



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Integración y Desarrollo de Base de Datos y Back-End para un Sistema de Control y Optimización de Rutas para Camiones Recolectores

Tomás Peña Bustamante

tomas.penab@usm.cl

Catherine Gómez
Profesora Guía

Resumen: En este trabajo se aborda la problemática de la gestión ineficiente de rutas de recolección de residuos sólidos en la Municipalidad de Viña del Mar, proponiendo como solución la implementación de una base de datos en Firestore integrada con aplicaciones móviles y web. La solución se centró en la recopilación, gestión y análisis de datos en tiempo real, utilizando funcionalidades de geolocalización y visualización. Se desarrolló un prototipo funcional que permitió gestionar rutas activas y localizaciones, simplificando y automatizando el registro y la visualización de datos operativos. La implementación de Firestore en el proyecto fue un éxito, lograda de manera simplificada gracias a la naturaleza flexible y escalable de esta plataforma. Sin embargo, se identificaron limitaciones en la gestión de relaciones complejas, proponiendo como mejora futura la incorporación de bases de datos relacionales para complementar la solución. Los resultados de este proyecto son relevantes para iniciativas similares que busquen optimizar la gestión operativa mediante tecnologías digitales.

Palabras Clave: Bases de Datos, Aplicaciones Móviles, Servicios Web, Sistemas Cloud, Back-End

1 Introducción

En el mundo actual, la gestión de datos se ha convertido en un pilar fundamental para el desarrollo eficiente de soluciones tecnológicas, especialmente en áreas donde la recopilación, el procesamiento y el análisis de información en tiempo real son esenciales para la toma de decisiones. En este contexto, el proyecto "Smart Waste" se presenta como una respuesta a los problemas de gestión y control de rutas de recolección de basura de la Municipalidad de Viña del Mar.

La problemática central radica en la falta de un sistema que permita monitorear, analizar y optimizar el recorrido de los camiones recolectores de basura, lo que resulta en rutas ineficientes y acumulación de residuos. La implementación de un sistema de gestión de datos adecuado permitirá no solo un control más riguroso de las operaciones, sino también la optimización de recursos y una mejora en la percepción del servicio por parte de la ciudadanía.

Este proyecto tiene como objetivo diseñar, desarrollar e implementar una base de datos eficiente y adaptable que permita capturar, almacenar y procesar la información de manera efectiva, brindando soporte a los distintos sistemas que interactúan en el proceso, como las aplicaciones móviles y web que forman parte del ecosistema de la solución.

1.1 Contextualización del Problema

En la municipalidad de Viña del Mar existen dos operadores encargados de la recolección de residuos sólidos: Cosemar, una empresa concesionada que opera con una flota privada, y la propia municipalidad, que gestiona una flota de camiones. La mayor parte de los sectores de la comuna son atendidos por Cosemar, mientras que la flota municipal se encarga del resto de los sectores.

El problema principal de la flota municipal radica en la falta de control sobre las rutas que siguen los camiones. A los conductores se les asigna un sector, pero no una ruta predefinida, lo que implica que la ruta que eligen puede no ser la óptima o, en algunos casos, no cubrir todos los lugares que deberían atenderse. Como consecuencia, se generan áreas sin cobertura adecuada y acumulación de residuos, lo que provoca insatisfacción ciudadana y un aumento en las quejas hacia el servicio de recolección municipal.

1.2 Solución Propuesta

Se busca crear un sistema que permita mediante el uso de tecnología GPS controlar la flota de camiones en tiempo real. Esta información y su posterior procesamiento permitirá generar rutas óptimas para cada sector, así como registrar y monitorear diversos parámetros de la operación en tiempo real.

Específicamente, la aplicación captura datos detallados de los camiones, como su ubicación, hora de paso por cada ubicación, velocidad, informes de fallas mecánicas de los propios camiones o de desvíos de la ruta asignada, como también el tonelaje de inicio y final. Y también se recopilan datos generales como la matrícula del camión y la identidad del conductor.

Esta información será procesada para generar análisis de inteligencia de negocios que apoyen la toma de decisiones por parte de los encargados de la flota municipal, estos análisis serán entregados a través de un servicio web estilo dashboard.

Como producto adicional de la solución se considera la entrega pública de la información en la forma de una aplicación móvil para la ciudadanía que permitirá ver en tiempo real la localización de los camiones recolectores, al estilo de las aplicaciones de buses Red y Uber.

1.3 Relevancia de la Gestión de Datos en el Proyecto

La relevancia de la gestión de datos en este proyecto se puede desglosar en varios aspectos clave:

- **Monitoreo en Tiempo Real:** El sistema gestionará datos en tiempo real, capturando la ubicación de los camiones en intervalos de pocos segundos. Esto es fundamental para la eficiencia de las rutas y el seguimiento en vivo de las operaciones. La cantidad masiva de datos generados requiere una infraestructura que soporte almacenamiento y procesamiento continuo, lo que resalta la importancia de una arquitectura bien diseñada, posiblemente utilizando bases de datos NoSQL, como Firestore, que pueden manejar este tipo de datos en tiempo real.
- **Almacenamiento y Consulta de Datos Históricos:** Además de los datos en tiempo real, también se almacenan datos históricos, como el peso inicial y final del camión, la duración de las rutas y las ubicaciones recorridas. Esta información es vital para la optimización de rutas futuras y la evaluación de la eficiencia operativa.
- **Escalabilidad:** Dado que el volumen de datos será considerable, sobre todo si se manejan varias rutas diarias y múltiples camiones, el sistema debe estar preparado para escalar a medida que crece el número de camiones, rutas, y localizaciones. La capacidad de la base de datos para gestionar estos volúmenes y consultas frecuentes sin degradar el rendimiento es esencial.
- **Integración Potencial:** Aunque no se tiene certeza sobre los sistemas preexistentes en la Municipalidad de Viña del Mar, la base de datos debe estar diseñada de manera flexible para facilitar una futura integración con otros sistemas municipales o de terceros. Esto podría implicar la exportación o importación de datos, por lo que la interoperabilidad es un aspecto que considerar.

- **Control de Acceso y Seguridad:** Es importante considerar los mecanismos de control de acceso a los datos. Los roles diferenciados entre camioneros y administradores deben gestionarse adecuadamente en el sistema para asegurar que solo personal autorizado acceda a información crítica.
- **Optimización de Recursos y Eficiencia Operativa:** Finalmente, la capacidad de analizar y gestionar los datos de manera efectiva tendrá un impacto directo en la eficiencia de las operaciones. A través del análisis de los datos recolectados, la Municipalidad podrá identificar cuellos de botella, optimizar rutas y mejorar la eficiencia en la recolección de basura, reduciendo costos y mejorando la percepción ciudadana.

1.4 Objetivo General

Desarrollar una base de datos eficiente y escalable que permita la gestión, monitoreo, y análisis de rutas de recolección de basura para la Municipalidad de Viña del Mar, facilitando la toma de decisiones y optimizando la operación del servicio de recolección de residuos a través de la implementación de un back-end que integre diversos sistemas.

1.4.1 Objetivos Específicos

- Diseñar un modelo de datos capaz de almacenar información relevante del proyecto y representar correctamente las relaciones entre las entidades.
- Implementar la integración de la base de datos con aplicaciones móviles que utilicen geolocalización.
- Desarrollar la integración con aplicaciones web enfocadas en la administración y visualización de los datos almacenados.
- Evaluar el sistema mediante pruebas de usabilidad, funcionalidad y rendimiento, asegurando su eficacia y robustez.
- Optimizar la eficiencia de la base de datos y garantizar su funcionalidad al integrarla correctamente con las aplicaciones.

2 Metodología

Este proyecto sigue un enfoque ágil, utilizando metodologías de desarrollo reutilizables para cada sprint, complementadas con la metodología Scrum para la planificación y gestión de las actividades. La metodología principal de Scrum se divide en tres sprints, cada uno orientado hacia la entrega de un componente clave del sistema.

2.1 Metodología Scrum

Scrum se aplica en el desarrollo general del proyecto, con sprints de 4 semanas:

- **Sprint 1:** Desarrollo de la Aplicación Móvil para Conductores, que permitirá la recolección de datos en tiempo real, como la localización y el peso de los camiones.
- **Sprint 2:** Implementación de la Aplicación Web para Administradores, enfocada en la visualización y análisis de los datos recolectados a través de un dashboard.
- **Sprint 3:** Creación de la Aplicación Móvil para Ciudadanos, donde se entrega información pública sobre las rutas y horarios de recolección en tiempo real.

2.2 Metodología de Desarrollo

Se incluyen también varias fases clave para garantizar el éxito del desarrollo de cada sprint y el proyecto en general:

- **Investigación y Análisis:** Esta fase inicial consiste en la identificación y comprensión profunda de los requerimientos funcionales y no funcionales del sistema. Aquí se evalúan las necesidades de los usuarios finales, tanto los administradores de la Municipalidad como los ciudadanos, así como los conductores que utilizarán la aplicación móvil. Además, en esta etapa se realiza una investigación técnica para seleccionar las herramientas más adecuadas para la implementación. Esto incluye la elección de la arquitectura tecnológica, el tipo de base de datos que será usada y las tecnologías para el desarrollo de las aplicaciones móviles y web. También se evalúan soluciones similares como Routeware y sistemas de control de flotas para extraer buenas prácticas.
- **Diseño:** Una vez definidos los requerimientos, se procede al diseño del sistema. En esta fase, se realiza el modelado relacional de la base de datos, que se traduce posteriormente en un modelo NoSQL de ser necesario. El diseño de la base de datos debe tener en cuenta los grandes volúmenes de datos que serán procesados en tiempo real, incluyendo la localización de los camiones, el peso y las rutas realizadas. Esto implica definir colecciones de documentos que representen entidades como los usuarios, las rutas, los camiones y los datos de localización, garantizando la correcta relación e integridad entre estos elementos. El diseño también abarca la arquitectura del sistema, la cual debe asegurar la modularidad y escalabilidad del proyecto. En este sentido, se define cómo interactuarán las aplicaciones móviles y web con el backend, asegurando que el sistema pueda manejar el flujo constante de datos y responder de manera eficiente a las consultas.
- **Implementación:** La implementación consiste en la programación de cada componente del sistema. En esta fase, se desarrollan las aplicaciones móviles y web, la creación de la base de datos, y la integración de esta en los sistemas móviles y web, incluyendo la captura y la lectura de datos.
- **Pruebas:** Las pruebas son fundamentales para asegurar que el sistema funcione correctamente y cumpla con los requerimientos planteados. Se realizarán pruebas unitarias de cada componente, asegurando que cada parte del sistema se comporte según lo esperado. Además, se llevarán a cabo pruebas de integración, que verificarán la correcta interacción entre la base de datos, el backend y las aplicaciones móviles y web. Una vez que el sistema está integrado, se ejecutarán pruebas de campo, en las que se comprobará la funcionalidad del sistema en condiciones reales de operación. En esta fase se debe verificar, por ejemplo, la precisión de la captura de la localización en tiempo real y la correcta visualización de los datos en el dashboard.
- **Despliegue:** La fase final es el despliegue del sistema. El sistema será implementado inicialmente en un entorno piloto, donde los usuarios clave (conductores y administradores de la municipalidad) podrán interactuar con la solución para validar su funcionalidad. Posteriormente, se desplegará en producción, donde estará disponible para su uso real por parte de todos los usuarios. El despliegue incluye también la documentación del sistema, en la cual se especificará cómo utilizar e interactuar con cada parte de la solución.
- **Mejora Continua:** A lo largo de todo el ciclo de vida del proyecto, se seguirán procesos de retroalimentación y mejora continua. En cada sprint, se revisarán los resultados y se ajustarán las funcionalidades en función de las pruebas y comentarios de los usuarios. Esto permitirá adaptar el sistema a las necesidades cambiantes y mejorar su rendimiento y capacidad.

2.3 Requerimientos Funcionales

Los requerimientos funcionales definen qué debe hacer el sistema para satisfacer las necesidades del usuario y cumplir con su propósito, se desarrollan las funciones del sistema en base a estos requerimientos.

