

# Especificación y desarrollo de aplicación móvil dirigida al mantenimiento de vehículos

---

## Autor

Julián Francisco Gómez Díaz

## Tutor

José Ambrosio Toval Álvarez

*Departamento de Informática y Sistemas*



Grado en Ingeniería Informática



Murcia, 30 de Junio de 2025



# Agradecimientos

Este trabajo no habría sido posible sin el apoyo, la compañía y la inspiración de muchas personas a lo largo de estos años. A todas ellas, gracias.

En primer lugar, quiero agradecer profundamente a mis padres. Su amor incondicional, su paciencia infinita y su fe constante en mí han sido mi base a lo largo de toda esta etapa. Gracias por estar siempre ahí, en los momentos de duda y en los de alegría, por darme las herramientas para avanzar con confianza. Cada logro que he alcanzado es también reflejo del compromiso y el sacrificio que han hecho por mí desde el primer día.

A mi hermano mayor, por ser un ejemplo de constancia y responsabilidad. Su forma serena de ver la vida, su capacidad para relativizar los problemas y su manera tranquila de acompañar sin presionar me han ayudado en más ocasiones de las que él imagina. Le agradezco profundamente su apoyo silencioso pero firme, su confianza en mí, y sobre todo, su manera de estar: sin hacer ruido, pero siempre estando.

A mis compañeros de carrera, con quienes compartí no solo clases y exámenes, sino también risas y momentos inolvidables. A Fabián, por su energía contagiosa y su inagotable optimismo; a Antonio, por su mente brillante y su infinita paciencia para explicarme una y otra vez lo que no entendía; a Manuel, mi compañero inseparable de trabajos, que con su exigencia amable y su confianza en mí me ha ayudado a crecer más de lo que imagina; a Pablo y Alejandro, porque más que compañeros han sido hermanos de camino. Nuestra amistad nació antes de la universidad, pero fue aquí donde se fortaleció de verdad: hemos vivido estos cuatro años juntos, hombro con hombro, apoyándonos en los momentos difíciles, compartiendo alegrías, frustraciones y muchas tardes que empezaban con apuntes y terminaban con risas.

A mi tutor, José Ambrosio Toval Álvarez, por su guía paciente, su disponibilidad y su capacidad de orientar. Gracias por respetar mi proceso y enriquecerlo con sus sugerencias siempre certeras.

A mi grupo de amigos, Plan X, por todas las conversaciones absurdas, los planes improvisados y la amistad genuina que me ofrecieron. Han sido un refugio y una alegría constante.

Y a quienes estuvieron a mi lado, brindándome apoyo y compañía, aunque hoy ya no formen parte de mi vida, también les agradezco. Su presencia dejó huella.

A todos ustedes, gracias. Este trabajo también les pertenece.



# **Declaración firmada sobre originalidad del trabajo**

D./Dña. **Julián Francisco Gómez Díaz**, con DNI **23832122M**, estudiante de la titulación de **Grado en Ingeniería Informática** de la Universidad de Murcia y autor del TF titulado “**Especificación y desarrollo de aplicación móvil dirigida al mantenimiento de vehículos**”.

De acuerdo con el Reglamento por el que se regulan los Trabajos Fin de Grado y de Fin de Máster en la Universidad de Murcia (aprobado C. de Gob. 30-04-2015, modificado 22-04-2016 y 28-09-2018), así como la normativa interna para la oferta, asignación, elaboración y defensa de los Trabajos Fin de Grado y Fin de Máster de las titulaciones impartidas en la Facultad de Informática de la Universidad de Murcia (aprobada en Junta de Facultad 27-11-2015)

DECLARO:

Que el Trabajo Fin de Grado presentado para su evaluación es original y de elaboración personal. Todas las fuentes utilizadas han sido debidamente citadas. Así mismo, declara que no incumple ningún contrato de confidencialidad, ni viola ningún derecho de propiedad intelectual e industrial

Murcia, a 30 de Junio de 2025



Fdo.: Julián Francisco Gómez Díaz  
Autor del TFG



# Resumen

El presente Trabajo Fin de Grado presenta la especificación, diseño e implementación de **Mi Garaje**, una aplicación móvil multiplataforma destinada a facilitar la gestión integral del mantenimiento de vehículos. El proyecto surge de la necesidad de contar con una herramienta digital, gratuita y sin restricciones que permita a los usuarios organizar de forma eficiente tareas como el mantenimiento preventivo, el control de gastos o la gestión de documentación relacionada con sus automóviles. A pesar de la existencia de diversas aplicaciones similares, la mayoría limita funcionalidades esenciales en sus versiones gratuitas, incluyen anuncios o carecen de opciones avanzadas de personalización y colaboración. Mi Garaje se plantea como una solución alternativa que elimina estas barreras y apuesta por una experiencia de usuario intuitiva, flexible y completa.

La aplicación ha sido desarrollada utilizando Flutter como framework, lo que permite generar aplicaciones nativas para Android e iOS a partir de un único código fuente, optimizando recursos y facilitando el mantenimiento. Para el backend se ha elegido Firebase, que aporta servicios de autenticación, base de datos en tiempo real y reglas de seguridad, permitiendo construir una arquitectura escalable y sin necesidad de servidores propios. Entre las características más destacadas de Mi Garaje se encuentra su sistema colaborativo mediante “familias”, que permite a varios usuarios gestionar conjuntamente uno o varios vehículos, compartiendo información y responsabilidades. Este sistema resulta especialmente útil en contextos como familias, parejas, compañeros de piso o pequeños grupos de trabajo.

La aplicación ofrece un alto grado de personalización: los usuarios pueden registrar un número ilimitado de vehículos, añadir nuevas categorías de actividades y crear subcategorías específicas, adaptando la herramienta a distintos tipos de vehículos, desde coches y motocicletas hasta furgonetas o caravanas. Las actividades pueden incluir datos como fecha, coste, observaciones e imágenes asociadas, lo que facilita conservar documentación como facturas o reportes técnicos.

La interfaz de usuario se ha diseñado siguiendo principios de usabilidad, con un diseño moderno, soporte para modo claro y oscuro, e internacionalización (inicialmente en español e inglés). Además, incluye un sistema de estadísticas que permite visualizar el historial de gastos y actividades mediante gráficos, ayudando al usuario a analizar el uso de su vehículo y tomar decisiones informadas. Los datos pueden exportarse en formato CSV sin limitaciones, una funcionalidad que otras aplicaciones suelen reservar para versiones de pago.

En cuanto a la seguridad, el proyecto implementa reglas estrictas en Firestore para proteger los datos y se apoya en Firebase Auth para la autenticación, permitiendo incluso el uso de cuentas anónimas que pueden vincularse posteriormente sin pérdida de información. Finalmente, Mi Garaje sienta las bases para futuras mejoras, como notificaciones push, copias de seguridad en la nube, mantenimiento predictivo con inteligencia artificial o un asistente de configuración inicial, lo que refuerza su potencial como solución tecnológica robusta y en evolución.

