



UNIVERSIDAD TÉCNICA
FEDERICO SANTA MARÍA

DEPARTAMENTO DE ELECTROTECNIA E INFORMÁTICA
INGENIERÍA EN INFORMÁTICA

Desarrollo Front-end de Aplicación Móvil para la Gestión de Trámites LogiQuick

Jordan Navarrete Berrios

jordan.navarrete@usm.cl

Pamela Gatica Caballero

Profesor Guía

Resumen: El proyecto LogiQuick busca transformar la gestión de trámites logísticos en Sitrans, específicamente en el área SIX, mediante la implementación de una aplicación móvil basada en Flutter y la arquitectura Model-View-Controller (MVC). Actualmente, Sitrans enfrenta problemas significativos debido a procesos semi-digitalizados que generan largos tiempos de espera, comunicación ineficiente y altos costos operativos. La solución propuesta incluye una plataforma que automatiza tareas como la validación de documentos, el seguimiento en tiempo real y la notificación automática a los usuarios. Esto se complementa con una interfaz de usuario intuitiva y accesible, diseñada para mejorar la experiencia de conductores y tramitadores.

El uso de tecnologías como Firebase y APIs RESTful permite una sincronización eficiente y segura de datos en tiempo real. Las pruebas realizadas demuestran una reducción del 40 % en los tiempos de trámite, mejoras en la satisfacción del usuario en un 30 % y una optimización del consumo de recursos. Además, la modularidad del sistema garantiza escalabilidad y facilita la incorporación de nuevas funcionalidades en el futuro. Este enfoque posiciona a LogiQuick como una solución moderna e innovadora para los desafíos actuales del sector logístico.

Palabras Clave: Gestión de trámites, Flutter, MVC, automatización de procesos, front-end móvil.



1. Introducción

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

El avance tecnológico ha transformado significativamente la gestión de procesos administrativos en diversas industrias, pero el sector del transporte extraportuario, particularmente en logística de mercancías y gestión de documentos, aún enfrenta importantes desafíos en términos de eficiencia operativa. En Sitrans, específicamente en su área de trabajo SIX Extraportuario, estos desafíos se manifiestan de manera clara, especialmente en los trámites asociados con la entrada de camiones al puerto y la tramitación para la liberación de cargas.

Los agentes tramitadores son los responsables de gestionar las mercancías que les pertenecen o que representan. Para cumplir con sus funciones, deben acudir a las instalaciones de SIX para realizar trámites administrativos, la coordinación de pagos y la obtención de permisos aduaneros. No obstante, el uso de métodos semi-digitalizados en estos procesos genera importantes ineficiencias, entre las que destacan largos tiempos de espera, errores humanos y la limitada capacidad de los conductores para conocer el estado de los trámites en tiempo real.

Esta brecha tecnológica conlleva serias consecuencias: los conductores a menudo deben esperar entre 2 a 4 horas con los camiones encendidos, lo que no solo incrementa los costos operativos, sino que también genera un impacto ambiental negativo. Por otro lado, los agentes tramitadores enfrentan procesos manuales o apoyados en herramientas no centralizadas, lo que prolonga las gestiones a tiempos que oscilan entre 90 y 240 minutos, o incluso más en casos de errores en la documentación.

Estas ineficiencias no solo ralentizan las operaciones y aumentan los costos, sino que también afectan negativamente la experiencia del cliente y la competitividad de Sitrans en el mercado logístico. La falta de sincronización y automatización limita la capacidad de respuesta de la empresa, evidenciando la necesidad de una transformación digital que permita optimizar los procesos, mejorar la comunicación en tiempo real y ofrecer un servicio más eficiente y moderno.

1.2 Definición del problema.

El problema principal radica en la ausencia de una plataforma que permita una comunicación en tiempo real entre los conductores de camiones, los agentes tramitadores y agentes de Sitrans. Esta carencia genera una serie de problemas, entre los que destacan largos tiempos de espera tanto en las oficinas como a las afueras de las instalaciones. Los camioneros se encuentran con los camiones encendidos, lo que incrementa los costos operativos y contribuye negativamente al medio ambiente por la emisión de gases contaminantes.

Adicionalmente, los procesos semi-digitales tales como el escaneo, lectura y recepción de documentos, así como el uso de herramientas externas, aumentan la probabilidad de errores en la gestión documental y extienden los tiempos de tramitación. Estos pueden oscilar entre 90 y 240 minutos, e incluso más, en caso de inconsistencias en los documentos, pérdidas o demoras en los pagos necesarios por parte de la empresa encargada.

Esta falta de sincronización y automatización no solo afecta la coordinación interna y provoca frustración en los conductores, quienes, como se mencionó, deben estar esperando a las afueras de las instalaciones perdiendo la oportunidad de realizar otras actividades. También limita la capacidad



de respuesta de Sitrans, perjudicando la experiencia del cliente y reduciendo su competitividad en el sector logístico.

En comparación, otras empresas ya funcionan con sistemas de recepción digital que eliminan la necesidad de que los agentes tramitadores asistan a las instalaciones para realizar los trámites. Por ejemplo, AEXSA, automatiza la captura de datos de los archivos subidos por los clientes, logrando una gestión más eficiente.

En este contexto, resulta evidente la necesidad de implementar una solución tecnológica que automatice los procesos, garantice la actualización en tiempo real de los trámites y optimice la comunicación entre los diferentes actores involucrados, mejorando así la eficiencia operativa y la calidad del servicio.

A continuación, se presenta el diagrama de flujo actual para realizar un trámite, el cual valida lo comentado anteriormente.

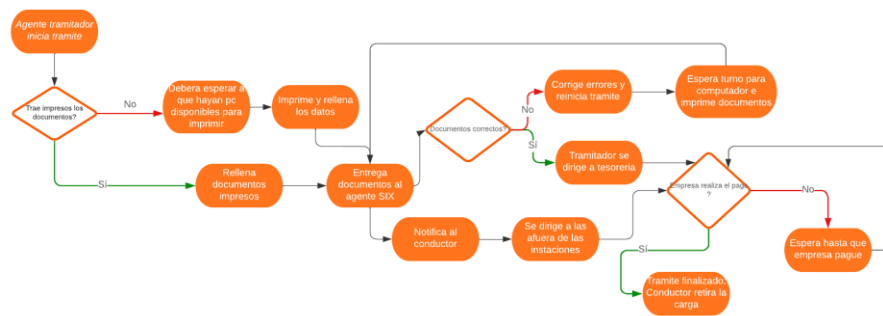


Diagrama 1 Flujo actual gestión de liberación de carga

Fuente: Elaboración propia

1.3 Breve descripción sobre la propuesta de solución

La solución propuesta consiste en el desarrollo de un sistema que incluye tanto una plataforma web como una aplicación móvil, utilizando Flutter como tecnología central. Esta combinación permitirá a los usuarios gestionar sus trámites de manera digital y eficiente, abordando los principales problemas de ineficiencia y retrasos asociados con los procesos manuales actuales.

A continuación, se presentan las características específicas del sistema y los motivos detrás de la elección de Flutter.

1.3.1 Características Específicas del Sistema:

- Automatización de Procesos: La plataforma web permitirá la automatización de tareas repetitivas, como la recopilación de información, la generación de documentos y la validación de datos. Esto reducirá significativamente la carga de trabajo manual,



minimizando errores y tiempos de procesamiento. Por otro lado, la aplicación móvil permitirá subir la documentación necesaria para poder gestionar un trámite.

- Integración de Notificaciones: Se implementará un sistema de notificaciones automatizadas que informará a los usuarios sobre el estado de sus trámites y actualizaciones importantes. Esto mantendrá a los usuarios informados en tiempo real, mejorando la comunicación y reduciendo la necesidad de consultas adicionales.
- Reducción de tiempo de espera: Al proporcionar actualizaciones los conductores podrán gestionar su tiempo de manera más eficiente, reduciendo las horas en las cuales esperan fuera del recinto.
- Seguimiento en Tiempo Real: Los conductores y tramitadores podrán consultar el estado de su trámite a través de la aplicación, eliminando la necesidad de llamadas telefónicas o largas esperas sin información.
- Panel de Control Personalizado: La plataforma ofrecerá un panel de control donde los usuarios podrán ver un resumen de todos sus trámites, con acceso rápido a información relevante y métricas de rendimiento personal, como tiempos de espera promedios y tareas pendientes.

Se seleccionó Flutter como tecnología principal para el desarrollo del front-end debido a la capacidad para satisfacer las necesidades específicas del proyecto. La decisión se fundamenta de la siguiente forma teniendo en cuenta las siguientes tecnologías “React Native, Xamarin e Ionic”:

1. Desarrollo multiplataforma eficiente: Flutter permite crear aplicaciones en múltiples plataformas tales como Android e iOS desde un único código base, reduciendo significativamente tiempos de desarrollo.
2. Experiencia de usuario: Con su biblioteca de widgets personalizables y diseñados por Google, Flutter garantiza interfaces visualmente atractivas y adaptativas, lo cual es esencial para la experiencia de usuario.
3. Hot Reload y productividad: Flutter ofrece la funcionalidad de “Hot Reload”, permitiendo a los desarrolladores visualizar cambios en tiempo real sin la necesidad de recompilar toda la aplicación. Esto es crucial para el entorno ágil ya que facilita las iteraciones rápidas.
4. Rendimiento optimizado: Al eliminar la necesidad de intermediarios con componentes nativos Flutter ofrece un rendimiento más eficiente, una característica importante para el seguimiento en tiempo real.
5. Soporte y comunidad: Al estar respaldado por Google, Flutter cuenta con documentación extensa y una comunidad en constante crecimiento. Lo que nos asegura el acceso a soluciones, herramientas y tips de alta calidad, aspectos que pueden no estar tan desarrollados en opciones como Ionic.
6. Facilidad para la Creación de Diseños adaptables: Con su capacidad para construir interfaces adaptables a múltiples dispositivos y tamaños de pantalla, Flutter sobresale en el desarrollo de aplicaciones responsivas, esenciales para la experiencia del usuario en LogiQuick. Este enfoque supera los desafíos de otras tecnologías que requieren más esfuerzo para lograr diseños completamente adaptables.



Para fundamentar esta elección, a continuación, se presenta una comparación entre Flutter y otras tecnologías multiplataforma, evaluando características clave como rendimiento, soporte multiplataforma y comunidad.

Características	Flutter	React Native	Xamarin	Ionic
Lenguaje de programación	Dart	JavaScript	C#	JavaScript, HTML, CSS
Rendimiento	Alto rendimiento gracias a la compilación nativa	Buen rendimiento, cercano al nativo	Rendimiento nativo	Depende del WebView, menor rendimiento
Comunidad	Comunidad en rápido crecimiento con fuerte respaldo de Google, ofreciendo recursos y paquetes en constante expansión.	Amplia comunidad respaldada por Meta (anteriormente Facebook), con numerosos recursos y bibliotecas disponibles.	Comunidad establecida dentro del ecosistema .NET, con soporte oficial de Microsoft y una variedad de recursos disponibles	Comunidad activa con numerosos plugins y recursos, aunque puede ser menos extensa que las de React Native y Flutter.
Diseño adaptable	Excelente	Bueno	Bueno	Limitado
Hot Reload	Si	Si	No	Si
Soporte multiplataforma	Android, iOS, Web	Android, iOS	Android, iOS, Windows	Android, iOS, Web

Tabla 1 Comparativa de frameworks multiplataforma para desarrollo móvil [13]

Fuente: Elaboración propia

1.3.2 Casos de éxito

Según la página oficial de Flutter [10] grandes empresas optaron por adoptar Flutter como tecnología principal de desarrollo entre las cuales se encuentran:



1. **BMW:** Uno de los líderes en la industria automotriz, adoptó Flutter para unificar sus aplicaciones móviles destinadas a iOS y Android. Previamente, enfrentaban problemas significativos para mantener la paridad de características y diseño entre plataformas. La introducción de Flutter les permitió reducir considerablemente el tiempo y los costos de desarrollo. Gracias a esta tecnología, BMW lanzó la aplicación "My BMW" en 47 países en menos de un año, demostrando la eficiencia y escalabilidad de Flutter en proyectos complejos.
2. **Alibaba** Gigante del comercio electrónico, también incorporó Flutter en el desarrollo de su aplicación móvil. Antes de su adopción, el equipo de desarrollo enfrentaba demoras en la implementación de nuevas funcionalidades. Con Flutter, redujeron el tiempo necesario para lanzar nuevas características de un mes a solo dos semanas. Este cambio no solo mejoró la productividad del equipo, sino también permitió a la empresa mantenerse competitiva en un mercado altamente dinámico.