Código de requerimiento	Nombre	Descripción
-------------------------	--------	-------------

RF-1	Registro de usuarios	El sistema debe permitir la creación de usuarios y el almacenamiento de su información, teniendo en cuenta un sistema de autenticación integrado
RF-2	Actualización y consulta de datos de usuarios	Debe ser posible actualizar los usuarios y consultar los datos almacenados de estos, incluyendo información como nombre, rol y sectores asignados
RF-3	Asignación de sectores a usuarios	El sistema debe permitir la asignación de sectores a usuarios específicos, vinculados a días de la semana
RF-4	Registro de rutas	El sistema debe permitir a los conductores crear rutas asociadas al conductor y el camión usado
RF-5	Registro de ubicaciones en rutas activas	El sistema debe registrar en tiempo real las ubicaciones obtenidas desde el dispositivo móvil de los conductores
RF-6	Registro de reportes en rutas	Los conductores deben poder registrar reportes durante sus rutas que representen los problemas que hayan tenido
RF-7	Obtención de ubicaciones en tiempo real	El sistema debe permitir a los ciudadanos obtener las ubicaciones en tiempo real de los camiones en ruta
RF-8	Visualización de datos históricos	Los administradores deben poder consultar y visualizar los datos históricos, incluyendo rutas, reportes, y otros datos almacenados, desde la aplicación web

Figura 1. Lista de requerimientos funcionales

2.4 Requerimientos No Funcionales

Los requerimientos no funcionales son aquellos relacionados con aspectos técnicos del sistema, que no definen una función específica para el usuario. Se realizan ajustes técnicos adicionales en base a estos requerimientos para garantizar la implementación correcta de las funciones del sistema.

Código de requerimiento	Nombre	Descripción
RNF-1	Protección de datos	El sistema debe garantizar la protección de la información almacenada mediante autenticación, validación y otros tipos de reglas de seguridad
RNF-2	Acceso controlado	Como parte de la seguridad, se debe controlar el acceso a través de roles de usuario

RNF-3	Eficiencia de consultas	Las consultas a la base de datos deben realizarse sin gastos excedentes de procesamiento y almacenamiento sin poner en riesgo el rendimiento del sistema
RNF-4	Consultas en tiempo real	Las consultas en tiempo real deben ser realizadas a la mayor velocidad posible para poder tener actualizaciones de información entre múltiples dispositivos en pocos segundos

Figura 2. Lista de requerimientos no funcionales

3 Marco Teórico

3.1 Modelos de Bases de Datos

Existen diferentes modelos para estructurar bases de datos. Los dos modelos más relevantes en el contexto de este proyecto son:

- **Bases de Datos Relacionales (RDBMS):** Organizan los datos en tablas que se relacionan entre sí mediante claves primarias y externas. Son útiles para datos estructurados y para mantener la integridad referencial. Las transacciones en RDBMS generalmente cumplen con el modelo ACID (Atomicidad, Consistencia, Aislamiento, Durabilidad), que garantiza la confiabilidad y seguridad de las operaciones. Aunque históricamente han sido las más usadas, presentan limitaciones en términos de escalabilidad cuando se manejan grandes volúmenes de datos en tiempo real. Herramientas comunes incluyen MySQL, PostgreSQL, y SQL Server.
- **Bases de Datos NoSQL:** Las bases de datos NoSQL suelen seguir el modelo BASE (Basically Available, Soft state, Eventual consistency), que prioriza la disponibilidad y la escalabilidad sobre la consistencia inmediata de los datos. Ideales para el manejo de datos no estructurados o semiestructurados y cuando se requieren operaciones de alta velocidad. No utilizan un esquema fijo, lo que las hace más flexibles. En este proyecto, por ejemplo, una base de datos NoSQL como Firestore sería más adecuada debido al manejo de datos en tiempo real y la posibilidad de escalar fácilmente con la cantidad de rutas y ubicaciones de los camiones.

3.2 Principios de Diseño de Bases de Datos

Los principios de bases de datos son fundamentales para garantizar un diseño, implementación y funcionamiento eficientes. Estos principios abarcan varios aspectos:

- **Normalización:** Este proceso organiza los datos en una base relacional para reducir la redundancia. En modelos relacionales, es común aplicar reglas de normalización para evitar la duplicación de datos y asegurar que cada pieza de información se almacene en un lugar específico.
- **Desnormalización:** A veces, para mejorar el rendimiento, es necesario introducir cierta redundancia en el diseño de la base de datos. Esto se hace en situaciones donde la normalización estricta podría llevar a consultas ineficientes. La desnormalización es un compromiso que busca equilibrar el rendimiento y la integridad.
- **Integridad de Datos:** Asegura que los datos almacenados en la base de datos sean precisos, completos y consistentes. Se implementa a través de restricciones como claves primarias, claves externas, reglas de validación y *checks* que evitan la entrada de datos incorrectos o inconsistentes.

- **Consistencia:** Garantiza que la base de datos pase de un estado válido a otro después de cualquier transacción. En sistemas de bases de datos distribuidas, este principio es fundamental para asegurarse de que todos los nodos contengan la misma información.
- **Escalabilidad:** La base de datos debe poder crecer y manejar un aumento en la cantidad de datos y usuarios sin perder rendimiento. Esto implica considerar la estructura de la base de datos y su arquitectura para soportar futuros aumentos en la carga de trabajo.
- **Seguridad de Datos:** Los sistemas de bases de datos deben contar con medidas de seguridad para proteger la información, como la implementación de controles de acceso, encriptación y auditorías. Esto garantiza que solo los usuarios autorizados puedan acceder o modificar los datos.
- **Disponibilidad y Redundancia:** Las bases de datos deben estar disponibles y accesibles incluso en caso de fallos. Esto se logra mediante la replicación de datos, la creación de copias de seguridad y la configuración de sistemas de alta disponibilidad.
- **Flexibilidad y Adaptabilidad:** El diseño de la base de datos debe permitir cambios y adaptaciones en función de las necesidades futuras. Esto implica crear un esquema que pueda ajustarse a la evolución del sistema sin requerir una reestructuración completa.

3.3 Tecnologías de Bases de Datos

Las tecnologías actuales de bases de datos ofrecen una variedad de opciones adaptadas a diferentes necesidades y casos de uso. Algunas de las tecnologías más relevantes incluyen:

3.3.1 Base de Datos Relacionales (SQL)

- **PostgreSQL:** Un sistema de base de datos relacional de código abierto que es muy popular por su robustez y escalabilidad. Ofrece características avanzadas como soporte para datos geoespaciales y JSON.
- **MySQL:** Otro sistema de base de datos relacional ampliamente utilizado, conocido por su rendimiento, fiabilidad y facilidad de uso. Es especialmente popular para aplicaciones web y es compatible con muchos sistemas operativos.
- **Microsoft SQL Server:** Un sistema de base de datos comercial que ofrece características avanzadas de seguridad, análisis y escalabilidad, ideal para aplicaciones empresariales de gran escala.

3.3.2 Bases de Datos NoSQL

- **MongoDB:** Una base de datos orientada a documentos que almacena datos en formato JSON. Es muy flexible y escalable, lo que la hace ideal para aplicaciones que requieren manejar grandes volúmenes de datos no estructurados o semi-estructurados.
- **Firebase Firestore:** Construida en la infraestructura de Google Cloud, es una base de datos Cloud NoSQL basada en documentos que permite la sincronización en tiempo real y se integra fácilmente con aplicaciones móviles y web. Es especialmente útil para proyectos que requieren una actualización en tiempo real y escalabilidad.
- **Cassandra:** Una base de datos distribuida y altamente escalable, diseñada para manejar grandes cantidades de datos a través de múltiples nodos. Es utilizada por empresas que requieren alta disponibilidad y tolerancia a fallos.

Si bien las bases de datos relacionales como PostgreSQL o MySQL ofrecen robustez y cumplen con el modelo ACID, para este proyecto se optó por una base de datos NoSQL, específicamente **Firestore**. Esta decisión se fundamenta en la necesidad de manejar grandes volúmenes de datos de ubicación en tiempo real, un escenario donde Firestore, con su arquitectura distribuida, ofrece mayor escalabilidad y rendimiento. Además, la sincronización en tiempo real que proporciona Firestore es crucial para la funcionalidad principal del proyecto; mostrar la ubicación de los camiones en la aplicación móvil. La integración con otras herramientas de Firebase, como la autenticación y el hosting, simplifica el desarrollo y la gestión del proyecto. Cabe mencionar que el equipo tiene una alta familiaridad con Firebase, incluyendo Firestore y el resto de las funciones que ofrece la plataforma de desarrollo.

3.4 Tecnologías del Proyecto

- **Flutter:** Framework de desarrollo de aplicaciones móviles multiplataforma de código abierto desarrollado por Google. Permite crear aplicaciones nativas de alto rendimiento para iOS y Android con una única base de código. En el contexto del proyecto, Flutter se utilizará para desarrollar dos aplicaciones móviles, una que permitirá principalmente recolectar la localización de los camiones recolectores y otra que permitirá a los ciudadanos visualizar la ubicación de los camiones recolectores en tiempo real.
- **Angular:** Framework de desarrollo web de código abierto basado en TypeScript, mantenido por Google. Permite construir aplicaciones web escalables y robustas con un enfoque en la organización del código y la eficiencia. En el contexto del proyecto, Angular se utilizará para desarrollar el dashboard web que proporcionará a los administradores información sobre las rutas de los camiones, análisis de datos e informes.
- **BigQuery:** Servicio de almacenamiento de datos en la nube, completamente gestionado y sin servidor, ofrecido por Google Cloud. Permite almacenar y analizar grandes conjuntos de datos con alta velocidad y eficiencia. En el contexto del proyecto, BigQuery se utilizará en el área de la inteligencia de negocios, los datos que se almacenarán en ella son los datos procesados después de ser recolectados en Firestore, se utilizará como conexión final hacia Looker Studio.
- **Looker Studio:** Herramienta de visualización de datos e informes en línea ofrecida por Google. Permite crear dashboards interactivos y personalizados a partir de diversas fuentes de datos. En el contexto de este proyecto, Looker Studio se utilizará para visualizar los datos almacenados en BigQuery y generar informes para los administradores, facilitando la toma de decisiones.

3.5 Soluciones Existentes

- **Uber:** Uber utiliza una infraestructura tecnológica robusta basada en el seguimiento en tiempo real de vehículos. A través de su aplicación móvil, los usuarios pueden visualizar la ubicación del conductor en un mapa en tiempo real (La cual también se obtiene a través de una aplicación móvil). Este sistema se apoya en una base de datos in-house de Uber, “Docstore” la cual está construida por encima de MySQL. ^[1]
- **Sistema RED (Chile):** RED es el sistema de transporte público de Santiago, Chile, que reemplazó al antiguo Transantiago. Utiliza una tecnología de geolocalización que permite a los usuarios seguir en tiempo real la ubicación de los buses. Esto mejora la eficiencia del servicio y proporciona a los ciudadanos información actualizada sobre la llegada y salida de buses, ayudando a reducir los tiempos de espera en paraderos. El sistema utiliza bases de datos en tiempo real para gestionar grandes volúmenes de información geoespacial y datos de tráfico.
- **Routeware:** Sistema avanzado de gestión de flotas adoptado principalmente en la industria de la gestión de residuos. La solución integra computadoras a bordo que permiten un seguimiento en tiempo real y análisis del rendimiento de los vehículos. Entre sus características más destacadas se encuentran la optimización de rutas, geoperimetraje, localización automática de vehículos y navegación con instrucciones paso a paso. Esto ayuda a mejorar la productividad de las flotas y a reducir las paradas no realizadas, optimizando las rutas y disminuyendo tanto el tiempo de trabajo como las emisiones de carbono. Además, Routeware permite una integración sencilla con sistemas de terceros, lo que facilita la gestión de datos y la generación de informes, siendo ideal para municipalidades y agencias de recolección de residuos que buscan optimizar sus operaciones ^[2]

4 Diseño del Modelo de Datos

El modelado de la Base de Datos se llevó a cabo en función de los requisitos específicos de los clientes, asegurando que se cubran todas las funcionalidades propuestas en la solución. En este proceso, se incluyeron las entidades necesarias para gestionar tanto los datos de usuarios como de camiones, independientemente de que estos no se encuentren inicialmente en una base de datos externa accesible por nuestros sistemas; por lo tanto, se incorporan desde cero en el diseño.

