



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA INGENIERÍA
EN INFORMÁTICA

GeoCuadrilla: Desarrollo Front-End de aplicación móvil para la recolección y transmisión de datos operativos de cuadrillas en terreno en Chilquinta transmisión.

Patricio Oyarzo Aguilera.

patricio.oyarzo@usm.cl

Catherine Gómez.
Profesor Guía.

David Larrondo.
Profesor Correferente.



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: GeoCuadrilla : Desarrollo Front-end de aplicación móvil para la recolección y transmisión de datos operativos de cuadrillas en terreno en Chilquinta transmisión.

Nombre del candidato(a): Patricio Ignacio Oyarzo Aguilera

Carrera / Grado: Ingeniería en Informática

Campus: Viña del Mar **Departamento:** Electrotecnia e Informática

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Catherine Gómez B., en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 23/09/2026

Firma:

Catherine BC

Estudiante o Candidato(a):

Fecha: 23/09/2026

Firma:

[Firma]

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

Resumen: Las empresas de distribución eléctrica enfrentan desafíos críticos en el monitoreo en tiempo real de cuadrillas de trabajo en terreno, lo que impacta negativamente en los tiempos de respuesta ante emergencias y la toma de decisiones operativas. Chilquinta experimenta esta problemática al carecer de visibilidad constante sobre la ubicación y estado de sus equipos de trabajo.

El objetivo de este trabajo es desarrollar GeoCuadrilla, un sistema integral de seguimiento y monitoreo que incluye una aplicación móvil desarrollada en *Flutter* para trabajadores en terreno y un panel web en Angular para el centro de control. La solución implementa georreferenciación mediante dispositivos móviles institucionales, integración con *Google Maps* API y transmisión de datos en tiempo real usando *RabbitMQ*.

La metodología aplicada incluye desarrollo bajo arquitectura de componentes, implementación de patrón *Provider* para gestión de estado, y pruebas de funcionalidad en ambiente controlado con validación de conectividad y precisión de ubicación.

Los resultados obtenidos muestran que la arquitectura implementada cumple con los criterios de aceptación no funcionales definidos para el sistema: la ubicación se obtiene en menos de 2 segundos, el consumo de batería se mantiene bajo el 10% por hora con tracking activo, la reconexión ante pérdida de conectividad ocurre en menos de 5 segundos, los tokens se almacenan cifrados y el inicio del tracking puede completarse en menos de 3 pasos. Estos resultados evidencian viabilidad técnica de la solución para el monitoreo en tiempo real de cuadrillas en terreno. Se espera que su implementación favorezca una mejor organización de recursos y una atención mas oportuna ante incidencias; sin embargo, la validación se limitó a un ambiente controlado, por lo que cuantificar esos beneficios operativos queda como trabajo futuro.

Estos avances se alinean directamente con el impacto esperado del proyecto, el cual contempla la optimización de recursos operativos, la mejora en la calidad de servicio eléctrico y el fortalecimiento de la gestión de emergencias en Chilquinta.

Palabras Clave: Monitoreo, *Flutter*, *Front-end* móvil, API, *RabbitMQ*.



Contenidos

1	Introducción	4
1.1	Contexto y relevancia del desarrollo de componentes específicos en el proyecto.	4
1.2	Definición del problema.	5
1.3	Breve descripción sobre la propuesta de solución.	5
1.4	Objetivos Generales y Específicos de la Tesina.	5
1.5	Justificación de proyecto.	6
1.6	Metodología.	7
1.7	Breve descripción de la organización del informe en capítulos.	8
2	Marco Teórico.....	9
2.1	Fundamentos del desarrollo front-end.	9
2.2	Arquitectura de software.	9
2.3	Tecnologías y frameworks utilizados.	10
3	Diseño e Implementación.	11
3.1	Diseño de Componentes.	11
3.1.1	Arquitectura y estructura de los componentes.	11
3.1.2	Uso de buenas prácticas y patrones de diseño.....	15
3.1.3	Integración con el sistema general.	17
3.2	Diseño de la Interfaz.....	20
3.3	Detalles de la codificación y desarrollo.	25
3.4	Pruebas y Validación	26
3.4.1	Estrategias de testing aplicadas a los componentes.....	27
3.4.2	Resolución de bugs y optimización.....	29
3.5	Integración con Otros Componentes.	29
4	Conclusiones, logros, limitaciones y trabajo futuro.	30
4.1	Conclusiones.	30
4.2	Logros.....	31
4.3	Limitaciones.	31
4.4	Trabajo Futuro.....	33
5	Agradecimientos.	34
6	Referencias.	35
7	Anexos.....	37

1 Introducción

1.1 Contexto y relevancia del desarrollo de componentes específicos en el proyecto.

El aumento exponencial del avance tecnológico y la creciente demanda por servicios eléctricos confiables han expuesto la necesidad de modernizar las herramientas de gestión operativas de las empresas distribuidoras. En Chilquinta, la supervisión de las cuadrillas que realizan trabajos en terreno como mantenciones y reparaciones depende en gran medida de procesos manuales o semiautomatizados que no entregan una visibilidad notoria en tiempo real, +esta carencia provoca esperas prolongadas, asignaciones ineficientes de los recursos y dificultades para priorizar intervenciones, afectando tanto a la continuidad del servicio como la experiencia del usuario.

Frente al contexto de esta problemática, Geocuadrilla propone como solución a las carencias encontradas, una plataforma integrada que combina una aplicación móvil ligera para la captura automática de ubicaciones y datos operativos, junto con un panel web de monitoreo central. La solución descrita incorpora la georreferenciación con *Google Maps*, mensajería de datos operativos en tiempo real mediante *RabbitMQ* y arquitecturas modernas que permiten despliegues más rápidos y mejora en la toma de decisiones respecto a un accidente o problema en la red eléctrica. El diseño propuesto considera la simplicidad para el usuario en terreno (UI/UX), tolerancia en la conectividad y mecanismos para garantizar la persistencia de los datos operativos.

Para validar la propuesta se ejecutarán pruebas en ambiente controlado, midiendo métricas como latencia de actualización, precisión de ubicación y tiempo desde el evento hasta la visualización en el centro de control. Las pruebas combinan pruebas automatizadas y evaluaciones en dispositivos reales: se emplea el *framework flutter_test* para *unit* y *widget tests* con clientes HTTP *mockeados* (p. ej. *MockClient*) para simular respuestas del *back-end*, y pruebas de integración manuales en dispositivos físicos (Samsung Galaxy A52, Xiaomi Redmi Note 11) para medir comportamiento en condiciones reales de uso. Los resultados obtenidos confirman el cumplimiento de los requerimientos no funcionales definidos para el sistema. En cuanto a *performance*, la ubicación se obtiene en menos de 2 segundos; el consumo de batería se mantiene por debajo del 10% por hora con el *tracking* activo; la reconexión ante pérdida de conectividad ocurre automáticamente en menos de 5 segundos luego de la recuperación de esta; los tokens de acceso se almacenan de forma cifrada, garantizando la seguridad de información; y la tarea de inicio de *tracking* puede completarse en menos de 3 pasos, validando la usabilidad de la solución. En conjunto, estos resultados evidencian que la arquitectura propuesta cumple con los criterios de aceptación establecidos, adicionalmente se destaca una alta homogeneidad en las pruebas manuales de los dispositivos físicos.

Para evaluar latencia y entrega de mensajes se monitorea *RabbitMQ* con su interfaz de administración (*RabbitMQ Management*) y se registran métricas de publicación/consumo; además se usan herramientas de *profiling* nativas (Android *Profiler* / Xcode Instruments) y logs (logger, Logcat) para medir tiempos desde la generación del *LocationMessage* hasta su procesamiento. La precisión de ubicación se valida comparando lecturas de *geolocator* con coordenadas de referencia y usando emuladores con ubicación simulada.

1.2 Definición del problema.

En la operación de mantenimiento y atención ante emergencias, Chilquinta presenta dificultades para conocer en tiempo real la ubicación exacta de las cuadrillas. Esta falta de visibilidad de ubicación impide la asignación eficiente de recursos y desplazamientos redundantes, provocando mayores costos operativos y malestar en los clientes.

1.3 Breve descripción sobre la propuesta de solución.

Geocuadrilla es una plataforma digital para monitorear, coordinar y optimizar el desplazamiento de cuadrillas en terreno. Consta de una aplicación móvil para los equipos en terreno que captura la información operativa en específico, su georreferenciación y un panel de control web para los operadores y superiores que centraliza visualización en mapas, seguimiento en tiempo real y rutas. El *front-end* de la aplicación móvil entrega una experiencia de captura y envío de datos desde terreno, con énfasis en usabilidad y tolerancia a conectividad intermitente. Respecto al panel web, provee la interfaz principal de supervisión, visualizando mapas, filtros por cuadrilla y herramientas de búsquedas/filtrado.

1.4 Objetivos Generales y Específicos de la Tesina.

Objetivo general: Desarrollar una interfaz *Front-end* móvil mediante *Flutter* que permita la captura y registro de datos operacionales georreferenciados de las cuadrillas de trabajo de Chilquinta, optimizando la recolección de información en terreno a través de una experiencia de usuario intuitiva y funcional.