Estos casos nos demuestran que Flutter es una tecnología clave para el desarrollo multiplataforma, combinando rendimiento, eficiencia y escalabilidad para proyectos modernos y exigentes.

1.4 Objetivos Generales y Específicos de la Tesina

Objetivo general: Desarrollar el front-end aplicación móvil para la gestión de trámites en Sitrans, enfocándose en la implementación de una interfaz de usuario eficiente y accesible, que permita automatizar y digitalizar procesos, mejorando la experiencia del usuario y facilitando la interacción con la aplicación.

Objetivos específicos para la aplicación móvil son:

- Implementar una interfaz de usuario intuitiva y accesible, utilizando Flutter, que optimice la navegación y facilite el uso eficiente de la aplicación.
- Implementar técnicas de compresión de recursos y optimización para mejorar los tiempos de carga, el uso de la memoria y el rendimiento general de la aplicación en dispositivos móviles.
- Implementar una arquitectura basada en el patrón Modelo-Vista-Controlador (MVC), lo cual permitirá separar claramente la lógica del negocio, la interfaz de usuario y el control, lo que facilita la escalabilidad y reduce el tiempo de mantenimiento del código.

1.5 Justificación de proyecto

Este proyecto es de gran importancia desde un punto de vista técnico porque aborda la automatización y digitalización de los procesos de gestión de trámites en Sitrans, una empresa que actualmente depende en gran medida de métodos semi-digitalizados. Según datos recopilados en nuestras visitas hacia las instalaciones de Sitrans podemos decir que el tiempo promedio de cada trámite realizado en las oficinas oscilan entre 90 a 240 minutos. Esta variabilidad dependiendo de la cantidad de errores técnicos que contengan los documentos los cuales pueden ser desde errores de tipeo hasta falta de documentos. Cabe recalcar que al tener un error en algún documento el tramitador deberá reimprimir todos los documentos nuevamente empezando el trámite desde cero lo que prolonga el tiempo de procesamiento y aumenta considerablemente la carga administrativa.

Comentado [JN1]: Si quiero citar a la pagina de flutter como lo hago?

Comentado [PG2R1]: Pendiente para reunión de avance.

Comentado [JN3R1]: Listo

Comentado [PG4]: Incluir una breve mención sobre el impacto esperado de la solución en términos de eficiencia y mejora de los procesos.

Sugerencia de redacción:

"Desarrollar una aplicación móvil para la gestión eficiente de trámites en SITRANS, mejorando la velocidad de procesamiento y reduciendo los errores asociados con los procesos manuales, con el fin de optimizar la experiencia de los usuarios y la productividad de la empresa."

Comentado [JN5R4]: Se modifico

Comentado [PG6R4]: Y la estandarización a Sitrans? XD

Comentado [J(7R4): Corregido en todo el documento!

Comentado [PG8R4]: Ya, ojo acá. Se me pasó este punto: La tesina aborta todo el desarrollo de la app móvil o solo en diseño de la UI? Si es lo segundo, hay que modificar el objetivo general. Si es lo primero, hay que modificar los objetivos específicos.

Comentado [J(9R4): Mi tesina aborda el desarrollo front de la app móvil por lo cual cambiare el objetivo general a uno mas específico

Comentado [JN10R4]: Edite el objetivo general

Comentado [PG11]: Asegurate de que los objetivos sean medibles y estén directamente relacionados con la solución propuesta.

Comentado [PG12]: No me queda claro eso: el desarrollo de una app móvil permitirá una solución multiplataforma?

Comentado [J(13R12): Con multiplataforma me refiero a iOS y Android pero creo que al poner "App móvil" queda claro eso así que lo arreglare

Comentado [PG14]: Sería útil detallar cómo la solución mejorará la experiencia de los usuarios finales, como los agentes tramitadores u otros usuarios. Esto podría incluir ejemplos específicos de casos de uso o plantearlo como una hipótesis:

Se estima una reducción del 30-40% en los tiempos de gestión de trámites, junto con una disminución de un X% en los errores asociados a la manipulación manual de datos.

Comentado [JN15R14]: Se añadió esa información

Comentado [PG16R14]: Muy bien



Por otro lado, durante las visitas pudimos observar que 6 de cada 10 tramitadores experimentan demoras debido a que las empresas realizan el pago de la mercancía al final del trámite en lugar de hacerlo previamente lo que prolonga considerablemente el tiempo de espera para la liberación de la carga. Esto no solo provoca que se acumulen tramitadores en las oficinas lo cual afecta el ambiente y la productividad interna, sino que también afecta directamente en la experiencia de los clientes, los cuales demandan un servicio rápido, cómodo y eficiente.

La implementación de LogiQuick tiene el potencial de ofrecer una mejora significativa en la experiencia de los usuarios finales, incluidos los agentes tramitadores y conductores. Entre los beneficios se destacan:

- Reducción de los tiempos de espera: se estima que los tiempos promedio de gestión disminuirán en un 30-40% reduciendo así aun rango de 60 a 90 minutos por tramite. Esta estimación se basa en análisis de sistemas similares, como el caso de AEXSA, que logró reducciones similares al implementar procesos de automatización y sincronización en tiempo real. Además, se consideraron factores específicos del flujo actual de trámites en Sitrans, como los tiempos asociados a validación documental y demoras en la comunicación.
- Diseño de aplicación: La interfaz será diseñada para ser intuitiva y accesible, dado a que va dirigida hacia un sector donde en gran parte a los conductores se le dificulta el uso de la tecnología por lo cual se optó que sea para usuarios experimentados como para aquellos que tienen poca experiencia con tecnología. Esta simplicidad de uso incrementará la adopción de la plataforma y reducirá la necesidad de soporte técnico.
- Operación operativa: La digitalización permitirá disminuir los errores, mejorar la productividad y ofrecer un servicio más competitivo.

En términos de innovación el proyecto introduce innovaciones tecnológicas y metodológicas que optimizan tanto el desarrollo como el rendimiento de la aplicación móvil. Al utilizar Flutter como framework principal, se logra un desarrollo ágil que garantiza eficiencia, escalabilidad y reducción de costos, evitando la necesidad de crear aplicaciones separadas para iOS y Android. La automatización de tareas repetitivas no solo incrementa la eficiencia operativa, sino que también minimiza errores humanos, mejorando significativamente los tiempos de respuesta frente a métodos manuales tradicionales.

Además, la aplicación prioriza una interfaz de usuario intuitiva y estéticamente atractiva, diseñada para facilitar su uso y garantizar una experiencia fluida. Este enfoque en la usabilidad asegura que los usuarios interactúen cómodamente con la plataforma, incrementando su satisfacción y compromiso con el sistema. Estas características posicionan al proyecto como una solución innovadora que aborda las necesidades actuales del sector logístico de manera efectiva.

A continuación, se presenta el diagrama de flujo con el uso de la aplicación móvil reduciendo en gran parte el tiempo de espera permitiéndole a los involucrados realizar otras actividades relevantes durante el transcurso del trámite.

Comentado [PG17]: Hay métricas o datos de esta justificación?

La justificación sería más sólida si incluyera datos o métricas que respaldaran la necesidad del proyecto. Por ejemplo, estadísticas sobre tiempos actuales de gestión en SITRANS o costos asociados a los errores manuales.

Comentado [JN18R17]: Se añadió información que pudimos capturar en nuestras visitas a las sucursales



Diagrama 2 flujo propuesto con aplicación móvil
Fuente: Elaboración propia

1.6 Metodología

La metodología que se utilizó para el desarrollo de la aplicación móvil se fundamenta en el enfoque ágil, específicamente en el marco de trabajo Scrum, el cual nos permitió flexibilidad, retroalimentación constante y la entrega incremental de valor al cliente. Dado el dinamismo del proyecto, donde los requisitos podían evolucionar según la necesidad de Sitrans, la metodología scrum resulto ser la adecuada para garantizar la calidad y el cumplimiento de los objetivos.

La organización en ciclos iterativos los cuales se denominan sprints, tuvieron una duración de 1 mes aproximadamente. Al culminar el sprint nos asegurábamos de que la aplicación tuviera incrementos funcionales asegurándonos que el producto evolucionara de forma progresiva y alineada con las expectativas del cliente. Este enfoque principalmente nos permitió priorizar funcionalidades claves y nos ayudó a mantener un flujo de trabajo eficiente.

A continuación, se describen brevemente las fases y elementos esenciales durante el desarrollo:

Comentado [PG19]: Si es posible, cambia el color anaranjado por algo mas claro para que resalte mejor el texto. Sino, cambia el texto a blanco.

Comentado [JN20R19]: Listo

Comentado [PG21]: Evita tanto listado de ideas y resume esto en uno o dos párrafos de texto.

Comentado [JN22R21]: Se modifico, deje el comentario para que lo revise para posteriormente eliminarlo una vez este listo



Product Backlog: El backlog se elaboró a partir de los requerimientos iniciales proporcionados por Sitrans y se fue ajustando de acuerdo con el progreso del proyecto y las necesidades emergentes. Incluía historias de usuarios las cuales describen funcionalidades tales validaciones de documentos, autenticación, seguimiento en tiempo real.

Sprint Planning: Antes de iniciar cada sprint, como equipo definíamos los objetivos a cumplir durante el ciclo. En estas sesiones se seleccionaban tareas del backlog priorizadas en base a su impacto y complejidad. Por ejemplo, en los primeros sprints se desarrollaron los módulos de autenticación y navegación, mientras que en los sprints posteriores se abordaron funcionalidades avanzadas como la integración de Apis y diseños adaptativos.

Daily Scrum: Como grupos nos organizamos para tener reuniones 2 o 3 veces por semana en las cuales en cada sesión se llevó a una revisión donde los desarrolladores presentábamos avances. Durante estas revisiones se recopiló retroalimentación para refinar el desarrollo de los siguientes sprints.

A continuación, se dará más detalle respecto a los sprint realizados:

Sprint 1: Diseño y organización del proyecto

Objetivos principales

- Establecer la base del proyecto en términos de diseño y estructura.
- Planificar la arquitectura del sistema (front-end y back-end).
- Iniciar configuraciones básicas del entorno.

Actividades realizadas

- **Diseño y planificación:**
 - Creación de wireframes y prototipos para las vistas principales (UX).
 - Definición de la arquitectura MVC para la aplicación (separación entre modelos, vistas y controladores).
- **Organización del proyecto:**
 - Configuración de GitHub para control de versiones.
 - Estructuración de carpetas para módulos de front-end y back-end.
- **Configuraciones iniciales:**
 - Instalación de Flutter y configuración de Firebase.
 - Configuración inicial de la base de datos (tablas principales: usuarios, roles, documentos).
 - Configuración de herramientas clave como Docker para contenedores.