El diseño del modelo y el posterior desarrollo de código, entre otras configuraciones del proyecto en general son realizadas en inglés como buena práctica para mantener consistencia, adaptabilidad a estándares y herramientas, para tener un acceso más facilitado al buscar recursos en internet y para evitar problemas de codificación debido a caracteres especiales.

4.1 Modelo Relacional

El modelo relacional se realizó en dbdiagram.io, haciendo uso de su Database Markup Language, un lenguaje open-source diseñado para definir y documentar esquemas y estructuras de bases de datos, a través de la misma página se crea automáticamente la visualización del modelo relacional en base al Database Markup Language que se escribe.

El modelo constituye una base sólida para la organización estructurada de los datos, sirviendo como punto de partida para su posterior adaptación al sistema NoSQL implementado en Firebase Firestore:

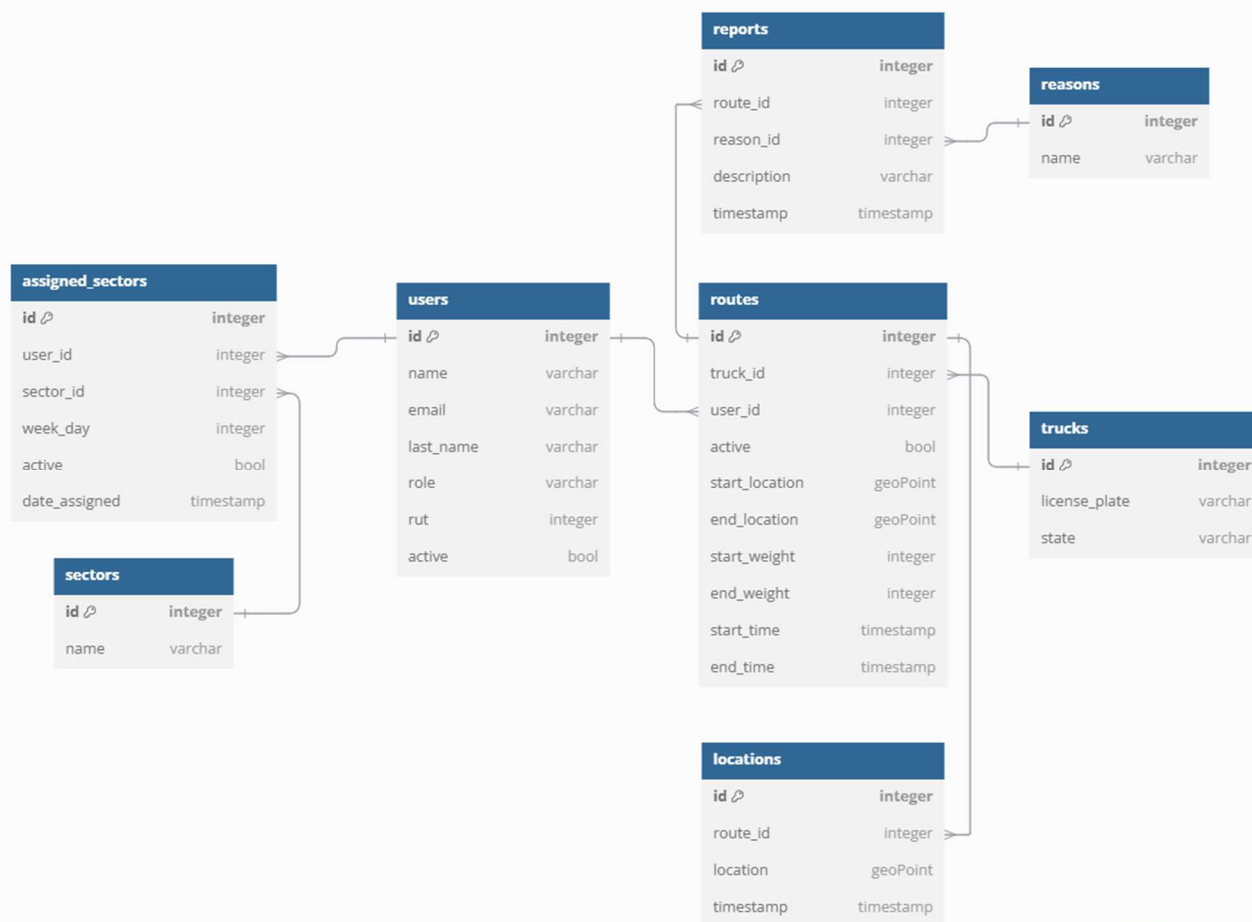


Figura 3. Modelo Relacional (Fuente: elaboración propia a través de dbdiagram.io)

En este modelo, la normalización busca minimizar la redundancia de datos, garantizando que cada entidad esté optimizada y sea independiente. Se han creado tablas para entidades que podrían necesitar campos adicionales en el futuro o que ya los tengan, tales como “sectors”, “reasons” y “trucks” con la intención de facilitar la escalabilidad y flexibilidad del sistema.

4.1.1 Tablas

- Usuarios (users): Representa a los usuarios que interactúan con el sistema.
- Rutas (routes): Almacena las rutas de recolección.
- Camiones (camiones): Vehículos asignados a las rutas.
- Ubicaciones (locations): Puntos de geolocalización registrados a lo largo de las rutas.
- Reportes (reports): Incidentes o problemas reportados en las rutas.
- Razones (reasons): Razones o causas que justifican los reportes.
- Sectores asignados (assigned_sectors): Sectores de la ciudad asignados a usuarios.
- Sectores (sectors): Diferentes zonas geográficas de la ciudad

4.1.2 Relaciones

- Usuarios (users) y Rutas (routes): Los usuarios pueden estar asociados con múltiples rutas de recolección. Esto permite rastrear qué usuario está a cargo de cada ruta en distintos días o momentos.
- Rutas (routes) y Camiones (trucks): Cada ruta está asignada a un camión específico, aunque un camión puede ser utilizado en varias rutas diferentes. Esto permite organizar el uso de los vehículos y rastrear qué camión realizó cada ruta.
- Rutas (routes) y Ubicaciones (locations): A lo largo de cada ruta se registran múltiples ubicaciones que indican el recorrido y la posición en tiempo real del camión durante la recolección.
- Rutas (routes) y Reportes (reports): Las rutas pueden tener múltiples reportes asociados, los cuales registran cualquier incidente o problema ocurrido durante la recolección en una ruta específica.
- Reportes (reports) y Razones (reasons): Cada reporte puede tener una razón asociada, que especifica el motivo o causa del incidente.
- Usuarios (users) y Sectores Asignados (assigned_sectors): Los usuarios pueden estar asignados a múltiples sectores de la ciudad en días específicos. Esta relación permite definir qué áreas geográficas son responsabilidad de cada usuario.
- Sectores (sectors) y Sectores Asignados (assigned_sectors): Cada sector puede estar asignado a múltiples usuarios en días distintos.

4.2 Modelo Firestore

En este nuevo contexto se adapta el modelo relacional a la jerarquía basada en colecciones, subcolecciones y documentos de Firestore. Algunos caminos de datos permiten consultas eficientes al alinear los datos necesarios para cada funcionalidad en la misma colección o subcolección y así prevenir búsquedas complejas para unir datos desde múltiples ubicaciones, la manera en que estos caminos de datos se han realizado son considerando de que depende cada colección, por ejemplo, las rutas tienen que pertenecer a un usuario, y las localizaciones deben pertenecer a una ruta.

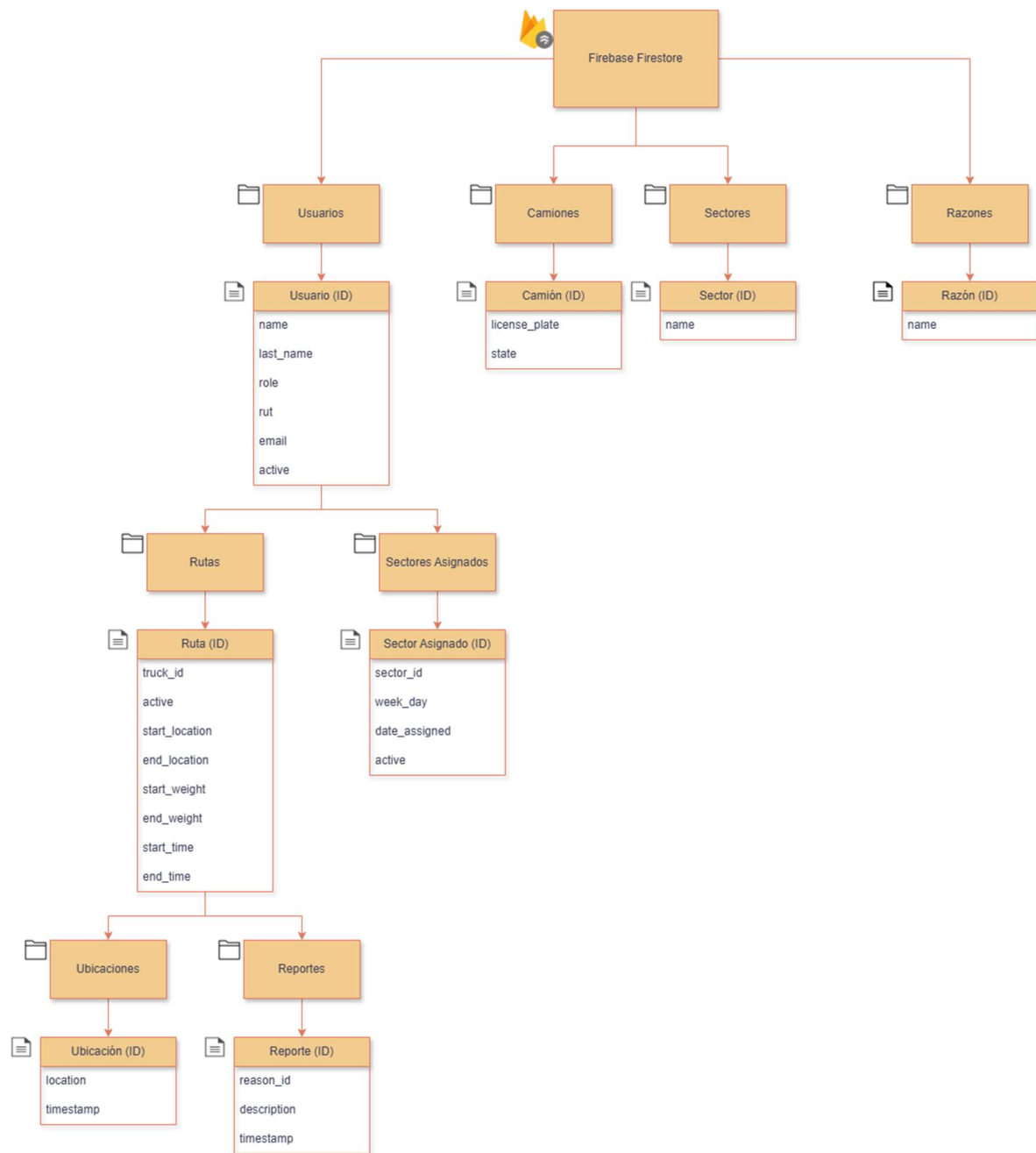


Figura 4. Jerarquía del modelo Firestore (Fuente: elaboración propia)

Descripción de campos de cada colección y subcolección:

- **Usuarios** (/users/{userId}):