---

# **Abstract**

This Final Degree Project develops the design, specification, and implementation of a cross-platform mobile application called Mi Garaje (My Garage), whose purpose is to assist users in the comprehensive management of the maintenance of their vehicles. The origin of this proposal arises from the growing need to digitally and efficiently organize all activities related to the care, monitoring, and administration of automobiles, in an environment where the dependence on personal and professional transportation is increasingly significant. Modern automobiles require regular maintenance, document tracking, and expense control, aspects that are often carried out in a dispersed, informal way or are simply not recorded, which can result in technical problems, financial losses, or legal penalties due to expired documentation. Although there are multiple technological solutions available in app stores, a detailed analysis of the most popular ones reveals that the majority present significant limitations, especially in their free versions. These restrictions affect essential functionalities such as the management of multiple vehicles, access to statistics, customization of categories, or collaboration between users. Furthermore, many of these applications impose advertisements or have unintuitive interfaces. Faced with this situation, this project proposes a solution that is completely free, without advertisements, customizable, collaborative, and cross-platform, which combines the power of modern technologies with a carefully designed and accessible user experience.

To achieve this objective, Flutter was chosen as the development framework. Flutter allows the creation of native applications for Android and iOS from a single code base, which optimizes the necessary resources and facilitates maintenance. This technology, developed by Google, is recognized for its performance, development speed, and active community, characteristics that make it an ideal option for cross-platform projects. Regarding the backend, Firebase was chosen, offering authentication services, real-time database, and security rules. This technological choice has made it possible to build a scalable, robust, and secure architecture without the need to implement and maintain our own server. Firebase, being a serverless platform, allows focusing on business logic and user experience without having to worry about infrastructure.

One of the most innovative and distinctive elements of Mi Garaje is its collaborative approach. Through the “families” system, multiple users can jointly manage one or several vehicles. This is especially useful in real contexts such as families who have several vehicles and want to control all the vehicles together, couples who share vehicles, housemates, friends, or small work groups. Each family can invite new members

through a unique family code. All members can access vehicle information, record activities, modify data, view statistics, and export information. This functionality makes it possible to decentralize maintenance management, share responsibility, and avoid duplications or misunderstandings. In the development of this functionality, special attention has been paid to security and privacy, allowing precise control over the data accessible by each user and ensuring that there are no information leaks between families or unauthorized users.

In addition to this collaborative approach, Mi Garaje stands out for its high level of customization. The user can manage an unlimited number of vehicles, registering data such as make, model, and other relevant attributes. Various activities can be associated with these vehicles, classified into three main groups: maintenance, refueling, and documentation. For each of the predefined activities, there is a list of sub-activities, such as under maintenance “brakes,” “battery,” or “tires.” However, one of the most powerful aspects is that these categories are not rigid; rather, they can be deleted, and new types can be added that can be fully customized by the user. Thus, a person can create new subcategories such as “exterior wash,” “winter tire change,” or “emissions check.” This allows the application to adapt both to those who have a single urban vehicle and to those who manage motorcycles, vans, electric cars, or small fleets. Likewise, each activity allows recording the date, cost, observations, and attaching images, which is useful for keeping invoices, technical documents, photos of the vehicle’s condition, or evidence of repairs.

In addition to the customization of subtypes of activities, Mi Garaje also allows the customization of vehicle types and general activity categories. Although the default main categories — maintenance, refueling, and documentation — cannot be deleted, the user has the freedom to add new general categories of activities according to their needs. This further expands the flexibility of the application, adapting to very varied and particular uses. For example, someone could create an additional category such as “personalization” to manage aesthetic or mechanical modifications, or “trips” to record kilometers and expenses associated with long journeys. Similarly, vehicle types are not limited to the predefined ones, allowing the creation of new personalized types to better reflect the diversity of the user’s vehicle fleet, whether it be a classic motorcycle, a latest-generation electric vehicle, or a camper van.

The user interface has been designed following principles of usability and modern visual design. The aim has been for the experience to be intuitive, clear, and accessible for any user profile, regardless of their technical level. Navigation is structured through a bottom bar with tabs for the main functionalities: statistics, home, profile. Support has been implemented for light mode and dark mode, configurable automatically according to the system or manually by the user. Internationalization has also been included, starting with Spanish and English, but prepared to support more languages in future versions. The entire text system is managed through localization files, allowing the app to be translated without modifying the code. In addition, visual preferences (langua-

---

ge, theme) are stored locally using SharedPreferences, ensuring they are maintained between sessions.

A key functionality of the application is the statistics system. Through visual graphs (bars and pie charts), the user can analyze their maintenance and expense history. One can observe the monthly evolution of costs, the distribution of activities by type, the average cost per intervention, and other relevant data. These statistics allow the user to have a clear vision of how they use their vehicle, identify months with higher expenses, assess whether they carry out maintenance regularly enough, and make informed decisions. For example, if corrective maintenance costs are repeatedly high, it could be deduced that preventive maintenance is lacking. This functionality integrates smoothly with the rest of the application and updates in real time as new activities are added.

In addition to visualizing the data, the user can export them in CSV format, allowing them to work with them externally, either in spreadsheets, accounting applications, or simply as a backup. This export includes both the vehicle data and the registered activities, with their respective fields. This functionality, which is usually restricted to paid versions in other applications, is offered completely free of charge and without limitations. The generated file follows a structured and clean format, easily interpretable by data analysis tools. This export capability represents an added value for those who want greater control, transparency, or backup of their information.

Security is another critical aspect in the development of the project. All read and write operations in Firestore are protected through security rules that restrict access to documents according to the user's identity. A user cannot access another user's data unless they belong to the same family. Principles of least privilege are applied, and the database structure was designed to minimize duplications and avoid inconsistencies. Furthermore, authentication operations are managed by Firebase Auth, which offers secure integration with Google accounts, email, and anonymous sessions. The latter option allows the app to be used without registering, facilitating exploration before creating an account. Later, the anonymous account can be linked to a persistent account without losing the data already entered. This entire system guarantees a flexible, secure, and user-centered experience. Moreover, at any time, the account can be linked with Google without losing the data from the account.

During the validation process, functional tests were carried out on different Android devices, including emulators and physical mobiles. The tests covered complete flows of authentication, navigation, vehicle and activity creation, statistics, data export, family system, language and visual theme change, configuration persistence, and general performance. The results were satisfactory, with some minor errors detected and corrected during development. The system demonstrated stability, good responsiveness, and adequate adaptation to different resolutions and screen sizes. Thanks to this constant testing approach, a mature, usable, and goal-consistent product was achieved.

---

Regarding code organization, a clear and maintainable modular structure was followed. Models, Firebase services, business logic, state controllers using Riverpod, and interface widgets were separated. Good programming practices, use of explanatory comments, style conventions, and version control using Git were employed. In addition, technical documentation has been generated that includes system architecture, flow diagrams, database design, technological justification, usage examples, and explanations of the main modules. This documentation will facilitate future collaborations, maintenance, or publication as an open-source project.

On a formative level, the development of this project has been a comprehensive experience that has allowed putting into practice a wide range of knowledge acquired during the degree. Concepts of software design, requirements analysis, databases, mobile development, client-server architecture, state management, and interface design have been used. Likewise, transversal skills such as autonomous decision-making, problem-solving, adaptability, and attention to detail have been developed. The process has required autonomy, critical capacity, sustained effort, and result orientation, fundamental qualities in any professional environment. The ability to think in terms of user experience has also been strengthened, valuing the importance of design and anticipating the real impact of technical decisions.