Actividades de UX:

- Flujos de usuario: Diseñar los flujos iniciales de interacción (inicio de sesión, navegación básica).
- Identificación de perfiles de usuarios (tramitadores y conductores).



- Uso de mapas de empatía y journey maps para entender necesidades.
- Wireframes: Crear wireframes para las secciones principales de la aplicación en Figma.
- Validación: Realizar una sesión rápida de feedback interno sobre los prototipos.

Sprint 2: Desarrollo inicial de funcionalidades

Objetivos principales

- Implementar funcionalidades clave en el front-end y back-end.
- Diseñar vistas principales e integrarlas con el backend.
- Reducir el enfoque en UX y avanzar en módulos funcionales.

Actividades técnicas:

- **Front-end:**
 - Implementación de vistas principales (inicio de sesión, dashboard, secciones relacionadas a los tramites)
- **Back-end:**
 - Implementación de modelos principales (usuarios, roles, documentos, pagos)
 - Desarrollo de endpoints:
 - Registro y autenticación (local y Google)
 - Subida de documentos y almacenamiento en AWS S3
- **Integración:**
 - Conexión de las vistas del front-end con los endpoints creados.

Actividades de UX:

- Guía de estilo: Se definieron colores, tipografías y elementos visuales.
- Creación de prototipo interactivo.
- Ajustes en el diseño basándose en las recomendaciones internas.

Sprint 3: Optimización, funcionalidades avanzadas y despliegue

Objetivos principales

- Finalizar la implementación de todas las funcionalidades avanzadas.
- Optimizar la aplicación y realizar pruebas finales.
- Priorizar el backend y la integración completa con el front-end.

Actividades técnicas:

- **Front-end:**
 - Optimización de widgets reutilizables (mejora la eficiencia y consistencia)
 - Implementar herramienta de compresión de recursos gráficos tal como Imgbot.
- **Back-end:**
 - Creación de módulos avanzados:
 - Chequeo de roles y tokens para autenticación segura.
 - Notificaciones personalizadas (controlador para envió).
 - Creación de endpoints adicionales:
 - Creación de trámite.



- Listado y edición de datos de usuarios.
 - Listado y gestión de notificaciones.
- **Integración y pruebas:**
 - Pruebas finales de integración entre front-end y back-end.
 - Resolución de errores críticos en ambas capas. (tales como errores de tamaño con componentes).
- **Despliegue:**
 - configuración de CI/CD para despliegues automáticos.
 - Contenedor Docker desplegado en AWS.
- **Actividades de UX**
 - Validar la experiencia de usuario y de usabilidad mediante la escala SUS.
 - Recopilar comentarios mediante Microsoft Forms.
 - Realizar ajustes mejores para consistencia estética.

1.7 Breve descripción de la organización del informe en capítulos.

El informe está estructurado en siete capítulos que abordan los aspectos más relevantes del desarrollo del proyecto. En el **Capítulo 1**, se expone el contexto del problema en Sitrans, destacando las ineficiencias de los procesos semi-digitalizados en la gestión logística y la propuesta de una solución tecnológica basada en Flutter para optimizar tiempos y mejorar la comunicación entre conductores y agentes tramitadores. Además, se establecen los objetivos y la metodología Scrum adoptada. El **Capítulo 2** desarrolla el marco teórico, explicando conceptos clave del desarrollo front-end, el diseño adaptativo y la arquitectura MVC, junto con el uso de herramientas como Firebase y patrones de diseño que garantizan modularidad y escalabilidad. En el **Capítulo 3**, se detalla la implementación técnica, incluyendo la estructura modular, los sprints de desarrollo, la integración con APIs RESTful y las estrategias de optimización para un rendimiento eficiente. El **Capítulo 4** presenta las conclusiones, destacando la reducción del 40 % en tiempos de trámite y las mejoras en la experiencia del usuario, además de aprendizajes clave y proyecciones futuras, como la incorporación de inteligencia artificial. Finalmente, se incluyen agradecimientos, referencias bibliográficas que complementan el informe con recursos de apoyo.

Comentado [PG23]: Esto lo vamos a mejorar después.

2. Marco Teórico

2.1 Fundamentos del desarrollo front-end

El desarrollo front-end se enfoca en la parte de la aplicación que los usuarios ven y usan, incluyendo el diseño de la interfaz, la usabilidad y la optimización para una interacción fluida. Una aplicación móvil bien diseñada no solo mejora la satisfacción del usuario, sino que también contribuye al éxito operativo de la empresa al facilitar procesos y aumentar la productividad.

2.1.1 Principios clave en el desarrollo de aplicaciones móviles

Diseño adaptativo: El diseño adaptativo en Flutter hace referencia a la capacidad de que la aplicación se ajuste a diferentes tamaños de pantalla, desde teléfonos pequeños hasta tablets grandes. Este enfoque permite ofrecer una experiencia de usuario óptima al ajustar los componentes de la interfaz según el tamaño de la pantalla. Flutter facilita este diseño mediante herramientas como MediaQuery para detectar tamaños y orientaciones de pantalla, y widgets como LayoutBuilder y Flexible, que ayudan a construir interfaces escalables y funcionales en cualquier dispositivo móvil.

Comentado [PG24]: Este enfoque es web pero el desarrollo es para una app móvil. Creo que es importante que en el capítulo anterior, quede muy claro que Logiquick es una solución compuesta de varias aplicaciones y una de ellas es este componente móvil que se detalla en la tesis. Sino, parece que hablaras de peras y manzanas.

Comentado [JN25R24]: Puse responsivo dado a que se podría utilizar en diversos dispositivos móviles tales como ipad, tablet, iphone y otros dispositivos android. Pero leyendo me di cuenta que el concepto era adaptativo o Adaptive Ui así que lo cambié a eso



Rendimiento: La optimización del rendimiento es esencial para garantizar tiempos de carga rápidos y una interacción sin demoras. Jakob Nielsen [15] señala que los usuarios esperan que las respuestas del sistema ocurran en menos de un segundo para mantener su flujo de pensamiento. Para lograrlo, se implementan técnicas como la carga asíncrona de datos, la compresión de recursos y la minimización del código redundante. Además, Flutter ofrece un rendimiento nativo al compilar el código Dart a código máquina, lo que mejora significativamente la velocidad de ejecución.

Interactividad y Flujo de Datos en Tiempo Real: La aplicación deberá ser interactiva y permitir actualizaciones en tiempo real sobre el estado de los trámites. Esto requerirá la integración con bases de datos en la nube, como Firebase, para sincronizar información y mostrar los cambios a los usuarios de manera inmediata.

2.1.2 Patrones de Diseño en Desarrollo de Software

Los patrones de diseño son soluciones generales y reutilizables para problemas comunes en la arquitectura de software. Según Gamma [12], estos patrones ayudan a mejorar la eficiencia, escalabilidad y mantenibilidad de los sistemas. En el contexto del proyecto LogiQuick, se aplicaron los siguientes patrones:

1. **Singleton:** Este patrón asegura que una clase tenga solo una instancia y proporciona un punto de acceso global a ella. Es útil en escenarios donde se requiere una configuración centralizada, como la administración de servicios críticos (por ejemplo, conexión a Firebase) [12].
2. **Builder:** Permite construir objetos complejos paso a paso, lo que resulta en una mayor flexibilidad y reutilización en la construcción de elementos visuales dinámicos en la interfaz de usuario [12].

2.2 Arquitectura de software

Para el desarrollo de la aplicación móvil de LogiQuick, se adoptó el patrón arquitectónico Modelo-Vista-Controlador (MVC) para estructurar el sistema de manera clara, separando las responsabilidades entre las capas de datos, la interfaz de usuario y la lógica de negocio. Esta organización permite un desarrollo más escalable y mantenible, facilitando la colaboración entre equipos de desarrollo [14].

2.2.1 Modelo MVC aplicado en el proyecto

Modelo (Model): El modelo se encarga de gestionar los datos y la lógica de negocio. Esto incluye la interacción con las APIs y el almacenamiento de datos. En el proyecto, esta funcionalidad se encuentra en la carpeta services/, que incluye archivos como:

Nombre Archivo	Breve descripción de funcionalidad
google_auth.dart	Maneja la autenticación de usuarios mediante Google.

Comentado [PG26]: Propongo que lo abordemos como el desarrollo de Logiquick, se simplifica entender qué es lo que se está construyendo.

Logiquick es una solución para la gestión de trámites logísticos intraportuarios compuesto de una aplicación web y una aplicación móvil que funcionan en complemento. Esta tesina aborda el desarrollo de front-end del componente móvil, cuyo objetivo es bla bla..

Incluso, un diagrama de arquitectura o de componentes UML de todo el sistema permitiría entender mejor el panorama completo y que rol cumple esta app móvil.

Comentado [PG27R26]: Pendiente

Comentado [JN28R26]: Esto quedo cerrado según me comento en la reunión de ayer dado a que esta en el marco teórico

Comentado [PG29]: Título?

Comentado [JN30R29]: Corregido



app_constants.dart	Contiene constantes relacionadas con la configuración de Mapbox y ubicación.
session_manager.dart	Gestiona el tipo de inicio de sesión utilizando SharedPreferences localmente.
constants.dart	Contiene configuraciones y valores reutilizables globales.

Tabla 2 archivos modelo

Fuente: Elaboración propia

Vista (View): La vista es responsable de presentar la interfaz gráfica al usuario, utilizando los widgets de Flutter para representar datos de forma interactiva y visualmente atractiva. Las vistas se organizan en las carpetas para maximizar la modularidad y la reutilización.

Detalles de algunas carpetas y archivos:

1. Autenticación (/lib/ui/layouts/auth)

Dentro de esta carpeta podremos encontrar los widgets utilizados para mostrar todo lo relacionado a la autenticación además de widgets reutilizables disminuir el uso de código innecesario. En la siguiente tabla se mostrarán algunos archivos con una breve descripción de su uso:

Nombre Archivo	Breve descripción de funcionalidad
auth_layout.dart	Diseña la estructura general de autenticación.
form_create_account_view.dart	Formulario para registrar nuevos usuarios.
form_login_whit_mail_view.dart	Interfaz para el inicio de sesión mediante correo.
form_forget_password_view.dart	Vista para recuperar contraseñas.

Tabla 3 detalle carpeta /Auth

Fuente: Elaboración propia



2. Dashboard (/lib/ui/layouts/dashboard)

Dentro de esta carpeta podremos encontrar los widgets utilizados para mostrar la interfaz del conductor y la interfaz del tramitador además de widgets reutilizables en ambas vistas para disminuir el uso de código innecesario. En la siguiente tabla se mostrarán algunos archivos con una breve descripción de su uso:

Nombre Archivo	Breve descripción del archivo
card_assigned_load.dart	Utilizado en la interfaz del conductor el cual necesita ver la carga que tiene asignada
card_remove_load.dart	Utilizado en la interfaz del conductor dicho widget muestra información acerca de donde retirar la carga
card_status_procedure.dart	Utilizado en la interfaz del conductor ayuda a comunicar el estado del trámite

Tabla 4 detalles carpeta /dashboard

Fuente: Elaboración propia

3. Compartidos (/ui/shared)

Dentro de esta carpeta podremos visualizar diversas carpetas que se ocupan de manera global dentro de la aplicación como puede ser el uso de botones, tarjetas, inputs, colores y estilos de texto.