Objetivos Específicos:

- Diseñar e implementar interfaces de usuario responsivas y adaptables para dispositivos móviles, considerando las necesidades específicas de trabajadores en terreno, teniendo en cuenta las condiciones climáticas.
- Desarrollar componentes *Front-end* para la captura de datos de geolocalización, incluyendo el seguimiento en tiempo real y actualización automática de posición.
- Implementar funcionalidades para la captura automática de los datos temporales (Fecha/Hora) y geográficos, con actualización dinámica de información y sincronización de estados de la aplicación.
- Implementar sistemas de validación de datos y mecanismos de retroalimentación visual mediante *SnackBar*s, alertas y cambios de estado, mejorando la usabilidad y prevención de errores.

1.5 Justificación de proyecto.

La implementación de GeoCuadrilla representa un avance significativo en la gestión operativa, al permitir monitoreo y coordinación en tiempo real de cuadrillas en terreno. Desde el punto de vista técnico, se utiliza *Flutter* para el desarrollo de la aplicación móvil y *Angular* para el panel web, pudiendo así lograr interfaces modernas, responsivas, facilitando la adopción o usabilidad por parte de los usuarios finales. Por otro lado, el uso de la API de Google Maps y RabbitMQ permite visualizar los datos georreferenciados y transmitir los datos operativos en tiempo real.

"Flutter permite a los desarrolladores escribir una vez y desplegar en múltiples plataformas. Esto se logra mediante el uso de widgets personalizables que se adaptan a la plataforma de destino" [1]

Se espera que uno de los principales beneficios sea una mejora significativa en la eficiencia operativa y una reducción de los tiempos de localización y de respuesta, aspecto que deberá confirmarse en una futura validación en condiciones reales. Una experiencia de usuario intuitiva tanto para técnicos en terreno como para los supervisores del centro de control. El front-end móvil desarrollado promueve la reutilización de componentes, facilita la integración de futuras funcionalidades, asegurando la posible escalabilidad del sistema ante un crecimiento operacional.

1.6 Metodología.

Para el desarrollo del proyecto se optó por una metodología ágil específicamente Scrum, debido a su flexibilidad y capacidad para adaptarse a cambios o mejoras frecuentes durante el desarrollo de éste. El equipo se organizó en dos grupos principales, para dividir el desarrollo del panel web y la aplicación móvil. Cada grupo tuvo un desarrollador *Front-end* y un desarrollador *Back-end*. La coordinación y el seguimiento del avance se realizó mediante reuniones semanales distribuidas en dos días a la semana, martes y jueves a través de la plataforma Discord, donde en cada reunión se presentan los progresos, planifican posibles tareas futuras y resuelven dudas técnicas o de integración de servicios.

El proyecto se estructuró en tres *sprints*, permitiendo una entrega incremental de las funcionalidades y una validación continua de los objetivos planteados. La elección de Scrum permitió mantener una comunicación fluida entre los equipos de desarrollo, priorizar tareas según el valor entregado al usuario responder de manera eficiente los desafíos surgidos durante el desarrollo. La metodología facilitó la comunicación, colaboración, transparencia y la adaptación a los requerimientos propios de un entorno de desarrollo ágil. Luego de realizar estas reuniones, se revisaba la validación de los requerimientos y avances por medio de reuniones con el patrocinador.

Los *sprints* son organizados de la siguiente manera:

Durante el Sprint 1 (18-29 de agosto), se consolidó la base funcional del sistema GeoCuadrilla. La desarrolladora web creó el panel inicial con mapa principal, visualización básica de cuadrillas y pestañas de navegación del prototipo. Paralelamente, el equipo móvil implementó *login*, controles de jornada y captura de coordenadas GPS. El integrador habilitó *endpoints* y acceso al servidor privado, culminando en una revisión positiva con el patrocinador.

En el Sprint 2 (primera semana de septiembre), se integraron funciones operativas claves. La desarrolladora web añadió visualización de archivos KMZ para líneas eléctricas, filtros y conexión en tiempo real entre aplicación móvil y panel, permitiendo seguimiento de coordenadas en mapa. El integrador coordinó el servidor y datos, incorporando alertas y mejoras solicitadas por el patrocinador.

El Sprint 3, fase final, se enfocó en la consolidación mediante pruebas en notebooks y móviles reales, simulando escenarios operativos. Se refinó la interfaz web con ajustes visuales y optimizaciones UX, basados en *feedback* del patrocinador, alineando el sistema con necesidades empresariales y cumpliendo plazos de entrega.



1.7 Breve descripción de la organización del informe en capítulos.

Este informe sobre el desarrollo del *front-end* de la aplicación móvil GeoCuadrilla para Chilquinta se organiza en capítulos que siguen una estructura lógica y secuencial, guiando al lector desde el contexto inicial hasta las implicancias prácticas del proyecto. El Capítulo 1, Introducción, presenta el contexto y relevancia del desarrollo, define el problema de monitoreo de cuadrillas en terreno, describe la propuesta de solución basada en *Flutter* y *Angular*, detalla objetivos generales y específicos, justifica la iniciativa y explica la metodología ágil Scrum con *sprints* y reuniones semanales. El Capítulo 2, Marco Teórico, expone fundamentos del desarrollo *front-end* móvil, arquitectura basada en componentes con patrones como *Provider* para gestión de estado, navegación declarativa y diseño responsivo, además de tecnologías clave como *Flutter*, *Dart*, Google Maps API, *RabbitMQ* y librerías de geolocalización. El Capítulo 3, Diseño e Implementación, profundiza en la arquitectura de componentes (UI, servicios y modelos), diseño de interfaz con paleta de colores y *wireframes* iterativos, detalles de codificación, pruebas unitarias, de widgets e integración, resolución de bugs como reconexión automática y optimizaciones de batería, e integración con *backend* vía API REST y mensajería AMQP. Finalmente, los Capítulos 4 a 7 abordan conclusiones con lecciones aprendidas, agradecimientos y referencias bibliográfica facilitando una comprensión integral del proceso de desarrollo y sus resultados.

2 Marco Teórico.

2.1 Fundamentos del desarrollo front-end.

El desarrollo *front-end* móvil se ha consolidado como una disciplina especializada dentro de la Ingeniería de Software. Las aplicaciones móviles han evolucionado hacia sistemas de software más complejos altamente utilizados por los usuarios finales en todo el mundo. Debido al aumento de uso de teléfonos móviles a nivel mundial [2] [3] [4], el desarrollo *front-end* ha sido pieza esencial para la atracción de posibles usuarios finales. [5] [6] Esta evolución ha establecido varios principios arquitectónicos específicos para el desarrollo de software, los cuales se fundamentan en principios derivados de la investigación en arquitectura de software, aplicados posteriormente al desarrollo móvil.

La investigación muestra que las aplicaciones móviles han estado experimentando una transformación significativa desde sistemas monolíticos rígidos a arquitecturas basadas en componentes. Estudios revelan que los principios encontraron amplio uso de estructuración de software de escritorio tradicional o mejor dicho muy similar a aplicación de ordenador, pero el software móvil inicialmente carece de tales estructuras. [7] [8] [9] [10]

2.2 Arquitectura de software.

- Arquitectura basada en componentes.
 - Se implementa una arquitectura basada en componentes, donde cada pantalla o funcionalidad principal es independiente de cada una, ya que esto se desarrolla como un widget independiente. Esto permite que los elementos se conviertan en módulos autónomos, facilitando la reutilización de estos mismos, el mantenimiento y la escalabilidad del sistema. Esta arquitectura es fundamental en el desarrollo con *Flutter* y *front-end* moderno, ya que promueve la separación de responsabilidades.
- Patrón *Provider* (gestión de estado).
 - En la gestión de estado, comunicación con interfaz de usuario (UI) y lógica de negocio, se utiliza este patrón, debido a que permite inyectar servicios de autenticación, localización en cualquier parte de la aplicación, dando como resultado que los datos se reflejen automáticamente en la interfaz de usuario (UI). Por su lado, que la lógica y la presentación estén separadas nos facilita la mantenibilidad del código y permite pruebas unitarias.

"Provider es una solución de gestión de estado para Flutter que ofrece una forma elegante y eficiente de compartir y manejar el estado en tu aplicación. Su diseño basado en el patrón de diseño de proveedores permite una integración fluida con el sistema de widgets de Flutter". [11] [12]

- Patrón de navegación declarativa.
 - Navegar entre las diferentes pantallas de la aplicación se implementa mediante un patrón declarativo, utilizando rutas para llamar a la vista en específico y el widget *Navigator*, lo que ayuda a definir una manera más centralizada sobre los flujos de navegación entre vistas, facilitando la gestión de rutas, la protección de accesos, al desarrollo de nuevas vistas o funcionalidades sin dar de baja la aplicación o no dañando las vistas que ya están creadas. La extensión futura del sistema con nuevas vistas o funcionalidades, no se verían afectadas con las ya creadas en versiones anteriores de la aplicación.
- Diseño responsivo y adaptativo.
 - El diseño de la interfaz de usuario (UI) considera los principios de la responsividad y adaptabilidad, utilizando widgets, para la adaptación en distintas pantallas de dispositivos móviles sin importar las dimensiones de estas, para que la aplicación se visualice correctamente. Esta práctica es esencial para garantizar la accesibilidad y usabilidad en contextos móviles, teniendo en cuenta la diversidad de estos mismos.

