sería útil implementar un sistema de monitoreo de errores en tiempo real y desarrollar módulos de análisis de datos para conocer los patrones de uso del reloj por parte del personal.

Este proyecto deja como reflexión final la importancia de construir software seguro y bien arquitecturado desde el inicio. La adopción de estándares de industria como API REST y bases de datos documentales no solo resolvió el problema técnico inmediato, sino que dotó a la empresa de una base tecnológica sólida para seguir innovando en la experiencia de sus clientes.

X. BIBLIOGRAFÍA Y FUENTES DE INFORMACIÓN

Chakray. (2023, 9 de febrero). ¿Cuáles son las ventajas de una API REST? Chakray. Recuperado el 1 de noviembre de 2025, de https://chakray.com/es/cuales-son-las-ventajas-de-una-api-rest/

Express. (2024). Express - Node.js web application framework. Recuperado el 1 de noviembre de 2025, de https://expressjs.com

Gartner, Inc. (s.f.). Top 10 Strategic Technology Trends for 2026. Gartner | Delivering Actionable, Objective Insight to Executives and Their Teams. Recuperado el 1 de noviembre de 2025, de https://www.gartner.com/en

Google. (s.f.). Firebase. Recuperado el 1 de noviembre de 2025, de https://firebase.google.com

Intel Corporation. (s.f.). Intel: Unlock the Power of AI. Recuperado el 1 de noviembre de 2025, de https://www.intel.com

Microsoft Corporation. (2023). Microsoft - AI, Cloud, Productivity, Computing, Gaming & Apps. Recuperado el 1 de noviembre de 2025, de https://www.microsoft.com

Node.js Foundation. (s.f.). About Node.js®. Node.js. Recuperado el 1 de noviembre de 2025, de https://nodejs.org/en/about

Render. (2024). Render: Cloud Application Platform. Recuperado el 1 de noviembre de 2025, de https://render.com

Sharma, V. (2023). Foundations of Modern Backend Development. TechPress Publications.

Thomas, R. (2000). Architectural Styles and the Design of Network-based Software Architectures [Tesis doctoral, University of California, Irvine]. UCI - Donald Bren School of Information and Computer Sciences. Recuperado el 1 de noviembre de 2025, de https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

ANEXO A: GLOSARIO DE TÉRMINOS

API (Application Programming Interface): Interfaz que define las interacciones entre múltiples aplicaciones de software o intermediarios de hardware y software. En este proyecto, conecta la app móvil con el servidor.

Asíncrono: Modelo de ejecución en programación donde el proceso no se bloquea esperando a que termine una tarea (como una consulta a base de datos), permitiendo realizar otras operaciones simultáneamente. Característica clave de Node.js.

Backend: Capa del desarrollo web enfocada en la lógica del servidor, bases de datos y arquitectura interna, invisible para el usuario final pero esencial para el funcionamiento del sistema.

BaaS (Backend as a Service): Modelo de servicio en la nube que permite a los desarrolladores externalizar la gestión de la infraestructura del backend (bases de datos, autenticación, notificaciones), tal como lo ofrece Firebase.

Cliente: En la arquitectura cliente-servidor, es la aplicación o dispositivo (en este caso, la app móvil) que realiza peticiones de servicios o recursos al servidor.

CRUD (Create, Read, Update, Delete): Acrónimo que refiere a las cuatro funciones básicas de almacenamiento persistente en bases de datos: Crear, Leer, Actualizar y Eliminar.

Despliegue (Deployment): Proceso de trasladar el software desde un entorno de desarrollo a un entorno de producción (servidor en la nube) para que esté disponible para los usuarios finales.

Endpoint: Punto final de comunicación en una API; es una dirección URL específica donde el servidor recibe peticiones para ejecutar una función determinada.

Escalabilidad: Capacidad de un sistema informático para manejar una cantidad creciente de trabajo o para ser ampliado con el fin de acomodar dicho crecimiento.