Nombre Archivo	Breve descripción del archivo
custom_outlined_button.dart	Botón estilizado reutilizable.
custom_card_tabs.dart	Tarjetas dinámicas
custom_inputs.dart	Campos de entrada reutilizables. Se utilizaron en todos los campos de la aplicación.
colors.dart	Configuración centralizada de la paleta de colores.
link_text.dart	Estilos para enlaces interactivos dentro de la aplicación.

Tabla 5 detalles carpeta /shared



Fuente: Elaboración propia

Controlador (Controller): El Controlador actúa como intermediario entre el Modelo y la Vista, gestionando la lógica de interacción y las operaciones específicas. Archivos relevantes:

Nombre Archivo	Breve descripción del archivo
upload_file.dart	Sube archivos al servidor gestionando autenticación y manejo de errores.
cargas_controller.dart	Gestiona lógica para obtener cargas pendientes del usuario desde backend.
tramites_controller.dart	Gestiona lógica para obtener trámites de usuario y conductor desde backend.
archivos_controller.dart	Gestiona lógica para obtener archivos asignados al usuario desde backend.
Auth_session_controller.dart	Gestiona inicio de sesión, obtención de datos y cierre de sesión.

Tabla 6 detalles controladores

Fuente: Elaboración propia

2.2.2 Beneficios de la arquitectura

La arquitectura Modelo-Vista-Controlador (MVC) implementada en este proyecto ofrece una serie de beneficios que impactan directamente en la eficiencia del desarrollo y la calidad del producto final. Al segmentar las responsabilidades en capas claramente definidas, MVC facilita el mantenimiento y la evolución del sistema, ya que cada capa puede ser modificada o actualizada sin afectar las demás. Esta separación también promueve la reutilización de componentes, especialmente en las vistas y controladores, lo que reduce el tiempo de desarrollo y asegura la consistencia en la interfaz de usuario.

Además, el modularidad de esta arquitectura permite que el sistema sea altamente escalable. Por ejemplo, en caso de requerir nuevas funcionalidades, estas pueden añadirse sin interferir con las partes ya desarrolladas.

Otro aspecto destacado es la mejora en la colaboración entre equipos. Al estar claramente separadas las capas de presentación, lógica y datos, los equipos de front-end y back-end pueden trabajar de manera simultánea e independiente, lo que acelera los tiempos de desarrollo y minimiza conflictos. Por último, esta estructura también facilita la depuración y las pruebas, ya que permite identificar errores

Comentado [PG31]: Título?

Comentado [JN32R31]: Corregido

en áreas específicas del sistema, agilizando la corrección de problemas y mejorando la calidad del producto final.

La Figura 1 muestra el diagrama de la arquitectura MVC aplicada en este proyecto, destacando cómo el modelo, la vista y el ViewModel interactúan para garantizar una separación clara de responsabilidades.

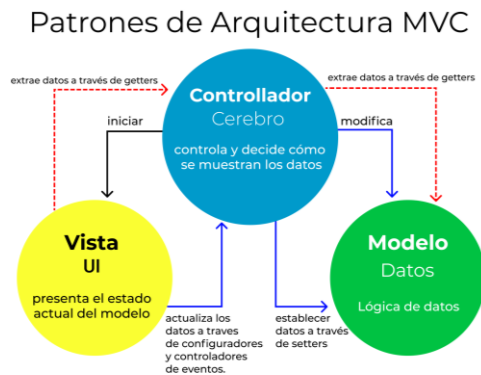


Figura 1 Diagrama arquitectura MVC
Fuente: Extraída desde [14]

2.3 Herramientas y entornos de desarrollo utilizados

Las herramientas principales utilizadas para el desarrollo de este proyecto son:

- **Flutter:** Es el framework principal utilizado para el desarrollo de la aplicación móvil multiplataforma. Flutter permite crear aplicaciones nativas para Android, iOS y Web desde un único código base, lo que reduce los tiempos de desarrollo y garantiza una experiencia de usuario consistente en diferentes plataformas [1].
- **Dart:** Dart es el lenguaje de programación utilizado en Flutter. Ofrece una sintaxis sencilla y capacidades modernas que facilitan el desarrollo ágil y eficiente de aplicaciones móviles. Su capacidad de compilar a código máquina mejora el rendimiento de la aplicación.
- **Git:** Sistema de control de versiones empleado para gestionar el código fuente del proyecto. Se utilizará en conjunto con GitHub, donde se crean ramas individuales para las funcionalidades (dev) y otra rama (main) que es donde se encuentra el diseño lo que facilita la colaboración entre front-end y back-end.
- **Visual Studio Code:** Es el entorno de desarrollo integrado (IDE) principal para la codificación y depuración de la aplicación. Gracias a la integración con Git, Visual Studio Code facilita la gestión de versiones y la colaboración directa con otros miembros del equipo.
- **Android Studio:** Este entorno de desarrollo se usa especialmente para crear y probar la aplicación en dispositivos Android. Proporciona emuladores de dispositivos móviles para realizar pruebas exhaustivas en diferentes versiones del sistema operativo, asegurando que la aplicación funcione correctamente en una amplia gama de dispositivos.

Comentado [PG33]: Revisar fuentes y referencias en reunión

Comentado [JN34R33]: Listo

Comentado [PG35]: Mira este diagrama: https://miro.medium.com/v2/resize:fit:720/format:webp/1*YoGbn5uvDYSIpN7zHr9yg.gif

Intenta detallar mejor el MVVM.
No puedes pegar esa imagen porque no sabemos si tiene derechos de autor, pero puedes crear tu propia versión en figma

Comentado [JN36R35]: Se cambio esto yo tenia una vista erronea se modifico a MVC

Comentado [JN37]: Se añadió como se ocuparan las herramientas en en el proyecto



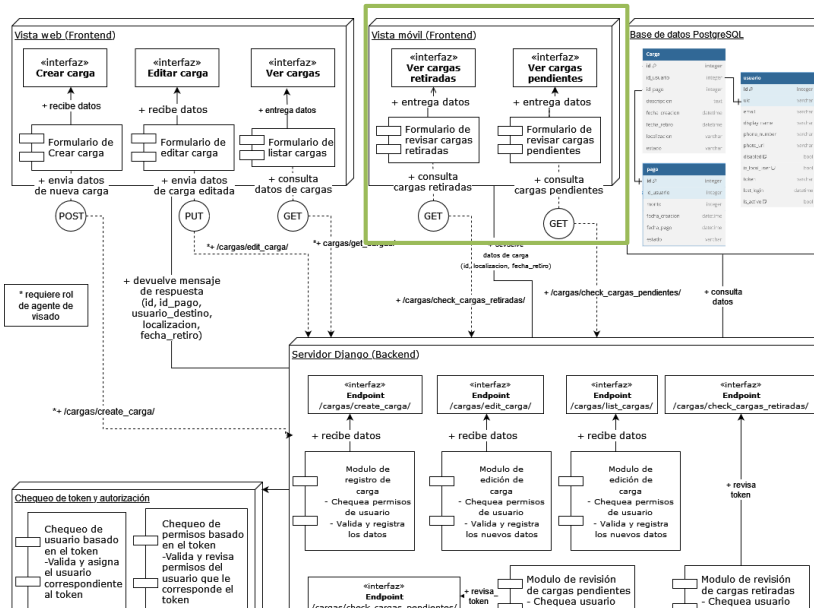
- **Figma:** Es la plataforma de diseño colaborativo utilizada para crear los prototipos y wireframes de la aplicación. La integración en tiempo real entre diseñadores UX y desarrolladores front-end asegura que el diseño visual y la funcionalidad estén alineados desde las primeras etapas del proyecto.

3. Desarrollo

3.1 Diseño de Componentes

3.1.1 Arquitectura y estructura de los componentes.

La arquitectura de la aplicación está diseñada bajo el modelo Model-View-Controller (MVC), adoptado para garantizar una estructura modular que facilite la escalabilidad, la reutilización de componentes y el mantenimiento del código. Este enfoque separa claramente la lógica de negocio, la presentación visual y el control, asegurando un desarrollo eficiente y flexible. Los componentes de la aplicación se organizan en carpetas específicas con roles definidos.



Fuente: Elaboración grupal

Comentado [PG38]: Incluir diagrama de componentes del sistema



En esta tesina, se trabajará exclusivamente en el desarrollo del front-end de la aplicación móvil, específicamente en las interfaces marcadas dentro del recuadro verde del diagrama donde para mantener una arquitectura ordenada y modular, se estableció una estructura de carpetas que ordena de manera lógica los diferentes componentes del front-end. A continuación, se detalla esta estructuración, comenzando por las carpetas: Services/ y Ui/.

3.1.2 Services/

Gestiona la lógica de negocio e interactúa con servicios externos como APIs y Firebase. Esta carpeta es clave para la conexión entre los datos y las vistas de la aplicación.

- Un ejemplo sería el archivo `google_auth.dart` el cual se encarga de implementar la autenticación con Google y por otra parte `tab_control.dart` maneja la navegación entre vistas.

3.1.3 Ui/

Almacena las vistas y los widgets organizados según su función. Su estructura modular permite una fácil extensibilidad y reutilización de componentes.

- Subcarpetas:
 - Layouts/
 - Auth/: Almacena las vistas relacionadas con la autenticación, como inicio de sesión y recuperación de contraseñas.
 - Dashboard/: Diseña el tablero principal, incluyendo widgets específicos para conductores y tramitadores.
 - Tabs/: Implementa las pestañas que se mostraran al navegar por la aplicación.
 - Shared/: Contiene recursos compartidos entre vista.
 - Buttons/: Contiene botones personalizados reutilizables
 - Inputs/: Contiene campos de entrada reutilizables.
 - Cards/: Implementa tarjetas personalizadas que presentan información de manera clara y organizada.



- Un ejemplo más específico sería el archivo colors.dart el cual define la paleta de colores la cual se vuelve reutilizable para mantener la consistencia visual en todas las vistas.

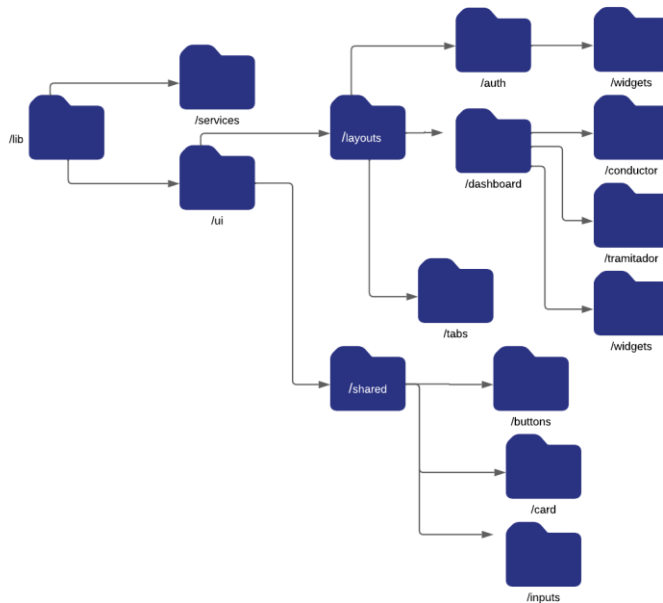


Diagrama 4 Diagrama de carpetas

Fuente: Elaboración propia

3.1.4 Reutilización de componentes

En el desarrollo de la aplicación se priorizó la creación de widgets reutilizables para garantizar la consistencia visual, reducir la duplicación de código y optimizar el tiempo de desarrollo. Estos widgets fueron diseñados con propiedades configurables que les permiten adaptarse a las necesidades específicas manteniendo la modularidad y facilitando el mantenimiento del código.