As a final degree project, Mi Garaje not only meets the required academic criteria but also has real viability as a product. Its publication in app stores could benefit a wide spectrum of users: from individuals who only wish to keep basic control of their car to families, couples, small fleets, or industry professionals who need a reliable and free solution. In the future, different lines of improvement are proposed, such as the implementation of automatic reminders through push notifications, expansion to new languages, a cloud backup system, or an initial app setup assistant.

In conclusion, this Final Degree Project goes beyond a mere academic exercise; it represents a genuine technological proposal aimed at addressing a common, everyday need through an application that stands out for its thoughtful design, solid implementation, and considerable potential for widespread use. Mi Garaje positions itself as a reliable, entirely free solution without limitations, offering an alternative to a market filled with incomplete or paywalled options. Its collaborative philosophy, extensive customization options, user-friendly interface, and modern architecture demonstrate how computer engineering can deliver tangible value, promote efficiency, and bring order to people's daily lives. This project not only marks the culmination of an enriching educational journey but also sets the stage for new opportunities for growth, continuous learning, and professional advancement.

Moreover, Mi Garaje has been conceived as a living project, ready to adapt to the evolving needs of its users by providing a flexible and scalable platform. Thanks to its open and modular architecture, it is prepared for future integrations with external services such as repair shops, insurance companies, or navigation systems. This vision paves the way for the creation of a connected ecosystem that will further enhance

---

the comprehensive management of vehicles. Ultimately, Mi Garaje aims to become an indispensable tool for anyone who values the efficient, organized care of their personal and professional mobility assets.

Beyond the technical and functional features already developed, Mi Garaje establishes the basis for future improvements aimed at enhancing its usability and value. Planned developments include the integration of push notifications to remind users about upcoming maintenance tasks or expiring documents, the expansion of the family system with internal chat and an activity history for better coordination, and the implementation of a user-configurable cloud backup and restore system to facilitate data migration and recovery. Additionally, the app aims to incorporate predictive maintenance powered by artificial intelligence, capable of analyzing usage patterns and activity history to estimate when interventions such as oil changes, brake replacements, or inspections will be needed. Finally, a step-by-step initial setup assistant is envisioned to guide users in configuring vehicles, activities, and preferences, reducing the learning curve and ensuring an intuitive first experience.



# Índice general

<b>1. Introducción</b>	<b>1</b>
<b>2. Objetivos y Metodología</b>	<b>3</b>
<b>3. Análisis de aplicaciones actuales</b>	<b>7</b>
3.1. Aspectos claves de las aplicaciones existentes . . . . .	7
3.2. Análisis comparativo . . . . .	8
3.3. Propuesta de aplicación: Mi Garaje . . . . .	11
3.3.1. Funcionalidades principales . . . . .	11
3.3.2. Funcionalidades innovadoras . . . . .	11
<b>4. Especificación de la aplicación</b>	<b>13</b>
4.1. Requisitos funcionales . . . . .	13
4.2. Requisitos no funcionales . . . . .	15
4.3. Modelo de dominio . . . . .	16
4.4. Funcionalidades . . . . .	17
4.4.1. Gestión del perfil de usuario . . . . .	18
4.4.2. Gestión de vehículos y actividades . . . . .	18
4.4.3. Personalización de tipos de datos . . . . .	18
4.4.4. Gestión de familias . . . . .	19
4.4.5. Estadísticas . . . . .	19
4.4.6. Exportación de datos . . . . .	19
4.4.7. Personalización . . . . .	20
4.4.8. Feedback al usuario . . . . .	20
4.5. Documento de Especificación de Requisitos de Software . . . . .	20
<b>5. Planificación para el desarrollo de la aplicación</b>	<b>21</b>
5.1. Diagrama de Gantt . . . . .	22
5.2. Impacto de la planificación . . . . .	22
<b>6. Marco Tecnológico</b>	<b>23</b>
6.1. Criterio de selección de la tecnología . . . . .	23
6.2. Análisis comparativo de tecnologías . . . . .	23
6.3. Arquitectura general de la aplicación . . . . .	25

---

<b>7. Desarrollo de la Aplicación</b>	<b>27</b>
7.1. Estructura del Proyecto . . . . .	27
7.1.1. Organización Física del Proyecto . . . . .	28
7.2. Backend con Firebase . . . . .	29
7.2.1. Gestión de la Autenticación de Usuarios . . . . .	29
7.2.2. Estructura y Almacenamiento de Datos . . . . .	30
7.2.3. Reglas de Seguridad en Firebase Firestore . . . . .	31
7.3. Persistencia local y preferencias del usuario . . . . .	32
7.3.1. SharedPreferences . . . . .	32
7.3.2. Precarga de datos desde JSON local . . . . .	33
7.4. Gestión del estado . . . . .	33
7.4.1. Arquitectura de estado . . . . .	34
7.4.2. Ejemplo: AuthNotifier . . . . .	34
7.5. Diseño de la interfaz y prototipado . . . . .	37
7.5.1. Prototipo de interfaz . . . . .	37
7.5.2. Mapa de navegación . . . . .	39
7.6. Sistema de rutas . . . . .	40
7.7. Capa de presentación . . . . .	42
7.8. Personalización del usuario . . . . .	48
7.8.1. Cambio de tema . . . . .	48
7.8.2. Soporte multilingüe . . . . .	48
<b>8. Conclusiones y vías futuras</b>	<b>49</b>
<b>Glosario</b>	<b>51</b>
<b>Bibliografía</b>	<b>53</b>
<b>A. Anexo I: Documento de Especificación de Requisitos de Software (SRS)</b>	<b>55</b>
<b>B. Anexo II: Interfaz de usuario final de la aplicación</b>	<b>65</b>

# Índice de figuras

4.1. Modelo de dominio. . . . .	16
5.1. Diagrama de Gantt. . . . .	22
6.1. Arquitectura general. . . . .	25
7.1. Diagrama de colecciones en Firebase Firestore. . . . .	31
7.2. Reglas de seguridad en Firebase Firestore. . . . .	32
7.3. Diseño prototipado inicial. . . . .	38
7.4. Mapa de navegación. . . . .	39
7.5. Interfaz de la clase CarTabView. . . . .	44
7.6. Interfaz de la clase ActivityCard. . . . .	45
7.7. Interfaz de la clase DialogAddActivity. . . . .	47



# **Índice de tablas**

3.1. Análisis comparativo de las versiones gratuitas de las apps. . . . .	9
3.2. Análisis comparativo de las versiones de pago de las apps. . . . .	10
5.1. Listado de tareas a realizar. . . . .	21
6.1. Análisis comparativo de las tecnologías existentes. . . . .	24



# Índice de Códigos

7.1. Clase AuthState. . . . .	35
7.2. Método build() de la clase AuthNotifier. . . . .	36
7.3. Clase AuthWrapper. . . . .	36
7.4. Clase AppRoutes. . . . .	41
7.5. Método _sliderTransition de la clase AppRoutes. . . . .	42



# 1. Introducción

El avance de las tecnologías móviles ha propiciado el desarrollo de aplicaciones destinadas a mejorar diversos aspectos de la vida cotidiana, entre ellos, la gestión y el mantenimiento de vehículos. En un contexto en el que la movilidad y el uso del automóvil son esenciales, disponer de herramientas digitales que permitan llevar un control eficiente y personalizado del estado y actividades de los vehículos se convierte en una necesidad creciente.