Express.js: Framework de infraestructura web minimalista y flexible para Node.js que proporciona un conjunto robusto de características para aplicaciones web y móviles.

Framework: Entorno de trabajo o esquema que proporciona una base estándar para desarrollar aplicaciones de software de manera más rápida y estructurada.

Frontend: Parte de una aplicación web o móvil con la que el usuario interactúa directamente (interfaz gráfica).

Git: Sistema de control de versiones distribuido, utilizado para rastrear cambios en el código fuente durante el desarrollo de software.

GitHub: Plataforma de alojamiento de código para el control de versiones y la colaboración, que permite trabajar en equipo sobre repositorios Git.

HTTP (Hypertext Transfer Protocol): Protocolo de comunicación que permite la transferencia de información en la World Wide Web. Es la base de la comunicación de datos para la API RESTful.

IoT (Internet of Things): Red de objetos físicos ("cosas") que llevan integrados sensores, software y otras tecnologías con el fin de conectar e intercambiar datos con otros dispositivos a través de Internet.

JSON (JavaScript Object Notation): Formato ligero de intercambio de datos, fácil de leer y escribir para los humanos y fácil de analizar y generar para las máquinas.

Latencia: Tiempo que tarda un paquete de datos en viajar desde su origen hasta su destino. En sistemas de tiempo real, se busca que sea mínima.

Node.js: Entorno de ejecución para JavaScript construido con el motor V8 de Chrome, orientado a eventos y diseñado para construir aplicaciones de red escalables.

NoSQL: Categoría de sistemas de gestión de bases de datos que no utilizan el esquema tabular de filas y columnas (relacional), sino modelos flexibles como documentos, clave-valor o grafos.

NPM (Node Package Manager): Gestor de paquetes predeterminado para Node.js, utilizado para instalar y administrar dependencias (librerías) del proyecto.

PaaS (Platform as a Service): Entorno de desarrollo e implementación en la nube que permite entregar aplicaciones sin la complejidad de gestionar servidores físicos (ej. Render).

Postman: Plataforma de colaboración para el desarrollo de APIs, utilizada comúnmente para realizar pruebas de peticiones HTTP y documentar endpoints.

RESTful: Estilo de arquitectura de software para sistemas distribuidos que se adhiere a las restricciones de REST (Representational State Transfer) y utiliza estándares HTTP.

Stakeholder: Persona, grupo u organización que tiene interés o participación en el proyecto y puede afectar o ser afectado por sus resultados (ej. el Asesor Empresarial).

Token de Autenticación: Credencial de seguridad digital que verifica la identidad de un usuario para concederle acceso al sistema.

ANEXO B: ESPECIFICACIONES TÉCNICAS DEL SUBSISTEMA MÓVIL

1. Introducción y Alcance del Subsistema

El presente anexo detalla las especificaciones técnicas exclusivas para el subsistema de control móvil del proyecto "Malbouche". Este módulo tiene como objetivo principal permitir la gestión remota, autenticación de usuarios y programación de eventos del reloj análogo mediante una arquitectura cliente-servidor.

El sistema móvil se comunica con un backend dedicado desarrollado en Node.js, el cual gestiona la lógica de negocio y la persistencia de datos en la nube utilizando Google Cloud Firestore. Este enfoque garantiza la sincronización en tiempo real de los comandos de movimiento y la configuración de eventos sin depender de la infraestructura web del establecimiento.

2. Requisitos Funcionales del Backend Móvil

A continuación se describen los requerimientos funcionales que rigen el comportamiento de la API y la base de datos móvil:

- RF 1 - Gestión de Sesiones: El sistema deberá validar las credenciales de acceso (correo electrónico y contraseña) mediante un endpoint de autenticación, permitiendo el ingreso únicamente a usuarios registrados en la colección usuarios de Firestore con roles activos.
- RF 2 - Control Manual del Reloj: La API debe exponer endpoints para recibir comandos de movimiento inmediato (izquierda, derecha, modo "loco", personalizado) desde la aplicación móvil y registrarlos en la base de datos para su ejecución por el dispositivo IoT.