2.3 Tecnologías y frameworks utilizados.

La capa *front-end* de la aplicación fue implementada íntegramente con Flutter y el lenguaje Dart, por lo que no se emplearon *frameworks* web como React o Angular; Flutter actúa tanto como *framework* UI como *runtime*, permitiendo una única base de código para Android e iOS. El manejo del estado se organiza con la librería *provider* (*ChangeNotifier* y *ChangeNotifierProxyProvider*) y los servicios principales —*AuthService*, *RabbitMqService* y *LocationService*— se registran en *MultiProvider* en *main.dart* para inyección de dependencias y notificación de cambios. La comunicación con APIs se realiza mediante el paquete *http* para REST y *dart_amqp* para publicar mensajes en RabbitMQ; las constantes y endpoints se centralizan en *EnvironmentConfig*. Para autenticación se integra *msal_auth* (Microsoft OAuth) [13] y se protege la persistencia de credenciales con *flutter_secure_storage*. La geolocalización se obtiene con *geolocator* y la visualización con *google_maps_flutter*, mientras que *flutter_background* permite mantener el tracking en segundo plano. Las pruebas se apoyan en *flutter_test* con uso de *mocks*, y las optimizaciones incluyen reducir la frecuencia de muestreo GPS y minimizar *rebuidls* con *Consumer*. Adicionalmente, se aplican medidas de seguridad en el *front-end* móvil: almacenamiento cifrado de tokens, uso de HTTPS y manejo controlado de permisos, quedando como mejoras recomendadas el *certificate pinning* y validaciones adicionales.

3 Diseño e Implementación.

3.1 Diseño de Componentes.

3.1.1 Arquitectura y estructura de los componentes.

La aplicación Geocuadrilla está diseñada con una arquitectura basada en “*Provider*” o como su traducción la define “*Proveedor*”, esta arquitectura permite gestionar los cambios de estado de los widgets utilizados sin tener la necesidad de realizar el cambio de forma manual, esto se logra de forma automática debido a que está constantemente “*escuchando*” los cambios de estados de servicios e interacciones del usuario. Los principios utilizados en el desarrollo de la aplicación serían la separación de responsabilidades y la arquitectura limpia (*Clean Architecture*) vista en la figura 1.

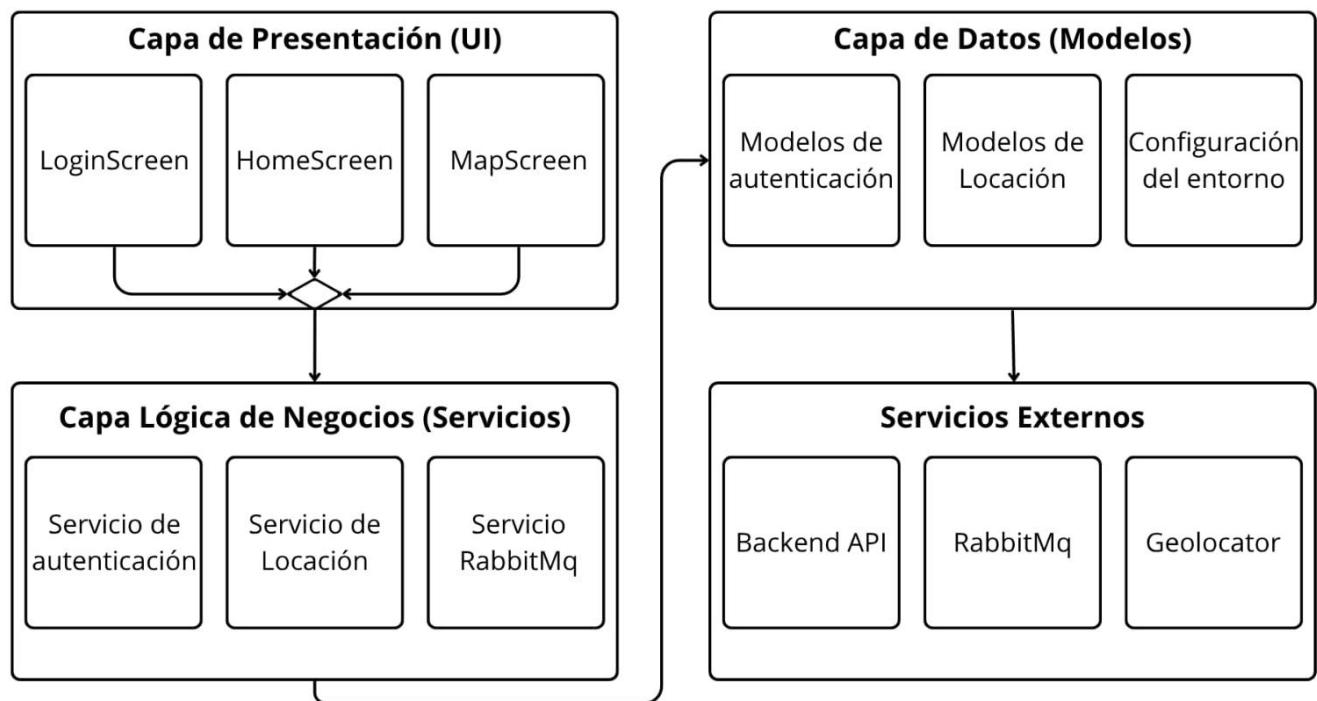


Figura 1. Arquitectura y estructura de los componentes.

Fuente: Elaboración propia.

Componentes principales.

1. Capa de Presentación (UI).

- *LoginScreen*: Interfaz de usuario que su principal función es la autenticación con Microsoft OAuth.
- *HomeScreen*: *Dashboard* principal con información que el usuario requiere, información del usuario, estado del tracking y controles de manejo de esta.
- *MapScreen*: Visualización del mapa con el uso de la Api de Google Maps, donde se muestra la ubicación del usuario junto con el tráfico en tiempo real.

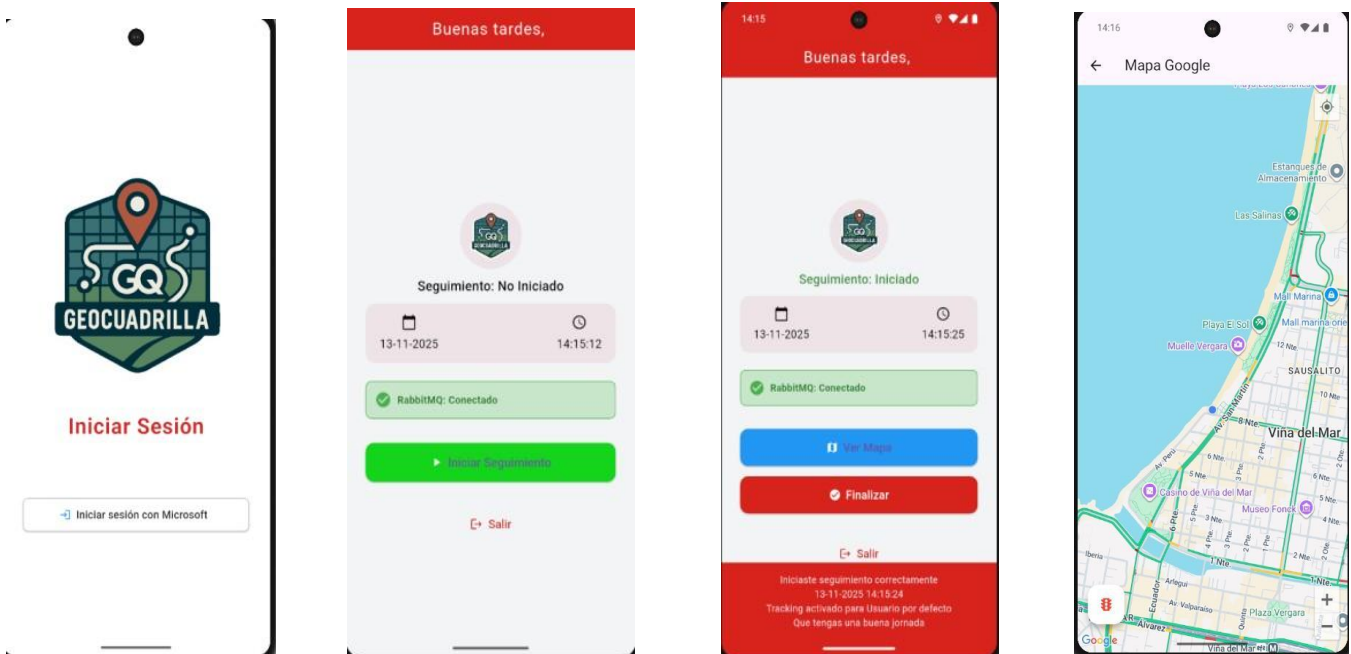


Figura 2. Capturas de pantalla de la aplicación(LoginScreen, HomeScreen, MapScreen).

Fuente: Elaboración propia.

2. Capa lógica de negocios (Servicios).

Servicio de autenticación:

- Responsable de la autenticación del usuario mediante Microsoft (MSAL).
- Responsable de la gestión de tokens (*access token* y *refresh token*).
- Responsable de la persistencia segura de las credenciales mediante el uso de Flutter Secure Storage.
- Responsable de la notificación de cambios en el estado de autenticación del usuario.

Servicio de Ubicación:

- Responsable de la obtención de ubicación GPS del dispositivo móvil.
- Responsable del control del tracking (inicio/finalización).
- Responsable del envío periódico de ubicaciones con un intervalo de 5 segundos.
- Responsable de la gestión de permisos de ubicación del dispositivo.
- Responsable de la integración del servicio en segundo plano (*Flutter Background*).

Servicio RabbitMq:

- Responsable de la conexión con RabbitMq mediante el protocolo AMQP.
- Responsable de la publicación de mensajes de ubicación en colas.
- Responsable de la reconexión automática en caso de fallo o desconexión.
- Responsable del monitoreo del estado de conexión.