Actualmente, aunque existen múltiples aplicaciones destinadas a la gestión de vehículos, muchas de ellas presentan limitaciones importantes, como modelos de pago restrictivos, interfaces poco intuitivas, funcionalidades limitadas en sus versiones gratuitas o una falta de enfoque en la personalización y la colaboración entre usuarios. Estas carencias generan una oportunidad clara para el desarrollo de una solución que no solo cubra las funcionalidades básicas del mantenimiento, sino que además sea **gratuita, personalizable, colaborativa y fácil de usar**.

En este marco, surge el presente Trabajo Fin de Grado, cuyo objetivo principal ha sido el **diseño, especificación y desarrollo de una aplicación móvil multiplataforma**, denominada **Mi Garaje**, destinada a la gestión integral del mantenimiento de vehículos. Esta herramienta está pensada tanto para usuarios individuales como para grupos, como familias o pequeños equipos de trabajo, que necesiten una solución compartida para registrar actividades, consultar historiales, analizar gastos y mantener al día el estado de sus vehículos.

Para alcanzar este objetivo, se ha seguido una metodología estructurada, de carácter tradicional, basada en una planificación detallada mediante un **diagrama de Gantt**, que ha permitido dividir el trabajo en fases bien delimitadas: análisis del mercado y de las aplicaciones existentes, especificación de requisitos, diseño de la arquitectura de la aplicación, desarrollo técnico y redacción de la documentación.

Desde el punto de vista técnico, el proyecto se ha desarrollado utilizando una tecnología que permite generar aplicaciones nativas desde un único código fuente para Android e iOS, e integrando servicios en la nube para la autenticación, almacenamiento y sincronización de datos en tiempo real. Esta combinación tecnológica ha ofrecido una gran flexibilidad, escalabilidad y facilidad de mantenimiento, a la vez que ha permitido implementar funcionalidades avanzadas sin necesidad de una infraestructura compleja. Asimismo, se ha adoptado una solución moderna para la gestión del estado, garantizando una estructura limpia, modular y fácilmente escalable.

El diseño de la interfaz de usuario se ha concebido con un enfoque centrado en **la usabilidad y la accesibilidad**, utilizando herramientas de prototipado interactivo que han guiado la implementación visual. Se ha prestado especial atención a aspectos como el soporte multilingüe (español e inglés), la personalización visual mediante temas claro y oscuro, y la organización intuitiva de las funcionalidades clave.

Entre las características implementadas destacan: la gestión de múltiples vehículos, el registro de actividades organizadas por tipo y subtipo, la exportación de datos en formato CSV, la posibilidad de personalizar completamente los tipos de datos, un sistema de estadísticas visuales para el análisis de costes y actividades, y la funcionalidad de familias que permite compartir la gestión con otros usuarios en tiempo real.

Además, se han realizado pruebas funcionales en dispositivos reales para validar la estabilidad, rendimiento y fiabilidad de la aplicación, así como para ajustar detalles de interfaz y navegación a partir del feedback recogido durante el desarrollo. El código de la aplicación se encuentra disponible de manera abierta y libre en el siguiente repositorio de GitHub: <https://github.com/julisco23/tfg-mi-garaje>. En dicho repositorio también se encuentra publicada la primera versión oficial y definitiva de la aplicación, la cual representa el resultado final del proceso de desarrollo.

En definitiva, el presente trabajo no solo se centra en la implementación de una aplicación funcional, sino que abarca todo el ciclo de vida del software: desde el análisis del problema y la detección de necesidades reales, hasta el desarrollo, validación, documentación y propuesta de mejoras futuras. Con ello, se pretende ofrecer una solución sólida y extensible, alineada con los principios de la ingeniería del software moderna y preparada para continuar evolucionando en respuesta a las necesidades cambiantes de los usuarios.

En cuanto a la estructura del documento, en el **Capítulo 1** se presenta la introducción al proyecto, contextualizando la problemática, los objetivos generales y las motivaciones que han guiado su desarrollo. En el **Capítulo 2** se exponen los objetivos específicos y la metodología adoptada. El **Capítulo 3** realiza un análisis de las aplicaciones existentes y destaca las funcionalidades relevantes que justifican la propuesta de **Mi Garaje**. En el **Capítulo 4** se detallan los requisitos del sistema y su diseño funcional. El **Capítulo 5** aborda la planificación temporal del proyecto. El **Capítulo 6** presenta el marco tecnológico, analizando diversas alternativas y justificando la elección de Flutter como tecnología principal. A continuación, el **Capítulo 7** se centra en el desarrollo de la aplicación, desde su arquitectura hasta la interfaz de usuario. Finalmente, el **Capítulo 8** recoge las conclusiones alcanzadas y plantea posibles líneas de trabajo futuro.

---

## 2. Objetivos y Metodología

El objetivo principal es el diseño y desarrollo de una aplicación móvil multiplataforma, denominada **Mi Garaje**, que permita a los usuarios gestionar de manera eficiente, accesible y personalizada el mantenimiento de sus vehículos. La aplicación busca cubrir las principales necesidades detectadas en el análisis del mercado actual, superando las limitaciones observadas en soluciones existentes, tanto en funcionalidad como en accesibilidad y modelo de negocio.

Para alcanzar este objetivo general, se han definido una serie de **objetivos específicos** que abarcan todas las fases del ciclo de vida del desarrollo de software:

- **Análisis de mercado y evaluación de aplicaciones existentes:** Investigar las soluciones actuales disponibles en las principales plataformas de distribución (Google Play, App Store), identificando sus funcionalidades clave, carencias y oportunidades para innovar.
- **Definición precisa de los requisitos funcionales y no funcionales:** Establecer, de manera estructurada y con enfoque centrado en el usuario, las funcionalidades esenciales que debe ofrecer la aplicación. Esto incluye la identificación de los distintos tipos de entidad, flujos de navegación, restricciones técnicas y atributos de calidad como usabilidad, accesibilidad, rendimiento y escalabilidad.
- **Selección fundamentada de la tecnología de desarrollo:** Evaluar distintas tecnologías móviles (Kotlin, Swift, React Native, Flutter), considerando criterios como compatibilidad multiplataforma, curva de aprendizaje, rendimiento, mantenimiento y comunidad de soporte. Finalmente, se ha optado por **Flutter**, en combinación con **Firebase**, para aprovechar su flexibilidad y capacidad de integración.
- **Diseño de una interfaz de usuario intuitiva, accesible y personalizable:** Concebir la experiencia visual y de interacción de la aplicación desde una perspectiva de diseño centrado en el usuario, priorizando la claridad, simplicidad y adaptabilidad. Se ha empleado la herramienta **Figma** para el desarrollo de *wireframes* y prototipos de alta fidelidad que guían la implementación final.
- **Implementación técnica de la aplicación con arquitectura modular:** Desarrollar la aplicación utilizando **Flutter**, estructurada en módulos bien definidos que faciliten el mantenimiento y la escalabilidad del proyecto. Se ha empleado

**Riverpod** como solución para la gestión del estado, lo que garantiza un control eficiente y reactivo de la información entre la interfaz y el backend.