- RF 3 - Gestión de Eventos Programados: El sistema permitirá realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre la colección eventos. Se validará que las fechas y horas de inicio/fin sean futuras y que no existan conflictos de horario con otros eventos prioritarios.

- RF 4 - Administración de Usuarios: El backend permitirá a los usuarios con rol de "Administrador" registrar nuevos operadores, asignar roles y modificar permisos, actualizando los documentos correspondientes en la base de datos.

- RF 5 - Sincronización en Tiempo Real: La base de datos Google Cloud Firestore deberá mantener sincronizados los estados de los documentos (eventos y movimientos) entre la aplicación móvil y el controlador del reloj, asegurando una latencia mínima en la actualización de estados.

3. Especificación de la API RESTful

La comunicación entre la aplicación móvil y el servidor se realiza a través de una API RESTful. La siguiente tabla detalla los principales endpoints desarrollados:

Método	Endpoint	Descripción	Cuerpo de la Petición (JSON)
POST	/api/auth/login	Autentica al usuario y devuelve un token de sesión.	{ "email": "...", "password": "..." }
GET	/api/events	Obtiene la lista de todos los eventos programados.	N/A
POST	/api/events	Registra un nuevo evento en la base de datos.	{ "name": "...", "startTime": "...", "type": "..." }
PUT	/api/events/:id	Actualiza la información de un evento existente.	{ "startTime": "...", "status": "active" }

DELETE	/api/events/:id	Elimina un evento de la programación.	N/A
POST	/api/movements	Envía un comando de movimiento inmediato al reloj.	{ "direction": "left", "speed": 100 }
GET	/api/users	Recupera el listado de usuarios (Solo Admin).	N/A

4. Interfaces de Usuario (Wireframes)

Las siguientes figuras ilustran el diseño de la interfaz de la aplicación móvil, la cual interactúa directamente con los servicios descritos anteriormente.

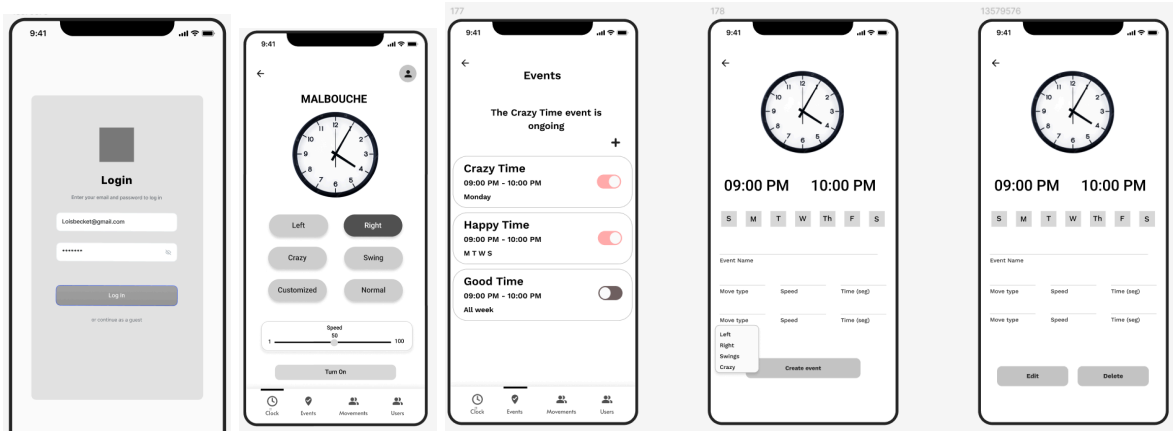

Figura B.1. Prototipos de interfaz de la aplicación móvil para el control del reloj Malbouche.