Campo	Descripción
id	Identificador único del usuario
name	Nombres del usuario
last_name	Apellidos del usuario
role	Rol del usuario en el sistema, por ejemplo “conductor “
rut	Rut del usuario

email	Correo electrónico del usuario
active	Usuario activo o no activo (Booleano)

Figura 5. Lista de campos de colección de usuarios

- **Sectores Asignados**

(/users/{userId}/assigned sectors/{assigned sectorId}):

Campo	Descripción
id	Identificador único del sector asignado
sector_id	Identificador del sector de recolección de basura que tiene asignado el usuario
week_day	Día de la semana en que el usuario tiene asignado el sector (Representado con 0 a 6)
date_assigned	Fecha en la que fue realizada la asignación de sector al usuario
active	Asignación activa o no activa (Booleano)

Figura 6. Lista de campos de sectores colección de sectores asignados

- **Rutas** (/users/{userId}/routes/{routeId}):

Campo	Descripción
id	Identificador único de la ruta
truck_id	Identificador del camión que realizará la ruta
active	Ruta activa o no activa (Booleano)
start_location	Ubicación de origen de ruta (geoPoint)
end_location	Ubicación de finalización de ruta (geoPoint)
start_weight	Peso al inicio de la ruta
end_weight	Peso al finalizar la ruta
start_time	Hora de inicio de ruta (Timestamp)
end_time	Hora de finalización de ruta (Timestamp)

Figura 7. Lista de campos de colección de rutas

- **Reportes** (/users/{userId}/routes/{routeId}/reports/{reportId}):

Campo	Descripción
id	Identificador único del reporte
reason_id	Identificador de la razón del reporte
description	Descripción del reporte realizado
timestamp	Hora de la realización del reporte (Timestamp)

Figura 8. Lista de campos de colección de reportes

- **Ubicaciones** (/users/{userId}/routes/{routeId}/locations/{locationId}):

Campo	Descripción
id	Identificador único de la ubicación
location	Ubicación (geoPoint)
timestamp	Hora en que se estuvo en esta ubicación (Timestamp)

Figura 9. Lista de campos de colección de ubicaciones

- **Camiones** (/trucks/{truckId}):

Campo	Descripción
id	Identificador único del camión
license_plate	Patente del camión
state	Estado del camión, funcional o no funcional

Figura 10. Lista de campos de colección de camiones

- **Razones** (/reasons/{reasonId}):

Campo	Descripción
id	Identificador único de la razón
name	Nombre de la razón

Figura 11. Lista de campos de colección de razones

- **Sectores** (/sectors/{sectorId}):

Campo	Descripción
id	Identificador único del sector
name	Nombre del sector

Figura 12. Lista de campos de colección de sectores

4.2.1 Campos de Referencia en NoSQL

Las claves foráneas del modelo relacional son adaptadas como “campos de referencia” cuando la subcolección no logra tener como colección padre a la llave foránea que tenía en el esquema relacional, por ejemplo, camiones (/trucks) es una colección que está en el nivel inicial de las colecciones y no tiene subcolecciones, por lo que no es posible revisar esa colección como colección padre si se empieza desde un documento que está en la subcolección de rutas (/users/{userId}/routes), la colección /trucks no es una colección padre de este camino de datos.

Inversamente, se puede considerar redundantes los campos de referencia de caminos de datos que, si puedan ser adaptados a una sola ruta de colecciones, por ejemplo, el camino /users/{userId}/routes representa a la colección /users como una colección padre de la subcolección de rutas, por lo que podemos obtener la referencia a través de ella en lugar de crear un campo de referencia apuntando a /users en la subcolección rutas.

Las subcolecciones solo pueden tener una colección padre, por lo que esto impide la posibilidad de que una colección que depende de dos colecciones haga redundante el uso de campos de referencia por completo, en el ejemplo anterior mencionamos que la colección /trucks está en el nivel inicial y no tiene subcolecciones aunque la colección /routes depende de esta, esto debido a que la colección de rutas ya tiene a la colección de usuarios como su colección padre y el único camino para llegar a ella es a través de la colección /users, dejando fuera del camino de datos a la colección de camiones, la cual se puede considerar una colección de menor importancia para las rutas al compararla con la colección de usuarios.

También aporta a la normalización la mantención de colecciones creadas en la normalización del esquema relacional, esto refiriéndose a las colecciones /reasons y /sectors, aunque aún contengan solo un campo, están consideradas como colecciones de referencia, permitiendo hacer el uso de la base de datos para hacer estas listas en lugar de realizar *hard-coding* de los datos en las aplicaciones web y móvil.

Con estos esfuerzos se mantiene la normalización del diseño de la base de datos hasta cierto punto, intentando evitar desnormalización excesiva al adaptarla a Firestore y promoviendo una base de datos ligera.

5 Implementación

5.1 Creación de la Base de Datos

La base de datos se crea automáticamente al realizar la implementación en las aplicaciones web y móvil, esto refiriéndose a que cada colección se crea cuando los documentos se crean, no existen las colecciones sin

documentos y no es necesario hacer explícito uso de la consola de Firestore para hacerle saber de las colecciones a crear.

Sin embargo, a través del desarrollo del proyecto se utiliza varias veces la consola Firestore para hacer el amplio set de pruebas de los datos que se crean, editan, leen o eliminan.

5.2 Integración con Firebase Authentication

Firebase cuenta con su propio sistema de autenticación, el cual incluye poder autenticar con programas externos como Google, Facebook, número de teléfono, etc. En el proyecto, las cuentas son propias del sistema Firebase y es aquí donde deben ser creadas con un simple sistema de correo electrónico y contraseña. Al autenticarse por primera vez Firebase Authentication crea automáticamente un usuario con su propio UID y los campos de correo electrónico y contraseña.

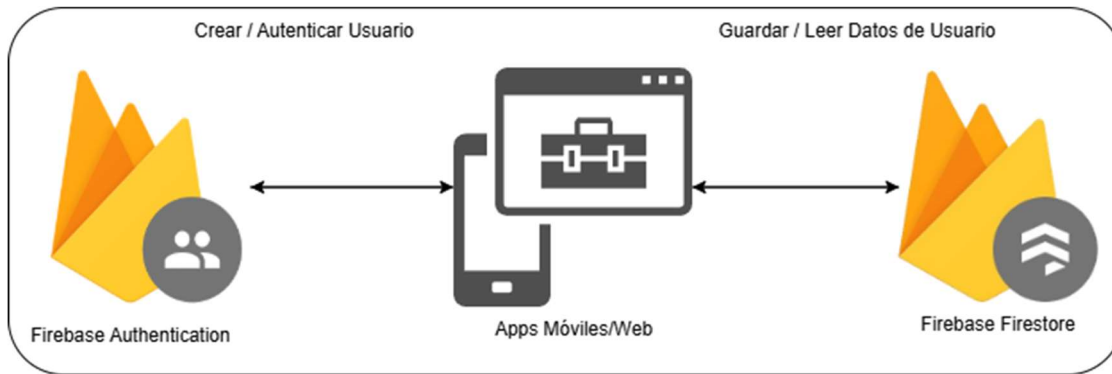


Figura 11. Flujo de la integración Firebase Authentication-Firestore (Fuente: elaboración propia)

La integración se realiza a través de codificación propia, una vez creado un usuario se crea un nuevo documento en la colección `/users` representando al usuario creado, usando la UID creada por Firebase Authentication como ID del documento. A este documento se le añaden los campos que tengan relevancia para cuando se deba recuperar información pública del usuario, como su nombre, rol, entre otros.

De la misma manera, para recuperar los datos que se guardan del usuario que inició sesión en la aplicación (Web o Móvil) se recupera la UID del usuario actualmente autenticado y esta es utilizada nuevamente para buscar el documento correspondiente al usuario en Firestore.

5.3 Integración con App Móvil (Conductor)

La aplicación móvil es la pieza central para la recopilación de datos en este proyecto, encargándose de registrar rutas y su información relevante, como los campos ingresados al momento de su creación, los reportes generados durante el trayecto, el usuario responsable de la ruta y cada localización registrada a lo largo del recorrido.

La aplicación está integrada con Firebase Authentication, lo que permite identificar al usuario autenticado. A partir de esta integración, se consulta el documento correspondiente al usuario autenticado en la colección `/users/{userId}` y se utiliza esta información como referencia para crear y asociar nuevas rutas.

5.3.1 Creación de Rutas

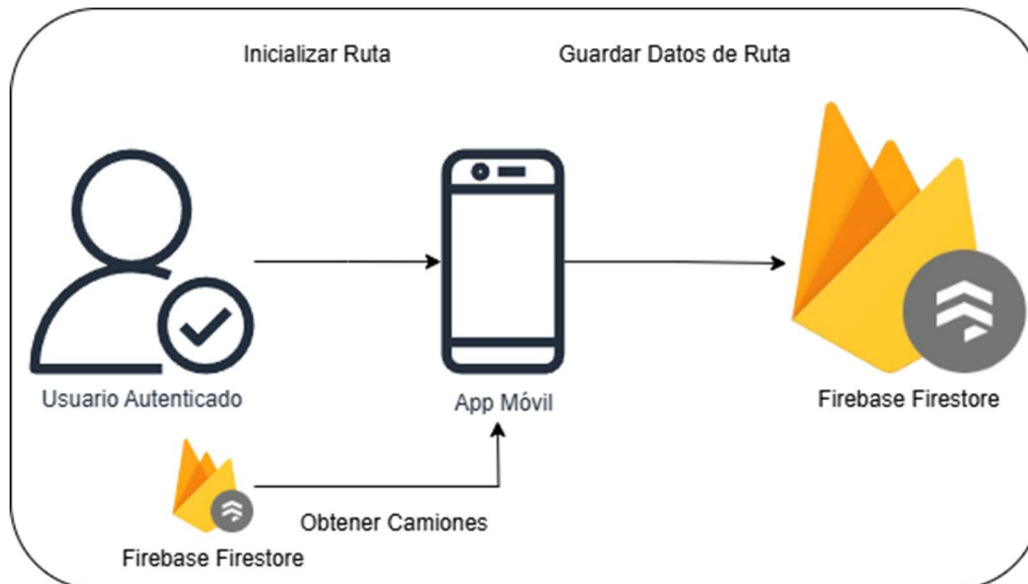


Figura 12. Flujo de la carga de ruta a Firestore (Fuente: elaboración propia)

El proceso de registro de una nueva ruta en la aplicación móvil del conductor es un componente clave del sistema, la información analizable del sistema nace de la información que proveen las rutas que realizan los conductores de los camiones recolectores. Este procedimiento incluye validaciones y operaciones esenciales para asegurar la integridad de la información.

1. **Ingreso de datos manuales**

- El conductor ingresa información básica para la creación de la ruta, incluyendo la patente del camión y el peso inicial.

2. **Validación de la patente**

- Se realiza una consulta a Firestore en la colección `/trucks`
- Se verifica si la patente ingresada coincide con un camión registrado.
- Se utiliza la ID del camión registrado para el campo `truckId` de la ruta.

3. **Inicialización de la ruta**

- Al completarse la validación, se crea un nuevo documento en la colección `/routes` bajo la estructura `/users/{userId}/routes/{routeId}`, la ruta se marca como activa (`active: true`) y otros campos relevantes se inicializan automáticamente, como:
 - `start_time`: Fecha y hora de inicio de la ruta.
 - `start_location`: Coordenadas actuales del conductor.
 - `start_weight`: Peso inicial ingresado.

En caso de ya tener una ruta activa asociada al usuario, la capacidad de crear una nueva ruta se bloquea en la aplicación, al iniciar sesión o entrar a la aplicación con una sesión ya iniciada se realiza una consulta en la subcolección `/routes`, en caso de tener una ruta activa, ésta se inicializa en el dispositivo del usuario y se le redirige inmediatamente a la página correspondiente.