3. Capa de Datos (Modelos).

En la capa de datos de la aplicación se definen y utilizan tres modelos fundamentales que estructuran la información y garantizan la comunicación correcta entre la interfaz, la lógica de negocio y los servicios externos. El primer modelo para destacar es el de autenticación donde se agrupan estructuras como *“LoginRequest”*, que contiene las credenciales que envió el *Back-end* y *“AuthResult”* junto a *“UserInfo”*, que representan la respuesta del servidor con el token, *“refreshToken”*, *“expiresAt”* y los datos del usuario; estos objetos permiten almacenar de forma segura las credenciales en *“Flutter Secure Storage”* y gestionar el estado de sesión desde el servicio de autenticación. El modelo de ubicación está representado por el paquete, que se compone por un identificador de la patrulla en terreno (*crewId*), latitud junto con la longitud donde se ubica el usuario y el correo institucional que se utilizó para el inicio de sesión en la aplicación, se usa este modelo para serializar las lecturas del GPS en JSON y publicarlas periódicamente en la cola mediante RabbitMQ. Finalmente, la configuración del entorno centraliza todas las constantes de configuración (*endpoints*, parámetros de conexión de RabbitMQ, nombres de colas y *timeouts*), evitando errores y facilitando cambios entre entornos. Además, los modelos implementan métodos de serialización por ejemplo *“toJson”* o *“fromJson”* que utilizan formatos estandarizados para evitar problemas de zona horaria y simplificar el procesamiento en el *back-end*. Esta organización mejora la trazabilidad, prueba unitaria y la mantenibilidad del proyecto.

3.1.2 Uso de buenas prácticas y patrones de diseño.

En este apartado se describen las buenas prácticas y patrones de diseño que se utilizan para estructurar la aplicación. El objetivo fue mantener una separación clara entre la lógica de negocio y la interfaz de usuario, facilitando el mantenimiento, las pruebas y la escalabilidad del proyecto.

En primer lugar, se emplea el patrón “*Provider*” para la gestión del estado global de la aplicación. Se configura un “*MultiProvider*” donde se registran servicios como la autenticación, ubicación y la comunicación con RabbitMQ, de manera que varios widgets puedan escuchar los cambios en estos servicios sin acoplarse directamente a su implementación. Con esto se busca que la UI reaccione de forma eficiente a las actualizaciones de datos y que el código de presentación se mantenga limpio (Listing 1).

```
MultiProvider(  
  providers: [  
    ChangeNotifierProvider(                                create:  
      (_) => AuthService()..loadSavedAuth(),  
    ),  
    ChangeNotifierProvider(                                create:  
      (_) { final rabbitMqService =  
RabbitMqService();  
rabbitMqService.connect();                                return  
rabbitMqService;  
      },  
    ),  
    ChangeNotifierProxyProvider<RabbitMqService,  
LocationService>(                                create: (context) =>  
  
LocationService(Provider.of<RabbitMqService>(context,  
listen: false)),                                update: (context, rabbitmq,  
location) =>                                location ??  
LocationService(rabbitmq),  
    ),  
  ],  
  child: const MyApp(),  
)
```

Listing 1. Ejemplo de uso de patrón Provider.

Fuente: Elaboración propia.

Además, se aplica la inyección de dependencias en los servicios, suministrando sus dependencias a través del constructor. Por ejemplo, el servicio de ubicación recibe una instancia de RabbitMQ, lo que permite probar cada componente de forma aislada y reducir el acoplamiento entre los módulos. Al combinar esta técnica con “*ChangeNotifier*” se logra que los componentes de servicios notifiquen a los widgets cuando cambian el estado, manteniendo sincronizada la interfaz con la lógica interna de la aplicación(Listing 2).

```
class LocationService extends ChangeNotifier {  
  final RabbitMqService _rabbitMq;  
  ...  
  LocationService(this._rabbitMq); // Inyección de dependencia  
}
```

Listing 2. Implementación de la clase LocationService con inyección de dependencias.

Fuente: Elaboración propia

Finalmente, se estructuran algunos servicios como si fueran “*singletons*” dentro del árbol de “*providers*”, instanciándolos solo una vez y compartiéndolos en toda la aplicación. Gracias a esto se mantiene un estado consistente entre distintas pantallas, evitando duplicar recursos y centralizando la gestión de funcionalidades críticas, como la sesión de usuario o la comunicación con los servicios externos.

Como buenas prácticas aplicadas se puede destacar el código limpio (*Clean Code*), donde en diferentes partes del código se ven reflejadas, por ejemplo, nombre de funciones descriptivos, funciones pequeñas y enfocadas, y manejo de errores robustos. Se utilizan nombres descriptivos en las funciones para que su responsabilidad sea evidente a simple vista y diseñen métodos pequeños y enfocados, donde cada uno se encarga de una tarea específica, como verificar permisos, iniciar el seguimiento o enviar datos. A esto se suma el manejo robusto de errores: encapsulando las operaciones críticas en bloques, registrando mensajes claros al capturar excepciones y realizando retorno de valores controlados para evitar fallos inesperados.

3.1.3 Integración con el sistema general.

En este apartado se presenta el proceso de integración del *front-end* de la aplicación móvil con el ecosistema de Geocuadrilla, combinando comunicación mediante API Rest y mensajería AMQP. Desde la interfaz, se gestiona el inicio de sesión con Microsoft. Enviando credenciales o tokens al *back-end* y recibiendo las respuestas en formato “JSON” que incluyen el *Json web token (JWT)*, datos del usuario y tiempos de expiración. A nivel del código se encapsula las llamadas HTTP en una función genérica que envía peticiones “POST” con cabeceras estándar y el *bearer token* correspondiente. Para la mensajería en tiempo real se utiliza RabbitMQ, definiendo la configuración del cliente AMQP, las colas de ubicación y estado y el formato JSON de los mensajes que envía la aplicación [14]. Además, se implementa la obtención de ubicación mediante “*Geolocator*” y el servicio en segundo plano con “*FlutterBackground*”, de forma que el dispositivo envía periódicamente su posición aún inclusive con la aplicación minimizada. Finalmente, se detalla el flujo completo de integración que el usuario inicia sesión hasta que el sistema procesa y almacena ubicaciones, incluyendo la lógica de reconexión automática, el manejo de tokens y envío de mensajes a la cola (Figura 3 y 4).

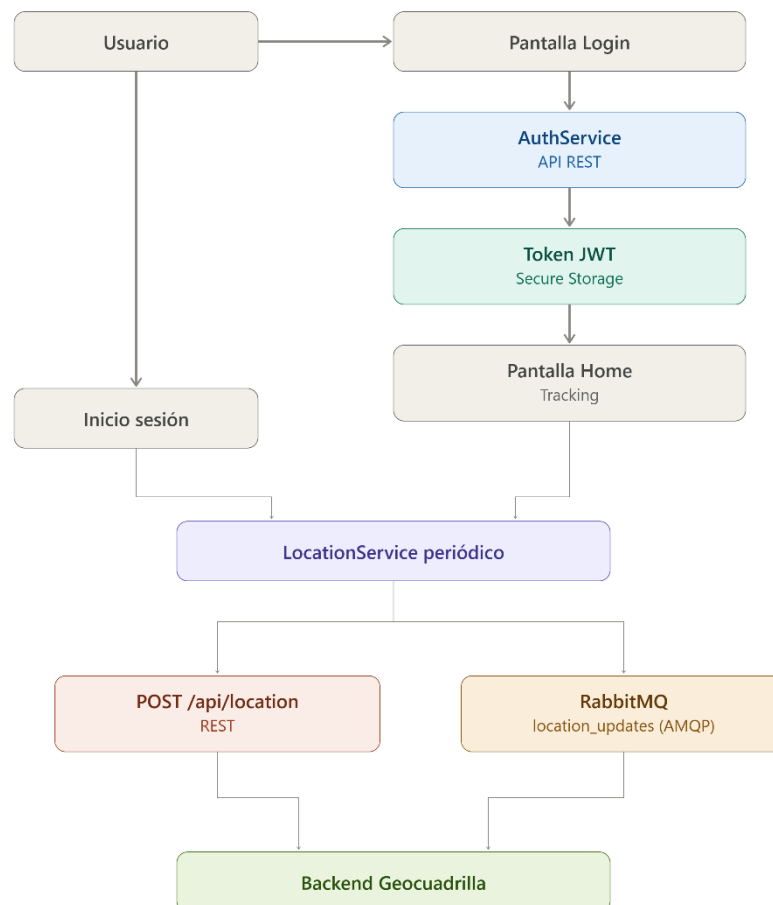


Figura 3. Diagrama general de flujo de la aplicación.

Fuente: Elaboración propia.