- **Validación funcional y experiencia de usuario en dispositivos reales:** Ejecutar **pruebas funcionales en dispositivos Android** reales, verificando el correcto funcionamiento de la aplicación, su estabilidad, y la satisfacción de los criterios definidos durante la fase de especificación. Estas pruebas han permitido depurar errores, optimizar el rendimiento y realizar ajustes finales en la interfaz.
- **Gestión del desarrollo y control de versiones:** Asegurar una trazabilidad adecuada del proceso de implementación mediante el uso de **Git** y **GitHub** para el control de versiones. Esto ha permitido mantener un historial ordenado de los avances del proyecto, facilitando la iteración y la reversión en caso de errores.

Dado el carácter individual del proyecto, se ha optado por una metodología de planificación tradicional, basada en fases bien definidas, cada una con objetivos, entregables y fechas de inicio y fin determinadas. Esta planificación se ha representado gráficamente mediante un **diagrama de Gantt**, que ha servido como herramienta principal para organizar y supervisar el progreso del proyecto.

La metodología seguida ha permitido abordar el proyecto desde una perspectiva secuencial e incremental, facilitando el análisis temprano de riesgos y la toma de decisiones fundamentadas en cada etapa. El enfoque adoptado ha proporcionado una visión clara del alcance del proyecto y ha permitido mantener una gestión eficaz del tiempo y los recursos disponibles.

Las fases principales de la metodología aplicada han sido las siguientes:

1. **Fase de análisis:** Investigación del mercado, identificación de carencias en aplicaciones similares y definición de la propuesta de valor de **Mi Garaje**.
  2. **Fase de especificación:** Redacción del documento de requisitos funcionales y no funcionales, diseño del modelo de dominio y definición de la arquitectura de la aplicación.
  3. **Fase de diseño:** Elaboración de prototipos interactivos y definición de la estructura de navegación, interfaz y experiencia de usuario.
  4. **Fase de desarrollo:** Construcción de la lógica de la aplicación, conexión con una plataforma de servicios remotos y organización modular del estado interno para garantizar escalabilidad y mantenimiento.
  5. **Fase de pruebas:** Validación funcional mediante pruebas manuales en dispositivos reales, detección y corrección de errores, y evaluación del rendimiento y usabilidad.
-

**6. Fase de documentación y entrega:** Redacción de la memoria del proyecto y estructuración de los contenidos para su presentación.

Gracias a esta estructura metodológica y a una gestión disciplinada del desarrollo, se ha conseguido completar el proyecto de forma integral, cumpliendo con los objetivos definidos y estableciendo una base sólida para futuras mejoras y ampliaciones.



### 3. Análisis de aplicaciones actuales

En la actualidad, el **mantenimiento de vehículos** constituye un aspecto esencial para garantizar el correcto funcionamiento y prolongar la vida útil del vehículo. La proliferación de aplicaciones dedicadas a la gestión de estas tareas ha dado lugar a una amplia variedad de herramientas diseñadas para facilitar el registro y seguimiento de las actividades.

En este apartado se lleva a cabo un análisis de las principales aplicaciones disponibles en Google Play orientadas a la gestión del mantenimiento de vehículos, centrando el estudio en aquellas que ofrecen una versión gratuita. La selección se ha basado en criterios de popularidad, determinada por una combinación entre el número de descargas y la valoración promedio otorgada por los usuarios, lo cual permite identificar herramientas consolidadas en el mercado.

Además, se incluye una **comparativa entre las versiones gratuitas y de pago** de dichas aplicaciones, con el fin de proporcionar una visión más completa de las funcionalidades avanzadas que se ofrecen bajo suscripciones u otros modelos *freemium*. Este análisis no solo permite detectar **puntos fuertes y limitaciones recurrentes**, sino que también facilita la identificación de **oportunidades de mejora y diferenciación**.

Finalmente, se presenta la propuesta de valor de la aplicación **Mi Garaje**, concebida para integrar las mejores funcionalidades observadas, resolver carencias detectadas y aportar innovaciones que optimicen la experiencia de usuario y la gestión integral del mantenimiento vehicular.

#### 3.1. Aspectos claves de las aplicaciones existentes

Como parte del estudio comparativo, se examinan las versiones gratuitas de las aplicaciones Notas de coche, Drivvo, Mis Coches y My Car, seleccionadas por su relevancia en el mercado. El análisis considera aspectos como las funcionalidades ofrecidas, compatibilidad multiplataforma, idiomas disponibles y la calidad de la experiencia de usuario.

**Notas de coche** [1] es una aplicación disponible únicamente para Android, con más de **100,000 descargas** y una valoración de **4,4 estrellas** en Google Play. Está disponible solo en español y a pesar de seguir un modelo *freemium*, de manera gratuita puedes acceder a la mayoría de funcionalidades claves. Destaca por su diseño moderno,

lo que facilita el acceso y la organización de la información. Entre sus principales funcionalidades se encuentran la **gestión de combustible, reparaciones, documentos, recordatorios e informes**, las cuales están bien estructuradas. Sin embargo, la sección de informes se centra exclusivamente en mostrar estadísticas sobre los costes, lo que deja espacio para una visión más completa del mantenimiento.

**Drivvo** [2] está disponible para Android, iOS y cuenta con una versión web. Con más de **1 millón de descargas** en Google Play y una valoración de **4,5 estrellas**, sigue el modelo *freemium*, ofreciendo versión gratuita y *premium*. Además, soporta una gran variedad de idiomas, lo que la hace accesible a usuarios internacionales. En cuanto a sus funcionalidades, ofrece una **amplia gama de herramientas** para la gestión del mantenimiento de nuestro vehículo, aunque su organización puede resultar algo confusa. Entre sus características principales destacan el **registro de repostajes y servicios**, aunque estas acciones requieren la introducción manual obligatoria de datos como el odómetro actual del vehículo y la ubicación, lo que puede ser tedioso. También incluye **gráficos y reportes interesantes**, pero su ubicación poco accesible limita su utilidad.

**Mis Coches** [3] está disponible únicamente para Android y soporta varios idiomas. Con más de **1 millón de descargas** en Google Play y una valoración de **4,5 estrellas**, esta aplicación permite **gestionar múltiples vehículos**. Su funcionalidad principal se centra en el repostaje, lo que hace que los gráficos y resúmenes se centren en los costes. Su interfaz, aunque funcional, es menos moderna que la del resto de alternativas.

**My Car** [4] está disponible para Android e iOS, y soporta idiomas como inglés y español. Con más de **100,000 descargas** en Google Play y una valoración de **4,3 estrellas**, la aplicación sigue el modelo *freemium*, ofreciendo funciones básicas de forma gratuita y características adicionales a través de versiones *premium*. Presenta características similares a las aplicaciones anteriores, destacándose principalmente por su simplicidad. Si bien permite registrar **repostajes, servicios y otros datos básicos**, su diseño y usabilidad podrían mejorarse para competir con alternativas más avanzadas.