5.3.2 Registro de Ubicación Actual

Una vez iniciada una ruta, la aplicación móvil redirige al conductor a la página correspondiente a la ruta activa. Al inicializar esta página, la aplicación comienza a obtener de manera continua la ubicación actual del dispositivo, utilizando el sistema de posicionamiento global (GPS). Este proceso es crucial para realizar el seguimiento en tiempo real de la ruta activa y actualizar la base de datos con los datos más recientes.

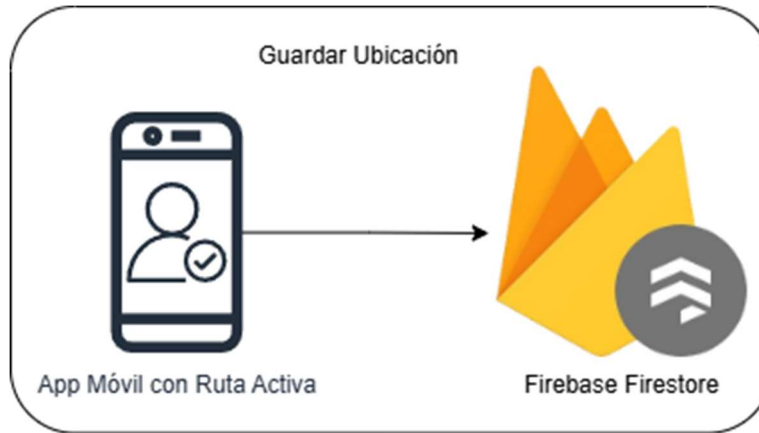


Figura 13. Flujo de la carga de ubicación a Firestore (Fuente: elaboración propia)

1. Obtención de la ubicación

- Se capturan las coordenadas geográficas (latitud y longitud) del dispositivo en intervalos regulares.
- Las coordenadas obtenidas se transforman al formato geoPoint, que permite almacenar datos geospaciales de manera nativa en la base de datos.

2. Almacenamiento en Firestore

- Cada nueva ubicación se almacena como un documento único dentro de la subcolección `/locations` asociada a la ruta activa.
- La estructura de la jerarquía es:
`/users/{userId}/routes/{routeId}/locations/{locationId}`

5.3.3 Registro de Reporte de Ruta

Los reportes de ruta son posibles de realizar una vez teniendo una ruta activa, la primera consulta de la página de reportes es la obtención de la colección `/reasons`, las razones se presentan en una lista seleccionable, el usuario debe seleccionar una de las razones e ingresar opcionalmente una descripción del reporte. Se obtiene automáticamente la localización y hora del reporte una vez ingresado a la subcolección.

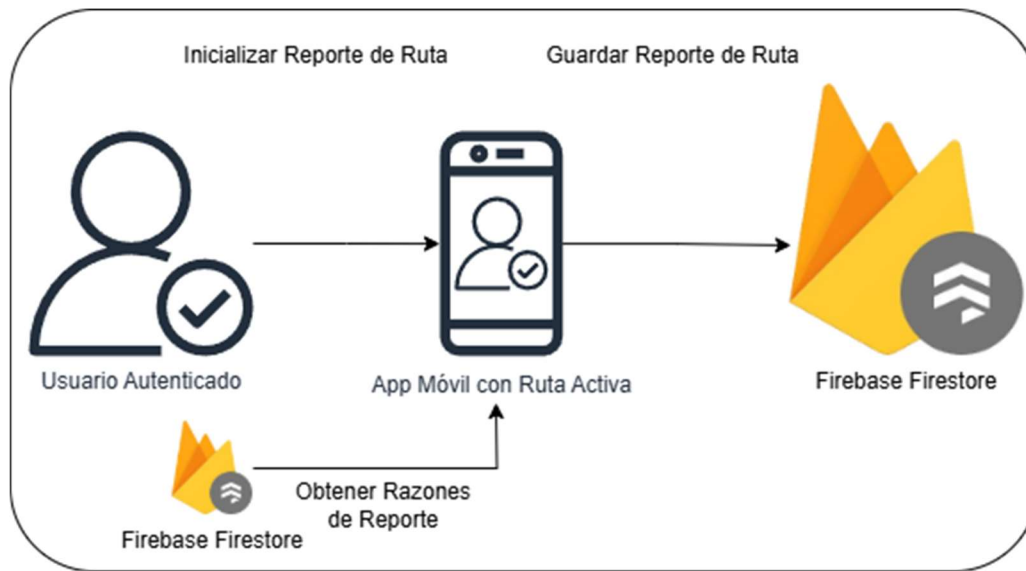


Figura 14. Flujo de la carga de reporte de ruta a Firestore (Fuente: elaboración propia)

1. **Consulta inicial a Firestore**
 - Se realiza una consulta a la colección `/reasons` en Firestore.
 - Esta colección almacena los posibles motivos de reporte, como "Obstáculo en el camino" o "Vehículo descompuesto".
 - Los datos obtenidos se presentan al usuario como una lista desplegable seleccionable.
2. **Ingreso de datos por el usuario**
 - El usuario debe:
 - Seleccionar una razón del reporte de la lista desplegable.
 - Ingresar, de manera opcional, una descripción que detalle la incidencia.
3. **Obtención automática de datos**
 - La aplicación obtiene automáticamente:
 - **Ubicación geográfica:** Utilizando el servicio de localización, se almacena la ubicación en formato geoPoint.
 - **Marca de tiempo:** Se genera un Timestamp en el momento exacto en que se crea el reporte.
4. **Almacenamiento en Firestore**
 - Una vez completado el formulario, los datos del reporte se guardan como un nuevo documento dentro de la subcolección `/reports` asociada a la ruta activa.
 - La estructura de la jerarquía es:
`/users/{userId}/routes/{routeId}/reports/{reportId}`

5.4 Integración con App Web (Administrador)

La aplicación web está dirigida a los usuarios administradores del sistema, cuenta con autenticación, un dashboard con varias visualizaciones basadas en la información de las rutas, gestión de usuarios y sus sectores y gestión de camiones, estas últimas gestiones de datos son las que interactúan directamente con la base de datos Firestore.

5.4.1 Gestión de usuarios

La gestión de usuarios en el sistema abarca las operaciones CRUD (Crear, Leer, Actualizar y Eliminar) e integra Firebase Authentication con Firestore para almacenar información adicional relevante sobre los usuarios. Estas operaciones se llevan a cabo en la colección `/users` de Firestore y garantizan una sincronización entre los datos de autenticación y los datos personalizados de los usuarios.

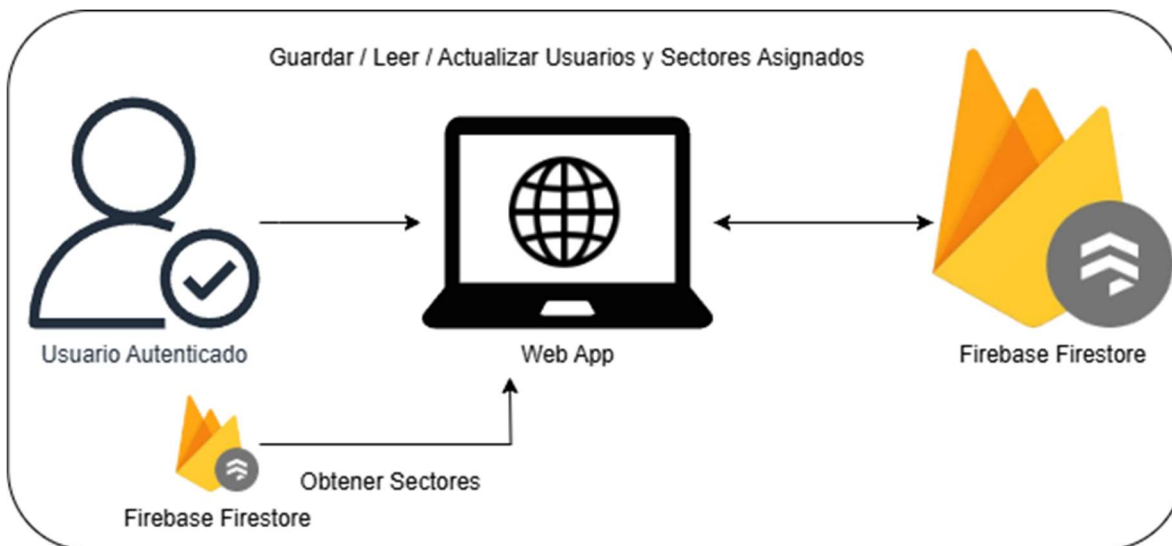


Figura 15. Flujo de interacción App Web-Firestore en gestión de usuarios (Fuente: elaboración propia)

Creación de Usuarios

La creación de usuarios en el sistema se realiza mediante Firebase Authentication, donde inicialmente se registra un usuario utilizando un correo electrónico y una contraseña. Este proceso genera automáticamente un UID que permite identificar de forma inequívoca al usuario en todos los sistemas integrados.

Posteriormente, la información adicional del usuario, como nombre, apellido, RUT y rol, se almacena en Firestore en un documento dentro de la colección `/users/{userId}`, utilizando el UID generado como ID del documento.

Lectura de Usuarios

La lectura de usuarios se implementa mediante consultas directas a Firestore. A través de la colección `/users`, se recuperan los documentos que representan a cada usuario. Esta implementación considera tanto la obtención de la lista completa de usuarios como la consulta de usuarios individuales mediante su UID rescatada al seleccionar un usuario de la lista de usuarios consultada originalmente, la consulta individual permite conseguir los datos del usuario en específico que no se hayan rescatado en la lista de usuarios. Cualquier filtro sobre la lista de usuarios de la consulta inicial es realizado a nivel de cliente sin consultas adicionales a la base de datos.

Actualización de Usuarios

El sistema permite actualizar parcialmente los datos almacenados de los usuarios. Para esto, se realizan actualizaciones directas al documento correspondiente en la colección `/users/{userId}`. Esta funcionalidad es esencial para mantener actualizada la información del usuario, como cambios en el nombre, correo electrónico o rol. No es posible la actualización de la ID del usuario.

Eliminación de Usuarios

La eliminación del usuario en Firestore significaría la eliminación de todas las subcolecciones que este ha creado, incluyendo las rutas y las ubicaciones guardadas, por lo tanto, la eliminación de usuarios no es realmente

una eliminación del sistema, los usuarios se marcan como inactivos a través del campo active del documento en Firestore.

Asignación de Sectores a Usuarios

El sistema incluye la capacidad de asignar sectores específicos a usuarios, los cuales están asociados a días de la semana. Este proceso se realiza mediante la creación de documentos dentro de la subcolección `/users/{userId}/assigned_sectors`.

1. Creación de asignaciones

- Se genera un nuevo documento en la subcolección `/assigned_sectors`, asociando el sector, el día de la semana y la fecha de asignación.
- Los sectores asignados son marcados como activos.

2. Desactivación de asignaciones

- Para desactivar un sector asignado, se actualiza el estado del documento correspondiente dentro de la subcolección, marcándolo como inactivo.

Consultas Avanzadas de Usuarios y Sectores

El sistema incluye capacidades avanzadas de lectura para la administración de usuarios y sectores:

- **Sectores Activos de un Usuario:** Recupera la lista de sectores asignados y activos para un usuario específico.
- **Usuarios Sin Asignaciones Activas:** Identifica a los usuarios que no tienen sectores activos en ciertos días de la semana.
- **Sectores No Asignados:** Permite identificar sectores que no han sido asignados a ningún usuario activo.

5.4.2 Gestión de camiones

Este módulo abarca las operaciones CRUD (Crear, Leer, Actualizar y Eliminar), las cuales se realizan sobre la colección `/trucks` en Firestore.