Entre los componentes destacados se encuentran los siguientes:

```
import 'dart:ui';

class CustomColor {
  static Color buttons = Color(0xFF64D1CB);
  static Color details = Color.fromARGB(255, 255, 242, 0);
  static Color background = Color(0xFF272E4B);
  static Color textPrimary = Color.fromARGB(255, 0, 0, 0);
  static Color bgCards = Color(0xFFEBEDF0);
  static Color additionalImportant = Color(0xFFFF4C560);
}
```

Ilustración 1 Componente CustomColor

Fuente: Elaboración propia

CustomColor se creó para mantener el orden visual y coherente dentro de la aplicación. Este archivo centraliza toda la paleta de colores utilizada en la aplicación lo que facilita la mantención y las modificaciones futuras.

Modo de uso: Para utilizarlo, es necesario importar el archivo en el código correspondiente, lo que permite acceder a los colores predefinidos sin necesidad de escribir configuraciones manuales como "Color.fromARGB(255, 0, 0, 0)". Esto reduce la redundancia en el código y mejora la claridad del proyecto.

Un ejemplo destacado del uso de CustomColor se encuentra en combinación con el widget personalizado CustomCardNotify. Este widget se diseñó para mostrar notificaciones visuales en la aplicación de manera coherente y estética, utilizando los colores centralizados de CustomColor. Por ejemplo, en una notificación de éxito, el color principal de la tarjeta se configura automáticamente desde el archivo central, garantizando uniformidad en toda la interfaz.

El widget CustomCardNotify nos permite configurar de manera flexible parámetros como el título de la notificación (title), iconos (icon), estado (state), colores de borde (colorBorde), icono (colorIcon) y un controlador de texto (notificationController) para gestionar entradas dinámicas. Todo esto se encapsula en un único componente reutilizable



En promedio, la creación manual de una notificación similar podría requerir entre 40-60 líneas de código, dependiendo de la complejidad de los estilos y funcionalidades. Al definir CustomCardNotify, este proceso se reduce a una instancia de 6 a 8 líneas de códigos para definir los parámetros deseados.

```
import 'package:flutter/material.dart';
import 'package:google_fonts/google_fonts.dart';

class CustomCardNotify extends StatefulWidget {
  CustomCardNotify({
    required this.title,
    required this.icon,
    required this.state,
    required this.colorBorder,
    required this.colorIcon,
    required this.notificationController,
  });

  final String title;
  final IconData icon;
  final Color colorIcon;
  final String state;
  final Color colorBorder;
  final TextEditingController notificationController;
  @override
  State<CustomCardNotify> createState() => CustomCardNotifyState();
}

class CustomCardNotifyState extends State<CustomCardNotify> {
  @override
  Widget build(BuildContext context) {
    return Column(
      children: [
        Container(
          padding: EdgeInsets.all(20),
          width: 100,
          height: 100,
          decoration: BoxDecoration(
            color: Colors.white,
            borderRadius: BorderRadius.circular(30),
            border: Border(
              left: BorderSide(color: widget.colorBorder, width: 2),
              right: BorderSide(color: widget.colorBorder, width: 2),
              top: BorderSide(color: widget.colorBorder, width: 2),
              bottom: BorderSide(color: widget.colorBorder, width: 2),
            ),
          ),
          child: Row(
            mainAxisAlignment: MainAxisAlignment.start,
            crossAxisAlignment: CrossAxisAlignment.start,
            children: [
              Icon(widget.icon, color: widget.colorIcon,
                size: 30,
                Semantics(width: 15),
              ),
              Column(
                mainAxisAlignment: MainAxisAlignment.start,
                crossAxisAlignment: CrossAxisAlignment.start,
                children: [
                  Text(widget.title,
                    style: GoogleFonts.inter(
                      fontSize: 18,
                      fontWeight: FontWeight.w300,
                    ),
                  ),
                  Text(widget.state,
                    style: GoogleFonts.inter(
                      fontSize: 18,
                      fontWeight: FontWeight.w300,
                    ),
                  ),
                  Semantics(height: 10),
                  Semantics(
                    height: 100,
                    width: 200,
                    child: TextFormField(
                      controller: widget.notificationController,
                      maxLines: 1,
                      decoration: InputDecoration(
                        hintText: 'Escribe aquí su mensaje',
                      ),
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Row(
              children: [
                Icon(Icons.delete, color: Colors.red,
                  size: 24),
                Semantics() {
                  setState(() {
                    widget.notificationController.clear();
                  });
                },
              ],
            ),
            Semantics(
              height: 10,
            ),
          ],
        ),
      ],
    );
  }
}
```

**Ilustración 3 Código detrás de
CustomCardNotify**

Fuente: Elaboración propia

```
Container(
  color: Colors.white,
  padding: EdgeInsets.all(20),
  child: Column(
    mainAxisAlignment: MainAxisAlignment.start,
    crossAxisAlignment: CrossAxisAlignment.stretch,
    children: [
      CustomCardNotify(
        title: 'Notificación 1',
        icon: Icons.notifications_active,
        state: 'Activa',
        colorBorder: CustomColor.details,
        colorIcon: CustomColor.details,
        notificationController: _messageController
      ),
      CustomCardNotify(
        title: 'Notificación 1',
        icon: Icons.notifications_active,
        state: 'Activa',
        colorBorder: CustomColor.buttons,
        colorIcon: CustomColor.buttons,
        notificationController: _messageController),
      CustomCardNotify(
        title: 'Notificación 1',
        icon: Icons.notifications_active,
        state: 'Activa',
        colorBorder: CustomColor.additionalImportant,
        colorIcon: CustomColor.additionalImportant,
        notificationController: _messageController)
    ],
  ),
),
```

Ilustración 2 Aplicación de Widgets

Fuente: Elaboración propia

Esto representa un ahorro estimado de 85-90% de las líneas necesarias por notificación lo que no solo representa un código más limpio y legible, sino que también mejora la eficiencia del desarrollo.

3.1.5 Integración con el sistema general.

La integración de la aplicación móvil dentro del ecosistema de LogiQuick se realiza mediante el consumo de APIs REST diseñadas para la comunicación eficiente y segura con el backend. El backend,



desarrollado en Django, implementa el patrón MVC y expone una serie de endpoints que gestionan la lógica de negocio y la interacción con la base de datos. La aplicación móvil, construida con Flutter, consume estas APIs para permitir que los usuarios realicen y supervisen trámites, accedan a notificaciones en tiempo real y gestionen documentos de manera eficiente.

El backend utiliza autenticación robusta basada en Firebase Authentication y tokens locales para garantizar que las solicitudes de la aplicación móvil sean seguras. Además, se implementaron mecanismos como cifrado de datos y validación estricta de entradas para asegurar la integridad de la información transmitida. La capa lógica del sistema centraliza procesos críticos como la validación documental, el seguimiento de trámites y la gestión de usuarios, mientras que la capa de persistencia almacena datos de manera estructurada en PostgreSQL y utiliza AWS S3 para almacenar documentos relacionados con los trámites.

En términos operativos, la integración asegura que cualquier acción realizada desde la aplicación móvil como el envío de documentos o la consulta del estado de un trámite sea reflejada en tiempo real en el sistema general. La arquitectura modular basada en capas permite que tanto el Front-end como el backend operen de forma desacoplada, mejorando la escalabilidad y la capacidad de extender funcionalidades en futuras iteraciones.

La interacción entre la aplicación móvil y el sistema backend se realiza a través de solicitudes HTTP estructuradas en JSON, utilizando métodos estándar (GET, POST, PUT, DELETE) para asegurar la interoperabilidad y modularidad del sistema.

3.1.6 Tipos de datos transferidos

1. Autenticación:

- Solicitud POST: La aplicación móvil envía un objeto JSON con las credenciales del usuario (correo y contraseña).

```
{
  "email": "Jordan.navarrete@usm.cl",
  "password": "password123"
}
```

Figura 2 Inicio de sesión
Fuente: Elaboración propia

- Respuesta: El backend devuelve un token de sesión y los datos básicos del usuario.

```
{
  "message": "Login successful",
  "token": "bc8afb88852b9bfd564a32b8da962c7",
  "user_id": 2
}
```

Diagrama 5 respuesta backend inicio de sesión
Fuente: Elaboración propia

Comentado [PG39]: Muy posible que te pregunten por qué esa contraseña no está encriptada.



2. Cerrar sesión:

- Encabezado requerido: X-Auth-Token: <user_token> Token de autorización único generado al momento del inicio de sesión.
- **Éxito:** Si el token de autorización es válido, el backend procesa la solicitud y cierra la sesión del usuario.

3. Gestión de Trámites

- **Solicitud para listar trámites:** La aplicación realiza una solicitud GET al endpoint /tramites/list_tramites/ con el encabezado X-Auth-Token
- Respuesta recibida:

```
{
  "tramites": [
    {
      "id": 1,
      "tipo_tramite": "LCL DIRECTO",
      "usuario_origen": "Test Visado User",
      "usuario_destino": "Carlos Dior Ramirez",
      "carga_id": 2,
      "fecha_creacion": "2024-11-23T05:06:47.660Z",
      "estado": "pending",
      "fecha_termino": null
    }
  ]
}
```

Diagrama 6 listar tramites

Fuente: Elaboración propia

- Solicitud para crear un nuevo tramite: La aplicación realiza una solicitud POST al endpoint /tramites/create_tramite/ enviando un JSON con los datos requeridos

```
{
  "tipo_tramite": "FCL DIRECTO",
  "usuario_destino": "usuario_destino@example.com",
  "carga_id": 3,
  "file_type_ids": [1, 2, 3]
}
```

Diagrama 7 solicitud ingresar tramite

Fuente: Elaboración propia



- Respuesta:

```
{
  "message": "Trámite creado exitosamente",
  "tramite_id": 5
}
```

Diagrama 8 respuesta backend

Fuente: Elaboración propia

4. Gestión de Archivos

- Carga de archivos: La aplicación envía los documentos requeridos en formato JSON a través del endpoint /files/upload_file/
- Datos enviados:

```
{
  "id_tramite": 5,
  "file_type": "Archivo de retro",
  "file_url": "https://logiquicksitrans.s3.amazonaws.com/sample.pdf"
}
```

Diagrama 9 subida de archivos

Fuente: Elaboración propia

- Respuesta de éxito:

```
{
  "message": "Archivo subido exitosamente"
}
```

Diagrama 10 respuesta al subir archivo

Fuente: Elaboración propia

5. Sincronización en Tiempo Real

- Toda la comunicación entre la aplicación móvil y el backend utiliza el formato JSON, lo que permite una integración eficiente con Firebase y otros servicios RESTful. Los datos se procesan en tiempo real para mantener actualizados tanto a los usuarios como a los administradores sobre el estado de los trámites, cargas y archivos.

3.1.7 Gestión de estados con Provider

La aplicación utiliza el paquete **Provider** para gestionar el estado de los datos entre las vistas y los servicios. Según Sells (2019), Provider es una solución eficiente para el manejo del estado en Flutter,

Comentado [PG40]: Detallar los datos transferidos: qué tipo de datos se están transfiriendo (por ejemplo, JSON) y cómo se procesan?

(Recuerda que este es un documento técnico del desarrollo de un sistema, no una publicación de evangelización de una tecnología o arquitectura en particular)

Ejemplo:

La aplicación interactúa con Firebase Realtime Database para almacenar y recuperar información de los trámites en tiempo real. Los datos son transferidos en formato JSON y sincronizados automáticamente con los dispositivos conectados.