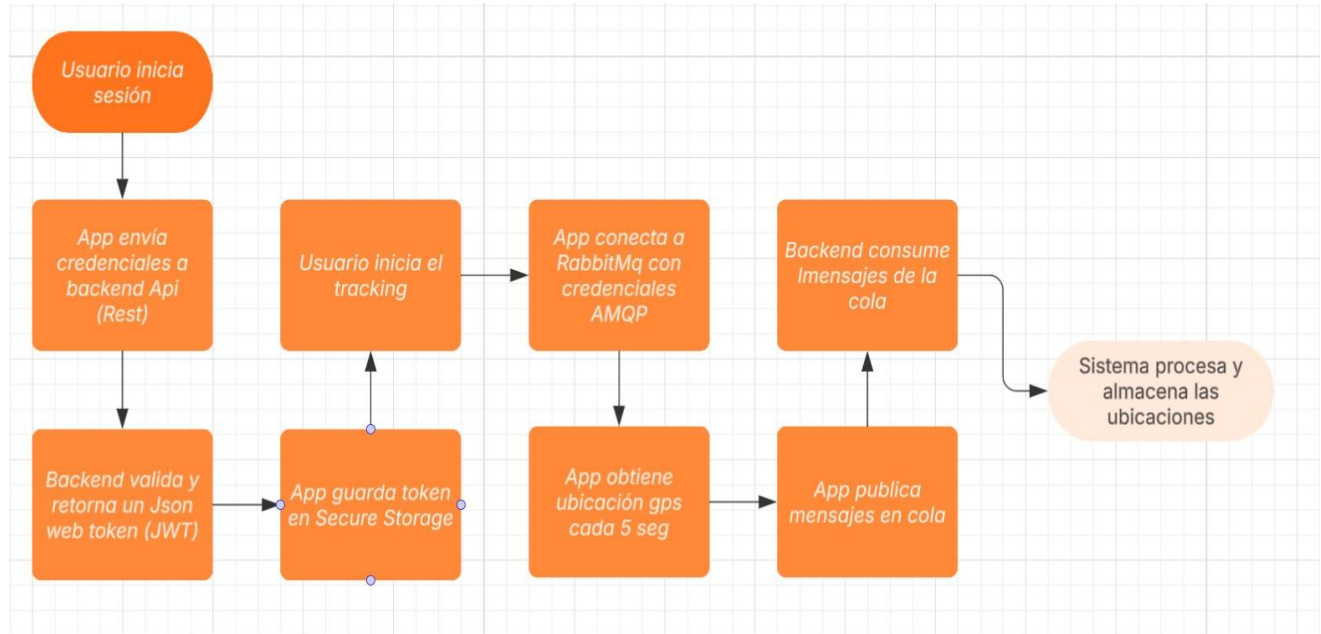


Figura 4. Flujo de autenticación y seguimiento de ubicación en la aplicación.

Fuente: Elaboración propia.

Backend API (Rest).

En primer lugar, desde el *front-end* se implementa la integración con el *backend* mediante una API REST sobre HTTP/HTTPS. Para la autenticación de usuarios se definen dos *endpoints* principales: un *login* tradicional (/api/Auth/login) que recibe credenciales de usuario y contraseña, y un *login* federado con Microsoft (/api/Auth/login/microsoft) que recibe el identificador de token emitido por el proveedor externo. En ambos casos, el *back-end* responde en formato JSON, retornando un indicador de éxito, el token JWT, el *refresh* token y los datos básicos del usuario, los cuales son consumidos por la aplicación móvil para inicializar la sesión.

Para estandarizar las llamadas desde la capa de presentación, se encapsula la lógica HTTP en un método “_postJson”, que recibe la URL y el cuerpo de la petición, serializa el contenido a JSON y agrega las cabeceras necesarias, de este modo, el *front-end* puede invocar distintos servicios REST reutilizando la misma función, manteniendo un formato de intercambio consistente y simplificando el mantenimiento del código.

RabbitMQ.

RabbitMQ permite complementar la integración REST con un mecanismo de mensajería asíncrona pensado para alto volumen y desacoplamiento entre componentes. Desde el *front-end* se implementa un cliente AMQP que se conecta a RabbitMQ usando la configuración de host, puerto, protocolo y credenciales de la aplicación, y trabaja principalmente con la cola *location_updates*, donde se realiza la publicación de los mensajes de ubicación generados por el dispositivo. Cada mensaje sigue un formato JSON con identificador de cuadrilla, marca de tiempo, latitud, longitud, datos del técnico y metadatos de precisión. El método *sendLocation* serializa este objeto y lo publica de forma asíncrona, permitiendo que el *back-end* procese las ubicaciones sin bloquear la experiencia de usuario en la aplicación (Listing 3).

```
{
  "crewId": "string",
  "timestamp": "2025-12-04T15:30:45.123Z",
  "latitude": -33.4489,
  "longitude": -70.6693,
  "emailTecnico": "tecnico@correo.cl",
  "metadata": {
    "accuracy": 5.0,
    "initial": false
  },
  "source": "flutter_app"
}
```

Listing 3. Ejemplo de formato de mensaje JSON enviado de la aplicación.

Fuente: Elaboración propia

3.2 Diseño de la Interfaz

Paleta de colores.

- Principal (Brand): #D8231F
- Fondo App: #F3F4F6
- Fondo *Cards*: #FFFFFF
- Fondo Secundario: #EEE3E6
- Texto Principal: #000000
- Texto Secundario: #666666

Estructura y diseño de la interfaz de usuario (UI).

La pantalla de Login se centra en ofrecer un acceso claro y directo al sistema. Se ubica el logo de Geocuadrilla centrado, seguido de un botón de acción para el *login* con Microsoft. Además, se incluyen mensajes de error y validación bajo los campos para que el usuario comprenda de inmediato si cometió algún error al autenticarse (Figura 5).



Figura 5. Captura de pantalla de inicio de sesión de la app.

Fuente: Elaboración propia.

La **HomeScreen** funciona como panel principal de control. En la parte superior se muestra un saludo personalizado con el nombre del usuario y datos en tiempo real como fecha y hora. Al centro se encuentran el identificador de cuadrilla y los botones para comenzar o detener el tracking, junto con indicadores de estado que reflejan la conexión con RabbitMQ y el proceso de tracking. También se agregan botones de navegación para abrir el mapa y cerrar sesión, entregando *feedback* inmediato en los botones según la acción ejecutada (Figura 6).

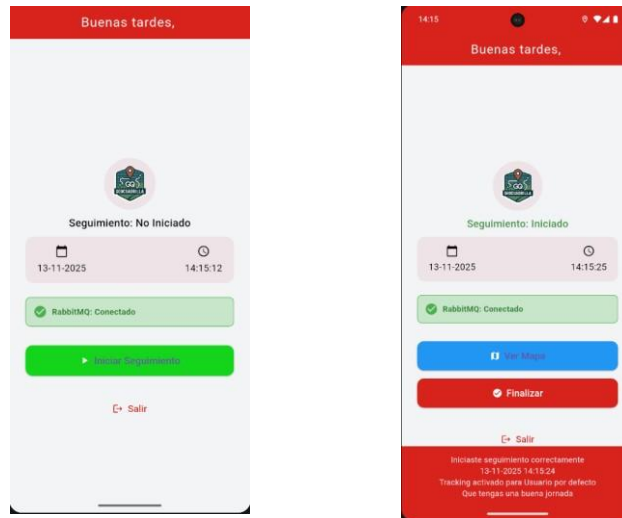


Figura 6. Captura de pantalla de página principal iniciando el tracking.
Fuente: Elaboración propia.

Por último, la **MapScreen** está enfocada en la visualización geográfica. Integra Google Maps para mostrar la posición actual del usuario mediante un marcador, actualizando la ubicación en tiempo real. La interfaz permite realizar zoom y desplazamiento sobre el mapa, manteniendo siempre centrada la posición del técnico cuando es necesario. El botón “Volver” simplifica el retorno a la pantalla principal, evitando distracciones y manteniendo la prioridad en la lectura rápida de la ubicación y de las coordenadas mostradas (Figura 7).

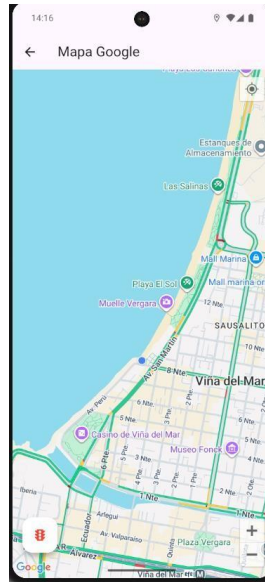


Figura 7. Captura de pantalla del funcionamiento del mapa.

Fuente: Elaboración propia.

Proceso de creación de wireframes y prototipos.

El proceso de diseño de la interfaz se realiza directamente en código *Flutter*, creando *wireframes* funcionales que se van refinando iterativamente. Se inicia con versiones muy simples de cada pantalla, centradas en la disposición de elementos clave (logo, formularios, botones y mapa), y luego se incorporan detalles visuales como colores, iconos y espaciados. Estas iteraciones se basan en *feedback* de técnicos de campo, quienes probaron distintas variantes y ayudaron a priorizar claridad por sobre elementos decorativos. A partir de sus comentarios se definen principios de diseño concretos: simplicidad en la estructura, accesibilidad mediante botones grandes y textos legibles, y *feedback* visual inmediato de los estados de conexión y tracking (Figura 8).



Figura 8. Prototipo inicial de la aplicación.

Fuente: Elaboración propia.

Responsividad y adaptabilidad a diferentes dispositivos.

Para asegurar que la aplicación se adapte correctamente a diferentes tamaños de pantalla, se implementan varias técnicas de diseño responsivo en Flutter. Se utilizó “*SingleChildScrollView*” en pantallas con formularios para permitir el desplazamiento en dispositivos pequeños y evitar recortes cuando aparece el teclado. El espaciado se maneja con “*Padding*” dinámico y “*SizedBox*” basados en porcentajes, de modo que los márgenes y separaciones escalen proporcionalmente. Cuando es necesario se recurre a “*MediaQuery*” para obtener las dimensiones del dispositivo y ajustar el “*layout*”. De esta forma, los componentes mantienen una jerarquía visual consistente en teléfonos con distintas resoluciones y orientaciones (Listing 4).

```
Container(  
  width: 88,  
  height: 88,  
  decoration: BoxDecoration(  
    shape: BoxShape.circle,  
    color: Color(0xFFEEE3E6),  
  ),  
)
```

Listing 4. Ejemplo de implementación de un contenedor circular en Flutter.