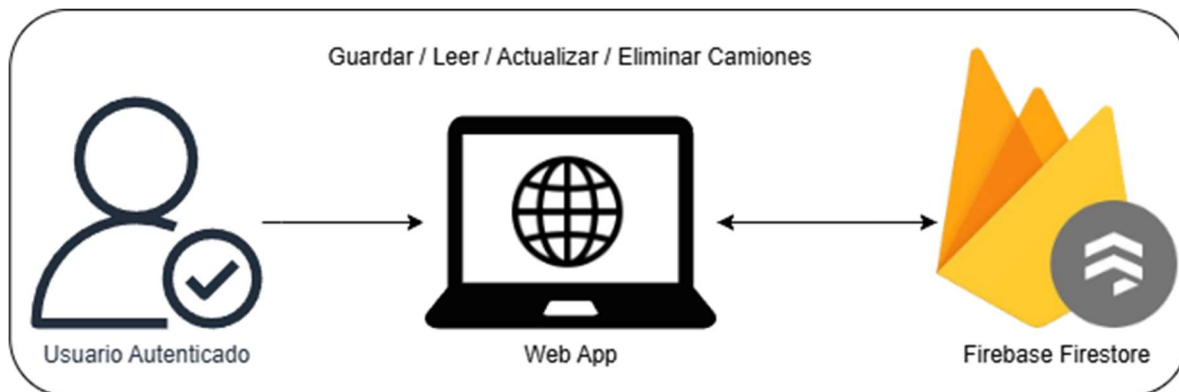


Figura 18. Flujo de interacción App Web-Firestore en gestión de camiones (Fuente: elaboración propia)

Creación de Camiones

Para registrar un nuevo camión, se crea un documento en la colección `/trucks` que almacena información clave como la matrícula del camión y su estado, el cual puede ser 'Funcional' o 'No Funcional'.

Lectura de Camiones

La lectura de datos permite acceder a la lista de camiones registrados en el sistema. Esta operación incluye la obtención de toda la información de los camiones en Firestore.

Actualización de Camiones

El sistema permite modificar los datos de un camión existente, como su matrícula o estado. Esto es útil, por ejemplo, para actualizar el estado de un camión que pasa de estar funcional a no funcional. No es posible actualizar la ID del camión.

Eliminación de Camiones

La eliminación de camiones implica borrar su registro de la base de datos Firestore. A diferencia de los usuarios esta opción se presenta como una eliminación real del documento, y se sugiere el uso de eliminación solo para camiones ingresados erróneamente al sistema, el campo “state” que representa el estado funcional del camión puede ser utilizado para actuar como marcador de documento activo o inactivo.

5.5 Integración con App Móvil (Ciudadano)

La aplicación móvil dirigida a la ciudadanía tiene como objetivo mostrar en tiempo real los camiones recolectores en rutas activas, con esto en mente las consultas a realizar son limitadas, esto es importante considerando que la aplicación móvil será distribuida sin la necesidad de autenticar al usuario.

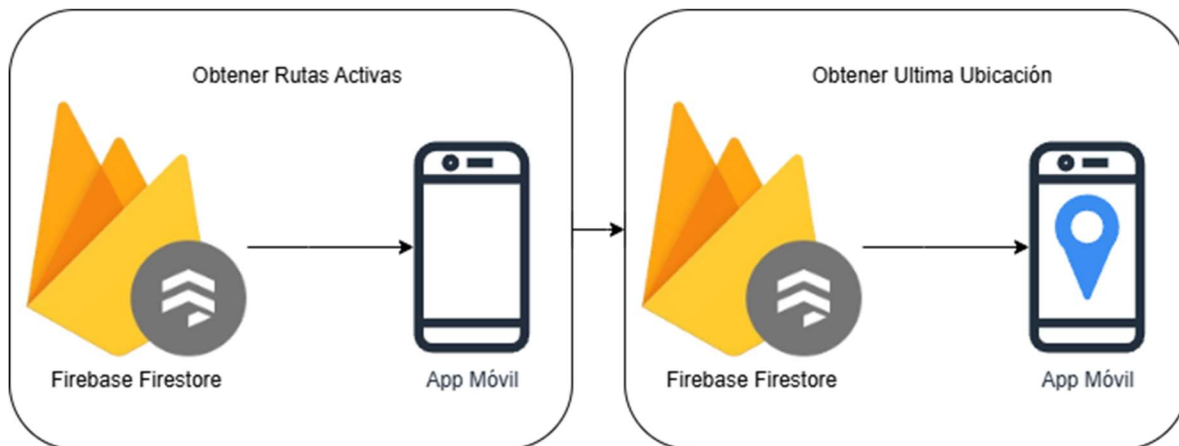


Figura 19. Flujo de la obtención de ubicación de camiones en tiempo real (Fuente: elaboración propia)

1. Consulta de rutas activas

- La consulta se realiza a la colección `/routes`
- Se accede únicamente a las rutas donde el campo `active` es igual a `true`.
- La información obtenida se actualiza en tiempo real mediante el uso de *streams* en Flutter, garantizando que cualquier cambio en la ubicación o el estado de las rutas se refleje instantáneamente en la interfaz.

2. Ubicación de los camiones

- Para cada ruta activa, se accede a la subcolección `/locations` para obtener la última ubicación registrada del camión.
- Estos datos se utilizan para actualizar los marcadores en el mapa integrado en la aplicación (Google Maps), lo que permite a los ciudadanos visualizar la posición de los camiones en tiempo real.

5.6 Integración con BigQuery

BigQuery es un data warehouse utilizado para la extracción, transformación y carga de datos (ETL), utiliza su propia base de datos y es parte del área de inteligencia de negocios, en el caso de este proyecto este data warehouse es utilizado para la posterior creación de visualizaciones y análisis en Looker Studio.

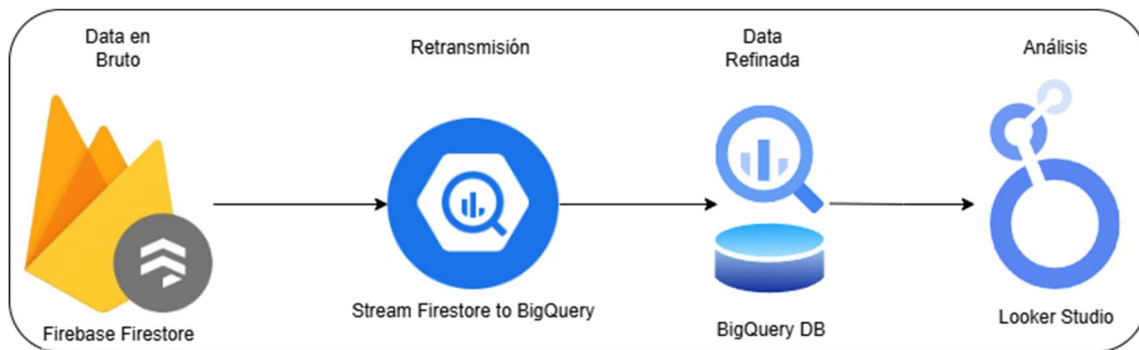


Figura 20. Flujo de la integración Firestore-BigQuery (Fuente: elaboración propia)

La integración con BigQuery en el proyecto se implementó mediante la extensión "Stream Firestore to BigQuery," la cual permite capturar en tiempo real los cambios en una colección de Firestore, facilitando el flujo continuo de datos hacia un dataset de BigQuery. Esta extensión rastrea automáticamente cualquier cambio en los documentos de la colección especificada, incluidos la creación, modificación y eliminación de documentos.

6 Reglas de Seguridad

La seguridad de la base de datos en Firestore se garantiza mediante la implementación de Reglas de Seguridad, un mecanismo que permite definir condiciones específicas para el acceso a los documentos y colecciones. Estas reglas funcionan evaluando cada solicitud de lectura, escritura o eliminación, asegurando que se cumplan los criterios establecidos antes de conceder el acceso.

El formato general para escribir estas reglas sigue la estructura básica:

```
match /<ruta_del_documento>/ {  
  allow read, write: if <condición>;  
}
```

Por ejemplo, la siguiente regla base permite tanto la lectura como la escritura en cualquier documento de la base de datos, pero únicamente si la solicitud se realiza antes de una fecha específica:

```
match /{document=**} {  
  allow read, write: if request.time < timestamp.date(2024, 11, 15);  
}
```

6.1 Autenticación

Uno de los pilares fundamentales de la seguridad en Firestore es la autenticación de usuarios. Este mecanismo asegura que solo los usuarios identificados tengan acceso a los datos almacenados. Para implementar un bloqueo general que exija autenticación en toda la base de datos, se utiliza una regla que permite la lectura y escritura únicamente si la solicitud incluye credenciales válidas:

```
match /{document=**} {  
  allow read, write: if request.auth != null;  
}
```

Además de la autenticación básica, es crucial establecer políticas de acceso diferenciadas según los roles y las responsabilidades de los usuarios. En el contexto de los conductores de la aplicación móvil, se garantiza que solo puedan acceder y modificar los datos que les correspondan, basándose en su ID único de usuario. La siguiente regla permite que un usuario acceda exclusivamente a sus propios datos de perfil y rutas:

```
match /users/{user=**} {  
  allow read, update: if request.auth.uid == userId;  
}
```

6.2 Validación de data

En el caso de la aplicación móvil destinada a la ciudadanía, donde no se requiere autenticación, se implementan reglas de seguridad que permiten la validación de datos para proteger la integridad de la información sin comprometer la accesibilidad pública. Un ejemplo es la regla que autoriza la lectura de documentos que representan rutas, siempre y cuando el campo “active” esté configurado como “true”. Esto asegura que solo se puedan visualizar rutas activas, manteniendo una experiencia coherente para los usuarios:

```
match /routes/{route} {  
  allow read: if resource.data.active == true;  
}
```

6.2.1 Inmutabilidad de Campos

Además de controlar el acceso, las reglas pueden usarse para restringir modificaciones de datos específicos, garantizando que ciertos campos permanezcan inmutables tras su creación. Por ejemplo, el siguiente fragmento impide la modificación del campo “location” de un documento, salvo que el nuevo valor coincida exactamente con el actual, preservando la consistencia de la información:

```
match /locations/{location} {  
  allow update: if resource.data.location ==  
    request.resource.data.location;  
}
```

6.2.2 Validación de Tipos y Contenidos

Firestore permite definir condiciones para validar el tipo y el rango de los datos almacenados, emulando las restricciones (constraints) típicas de bases de datos SQL. Por ejemplo, las siguientes reglas permiten la creación y actualización de documentos de tipo “route” solo si los campos “start_weight” y “end_weight” son de tipo “int” y tienen valores mayores o iguales a 0:

```
match /routes/{route} {  
  allow create, update: if request.resource.data.start_weight >= 0 &&  
    request.resource.data.start_weight is int;  
  allow create, update: if request.resource.data.end_weight >= 0 &&  
    request.resource.data.end_weight is int;  
}
```

Aunque es posible implementar reglas de seguridad para la validación de datos, no es recomendable utilizarlas como el principal mecanismo de validación. Es preferible que estas validaciones se realicen en el lado del cliente antes de enviar los datos a Firestore. El uso excesivo de reglas de validación dentro de Firestore puede afectar el rendimiento del sistema, debido a la necesidad de evaluar las condiciones en cada operación de escritura. Además, aumenta la complejidad de las reglas, dificultando su mantenimiento y comprensión. Por

ello, las reglas de seguridad deben priorizar el control de acceso, mientras que la validación de datos debe enfocarse en el cliente, aprovechando herramientas y frameworks que aseguren datos consistentes antes de interactuar con la base de datos.

6.3 Acceder Otros Documentos

Firestore permite definir reglas de seguridad que consideran no solo el documento sobre el que se realiza la operación, sino también documentos relacionados, accediendo a sus datos mediante consultas específicas. Esto es útil cuando las condiciones de acceso dependen de un campo almacenado en un documento padre o en una estructura jerárquica.