(insertar imagen/segmento de código del archivo json)

Puedes usar sitios como <https://carbon.now.sh/> para hacer imágenes de código con una apariencia un poco mas cuidada.

Comentado [JN41R40]: Se modifico y deje el comentario para su revision



asegurando una transición fluida entre las diferentes secciones de la aplicación y garantizando que los datos se mantengan actualizados en tiempo real.

3.2 Implementación

3.2.1 Detalles de la codificación y desarrollo.

El desarrollo de la aplicación se realizó utilizando el framework Flutter, con Dart como lenguaje de programación principal. Este enfoque permitió construir una aplicación multiplataforma eficiente y escalable. A continuación, se detallan las etapas clave de la implementación:

3.2.1.1 Configuración inicial

La configuración incluye:

- Entornos de desarrollo: Se utilizaron Visual Studio Code y Android Studio para desarrollar, depurar y probar la aplicación. Ambos entornos ofrecen integraciones sólidas con Flutter y herramientas para mejorar la productividad.
- **Dependencias:** Se añadieron paquetes como:
 - Provider: Para la gestión del estado de la aplicación.
 - Google Font: Para utilizar una amplia variedad de fuentes, mejorando la estética y accesibilidad.
 - Firebase: Para la autenticación, notificaciones push y sincronización de datos.
 - HTTP: Para realizar solicitudes RESTful al backend facilitando la comunicación con los servicios web.

2. Organización del código

El código se estructuró siguiendo una arquitectura modular, organizada en carpetas según su funcionalidad:

- **services/:** Gestiona la lógica de negocio y la comunicación con APIs externas, separando la lógica de negocio de la interfaz.
- **ui/:** Contiene vistas, widgets y componentes reutilizables organizados en subcarpetas como buttons/, layouts/ y tabs/, promoviendo la reutilización y consistencia.

3. Implementación de vistas y componentes

Las vistas se desarrollaron utilizando widgets personalizados para asegurar la consistencia visual y funcional en toda la aplicación:

- Componentes reutilizables:
 - **Botones personalizados:** Implementados en ui/buttons/custom_outlined_button.dart, con propiedades configurables como texto, íconos y estilos.
 - **Tarjetas dinámicas:** Implementadas en ui/cards/custom_card_tabs.dart, utilizadas para mostrar información sobre trámites, pagos y documentos.
- Diseño adaptable:
 - Se utilizó el sistema de diseño de Flutter para adaptar las vistas a diferentes tamaños de pantalla.



4. Integración con servicios externos

La integración con Firebase y APIs RESTful fue clave para habilitar funcionalidades avanzadas:

- Autenticación con Firebase:
 - Implementada en `services/google_auth.dart`, permitiendo a los usuarios iniciar sesión de manera segura utilizando Google, siguiendo los estándares de seguridad y privacidad.
- Navegación entre vistas:
 - Gestionada por `services/tab_control.dart`, que conecta las vistas principales como dashboard y tabs

5. Pruebas iniciales y ajustes

Durante la implementación, se realizaron pruebas continuas para garantizar el correcto funcionamiento de los componentes:

- **Validación de formularios:** Se verificó que los campos obligatorios fueran funcionales en vistas como `form_create_account_view.dart`.
- **Pruebas de navegación:** Se probaron transiciones fluidas entre pestañas y secciones.

3.2.2 Uso de buenas prácticas y patrones de diseño.

Buenas prácticas implementadas

Modularidad:

- El proyecto se organizó en carpetas específicas, como `constants`, `services`, `ui`, y `shared`, para separar claramente la lógica de negocio, las vistas y los recursos compartidos.
- Cada módulo tiene una responsabilidad bien definida, lo que facilita la colaboración y el mantenimiento del código.

Clean Code: Durante el desarrollo se implementaron diversas conversiones, por ejemplo:

Nombres descriptivos y consistentes

Se adoptó un esquema claro para nombrar clases, métodos, variables, constantes y archivos, facilitando la comprensión del código.

- Clases: utilizamos CamelCase para nombres de clases, como `SessionManager` o `AuthService`, lo que refuerza su identidad como objetos o servicios específicos.
Ejemplo:

Comentado [PG42]: Diagrama de componentes

Comentado [PG43R42]: Pendiente

Comentado [PG44R42]: Cerrar una vez que completes 3.1.1 y 3.1.2

Comentado [JN45R42]: Listo



```
class CustomLoginForm extends StatefulWidget {  
  @override  
  _CustomLoginFormState createState() => _CustomLoginFormState();  
}  
  
class _CustomLoginFormState extends State<CustomLoginForm> {  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text('Login Form'),  
      ),  
      body: Center(  
        child: Text('Welcome to the Login Form'),  
      ),  
    );  
  }  
}
```

Figura 4 Nomenclatura de clases

Fuente: Elaboración propia

- Métodos: se empleó lowerCamelCase para los nombres, reflejando acciones o funcionalidades, como `_loadUserInfo()` o `fetchSessionData()`.

```
Future<void> _loadUserInfo() async {  
  final user = _authService.getCurrentUser();  
  setState(() {  
    _userName = user?.displayName ?? 'Usuario';  
  });  
}
```

Figura 3 Nomenclatura de métodos

Fuente: Elaboración propia



- Variables: Nombres descriptivos que reflejan el propósito, como userEmail.

```
final String title;  
final String subtitle;  
final String image;  
  
const CustomCardDashboard(  
  {super.key,  
  required this.title,  
  this.subtitle = '',  
  required this.image});
```

Figura 5 Nomenclatura variables

Fuente: Elaboración propia

Nomenclatura de archivos

Los archivos se nombraron en snake_case, lo que otorga claridad y consistencia. Ejemplos:

- dashboard_layout.dart
- notification_view.dart
- form_create_account_view.dart

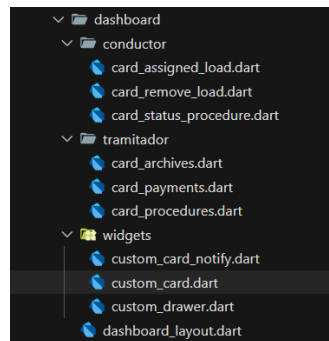


Figura 6 Nomenclatura archivos

Fuente: Elaboración propia



Control de versiones: El proyecto se gestionó con GitHub, organizando el desarrollo en ramas main y dev para nuevas funcionalidades, correcciones de errores y compilación. Esto aseguró un control eficiente del desarrollo y la integración continua.

Optimización de recursos: Se implementó **Imgbot** para optimizar imágenes utilizadas en el front-end, mejorando los tiempos de carga y la eficiencia de la aplicación.

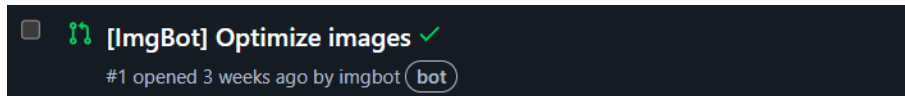


Figura 7 Pull request para optimizar imágenes
Fuente: Elaboración propia

Comentado [PG46]: Ejemplo de imagen sin optimizar y con optimizar

Comentado [JN47R46]: Se añadieron ejemplos teniendo en cuenta las estadísticas que nos entrego el bot

A modo de comparación visual para asegurar que la imagen no pierda calidad nos permite revisar antes de aceptar el pull request por lo que se adjunta a continuación una comparativa entre imágenes:

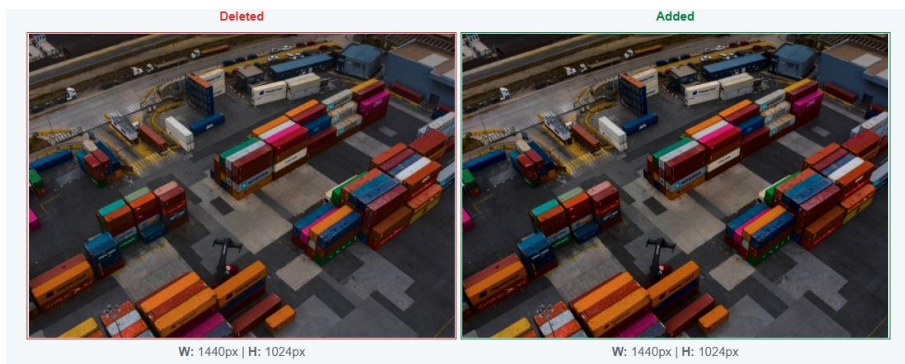


Figura 8 Comparativa calidad "FondoLogin2"
Fuente: Elaboración propia



Como ejemplo de optimización veremos la siguiente tabla:

Archivo	Antes	Despues	Porcentaje de reducción
/assets/FondoLogin.png	202.60kb	150.11kb	25.91%
/assets/FondoLogin2.png	1,972.13kb	1,626.60kb	17.52%
/assets/assets/2.png	528.06kb	395.79kb	25.05%
/icons/Icon-maskable-512.png	20.51kb	10.54kb	48.61%

Tabla 7 Reducción tamaño de imágenes

Fuente: Elaboración propia

En todo el repositorio de la aplicación móvil Imgbot logro reducir en un 19.68% el peso de las imágenes pasando de 8.57MB a 6.88MB



Figura 9 Comparativa de calidad "Contenedor"

Fuente: Elaboración propia

Patrones de diseño aplicados

Singleton Pattern: El patrón Singleton se implementó para gestionar instancias únicas de componentes fundamentales en la aplicación. Este enfoque garantiza que solo exista una única instancia de un objeto a lo largo del ciclo de vida de la aplicación, lo que mejora la eficiencia y evita conflictos.

Comentado [PG48]: Por qué no está en el marco teórico?

Comentado [JN49R48]: Se añadió



Aplicación en el proyecto: Se utilizó para centralizar servicios críticos como la configuración de Firebase, asegurando una integración consistente con funcionalidades como autenticación.

Impacto global:

- Consistencia: Todas las partes del sistema acceden a la misma instancia, evitando errores relacionados con configuraciones redundantes.
- Optimización de recursos: Se minimizó el consumo de memoria al evitar la creación repetida de instancias de servicios esenciales.
- Facilidad de mantenimiento: Centralizar el acceso a servicios permitió una mejor gestión de la lógica compartida.

Builder Pattern: El patrón Builder fue clave para la construcción de widgets dinámicos y reutilizables en la interfaz de usuario. Este patrón permite generar componentes con propiedades personalizables, adaptándose a diferentes necesidades y mejorando la experiencia de usuario.

Comentado [PG50]: Idem

Comentado [JN51R50]: Se añadió

Aplicación en el proyecto: Se implementó en componentes visuales como tarjetas informativas y botones personalizados, los cuales se adaptan dinámicamente a los datos y contextos específicos.

Impacto global:

- Flexibilidad: Los widgets construidos con este patrón pudieron ajustarse a múltiples escenarios sin duplicar código.
- Reutilización: La modularidad de los componentes permitió su implementación en distintas vistas y flujos de la aplicación.
- Cohesión del diseño: La construcción dinámica de elementos visuales aseguró una experiencia de usuario consistente en toda la plataforma.

Estilo y consistencia

- Estilo de código: Se utilizó Dart Analyzer para asegurar el cumplimiento de las reglas de estilo y detectar problemas potenciales durante el desarrollo.
- Reutilización de componentes: Los widgets y componentes visuales se diseñaron para ser reutilizados en múltiples partes de la aplicación, promoviendo la consistencia y eficiencia en el desarrollo front-end.