## 3.2. Análisis comparativo

Para facilitar la evaluación visual de las características principales de las aplicaciones analizadas en el apartado anterior, se presenta a continuación una **tabla comparativa (Tabla 3.1)**. Esta comparación incluye aspectos clave como la disponibilidad en plataformas, los idiomas soportados y las funcionalidades ofrecidas. Entre estas funcionalidades se encuentran la capacidad para gestionar categorías predefinidas (como tipos de vehículos o mantenimiento), así como la opción para que el usuario agregue o elimine nuevas categorías según sus necesidades.

---

Características	Drivvo	Mis Coches	Notas de coche	My Car	Mi Garaje
Plataformas disponibles	Android/ iOS/ web	Android	Android	Android/ iOS	Android/ iOS
Idiomas soportados	Español	Español/ Inglés/ Alemán/ Chino/ Otros	Español	Español/ Inglés	Español/ Inglés
Modo oscuro	No	No	No	Sí	Sí
Tipos de vehículos	Coche/ Motocicleta/ Autobús/ Camión	Único tipo de vehículo	Coche/ Moto/ Otro	Único tipo de vehículo	Fijos + Personalizados
Gestión de repostajes	Sí	Sí	Sí	Sí	Sí
Gestión de mantenimientos	Sí	No	Sí	No	Sí
Gestión de documentos	No	No	Sí	No	Sí
Añadir subtítulos personalizados	Sí	Sí	No	Sí	Sí
Estadísticas	Limitadas	Limitadas	Limitadas	Limitadas	Sí
Tipo de gratuidad	Parcial con anuncios	Parcial con anuncios	Parcial con anuncios	Parcial	Total
Número máximo de vehículos	2	Ilimitado	Ilimitado	3	Ilimitado
Número máximo de conductores	1	1	1	1	Ilimitado
Exportar datos	No	Sí	No	No	Sí
Adjuntar imágenes	No	No	Sí	Sí	Sí

Tabla 3.1: Análisis comparativo de las versiones gratuitas de las apps.

La tabla permite realizar una comparación rápida entre las distintas aplicaciones gratuitas, identificando sus fortalezas y debilidades, y resaltando las áreas en las que una nueva solución podría aportar mejoras o innovaciones. La última columna corresponde a la aplicación que voy a desarrollar, **Mi Garaje**, destacando sus características y su posicionamiento frente a las demás opciones del mercado.

Además, se incluye una segunda **tabla comparativa (Tabla 3.2)** que recoge algunas **características de las versiones de pago** de las aplicaciones analizadas. Aunque no se ha realizado un estudio exhaustivo de todas las funcionalidades *premium*, esta tabla ofrece una visión general de las opciones disponibles en el mercado y cómo se comparan con las funcionalidades que planeo implementar en mi aplicación.

Características	Drivvo	Mis Coches	Notas de coche	My Car	Mi Garaje
Precio/mes	4,90€	2€ (único pago)	3,99€	1,19€	Gratis
Sin anuncios	Sí	Sí	Sí	Sí	Sí
Exportar a Excel	Sí	No	Sí	Sí	Sí
Número máximo de vehículos	10	Ilimitado	Ilimitado	6	Ilimitado
Número máximo de conductores	2	1	1	2	Ilimitado
Otras funcionalidades	-	Gestión de viaje	-	-	No
	-	-	Permitir adjuntar imágenes	-	Sí
	-	-	Mejoras en gráficos	-	Sí
	-	Modo oscuro	-	-	Sí

**Tabla 3.2:** Análisis comparativo de las versiones de pago de las apps.

Es importante mencionar que **Mis Coches ofrece su versión de pago mediante un único pago de 2 €**, sin suscripciones mensuales ni anuales, mientras que el resto de aplicaciones cobran cuotas periódicas para acceder a las funciones *premium* o servicios completos.

### 3.3. Propuesta de aplicación: ***Mi Garaje***

En base al análisis de las aplicaciones actuales, ***Mi Garaje*** se plantea como una solución gratuita, sin suscripciones ni versiones de pago, que reúne las **mejores funcionalidades del mercado** y añade **características innovadoras** para ofrecer una experiencia de usuario más completa, personalizable y accesible.

#### 3.3.1. Funcionalidades principales

La aplicación permitirá al usuario gestionar múltiples vehículos sin ningún tipo de límite, asociándoles distintas actividades organizadas por tipo y subtipo. Entre las funcionalidades básicas que ofrecerá se encuentran:

- **Actividades predefinidas:** Por defecto, se contará con tres tipos de actividades: Repostaje, Mantenimiento y Documentación.
- **Subtipos de actividad:** Dentro de cada tipo, se incluirán subcategorías pre-configuradas (por ejemplo, dentro de Mantenimiento se podrá registrar cambios de frenos, batería, neumáticos, etc.).
- **Exportación de datos:** Todos los registros podrán exportarse en formato .csv, compatible con Excel, para facilitar su análisis o respaldo externo.
- **Multiplataforma:** La aplicación estará disponible tanto para Android como iOS desde el lanzamiento.
- **Interfaz intuitiva:** Se priorizará una experiencia de usuario clara, simple y eficiente, accesible para usuarios con distintos niveles de familiaridad tecnológica.
- **Idioma:** La aplicación permitirá alternar entre los idiomas español e inglés, ampliando su accesibilidad a usuarios internacionales.
- **Aspecto visual:** Se incluirá **modo claro y oscuro**, ajustable manualmente o según la configuración del dispositivo, mejorando la comodidad visual en distintos entornos.

#### 3.3.2. Funcionalidades innovadoras

Para destacar frente a las opciones actuales del mercado, ***Mi Garaje*** incorporará funcionalidades avanzadas que no están disponibles en las versiones gratuitas de otras aplicaciones, o que se encuentran notablemente limitadas:

- **Gestión de múltiples conductores:** Los usuarios podrán crear o unirse a un grupo familiar, permitiendo compartir vehículos con otros usuarios con los mismos privilegios de edición y seguimiento.
-

- **Categorías personalizables:** A diferencia de las apps actuales, que sólo permiten modificar los subtipos de actividades predefinidas, mi aplicación permitirá personalizar:
  - Tipos de vehículos.
  - Tipos de actividades.
  - Subtipos de actividades.
- **Estadísticas avanzadas:** Se generarán gráficos detallados no solo de los costes, sino también del historial de servicios, frecuencia de mantenimientos y evolución del rendimiento del vehículo, lo que facilitará la toma de decisiones para el usuario.
- **Adjuntar imágenes:** En cada registro de actividad, se podrá añadir de forma opcional una imagen (factura, foto del mantenimiento, documento escaneado, etc.) que quedará archivada junto a la actividad.

Con esta propuesta, **Mi Garaje** busca ofrecer una herramienta integral, flexible y gratuita que supere las limitaciones de las aplicaciones actuales. Gracias a su enfoque modular, su diseño intuitivo y sus capacidades de personalización, se adapta tanto a usuarios individuales como a grupos (familias, empresas, etc.) que deseen llevar un control detallado del uso y mantenimiento de sus vehículos.