Fuente: Elaboración propia

Consideraciones de accesibilidad y experiencia de usuario (UX).

En cuanto a accesibilidad, se cuida que el contraste entre texto y fondo cumpliera recomendaciones como WCAG AA, empleando tamaños de fuente de al menos 16sp para el contenido principal y botones con área táctil amplia (mínimo 48x48dp). La aplicación ofrece mensajes de error descriptivos cerca de los campos correspondientes y estados visuales claros para conexión, autenticación y tracking. Desde la perspectiva de UX, se busca que la navegación fuera intuitiva mediante rutas bien nombradas y flujos simples: *login*, pantalla principal y mapa. Todas las acciones clave muestran *feedback* inmediato (*loaders*, mensajes de éxito o error) y se solicitan confirmaciones antes de operaciones críticas como cerrar sesión o detener el tracking.



3.3 Detalles de la codificación y desarrollo.

El proyecto GeoCuadrilla gestionó su backlog mediante metodología Scrum con tres *sprints* bien definidos, priorizando tareas incrementales en desarrollo *front-end* móvil (Flutter), panel web (Angular) e integración *back-end* para entregar valor progresivo al patrocinador de Chilquinta.

Backlog por Sprint:

Sprint 1 (18-29 agosto): Consolidación de base funcional con panel web inicial (mapa principal, visualización básica de cuadrillas, pestañas de navegación), app móvil (*login* Microsoft OAuth, controles de jornada, captura GPS) y habilitación de *endpoints*/servidor privado por el integrador. Revisión positiva con patrocinador.

Sprint 2 (primera semana septiembre): Integración operativa clave incluyendo visualización KMZ de líneas eléctricas con filtros en panel web, conexión tiempo real app-panel vía RabbitMQ para seguimiento de coordenadas, alertas y mejoras por *feedback* del patrocinador. Coordinación de servidor/datos.

Sprint 3 (fase final): Consolidación con pruebas en notebooks/móviles reales simulando escenarios operativos, refinamiento de interfaz web (ajustes visuales, optimizaciones UX) basado en retroalimentación operativa, alineando con necesidades empresariales y plazos de entrega.

Control de Versiones y Despliegues.

Se utilizó Git en GitHub para control de versiones con *branches* por sprint (ej. *feature/sprint1*) y *pull requests* para revisiones colaborativas entre equipos *front-end/back-end*. No se implementaron CI/CD formales por el contexto académico, optando por despliegues manuales y pruebas locales en Discord; se recomienda GitHub *Actions* para automatización futura.

3.4 Pruebas y Validación

Para verificar que el *front-end* cumpliera con los requerimientos definidos, se diseñó una estrategia de validación que abarca tanto requisitos funcionales como no funcionales. En primer lugar, cada requisito funcional se asoció a un método de prueba específico: el *login* de usuario se validó mediante pruebas manuales complementadas con pruebas unitarias de validación de formularios, a su vez se implementaron reglas para que el usuario no pueda ir en un flujo incorrecto, esto se observa mediante alertas tanto al momento de iniciar el *tracking* o cuando se realiza un “mal inicio” del mismo, en este último caso el usuario es alertado mediante cambios de estado y color visuales de la *screen* y notificaciones enviadas al *Snackbar*; el inicio de sesión con Microsoft se revisó mediante pruebas de integración con la librería MSAL, en este caso las alertas se presentan de igual manera que en el mal inicio de tracking; el tracking de ubicación y la visualización en mapa se comprobaron en dispositivos reales, incluyendo escenarios con la aplicación en segundo plano, monitoreo del servicio en RabbitMQ Management y pruebas visuales en distintos equipos.

En segundo lugar, se evaluaron los requerimientos no funcionales utilizando criterios de aceptación medibles. Se definieron los siguientes umbrales: la ubicación debía obtenerse en menos de dos segundos; el consumo de batería no debía superar el 10% por hora durante el *tracking*; la reconexión automática al *bróker* debía completarse en menos de cinco segundos; los *tokens* debían almacenarse en un medio cifrado; la tarea de iniciar el *tracking* debía completarse en menos de tres pasos desde la pantalla principal.

Los resultados obtenidos confirmaron el cumplimiento de todos los criterios definidos. La ubicación promedió 1.8 segundos dependiendo del sector donde se realizaba la prueba, cumpliendo el umbral establecido. El consumo de batería se mantuvo por debajo del 10% por hora durante el *tracking*, y la reconexión automática al bróker se completó dentro del tiempo límite de cinco segundos. Asimismo, los tokens fueron almacenados correctamente en un medio cifrado, y la tarea de iniciar el *tracking* pudo completarse en menos de tres pasos desde la pantalla principal.

3.4.1 Estrategias de testing aplicadas a los componentes.

Pruebas Unitarias.

Como primera instancia se implementaron pruebas unitarias sobre los servicios del *front-end*, con el objetivo de validar su comportamiento de forma aislada y sin depender de recursos externos. En particular, se probó el servicio de autenticación utilizando un cliente HTTP simulado, verificando que ante credenciales válidas el servicio retorne un resultado exitoso y almacene correctamente el token recibido. La prueba se ejecutó de manera satisfactoria, cumpliendo con el comportamiento esperado en todos los casos evaluados. Este enfoque permite asegurar que la lógica de autenticación, gestión de tokens y persistencia funcione según lo esperado, incluso antes de integrar el módulo con la API real. Además, las pruebas unitarias facilitan detectar regresiones cuando se realizan cambios en el código, aportando una base de confianza para la evolución futura de la aplicación (Listing 5).

```
test('login devuelve true con credenciales validas', () async {  
  final mockClient = MockClient();  
  final authService = AuthService(client: mockClient);  
  
  when(mockClient.post(any, body: any, headers: any))  
    .thenAnswer((_) async => http.Response(  
      jsonEncode({'success': true, 'token': 'fake_token'}),  
      200,  
    ));  
  
  final result = await authService.login('user@test.com', 'password123');  
  expect(result, true);  
  expect(authService.accessToken, 'fake_token');  
});
```

Listing 5. Prueba unitaria del método login usando MockClient en Dart.
Fuente: Elaboración propia

Widget testing.

Como complemento a las pruebas unitarias, se realizaron pruebas de widgets enfocadas en verificar el correcto renderizado de la interfaz. En particular, se probó la pantalla de “Login” de forma aislada utilizando el “*WidgetTester*”, comprobando que se muestren los dos campos de texto correspondientes a email y contraseña, así como el botón “Iniciar con Microsoft”. Este tipo de prueba permite asegurar que la estructura visual mínima de la pantalla esté siempre presente, incluso después de refactorizaciones o cambios en el diseño. Además, contribuye a detectar errores tempranos en la configuración de widgets, rutas o textos, garantizando una experiencia de usuario consistente.

Integración testing.

Para complementar las pruebas unitarias y de widgets, se implementaron pruebas de integración que validan el flujo completo de la aplicación desde el inicio de sesión hasta el inicio del tracking. Estas pruebas automatizadas simulan la interacción de un usuario real: ingresan credenciales en la pantalla de Login, presionan el botón “Iniciar Sesión”, verifican la navegación correcta hacia la *HomeScreen* y comprueban la presencia del saludo personalizado. Luego continúan con el flujo ingresando el ID de cuadrilla, activando el botón “Comenzar Tracking” y validando que la interfaz muestre el estado “Detener Tracking”.

Todas las pruebas de integración implementadas se ejecutaron de manera exitosa, confirmando que el flujo completo de la aplicación desde la autenticación hasta el inicio del *tracking* funciona correctamente de extremo a extremo. En particular, se verificó que la navegación entre pantallas ocurra según lo esperado, que tanto la información del usuario como la de la cuadrilla se logre desplegar correctamente tras el inicio de sesión, y que los cambios de estado de la interfaz “Comenzar Tracking” y “Detener Tracking” reflejen fielmente las transiciones del flujo de tracking. Estos resultados permiten concluir que los distintos módulos de *front-end* no solo funcionan correctamente de forma aislada, sino que también se integran de manera coherente entre sí, lo que reduce el riesgo de fallos al momento de la interacción real del usuario con la aplicación.

3.4.2 Resolución de bugs y optimización.

Durante el desarrollo del *front-end* se realizó un ciclo sistemático de resolución de bugs y optimización con el objetivo de mejorar la estabilidad de la aplicación y la experiencia de los técnicos en terreno. A partir de las pruebas y del uso en dispositivos reales, se identificaron problemas críticos relacionados con el tracking en segundo plano, la reconexión con RabbitMQ y la gestión de tokens, junto con oportunidades de optimización en consumo de batería y rendimiento de la interfaz.

En primer lugar, se abordaron los bugs de funcionamiento. El problema más crítico era que el servicio de ubicación se detenía al bloquear la pantalla, lo que se solucionó configurando correctamente la ejecución en segundo plano mediante *FlutterBackground* y una notificación persistente. Luego se detectó que, ante una caída de RabbitMQ, la aplicación no intentaba reconectarse; para ello se implementó un mecanismo de reconexión automática con reintentos diferidos, evitando que el usuario perdiera el tracking por fallas transitorias de red. Finalmente, se identificó que los tokens expirados no se renovaban de forma automática, caso que quedó documentado como trabajo pendiente para una futura mejora del flujo de *refresh*.