En el siguiente fragmento se realiza una consulta escribiendo la ruta entera del documento a buscar y se utilizan los datos disponibles dentro de este nido de reglas, esta última regla se encuentra dentro de las reglas anteriormente mencionadas en este capítulo, por lo que se tiene el acceso a los documentos “route”, “user” y la “database” en la que se está trabajando. En este caso permitimos el acceso de lectura al documento “location” cuando el campo “active” de la data consultada es equitativo a “true”.

```
match /locations/{location} {  
  allow read: if  
get(/databases/{database}/documents/users/{user}/routes/{route}).data.  
active == true;  
}  
}
```

7 Optimización y Rendimiento

La optimización y el rendimiento en Firestore se logran mediante prácticas clave como el uso de límites en las consultas para evitar sobrecargas, la creación de índices compuestos para consultas complejas y la realización de pruebas de rendimiento que evalúan la capacidad del sistema en escenarios reales. Estas estrategias permiten un manejo eficiente de los datos, asegurando tiempos de respuesta rápidos y un sistema escalable que satisface las necesidades del proyecto. En general, se utilizan técnicas para asegurarse que las consultas a la base de datos sean necesarias, considerando, por ejemplo, casos en que la información que se consulta, ya se tiene.

7.1 Límites

Firestore permite gestionar eficientemente altos volúmenes de datos mediante la implementación de límites y paginación en las consultas. Estas capacidades no solo evitan la sobrecarga del servidor, sino que también optimizan el uso de recursos en las aplicaciones cliente, mejorando su rendimiento general al manejar únicamente los datos necesarios.

En el proyecto, las consultas con límites han sido integradas como una medida clave para:

1. **Evitar Sobrecargas Accidentales:**
 - Durante las fases de desarrollo y producción, establecer límites reduce el riesgo de realizar consultas que consuman un volumen excesivo de datos o recursos.
 - Esto es especialmente útil para evitar sobrepasar el umbral gratis de utilización de características Firebase.
2. **Optimización de Recursos:**
 - Reducir la cantidad de datos procesados disminuye el uso de recursos y el tiempo de respuesta de las consultas.
 - Esto beneficia tanto al servidor como al cliente.
3. **Casos de Uso Prácticos:**

- En operaciones simples, como la obtención de un solo documento, los límites simplifican las consultas.
- Por ejemplo, en la aplicación móvil para la ciudadanía, se utiliza un límite para recuperar únicamente la última ubicación registrada en la subcolección `/locations`, mejorando la eficiencia de la consulta.

7.2 Índices

En Firestore, los índices son estructuras diseñadas para optimizar la búsqueda y recuperación de documentos, mejorando el rendimiento de las consultas realizadas en la base de datos.

Firestore genera automáticamente índices de un solo campo para cada documento en la base de datos. Estos índices permiten realizar consultas simples, como búsquedas basadas en un único filtro u ordenamiento, sin necesidad de configuración adicional.

Para consultas más complejas que involucren múltiples filtros, órdenes específicos o una combinación de ambos, Firestore requiere índices compuestos. Estos índices no se crean de una manera totalmente automática, deben definirse de forma manual o mediante una configuración proporcionada por Firestore una vez que intentemos realizar una consulta que requiera de un índice compuesto.

En la aplicación móvil ciudadana, se implementó una consulta compuesta sobre la colección `/routes`. Esta consulta utiliza:

1. **Filtro:** El campo `active`, para incluir únicamente las rutas activas.
2. **Ordenación:** El campo `start_time`, ordenado de forma descendente para priorizar las rutas más recientes.

El índice compuesto necesario para esta consulta fue configurado de la siguiente manera:

- **Colección:** `/routes` (aplicada a todas las subcolecciones `/routes`).
- **Campos:**
 - `active` (ascendente).
 - `start_time` (descendente).

7.3 Pruebas

Algunas pruebas de rendimiento se llevaron a cabo utilizando la Firebase Emulator Suite para simular escenarios reales del sistema, específicamente la actualización de ubicaciones en rutas activas y las consultas de dichas ubicaciones en tiempo real por parte de los usuarios finales. Estas pruebas permiten evaluar la capacidad del sistema para manejar operaciones concurrentes.

7.3.1 Configuración de Pruebas Controladas

1. **Entorno de Pruebas:**
 - Se utilizó la Firebase Emulator Suite para replicar un entorno de Firestore en local.
 - El emulador permitió realizar operaciones sin costos asociados y sin afectar datos en producción.
2. **Escenario Simulado:**
 - Se crearon 50 usuarios con 1 ruta activa cada uno.
 - Cada ruta recibió actualizaciones de ubicación cada 2 segundos.
 - Los datos fueron consultados cada 3 segundos por un usuario simulado, buscando la última ubicación de cada ruta activa.
3. **Objetivo:**
 - Medir los tiempos promedio de:
 - **Actualización de ubicaciones:** Tiempo necesario para registrar nuevas ubicaciones en la subcolección `/locations`.

- **Consulta de ubicaciones:** Tiempo necesario para obtener la última ubicación registrada de las rutas activas.

7.3.2 Resultados de las Pruebas

Métrica	Promedio Obtenido	Descripción
Tiempo de actualización	13.82ms	Tiempo que demora en actualizar la última localización de una ruta
Tiempo de consultas	28.08ms	Tiempo que demora un dispositivo en consultar la última localización de una ruta

Figura 21. Resultados de pruebas de rendimiento Firestore

Análisis del Tiempo de Consulta:

- Un usuario que consulta la ubicación de las 50 rutas activas necesitaría aproximadamente 1.4 segundos (28 ms por ruta). Este tiempo se considera muy bueno para la operación del sistema en escenarios estándar.

7.3.3 Limitaciones Identificadas

Aunque las pruebas realizadas proporcionaron información clave sobre el rendimiento, es importante destacar las siguientes limitaciones:

- **Condiciones de Prueba Controladas:**
 - El entorno simulado no incluye latencias externas, como las relacionadas con la conexión a Internet o la latencia en la interfaz del usuario.
 - Las reglas de seguridad de Firestore, que podrían introducir demoras adicionales en producción, no se consideraron.
- **Desempeño en Campo:**
 - Las pruebas manuales realizadas en un dispositivo móvil emulado mostraron que la transmisión de datos desde la aplicación del conductor a la base de datos tiene una demora variable entre 2 y 15 segundos, con un promedio de 5 segundos.
 - Hubo ocasiones en las que los datos no se cargaban a la base de datos desde un dispositivo real.
 - La falta de carga de datos fue atribuida principalmente a dos causas humanas:
 - Falta de permisos de ubicación en segundo plano.
 - Se cerraba la aplicación en lugar de mantenerla en segundo plano.
 - La falta de carga de datos parcial fue atribuida a dos causas técnicas:
 - Problemas de conexión al estar en un área con muchos dispositivos cercanos, por ejemplo, en una congestión vehicular.
 - Problemas de conexión al estar en un área con poca cobertura.

7.3.4 Pruebas en Campo

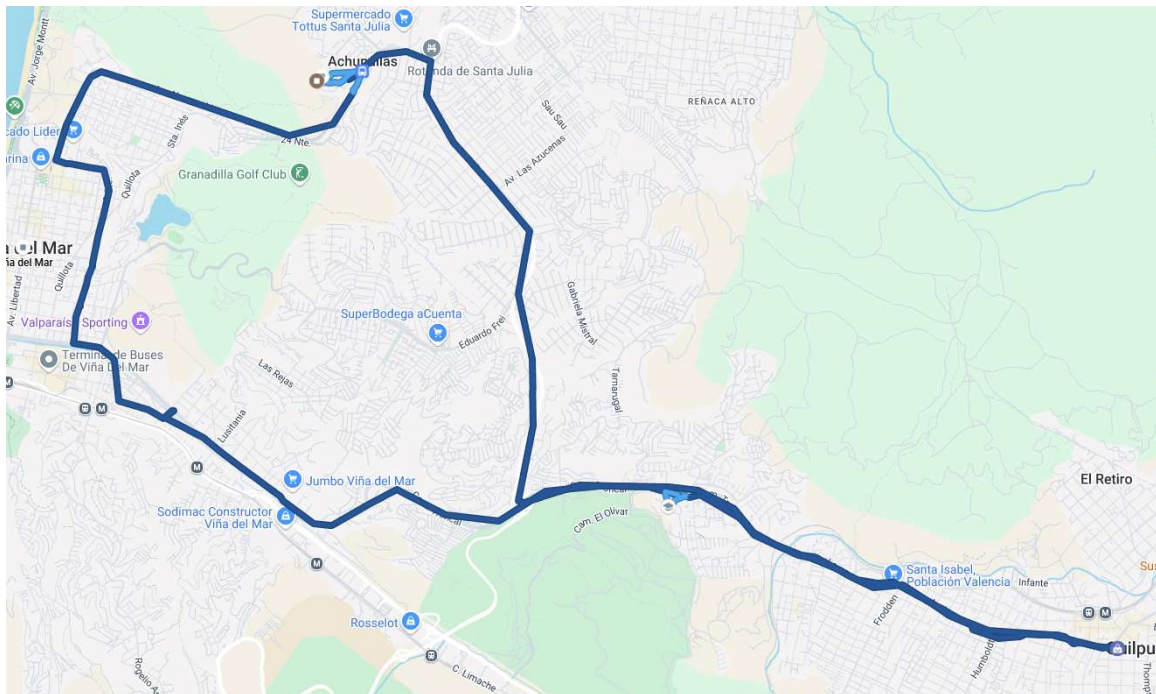


Figura 22. Línea de tiempo de ruta realizada en la vida real (Fuente: Timeline personal Google Maps)

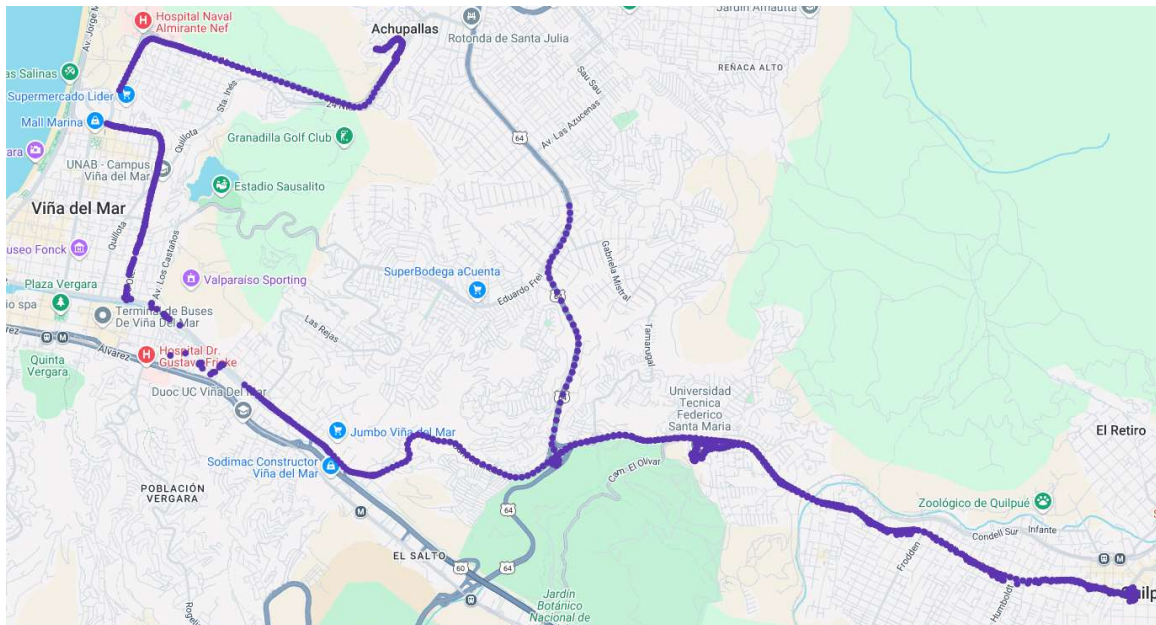


Figura 23. Línea de tiempo de ruta realizada a través de app móvil conductor (Fuente: Visualización Looker Studio)

La figura 11 representa la línea de tiempo obtenida automáticamente por Google Maps para una persona, mientras que la figura 12 muestra los puntos registrados por la aplicación móvil del conductor y visualizados en Looker Studio, con datos almacenados inicialmente en Firestore y transferidos a BigQuery. Estas visualizaciones permiten una comparación directa entre la ruta capturada por la aplicación y la ruta "real" registrada por Google Maps. Cabe considerar que la ruta de la aplicación móvil se inicializa en un punto intermedio en la carretera, no desde el inicio.