Para una descripción más detallada de cada una de estas funcionalidades, puede consultarse la **Sección 4.4**, donde se explican en profundidad las capacidades implementadas en la aplicación.

# 4. Especificación de la aplicación

La **especificación de requisitos de software (SRS)** establece de forma clara y estructurada lo que la aplicación debe hacer, cómo debe comportarse y bajo qué condiciones debe operar. Su propósito principal es definir tanto las funcionalidades como las características no funcionales que garantizarán que la aplicación cumpla con las necesidades y expectativas de los usuarios.

En esta sección se presentan los requisitos funcionales y no funcionales de **Mi Garage**, organizados a partir de las interacciones que los usuarios podrán realizar con el sistema. Estos requisitos proporcionan una base sólida para el diseño, desarrollo y validación de la aplicación.

Además, se incluye al final un **modelo de dominio**, que representa las entidades principales del sistema y sus relaciones. Este modelo servirá de guía conceptual para la implementación posterior y complementa la descripción de los requisitos aquí recogidos.

## 4.1. Requisitos funcionales

En esta sección se detallan los requisitos funcionales de la aplicación, definidos como **historias de usuario**. Cada historia refleja una necesidad específica que el sistema debe satisfacer desde el punto de vista del usuario. Las funcionalidades se agrupan según las áreas clave del sistema.

### Gestión de usuarios

1. Como usuario anónimo, quiero **iniciar sesión como invitado** para usar la aplicación sin necesidad de registro.
2. Como usuario anónimo, quiero **vincular mi cuenta a un correo electrónico** para conservar mis datos como cuenta registrada.
3. Como usuario, quiero **registrarme e iniciar sesión con email y contraseña** para acceder de forma segura.
4. Como usuario, quiero **registrarme e iniciar sesión con mi cuenta de Google** para evitar crear nuevas credenciales.

5. Como usuario, quiero **vincular mi cuenta a mi cuenta de Google** para unificar accesos.
6. Como usuario, quiero **actualizar mi perfil** (nombre e imagen) para mantener mis datos actualizados.
7. Como usuario, quiero **cerrar sesión** para proteger mis datos en dispositivos compartidos.
8. Como usuario, quiero **eliminar mi cuenta** para asegurar el borrado de todos mis datos.

## Gestión de familias

9. Como usuario, quiero **crear una familia** para facilitar que otros usuarios puedan gestionar mis vehículos.
10. Como usuario, quiero **unirme a una familia mediante un código** para gestionar vehículos compartidos con otras personas.
11. Como usuario, quiero **abandonar una familia** cuando ya no desee compartir el acceso.

## Gestión de vehículos

12. Como usuario, quiero **añadir un nuevo vehículo** introduciendo información relevante (marca, modelo, nombre, etc.) para tenerlo registrado en el sistema.
13. Como usuario, quiero **editar los detalles de un vehículo** ya registrado para actualizar información en caso de cambios o correcciones.
14. Como usuario, quiero **eliminar un vehículo** de mi lista para descartar registros que ya no son necesarios.
15. Como usuario, quiero **ver todos mis vehículos** registrados para tener una visión general de mi flota o historial de vehículos.

## Gestión de actividades

16. Como usuario, quiero **añadir una nueva actividad** a un vehículo específico, incluyendo subtípo, fecha y coste.
  17. Como usuario, quiero **modificar los datos de una actividad** registrada.
-

18. Como usuario, quiero **eliminar una actividad** de un vehículo.
19. Como usuario, quiero **ver la lista de todas las actividades** realizadas por un vehículo.

### **Estadísticas**

20. Como usuario, quiero **consultar el historial de actividades** de un vehículo.
21. Como usuario, quiero **visualizar un resumen** con el gasto total, número de actividades, gasto medio por actividad y gasto medio mensual.
22. Como usuario, quiero **visualizar un gráfico de barras del gasto mensual** acumulado durante un año.
23. Como usuario, quiero **consultar un gráfico circular** con la proporción del número de actividades por tipo.

### **Gestión de ajustes**

24. Como usuario, quiero **personalizar los subtipos de actividad** para adaptarlos a mis necesidades.
25. Como usuario, quiero **personalizar los tipos de vehículos**.
26. Como usuario, quiero **personalizar los tipos de actividad**.

### **Exportación de datos**

27. Como usuario, quiero **exportar los datos de mis vehículos** en formato CSV.

## **4.2. Requisitos no funcionales**

### **Compatibilidad**

1. El sistema debe ser compatible con dispositivos móviles Android (versión X o superior) e iOS (versión Y o superior).

### **Seguridad**

2. El sistema debe cifrar los datos personales del usuario tanto en tránsito como en reposo, cumpliendo con buenas prácticas y normativas.
-

## Accesibilidad

3. Como usuario, quiero que la aplicación se muestre en español e inglés.
4. Como usuario, quiero que la aplicación incluya un modo oscuro y un modo claro, seleccionable manualmente.

## Escalabilidad

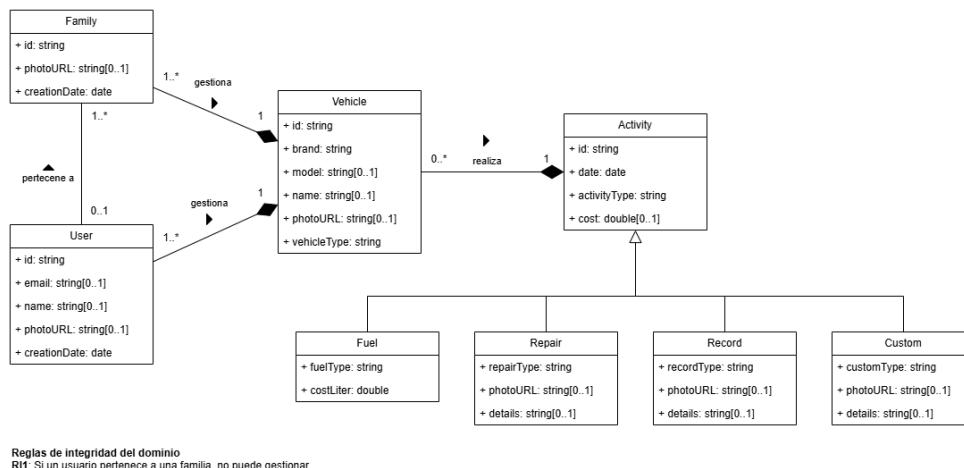
5. El sistema debe tener una arquitectura modular que permita escalar e incorporar nuevas funcionalidades sin comprometer el rendimiento.

## Usabilidad

6. Como usuario, quiero que la aplicación sea fácil de usar, cumpliendo con los estándares de usabilidad definidos en ISO/IEC 25010:2011.

### 4.3. Modelo de dominio

El **modelo de dominio** (**Figura 4.1**) representa los conceptos fundamentales de **Mi Garaje** y las relaciones entre ellos, sin entrar en detalles de implementación. Este modelo sirve como base conceptual para el diseño de la arquitectura y la estructura de la base de datos, asegurando una comprensión compartida entre los distintos actores del desarrollo.