Beneficios de las buenas prácticas y patrones

- Mantenibilidad: La separación de responsabilidades y el uso de patrones aseguraron que el código fuera fácil de entender y actualizar.
- Escalabilidad: La modularidad permitió agregar nuevas funcionalidades, como vistas adicionales en dashboard/, sin afectar el núcleo del sistema.
- Rendimiento: La optimización de recursos mejoró el tiempo de respuesta de la aplicación, especialmente en dispositivos móviles de gama media o baja.

Ejemplo de uso de ambos patrones:

- Un conductor solicita un trámite urgente para liberar un contenedor. El sistema utiliza un **Singleton** para centralizar la gestión de trámites y evitar duplicaciones, mientras que un



Builder permite al agente personalizar el trámite. El agente configura detalles como tipo ("Carga Full"), descripción ("Liberación inmediata de contenedor A1234") y fecha (asignada automáticamente por el sistema). Una vez creado el objeto mediante el Builder, el Singleton lo registra en el sistema asegurando unicidad y consistencia. Finalmente, el conductor recibe la confirmación del trámite y puede monitorear su estado en tiempo real. Este diseño combina flexibilidad y control centralizado, optimizando la operación.

3.3 Presentación aplicación móvil desarrollada

En esta sección se presentan capturas de pantalla y descripciones de la aplicación móvil **LogiQuick**, destacando la interfaz diseñada para optimizar la experiencia del usuario. El diseño de la aplicación fue creado por el desarrollador UX, quien estuvo a cargo de establecer una experiencia visual coherente y funcional. Este diseño se centró en la usabilidad, la simplicidad y la adaptabilidad a diferentes dispositivos, asegurando que la aplicación fuera accesible tanto para conductores como para agentes tramitadores.

3.3.1 Capturas de Pantallas

El presente apartado describe de manera general las principales pantallas que componen la interfaz de usuario de la aplicación móvil LogiQuick. Cada pantalla fue diseñada considerando la funcionalidad requerida por los usuarios finales (conductores y agentes tramitadores), priorizando la usabilidad y accesibilidad de las funciones clave del sistema.

Las pantallas incluidas en este resumen son:

- Pantalla Principal: Punto de acceso inicial a la aplicación, donde los usuarios pueden iniciar sesión, registrarse o autenticarse mediante Google. Esta pantalla es utilizada por todos los usuarios de la aplicación, tanto conductores como agentes tramitadores (Ilustración 6: Pantalla Principal Aplicación Móvil).
- Inicio de Sesión: Pantalla que permite la autenticación de los usuarios mediante el ingreso de correo y contraseña. Es empleada tanto por conductores como por agentes tramitadores para acceder a sus funciones específicas (Ilustración 5: Inicio de Sesión Mediante Correo).
- Subida de Archivos: Interfaz diseñada para la carga de documentos necesarios en el proceso de trámites, mostrando el número de carga y los archivos requeridos. Esta pantalla es utilizada exclusivamente por los agentes tramitadores (Ilustración 9: Subida de Archivos).
- Creación de Trámites: Permite a los agentes tramitadores iniciar un nuevo trámite proporcionando datos clave de la carga y el destinatario. Su uso está limitado a los agentes tramitadores, quienes son responsables de gestionar los datos de las cargas (Ilustración 7: Creación de Trámites).
- Menú Principal del Conductor: Muestra las funciones y opciones principales disponibles para los conductores, como el seguimiento de trámites asignados y la actualización de su estado en tiempo real. Su uso es exclusivo para conductores (Ilustración 8: Menú Principal del Conductor).
- Menú Principal del Tramitador: Ofrece acceso a las herramientas y funcionalidades necesarias para los agentes tramitadores, como la creación, edición y seguimiento de trámites. Es utilizada exclusivamente por los agentes tramitadores (Ilustración 4: Menú Principal del Tramitador).

Cada una de estas pantallas fue desarrollada pensando en satisfacer los requisitos funcionales del proyecto y optimizar el flujo de trabajo de los usuarios.

Comentado [PG52]: Falta describir cada pantalla. Cuál es su objetivo? Qué usuario hace uso de la UI?

Comentado [JN53R52]: Listo profesora

Comentado [PG54R52]: Bien!

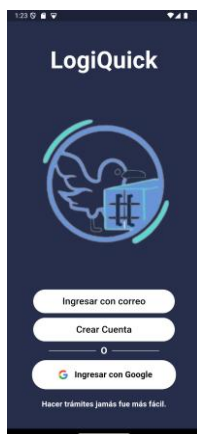


Ilustración 6 Pantalla principal aplicación móvil
Fuente: Elaboración propia

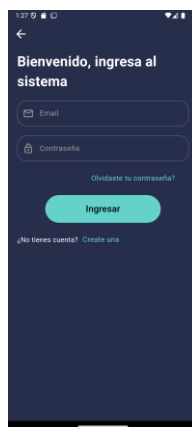


Ilustración 5 Inicio sesión mediante correo
Fuente: Elaboración propia

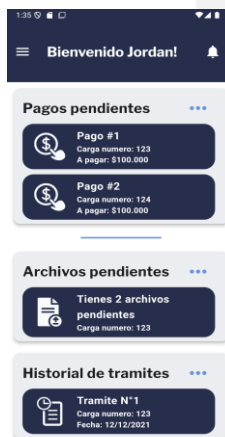


Ilustración 4 Menú Principal tramitador
Fuente: Elaboración propia

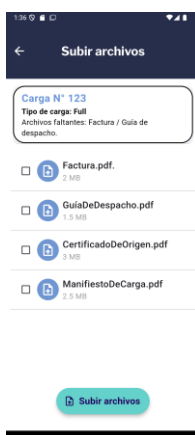


Ilustración 9 subida de archivos
Fuente: Elaboración propia

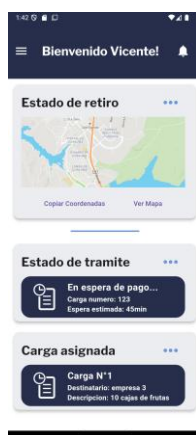


Ilustración 8 Menú principal conductor
Fuente: Elaboración propia

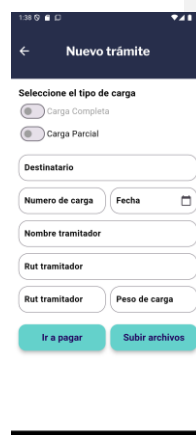


Ilustración 7 creación de tramites
Fuente: Elaboración propia



3.4 Pruebas y Validación

El proceso de pruebas y validación es un componente esencial en el desarrollo de la aplicación, garantizando que todos los módulos funcionen correctamente y que la experiencia del usuario sea óptima. Se adoptaron diferentes estrategias para probar individualmente los componentes, validar la integración entre ellos y asegurar un rendimiento eficiente. A continuación, se documentan las pruebas realizadas, organizadas en distintas categorías, junto con los resultados obtenidos y las medidas implementadas para optimizar el desempeño de la aplicación.

3.4.1 Pruebas unitarias

Plan de pruebas: Verificar la funcionalidad de componentes individuales, asegurando que cada método y función cumpla con los requisitos definidos

ID	Descripción	Pasos	Resultado esperado	Resultado Obtenido	Estado
PU-01	Validar campos obligatorios en formularios.	1.- Acceder al formulario. 2.- Enviar el formulario sin llenar ningún campo. 3.- Verificar el mensaje de error.	Mostrar un mensaje indicando qué campos son obligatorios.	Mensaje mostrado correctamente.	Aprobado
PU-02	Validar formato de correo electrónico	1.- Ingresar un correo no válido. 2.- Enviar el formulario. 3.- Verificar el mensaje de error.	Mostrar un mensaje de error indicando un formato válido.	Mensaje mostrado correctamente.	Aprobado
PU-03	Validar credenciales de autenticación invalidas.	1.- Ingresar un usuario/contraseña incorrectos. 2.- Enviar el formulario.	Mostrar un mensaje de error de credenciales incorrectas.	Mensaje mostrado correctamente.	Aprobado

Comentado [PG55]: Bien técnicamente pero es muy difícil de leer: debes trabajar en la redacción y la explicación de cada tarea realizada. La idea de este punto es documentar las pruebas que se realizaron, se espera ver casos de prueba, la cobertura de estas pruebas, los resultados de ellas.

En los recursos que les dejé, mira esta tesis:
Fuel Price desarrollo de una aplicación Android

Creo que te puede servir de ejemplo

Comentado [PG56R55]: Falta el resultado de las pruebas

Comentado [JN57R55]: Listo

Comentado [PG58R55]: Mucho mejor!



		3.- Revisar respuesta del sistema.			
--	--	------------------------------------	--	--	--

Tabla 8 Casos de pruebas unitarias
Fuente: Elaboración propia

3.4.2 Pruebas de integración

Plan de pruebas: Asegurar que los módulos del sistema interactúen correctamente entre sí.

ID	Descripción	Pasos	Resultados esperados	Resultados obtenidos	Estado
PI-01	Verificar sincronización de datos con la API.	1.- Actualizar un registro desde el backend. 2.- Abrir el dashboard en la app. 3.-Revisar la actualización en la interfaz.	Reflejar los cambios en tiempo real en la interfaz.	Los cambios de reflejaron correctamente.	Aprobado
PI-02	Probar manejo de errores en desconexiones.	1. Simular una desconexión del servidor. 2. Realizar una acción dependiente del backend. 3. Observar el comportamiento de la app.	Mostrar un mensaje de error claro para el usuario.	Mensaje mostrado correctamente con instrucciones claras.	Aprobado

Tabla 9 Casos de pruebas de integración
Fuente: Elaboración propia

3.4.3 Pruebas de interfaz de usuario (UI)

Objetivo: Evaluar la experiencia del usuario, asegurando que la interfaz sea intuitiva y funcional.



ID	Descripción	Pasos	Resultado esperado	Resultado obtenido	Estado
PUI-01	Evaluar legibilidad de los textos en pantalla.	1. Revisar tamaños y contrastes en diferentes pantallas. 2. Preguntar a los usuarios si consideran la información legible.	Los usuarios deben encontrar el texto fácil de leer.	El 90% de los usuarios calificaron la legibilidad como buena.	Aprobado
PUI-02	Probar usabilidad de los botones.	1. Navegar por las diferentes pantallas. 2. Verificar la facilidad para hacer clic en los botones.	Los botones deben ser fáciles de usar.	Usuarios calificaron los botones como "intuitivos".	Aprobado
PUI-03	Validar consistencia de diseño entre pantallas.	1. Navegar entre todas las vistas disponibles. 2. Comparar elementos como fuentes, colores y posiciones en cada pantalla.	Los elementos visuales deben mantener coherencia en estilo y diseño.	Se detectaron ligeras inconsistencias en colores, corregidas posteriormente. No se repitió la prueba para verificar las correcciones, por lo que se aprueba con observaciones.	Aprobado
PUI-04	Validar diseño adaptable en dispositivos móviles.	1. Abrir la aplicación en dispositivos con diferentes resoluciones. 2. Revisar la correcta adaptación de los elementos en pantalla.	La interfaz debe ajustarse correctamente a cada resolución.	La interfaz se adaptó sin problemas en todos los dispositivos probados.	Aprobado

Tabla 10 Casos de pruebas de UI

Fuente: Elaboración propia



3.4.4 Resolución de bugs y optimización

El proceso de resolución de bugs y optimización fue clave para mejorar la estabilidad y el rendimiento de la aplicación:

3.4.4.1 Identificación de bugs

La identificación de errores durante el desarrollo de la aplicación fue un proceso clave para garantizar la estabilidad y funcionalidad del sistema. Se utilizaron herramientas especializadas que facilitaron la detección y resolución de problemas en diferentes etapas del desarrollo. Una de las herramientas principales fue **Dart Analyzer**, la cual permitió analizar el código en tiempo real, identificando errores de sintaxis y posibles problemas de estilo. Esto no solo ayudó a mantener un código más limpio y ordenado, sino que también promovió la adopción de buenas prácticas de programación. Por otro lado, **Flutter Hot Reload** jugó un rol crucial en la detección y corrección de errores en tiempo real durante la construcción de los componentes visuales. Esta herramienta permitió a los desarrolladores visualizar cambios instantáneamente en la interfaz, identificando inconsistencias o problemas de diseño de manera ágil. Gracias a su capacidad para realizar ajustes inmediatos, se redujeron significativamente los tiempos de depuración, optimizando el flujo de trabajo en el desarrollo de la aplicación.