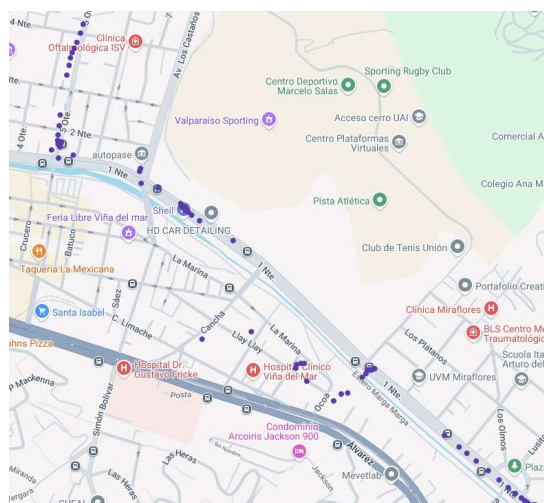


Figura 24. Falta de datos en calle 1 norte
(Fuente: Visualización Looker Studio)

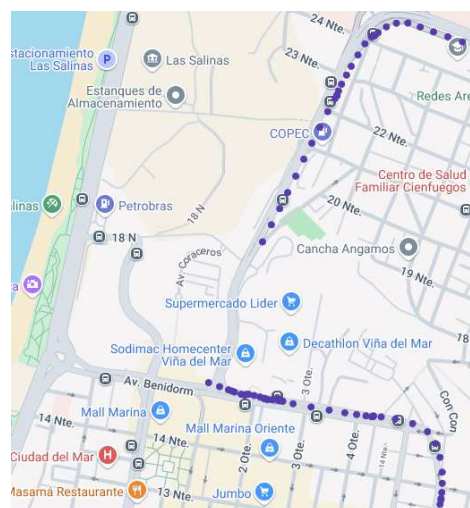


Figura 25. Falta de datos en Avenida Alessandri
(Fuente: Visualización Looker Studio)

Problemas específicos surgieron en ciertas áreas, como la falta de localizaciones cerca del "Hospital Dr. Gustavo Fricke" y en una calle cercana al "Mall Marina". Estas discrepancias se atribuyen a fallas en la obtención de la ubicación o a problemas de conexión a internet que impiden el registro oportuno en Firestore.

Una posible solución a estos problemas es mejorar la integración entre la base de datos y la aplicación móvil. Actualmente, los datos se transfieren en tiempo real, lo que depende de una conexión constante. Una estrategia más robusta sería implementar un sistema que almacene temporalmente las ubicaciones en el dispositivo y reintente la sincronización con Firestore una vez que la conexión a internet sea restablecida. Esto aseguraría la integridad y completitud de los datos en la base de datos, incluso en condiciones de conectividad inestable.

8 Resultados

El diseño inicial basado en un modelo relacional permitió una estructuración clara y normalizada de las entidades, lo que facilitó la adaptación al sistema NoSQL de Firestore. La jerarquía en Firestore permitió organizar los datos de manera intuitiva para su uso en las aplicaciones del proyecto. Este diseño jerárquico resultó clave para soportar las operaciones principales como la gestión de usuarios, rutas, reportes y localizaciones. Sin embargo, la complejidad de algunas consultas deriva del uso de campos de referencia, en lugar de realizar una desnormalización completa, esto limitó la flexibilidad del diseño y puede reducir ligeramente el rendimiento.

La integración con las aplicaciones móviles y web fue exitosa. En la aplicación para conductores, Firestore permitió la gestión de rutas activas y el registro en tiempo real de ubicaciones. Aunque el sistema funcionó correctamente en pruebas controladas, algunos problemas en campo se atribuyeron a factores externos, como la falta de capacitación de los usuarios o problemas de conexión en áreas con poca cobertura o congestión vehicular. Por otro lado, en la aplicación ciudadana, la consulta de rutas activas y localizaciones se realizó sin problemas, garantizando la funcionalidad esperada sin necesidad de autenticación. En la aplicación web, se logró una administración eficaz de usuarios, camiones y sectores. Además, la integración de Firestore con BigQuery permitió alimentar visualizaciones en Looker Studio, mostrando la información correcta sin demoras perceptibles.

Las reglas de seguridad implementadas aseguraron que solo los usuarios autorizados accedieran a los datos pertinentes, con restricciones adicionales para proteger la integridad de la información. Al mismo tiempo, estas reglas permitieron adaptarse a distintos casos de uso, como la consulta pública de rutas activas y la validación de datos en tiempo real para evitar errores en la información almacenada.

La implementación de límites e índices compuestos en Firestore permite mejorar automáticamente la eficiencia de las consultas, permitiendo manejar un alto volumen de datos sin comprometer el rendimiento. Por otro lado, la lógica de consultas implementada en las aplicaciones jugó un rol crucial para decidir cuándo realizar operaciones, optimizando así el uso de recursos tanto del servidor como del cliente.

En conjunto, estos aspectos aseguraron que la base de datos no solo cumpliera con los objetivos iniciales, sino que también proporcionara una plataforma lo suficiente sólida y escalable para el desarrollo del proyecto, permitiendo futuras ampliaciones y adaptaciones según sea necesario.

9 Conclusiones

En el contexto de un proyecto con recursos y tiempo limitados, la decisión de utilizar Firestore como base de datos se alineó con la necesidad de desarrollar un producto mínimo viable (MVP) que fuera escalable y fácil de integrar en las aplicaciones móviles y web. Firestore, con su estructura NoSQL, permitió una implementación rápida y sencilla de la base de datos, sin la necesidad de una compleja gestión de servidores o configuraciones adicionales. Esto fue especialmente útil para cumplir con los plazos establecidos, ya que las funciones clave del sistema, como la gestión de rutas, usuarios, camiones y reportes, pudieron implementarse con rapidez y eficiencia. El desarrollo del proyecto tomó aproximadamente tres meses desde la conceptualización inicial hasta la implementación final.

Además, la escalabilidad automática de Firestore permitió que el sistema pudiera manejar el crecimiento esperado de datos sin comprometer el rendimiento. Esto fue fundamental para poder cumplir con las expectativas de un sistema capaz de manejar en tiempo real la información sobre rutas y localizaciones de camiones recolectores, algo que se requería para asegurar una experiencia fluida en la aplicación ciudadana y en la aplicación para los conductores.

No obstante, a pesar de los beneficios que Firestore ofreció en términos de agilidad en el desarrollo y escalabilidad, también se identificaron algunas limitaciones inherentes a su estructura NoSQL. Si bien la flexibilidad de Firestore fue una ventaja en términos de adaptabilidad, el manejo de relaciones complejas entre datos no es tan eficiente como en una base de datos relacional. Esto se reflejó en algunos casos donde las consultas complejas entre entidades, como la relación entre usuarios, rutas y sectores, no se manejaron tan fácilmente como se habría hecho en un sistema SQL tradicional.

En este sentido, el diseño de la base de datos fue fundamental para el éxito del proyecto, especialmente en términos de cómo los datos se organizaban de manera jerárquica en Firestore. Sin embargo, al no hacer una desnormalización completa, surgieron algunas dificultades en cuanto a la complejidad de las consultas, lo que limitó algo la flexibilidad del sistema.

El enfoque en la implementación de un MVP significó que el equipo priorizó las integraciones y funciones esenciales sobre optimizaciones profundas. A pesar de esto, las decisiones tomadas respecto a la optimización del rendimiento, como la implementación de límites en las consultas y el uso de índices compuestos, garantizaron que el sistema pudiera escalar y operar con eficiencia bajo las condiciones previstas.

Durante el desarrollo del proyecto, fue necesario adquirir conocimientos adicionales sobre Firebase Authentication para integrar datos personalizados a los usuarios autenticados, así como sobre la integración de Firestore con BigQuery para alimentar el software de visualización Looker Studio. Además, se profundizó en características avanzadas de Firestore, como sus reglas de seguridad, la suite de pruebas emuladas y los

mecanismos de optimización. Aunque Firestore facilita la automatización y permite una implementación ágil, este aprendizaje es clave para maximizar su potencial en el proyecto ya sea actual como futuro. Como parte de la investigación, se exploraron sistemas equivalentes a Uber, lo que permitió identificar estrategias y enfoques útiles para desarrollar funcionalidades en tiempo real de manera eficiente.

La experiencia adquirida durante mi formación profesional fue fundamental para llevar a cabo este proyecto. El conocimiento previo en diseño de bases de datos, el uso de Firebase y el desarrollo de aplicaciones móviles con Flutter proporcionaron una base sólida para enfrentar los desafíos técnicos. El diseño de la base de datos, como uno de los aspectos más importantes del proyecto, requirió adaptarse a los requerimientos específicos del cliente, algo que también formó parte de mi preparación profesional. Además, las prácticas profesionales en desarrollo de software y bases de datos enriquecieron estas habilidades, permitiéndome aplicar soluciones efectivas y personalizadas para cumplir con los objetivos planteados.

El sistema ha sido probado con conductores reales de la Municipalidad de Viña del Mar, logrando resultados positivos en las pruebas manuales y captando el interés de los administradores de la flota. Aunque aún no está completamente implementado, se espera su explotación en el futuro. En el caso una continuación del desarrollo del proyecto esta vez con clientes aún más dispuestos a recibirlo se esperan mejoras en el sistema, como rediseños basados en nuevos requerimientos de usuarios, la posible migración total o parcial a bases de datos relacionales para manejar mejor las relaciones complejas y los cambios en el resto de las tecnologías integradas.

El proyecto logró cumplir con los objetivos iniciales y demostrar la viabilidad de la integración de Firestore con las aplicaciones móviles y web, cumpliendo con los requisitos funcionales de manera efectiva. Aunque existen áreas para mejorar, particularmente en el manejo de relaciones complejas entre datos, las decisiones tomadas fueron acertadas para cumplir con los tiempos y recursos limitados, permitiendo el desarrollo de un producto funcional y escalable dentro de las limitaciones impuestas.

Agradecimientos

Quisiera agradecer a mi familia por siempre darme el cariño, apoyo y siempre hacer lo posible por cumplir mis deseos, creer en mí, acomodarme como pudieran y aceptarme siempre como soy.

Agradecimientos a todos los amigos que he hecho en el transcurso de mi carrera universitaria, con los quienes he compartido siempre momentos divertidos y estoy extremadamente satisfecho de haberlos conocidos a todos. Agradecimientos especiales a mi amigo Renzo por un apoyo que necesitaba para finalizar mis estudios.

Y, por último, agradecimiento a mi equipo conformador del proyecto “Smart Waste”; Claudia Gómez, Maximiliano Munizaga y Zachir Faúndez. Este proyecto no habría sido posible de realizar con éxito sin ellos.

Referencias

[1] (2024, 15 febrero). *How Uber Serves Over 40 Million Reads Per Second from Online Storage Using an Integrated Cache* | Uber Blog.

<https://www.uber.com/en-CL/blog/how-uber-serves-over-40-million-reads-per-second-using-an-integrated-cache/#:~:text=Introduction,microservices%20from%20all%20business%20verticals>.

[2] (2024, 16 agosto). *Fleet management software for solid waste - Routeware*. | Routeware Fleet Management And Engagement Platform.

<https://routeware.com/>

[3] (2024). *Database Diagram Tool* | holistics.io. dbdiagram.io.

<https://dbdiagram.io/home>

[4] (2024, 26 noviembre). *Firestore Documentation* | Firebase. Firebase Documentation.

<https://firebase.google.com/docs/firestore>

[5] (2023, 9 diciembre). *Stream Firestore to BigQuery* | Firebase Extensions. Firebase Extensions Documentation.

<https://extensions.dev/extensions/firebase/firestore-bigquery-export>

[6] (2024, 26 noviembre). *Security rules in Firestore* | Firebase. Firebase Documentation.

<https://firebase.google.com/docs/firestore/security/overview>