En paralelo se aplicaron optimizaciones específicas. Para reducir el consumo de batería se incrementó el intervalo de obtención de ubicación de 1 a 5 segundos y se ajustó la precisión a un nivel medio cuando era posible, logrando una disminución importante en la cantidad de consultas GPS. Además, se implementó un mecanismo de caché local de ubicaciones para escenarios sin conexión, de modo que la aplicación almacene los puntos offline y los envíe en lote cuando RabbitMQ vuelva a estar disponible. Por último, se optimizaron los *rebuid*s de la UI utilizando *Consumer* en lugar de *context.watch* general, lo que redujo de forma significativa la cantidad de reconstrucciones innecesarias y mejoró la fluidez de la interfaz.

3.5 Integración con Otros Componentes.

La comunicación entre el *front-end* y los componentes externos se realiza mediante dos mecanismos complementarios: una API REST para operaciones síncronas y RabbitMQ (AMQP) [14] para telemetría asíncrona. Para la autenticación y consultas directas la aplicación usa el paquete *http* y los *endpoints* definidos en *EnvironmentConfig* (por ejemplo, *authEndpoint* y *microsoftAuthEndpoint*), intercambiando JSON y gestionando tokens JWT que *AuthService* persiste en *flutter_secure_storage* y adjunta en el *header Authorization: Bearer <token>* en solicitudes subsecuentes; se aplican *timeouts*, manejo de códigos HTTP y retroalimentación al usuario ante errores. Para el envío de ubicaciones, *LocationService* construye instancias de *LocationMessage* (con *crewId*, *timestamp* ISO8601 UTC, lat/long y metadata) y *RabbitMqService* las publica en la cola *location_updates* usando *dart_amqp*. Esta arquitectura con RabbitMQ ofrece tolerancia a fallos y desacoplamiento: los mensajes quedan en cola si el *back-end* o la red fallan y se procesan cuando sea posible. [15] El servicio implementa reconexión automática y exposición del estado (*isConnected*) al UI; se recomienda usar colas durables, *confirms/acks* y una *dead-letterqueue* para manejar mensajes problemáticos, además de validar idempotencia y considerar caché local para reenvío. En producción, ambas vías deben operar sobre HTTPS/TLS y contemplar *certificate pinning* y monitoreo para garantizar seguridad y observabilidad.

4 Conclusiones, logros, limitaciones y trabajo futuro.

4.1 Conclusiones.

A lo largo del desarrollo del proyecto, el equipo comprobó que la participación activa del cliente es clave para tomar decisiones acertadas y priorizar funcionalidades con un verdadero impacto. El trabajo conjunto permitió ajustar el alcance y validar tempranamente los requisitos del sistema GeoCuadrilla, orientado a apoyar la gestión operativa de cuadrillas en terreno para Chilquinta Transmisión mediante la integración de información de ubicación, estado operativo y contexto geográfico en una plataforma centralizada. Este proceso también evidencio la importancia de documentar sistemáticamente los cambios de software y las reuniones, para contar con un historial claro de acuerdos y modificaciones que respaldara las decisiones tomadas durante el desarrollo.

Desde el punto de vista del desarrollo móvil, la aplicación se diseñó con un enfoque centrado en la usabilidad, permitiendo una interacción intuitiva con las funcionalidades principales del sistema. La integración con los servicios *back-end* posibilitó la consulta y actualización de información en tiempo real mediante *SignalR* y solicitudes HTTP, además de la visualización de datos geoespaciales a través de *Google Maps*, contribuyendo a mejorar la disponibilidad de información para los usuarios durante sus labores operativas.

Entre los desafíos más relevantes estuvieron la coordinación de reuniones y la distribución equilibrada de tareas dentro del equipo. Pese a ello, se considera un logro significativo haber incorporado tecnologías no vistas en la carrera, apoyándose en las bases conceptuales adquiridas para aprenderlas de forma autónoma. Esto permitió construir una arquitectura segmentada, donde cada componente cumple una función específica, servicios *back-end* y un panel web de supervisión se comunica mediante servicios especializados, solicitudes HTTP y mecanismos de actualización en tiempo real, logrando integrar información operativa y geoespacial en una misma interfaz.

Aunque aún no es posible medir con exactitud el impacto del sistema en la sociedad, se proyecta que su uso contribuirá a mejorar la calidad del servicio eléctrico y en consecuencia, la vida diaria de los usuarios finales, al centralizar información que antes podía encontrarse dispersa o requerir consultas manuales. La validación realizada permitió comprobar el funcionamiento de los principales flujo del sistema en un ambiente controlado, aunque con un enfoque principalmente funcional y cualitativo, por lo que las mejoras concretas en tiempos de respuesta o eficiencia operacional quedan como aspectos a confirmar en una futura implementación en entorno productivo real.

4.2 Logros.

Uno de los principales logros de este proyecto fue el desarrollo de una aplicación móvil funcional, integrada a un ecosistema de software más amplio. Mediante una arquitectura basada en Flutter y Dart, se logró conectar la captura de ubicación en tiempo real con el panel web, estableciendo comunicación con servicios externos a través de una API REST para la autenticación y de RabbitMQ para el envío de paquetes de ubicación. Esta integración permitió que los datos generados desde el dispositivo llegaran de forma ordenada y confiable al sistema central.

Asimismo, se implementó el envío periódico de paquetes cada 5 segundos, lo que garantiza un rastreo continuo y una frecuencia de transmisión adecuada para el monitoreo operativo. Para ello, se incorporaron librerías claves como *Geolocator*, encargada de obtener la ubicación del dispositivo; *Flutter_background*, que permite mantener el seguimiento activo mientras la aplicación se ejecuta en segundo plano; *Provider*, utilizada para la gestión del estado de la aplicación; *Flutter_secure_storage*, que protege los *tokens* de sesión y *Msal_auth*, que habilita el inicio de sesión con Microsoft mediante correos institucionales.

El flujo de autenticación funciona de manera segura y confiable. La aplicación cuenta con mecanismos de reconexión que aseguran la continuidad de la operación ante pérdidas temporales de conectividad. En conjunto, estos avances demuestran que la solución no solo cumple con su funcionalidad principal, sino que también incorpora tecnologías modernas, escalables y orientadas al uso real en campo.

4.3 Limitaciones.

La dependencia de la conectividad constituye una de las principales restricciones del sistema desarrollado. Funcionalidades como la autenticación mediante *Microsoft* (MSAL), la transmisión de telemetría hacia *RabbitMQ*, la carga de mapas y la sincronización de información requieren disponibilidad de conexión para operar correctamente. Si bien la aplicación mantiene el seguimiento de la ubicación mediante servicios ejecutados en segundo plano, no se implementó una estrategia integral de funcionamiento *offline* que permitiera almacenar y sincronizar automáticamente la información generada durante períodos prolongados sin cobertura. En consecuencia, el desempeño de la aplicación podría verse afectado en sectores donde la conectividad sea limitada o intermitente.

Otra limitación corresponde a la ausencia de pruebas de carga y escalabilidad. Durante el desarrollo no se evaluó el comportamiento de la aplicación frente a un número elevado de usuarios concurrentes, múltiples cuadrillas transmitiendo información simultáneamente o un incremento significativo en el volumen de telemetría enviado al servidor. Asimismo, la solución depende parcialmente de servicios externos, como *Google Maps*, *RabbitMQ* y los mecanismos de autenticación de *Microsoft*, cuyos límites de uso, disponibilidad y costos operacionales deberán analizarse antes de una implementación institucional a gran escala.

En relación con el alcance de la investigación, este se circunscribió al diseño, desarrollo e implementación del *front-end* de una aplicación móvil destinada a apoyar la recolección y transmisión de datos operativos de cuadrillas en terreno de Chilquinta Transmisión. El desarrollo y administración de los componentes del *back-end*, la infraestructura de producción y los procesos asociados a la operación en un entorno real quedaron fuera de los objetivos de este trabajo. Por ello, la validación se enfocó en comprobar el funcionamiento de las funcionalidades implementadas, como la autenticación de usuarios, la navegación, la visualización de mapas, el seguimiento de ubicación y la transmisión de información, dentro de un ambiente controlado, sin realizar una evaluación durante la operación cotidiana de las cuadrillas.

Como consecuencia del alcance definido para esta investigación, tampoco fue posible cuantificar el impacto de la aplicación sobre los procesos operacionales. Aunque la solución fue concebida para facilitar el registro y la transmisión de información desde terreno, no se efectuó un estudio comparativo que permitiera obtener indicadores objetivos relacionados con la reducción de los tiempos de registro, la disminución de procesos manuales, la oportunidad en la transmisión de datos o la mejora en la disponibilidad de información para la toma de decisiones. En consecuencia, los beneficios observados se sustentan principalmente en la validación funcional de las características implementadas y no en métricas de desempeño obtenidas en condiciones reales de operación.

Respecto a la experiencia de usuario, las pruebas realizadas tuvieron un carácter funcional y exploratorio, orientado a verificar el correcto comportamiento de las pantallas, la navegación entre módulos y la ejecución de las funcionalidades implementadas. No se aplicaron metodologías estandarizadas para evaluar la usabilidad ni instrumentos de medición con una muestra representativa de operadores, por lo que no fue posible obtener indicadores relacionados con la eficiencia de uso, la tasa de éxito en las tareas, el tiempo promedio de ejecución o el nivel de satisfacción de los usuarios.