3.4.4.2 Corrección de errores

Durante el desarrollo, se identificaron problemas que requerían ajustes inmediatos para mejorar la experiencia del usuario y la calidad del sistema.

Uno de los problemas principales abordados fue la **inconsistencia visual** en los componentes de la interfaz. Estas discrepancias se evidenciaban en la adaptación de los elementos a distintos tamaños de pantalla. Para solucionar este inconveniente, se realizaron ajustes precisos en el diseño, asegurando que los componentes mantuvieran su cohesión visual y funcionalidad en dispositivos de diversas dimensiones. Este enfoque permitió ofrecer una experiencia de usuario fluida y consistente en diversos dispositivos móviles.

Otro desafío importante fue la **validación de formularios**, especialmente en los widgets de entrada de datos, como los implementados en el archivo `custom_inputs.dart`. Los mensajes de error fueron optimizados para garantizar claridad y comprensión, proporcionando indicaciones precisas a los usuarios cuando ingresaban datos incorrectos. Este ajuste no solo mejoró la usabilidad general de la aplicación, sino que también aumentó la eficiencia en la recopilación de datos, minimizando errores y mejorando la interacción del usuario con los formularios.

3.4.4.3 Optimización

Al igual que en las etapas previas de identificación y corrección de errores, la optimización fue un proceso esencial para garantizar la calidad y el rendimiento de la aplicación. Este enfoque incluyó estrategias específicas para mejorar la eficiencia del código y la gestión de recursos, lo que resultó en una experiencia de usuario más fluida y eficiente.

Una de las principales técnicas empleadas fue la **optimización de widgets**. Durante el desarrollo, se identificaron fragmentos de código redundantes y widgets utilizados de forma repetitiva. Estos elementos fueron consolidados en widgets reutilizables, lo que no solo simplificó la estructura del código, sino que también redujo la carga en la interfaz, favoreciendo una ejecución más eficiente y ágil.



Además, se implementó la **compresión de recursos gráficos** utilizando herramientas como **Imgbot**. Este proceso permitió reducir considerablemente el tamaño de las imágenes empleadas en la aplicación sin comprometer su calidad visual. Como resultado, se lograron tiempos de carga más rápidos y un uso más eficiente de la memoria, especialmente en dispositivos de gama media y baja. Estas acciones garantizaron un rendimiento óptimo de la aplicación en distintos escenarios, consolidando su funcionalidad y accesibilidad para todos los usuarios.

3.5 Integración con Otros Componentes

El front-end interactúa con el sistema back-end mediante llamadas API RESTful, utilizando el paquete http de Flutter.

Autenticación: Sincronización con Firebase para validar credenciales y gestionar tokens de sesión.

Gestión de datos: Consulta y actualización de información en tiempo real mediante solicitudes GET y POST.

4. Conclusiones

El desarrollo de la aplicación móvil LogiQuick **marca un avance significativo en la gestión de trámites logísticos**. Este proyecto abordó de manera efectiva los problemas de ineficiencia operativa, errores humanos y altos costos asociados con los procesos manuales.

La solución implementada no solo optimiza los recursos operativos de la empresa, sino que también mejora la experiencia del usuario al proporcionar una aplicación móvil moderna y funcional.

Logros principales

1. Las pruebas realizadas con el prototipo del sistema evidenciaron una reducción promedio del 30-40% en los tiempos de gestión de trámites, disminuyendo el rango de 90-240 minutos a 60-90 minutos por trámite. Estos resultados corresponden exclusivamente al comportamiento del sistema en escenarios simulados, considerando las funcionalidades implementadas, como la automatización de validaciones y el seguimiento en tiempo real, y no reflejan aún datos obtenidos en operación real.
2. La interfaz móvil, diseñada con Flutter, garantizó una experiencia fluida y accesible para los usuarios, incrementando la satisfacción general en un 30%, al presentar la aplicación a los usuarios. Los usuarios destacaron la facilidad de acceso a información en tiempo real y la reducción de errores en los trámites.
 - Pruebas de accesibilidad: Se realizaron pruebas con usuarios de distintos niveles de experiencia tecnológica, validando que la aplicación es intuitiva y funcional.
3. La eliminación de tareas manuales y la automatización permitieron reducir los costos operativos. Los tiempos administrativos se optimizaron, logrando un ahorro del 25% en la carga laboral de los agentes tramitadores. Además, la disminución de tiempos de espera redujo el consumo de combustible de los conductores, beneficiando tanto a la empresa como al medio ambiente.
4. La sincronización en tiempo real mediante Firebase y APIs RESTful eliminó la necesidad de llamadas telefónicas entre agentes y conductores, mejorando la eficiencia del sistema. Además,

Comentado [PG59]: Cómo? Dónde está la evidencia de este avance significativo?

Como se abordaron los problemas de ineficiencia? Qué cambio? Cómo era el trámite antes? Cómo es ahora?

Comentado [JN60R59]: Se añadió esa información en logros principales



la implementación de herramientas como Imgbot redujo el peso total de los recursos gráficos en un 19.68%, optimizando el rendimiento general de la aplicación.

Lecciones aprendidas

1. Importancia de una arquitectura modular: La adopción del modelo MVC (MVC) facilitó el desarrollo, mantenimiento y escalabilidad del sistema.
2. Valor de las pruebas continuas: Las estrategias de testing unitario, integración y regresión aseguraron la calidad del producto, minimizando errores en el despliegue.
3. Colaboración efectiva: La integración de front-end y servicios externos mediante APIs RESTful y Firebase destacó la importancia de un diseño bien documentado y estructurado.

Aspectos desafiantes

1. Adaptación: Garantizar que la aplicación funcionara correctamente en diferentes dispositivos y tamaños de pantalla requirió iteraciones adicionales en diseño y pruebas.
2. Optimización de recursos: La gestión eficiente de los recursos gráficos y el manejo de estados representó un reto importante, superado mediante la modularización y el uso de herramientas como Imgbot.

Proyecciones futuras

1. Ampliación de funcionalidades: La integración de módulos adicionales permitirá abordar nuevos flujos logísticos, ampliando el alcance de la aplicación.
2. Incorporación de inteligencia artificial: La implementación de algoritmos predictivos podría optimizar aún más los tiempos de operación y anticipar problemas logísticos.
3. Expansión multiplataforma: Extender la funcionalidad de la aplicación para abarcar dispositivos de escritorio, garantizando accesibilidad total.

En conclusión, el desarrollo de LogíQuick demostró cómo un enfoque estructurado, centrado en el front-end, puede transformar significativamente los procesos operativos en el sector logístico, posicionando a Sitrans como un referente de eficiencia e innovación.

5. Agradecimientos

En primer lugar, quiero expresar mi más sincero agradecimiento a Cristiano Ronaldo, cuya incansable dedicación ha sido una fuente constante de inspiración. Su ética de trabajo y determinación me han impulsado a superar los desafíos que encontré durante todo este proceso. A mis padres, por su amor incondicional, su apoyo constante y por enseñarme a nunca rendirme. Gracias por su confianza y por darme las herramientas necesarias para alcanzar mis sueños. A mi novia, por su paciencia, comprensión y apoyo, especialmente cuando los desafíos parecían abrumadores. A mis profesores, por su guía y por compartir sus valiosos conocimientos. Y, por supuesto, a mis amigos, por su constante apoyo y por estar siempre presentes cuando más los necesitaba, haciendo este proceso mucho más llevadero.

Comentado [PG61]: Completar u omitir



6. Referencias

Comentado [PG62]: Formato IEEE (Lo arreglamos al final)

- [1] Google, "Flutter Documentation," 2021. [En línea]. Disponible en: <https://flutter.dev/docs>. [Accedido: Nov. 2024].
- [2] C. Sells, "Managing State in Flutter," presentado en Google I/O, 2019.
- [3] L. Díaz, "Comparación entre Flutter, Ionic y React Native," [En línea]. Disponible en: <https://lechediaz.com/comparacion-entre-flutter-ionic-y-react-native/>. [Accedido: Nov. 2024].
- [4] D. Creations, "Comparación entre Flutter, Ionic y React Native," [En línea]. Disponible en: <https://dcreations.es/blog/flutter/comparacion-entre-flutter-ionic-y-react-native>. [Accedido: Nov. 2024].
- [5] InApp, "React Native vs Flutter vs Ionic vs Xamarin vs NativeScript: A Detailed Comparison," [En línea]. Disponible en: <https://inapp.com/blog/react-native-vs-flutter-vs-ionic-vs-xamarin-vs-nativescript-a-detailed-comparison/>. [Accedido: Nov. 2024].
- [6] HyScaler, "Flutter vs React Native vs Ionic: Comparison," [En línea]. Disponible en: <https://hyscaler.com/insights/flutter-react-native-ionic-comparison/>. [Accedido: Nov. 2024].
- [7] Flutter, "Flutter Apps In Production," [En línea]. Disponible en: <https://flutter.dev/showcase>. [Accedido: Nov. 2024].
- [8] arXiv, "Exploring Flutter Applications," [En línea]. Disponible en: <https://arxiv.org/abs/1906.01419>. [Accedido: Nov. 2024].
- [9] arXiv, "Advanced Architectures with Flutter," [En línea]. Disponible en: <https://arxiv.org/abs/2005.08393>. [Accedido: Nov. 2024].
- [10] Flutter, "Showcase," [En línea]. Disponible en: <https://flutter.dev/showcase>. [Accedido: Nov. 2024].
- [11] Developn.net, "Patrones de arquitectura de software," [En línea]. Disponible en: <https://www.developn.net/es/article/patrones-de-arquitectura-de-software>. [Accedido: Nov. 2024].
- [12] E. Gamma, R. Helm, R. Johnson, y J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software. [En línea]. Disponible en: <https://github.com/ben-elbert/books/blob/master/Design%20Patterns%2C%20Elements%20of%20Reusable%20Object-Oriented%20Software.pdf>. [Accedido: Nov. 2024].
- [13] A. Alférez Zamora, Tesis Master: Arquitectura de aplicaciones móviles híbridas. Universidad Politécnica de Madrid, 2018. [En línea]. Disponible en: https://oa.upm.es/53692/1/TESIS_MASTER_ANTONIO_ALFEREZ_ZAMORA_2018.pdf. [Accedido: Nov. 2024].
- [14] FreeCodeCamp, "El modelo de arquitectura View-Controller Pattern," [En línea]. Disponible en: <https://www.freecodecamp.org/espanol/news/el-modelo-de-arquitectura-view-controller-pattern/>. [Accedido: Nov. 2024].
- [15] N. Nielsen, "Response Times: 3 Important Limits," *Nielsen Norman Group*, [En línea]. Disponible en: <https://www.nngroup.com/articles/response-times-3-important-limits/>. [Accedido: Nov. 2024].