Adicionalmente, debido a las restricciones de tiempo y al alcance del proyecto, la aplicación fue validada sobre un conjunto acotado de dispositivos Android. En consecuencia, no se evaluó su comportamiento en una mayor diversidad de versiones del sistema operativo, tamaños de pantalla o configuraciones de hardware, factores que podrían influir en el rendimiento, la estabilidad y la experiencia de uso en una implementación a mayor escala.

En conjunto, estas limitaciones no disminuyen el aporte del trabajo desarrollado, sino que permiten identificar aspectos que podrán abordarse en futuras etapas del proyecto y en investigaciones posteriores, con el propósito de fortalecer la solución propuesta y facilitar su implementación en un entorno operativo real.

4.4 Trabajo Futuro.

Como trabajo futuro, la aplicación podría evolucionar hacia una herramienta integral de gestión operativa para las cuadrillas, incorporando funcionalidades que permitan registrar y gestionar incidencias en tiempo real. Entre las mejoras mas relevantes destaca la posibilidad de reportar hallazgos, evidencias, accidentes e imágenes asociadas a los problemas detectados, lo que facilitaría la documentación y el seguimiento de cada caso a lo largo de su ciclo de vida.

Asimismo, resultaría valioso incorporar funciones para iniciar y finalizar tareas, asignar estados de avance y registrar la magnitud del problema abordado, lo que permitiría una evaluación más precisa del impacto de las intervenciones realizadas. A esto se sumaría la implementación de un canal de comunicación entre cuadrillas cercanas o involucradas en un mismo caso, con el propósito de coordinar acciones, compartir información relevante y responder con mayor rapidez ante situaciones de emergencia.

En conjunto, estas mejoras fortalecerían la utilidad de la aplicación, optimizando tanto la operación diaria en terreno como la toma de decisiones a nivel de gestión.

Otra línea de trabajo futuro relevante es la incorporación de mecanismos de cumplimiento normativo en materia de protección de datos personales, en conformidad con la Ley N° 19.628 sobre Protección de la Vida Privada. En particular, resulta necesario que la aplicación contemple el consentimiento informado y expreso de los usuarios respecto del tratamiento de sus datos personales, conforme a lo establecido en el artículo 4°, especificando de manera clara la finalidad de la recolección, almacenamiento y eventual comunicación de dicha información. Asimismo, dado que funcionalidades como el registro de accidentes o incidentes de las cuadrillas pueden implicar el tratamiento de datos sensibles, entendidos como aquellos referidos al estado de salud de las personas, sería fundamental incorporar resguardos adicionales conforme al artículo 10° de la misma ley, que restringe el tratamiento de este tipo de datos salvo que exista consentimiento del titular, autorización legal expresa, o que su uso resulte necesario para fines vinculados a la determinación de beneficios de salud.

Cabe considerar además que, a partir del 1 de diciembre de 2026, entrará en vigor la Ley N° 21.719, que modifica sustancialmente el marco actual de protección de datos personales en Chile, estableciendo una nueva agencia de protección de datos personales con facultades fiscalizadoras y sancionatorias, derechos ampliados para los titulares de datos, y la obligación de notificar brechas de seguridad dentro de un plazo determinado. En consecuencia, se recomienda que futuras versiones de la aplicación sean diseñadas considerando estos nuevos estándares desde su etapa de desarrollo, incorporando principios como la trazabilidad del tratamiento de datos, el cifrado de la información sensible y personal, y mecanismos que permitan a los usuarios ejercer plenamente sus derechos sobre la información almacenada. Esto no solo aseguraría el cumplimiento legal de la herramienta en el corto y mediano plazo, sino que también fortalecería la confianza de los usuarios en su uso.



5 Agradecimientos.

Primero quiero agradecer a Dios por ayudarme en cada etapa de mi vida y estar presente en todas ellas. También deseo expresar mi más profundo agradecimiento a mi madre y a mi padre que en paz descanse, por su apoyo incondicional y por acompañarme en cada etapa de este proceso. Extiendo también mi gratitud a mi pequeña Daynara, cuyo cariño y paciencia fueron fundamentales para mantenerme motivado. Agradezco de manera especial a mi jefa de carrera, la profesora Pamela Gatica, por su orientación académica y confianza en mi trabajo.

A mi familia en general, gracias por el respaldo constante y por creer en mis capacidades. No puedo dejar fuera a mi perrito Kanito, cuya compañía fue un apoyo silencioso pero muy importante. Agradezco a mis mejores amigos Juanpi y Mati, por su amistad sincera y por estar presentes en los momentos más exigentes de este desafío. Para finalizar me gustaría agradecer a todas las personas que estuvieron en este gran camino, mis compañeros, mis amigos de universidad, a don Pedro Mella entrenador de básquetbol y en general a esta universidad.

6 Referencias.

- [1] Google, «Build for and integrate with multiple platforms,» Flutter documentation. [En línea]. Available: <https://docs.flutter.dev/platform-integration>. [Último acceso: Septiembre 2026].
- [2] Telefónica, «Smartphones en 2024: cuántos hay, tiempos de uso o principales utilizaciones,» Telefónica S. A., 2024. [En línea]. Available: <https://www.telefonica.com/es/salacomunicacion/blog/smartphones-2024/>. [Último acceso: Octubre 2025].
- [3] Exploding Topics, «Smartphone Usage & Adoption Statistics (2024),» 9 2024. [En línea]. Available: <https://explodingtopics.com/blog/smartphone-stats>.
- [4] Exploding Topics, «How Many People Own Smartphones? (2025–2029),» Semrush Inc., 2025. [En línea]. Available: <https://explodingtopics.com/blog/smartphone-stats>. [Último acceso: Octubre 2025].
- [5] Editorial Grupo AEA, «Programacion Web del Frontend al Backend,» Editorial Grupo AEA, 2023. [En línea]. Available: <https://dialnet.unirioja.es/descarga/libro/933116.pdf>. [Último acceso: Octubre 2025].
- [6] Autentia/Izertis, «Guia completa Front,» Autentia. [En línea]. Available: https://ahorasomos.izertis.com/autentia/wp-content/uploads/libros/Front_GuiaCompletaAutentia.pdf. [Último acceso: Octubre 2025].
- [7] Dialnet/Universidad de la Rioja, «Usabilidad en aplicaciones móviles,» Dialnet. [En línea]. Available: <https://dialnet.unirioja.es/servlet/articulo?codigo=5123524>. [Último acceso: Octubre 2025].
- [8] IEEE Software, «Trends and Challenges for Software Engineering in the Mobile Era,» IEEE Software. [En línea]. Available: <https://mpc.ece.utexas.edu/assets/pdf/ieeesoftware.pdf>. [Último acceso: Octubre 2025].
- [9] Babes-Bolyai University, «Automatic examining of software architectures on mobile applications codebases,» [En línea]. Available: <https://dobrean.ro/wp-content/uploads/2019/10/272automatic-examining-of-software-architectures-on-mobile-applications-codebases-1.pdf>. [Último acceso: Octubre 2025].
- [10] V. Giedrimas y S. Omanovič, «The impact of mobile architectures on component-based software engineering,» 2015 IEEE 3rd Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Riga, Letonia, 2015, pp. 1-6. doi: 10.1109/AIEEE.2015.7367317. [En línea]. Available: <https://ieeexplore.ieee.org/document/7367317/>. [Último acceso: Septiembre 2026].
- [11] Google, «Simple app state management,» Flutter documentation. [En línea]. Available: <https://docs.flutter.dev/data-and-backend/state-mgmt/simple>. [Último acceso: Septiembre 2026].
- [12] Sanghmitra, «The State Management Dilemma: BLoC vs. Provider in Modern Flutter Development,» International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 10, no. 5, pp. 326-336, Nov. 2024. doi: 10.32628/CSEIT241051027. [En línea]. Available: <https://ijsrcseit.com/home/article/view/CSEIT241051027>. [Último acceso: Septiembre 2026].



- [13] Microsoft, «Microsoft identity platform (MSAL) overview,» Microsoft Learn. [En línea]. Available: <https://learn.microsoft.com/entra/identity-platform/msal-overview>. [Último acceso: Septiembre 2026].
- [14] Broadcom Inc., «RabbitMQ Documentation.» [En línea]. Available: <https://www.rabbitmq.com/docs>. [Último acceso: Septiembre 2026].
- [15] N. Basavaraju, N. Alexander y J. Seitz, «Performance Evaluation of Advanced Message Queuing Protocol (AMQP): An Empirical Analysis of AMQP Online Message Brokers,» 2021 International Symposium on Networks, Computers and Communications (ISNCC), 2021, pp. 1-8. doi: 10.1109/ISNCC52172.2021.9615705. [En línea]. Available: <https://ieeexplore.ieee.org/document/9615705/>. [Último acceso: Septiembre 2026].



7 Anexos

1. Kirkpatrick, A., O Connor, J., Campbell, A., & Cooper, M. (Eds.). (2025). *Web content accessibility guidelines (WCAG) 2.1*. World Wide Web Consortium (W3C). <https://www.w3.org/TR/WCAG21/>
2. Rodríguez Mira, A. (2025, 8 de febrero). *Características para el diseño de interfaces de usuario (UI)*. Tokio School. <https://www.tokioschool.com/noticias/caracteristicas-ideales-que-deben-tener-interfaces-usuario/>
3. Ministerio Secretaría General de la Presidencia. (2024). *Ley N° 21.719: Regula la protección y el tratamiento de los datos personales y crea la Agencia de Protección de Datos Personales*. Biblioteca del Congreso Nacional de Chile. <https://www.bcn.cl/leychile/navegar?idNorma=1209272>