**Figura 4.1:** Modelo de dominio.

## Entidades

- **Usuario:** Puede registrarse, iniciar sesión, pertenecer a una familia y gestionar vehículos.
- **Familia:** Grupo de usuarios que colaboran en la gestión de vehículos compartidos.
- **Vehículo:** Entidad principal asociada a un usuario o familia. Incluye atributos como marca, modelo y tipo.
- **Actividad:** Evento asociado a un vehículo. Actividad es una clase abstracta con los subtipos:
  - Repostaje
  - Mantenimiento
  - Documento
  - Personalizado

## Relaciones entre entidades

- Un usuario puede estar asociado a una familia.
- Una familia está compuesta por varios usuarios.
- Un usuario gestiona uno o más vehículos.
- Una familia gestiona uno o más vehículos.
- Un vehículo puede tener múltiples actividades asociadas.

## Reglas de integridad

- Si un usuario pertenece a una familia, no puede gestionar vehículos de forma individual.

## 4.4. Funcionalidades

A continuación se describen las funcionalidades principales implementadas en la aplicación **Mi Garaje**, que permiten al usuario gestionar de forma eficiente el mantenimiento y seguimiento de sus vehículos:

---

#### 4.4.1. Gestión del perfil de usuario

La aplicación permite al usuario registrado gestionar y personalizar su perfil desde la sección correspondiente:

- **Actualización de datos personales:** el usuario puede modificar su nombre y su imagen de perfil.
- **Gestión de la cuenta:** se ofrecen diversas acciones según el estado del usuario:
  - Cerrar sesión, para desconectarse de la cuenta actual.
  - Eliminar cuenta, lo cual implica la eliminación permanente de los datos personales y asociados.
- **Vinculación con Google,** permitiendo iniciar sesión o registrar una cuenta mediante autenticación con Google, facilitando el acceso y sincronización.
- **Vinculación con correo electrónico,** ofreciendo la posibilidad de iniciar sesión o registrarse utilizando un correo electrónico y contraseña, lo que permite un acceso más tradicional y ampliamente compatible.

#### 4.4.2. Gestión de vehículos y actividades

El usuario puede:

- Añadir, editar o eliminar vehículos.
- Registrar actividades asociadas a cada vehículo (como mantenimiento, repostajes, reparaciones, o personalizadas).
- Visualizar un historial detallado de todas las actividades realizadas, ordenado cronológicamente y filtrado por tipo o por vehículo.

#### 4.4.3. Personalización de tipos de datos

Una de las funcionalidades más destacadas es la capacidad de personalización en la gestión de datos:

- **Subtipos de actividad:** el usuario puede modificar, añadir o eliminar subtipos dentro de las categorías predefinidas (repostaje, mantenimiento y documentación).
  - **Tipos de vehículos:** es posible editar la lista de tipos disponibles para adaptarse a las necesidades del usuario.
  - **Creación de nuevos tipos de actividad:** además de personalizar los subtipos, se permite la creación de nuevos tipos completamente personalizados, lo que brinda una gran flexibilidad para adaptarse a distintos contextos de uso.
-

#### 4.4.4. Gestión de familias

La aplicación permite compartir la información entre varios usuarios mediante un sistema de familias:

- Un usuario puede **crear una familia** y transferir automáticamente todos sus datos asociados (vehículos, actividades y configuraciones personalizadas).
- En la sección de perfil, se muestra un **código de familia** que otros usuarios pueden introducir para unirse.
- **Todos los miembros de una familia tienen acceso común a los mismos datos**, lo que permite a varios usuarios gestionar en conjunto los vehículos registrados.
- Cualquier miembro puede **abandonar la familia** en cualquier momento. En caso de que sea el último miembro en salir, la familia se elimina automáticamente.
- Al salir de una familia, el usuario inicia su experiencia como si fuera una **cuenta nueva**, comenzando con los datos iniciales por defecto (sin vehículos ni actividades registradas).

#### 4.4.5. Estadísticas

La aplicación incluye una sección de estadísticas donde el usuario puede analizar la información registrada a través de visualizaciones gráficas:

- **Resumen**: resumen de gasto total, número de actividades, gastos medios por actividad y mensual.
- **Gastos mensuales**: un gráfico de barras que muestra los gastos por mes, permitiendo identificar picos de gasto o tendencias.
- **Distribución de actividades**: un gráfico que refleja la cantidad de actividades registradas por tipo, ofreciendo una visión general del uso y mantenimiento de cada vehículo.

#### 4.4.6. Exportación de datos

Desde la sección de ajustes, el usuario tiene la posibilidad de exportar todos los datos registrados en formato .csv, facilitando el respaldo de información o su análisis externo en herramientas como Excel o Google Sheets.

---

#### 4.4.7. Personalización

Para mejorar la experiencia del usuario, se han incorporado las siguientes funcionalidades:

- **Cambio de tema:** permite alternar entre **modo claro** y **modo oscuro**.
- **Soporte multilingüe:** disponible en varios idiomas, adaptándose a las preferencias del usuario desde los ajustes de la aplicación.
- **Persistencia de preferencias:** las configuraciones personales del usuario se mantienen entre sesiones mediante `SharedPreferences`.

#### 4.4.8. Feedback al usuario

Para mejorar la experiencia del usuario y mantenerlo informado sobre el estado de las acciones realizadas, la aplicación implementa un sistema de notificaciones breves mediante mensajes tipo *toast*:

- Se utilizan *toasts* para **confirmar acciones completadas**, como el guardado de información, la unión a una familia o la exportación de datos.
- También se emplean para informar de errores, como problemas de autenticación, validación de formularios o errores de conexión.

Este sistema de mensajes garantiza una comunicación no intrusiva y rápida para entender lo que ocurre sin interrumpir su flujo de navegación.

### 4.5. Documento de Especificación de Requisitos de Software

El **Documento de Especificación de Requisitos de Software (SRS)** se incluye como **Anexo A**. En él se proporciona una descripción detallada de los requisitos funcionales y no funcionales del sistema, junto con aspectos técnicos relevantes y consideraciones complementarias.

Esta especificación sirve como base formal para el desarrollo del sistema, asegurando que se satisfagan adecuadamente las necesidades de los usuarios. Además, proporciona un marco estructurado que permite diseñar una aplicación escalable, mantenible y centrada en mejorar la experiencia del usuario en la gestión del mantenimiento de sus vehículos.

---

## 5. Planificación para el desarrollo de la aplicación

Para garantizar una correcta organización y el cumplimiento de los plazos establecidos, se ha elaborado una planificación detallada que estructura las tareas necesarias para el desarrollo de la aplicación. Esta planificación abarca desde el análisis inicial hasta las fases de diseño, desarrollo y entrega del proyecto.

A continuación, se presenta la **Tabla 5.1**, que recoge el listado de tareas principales, los entregables asociados y las fechas de inicio y finalización. Este esquema proporciona una visión general del cronograma de trabajo, permitiendo identificar claramente las distintas etapas del proceso y sus respectivos plazos.

Nº	Tareas a realizar	Entregable	Inicio	Fin
1	Analizar aplicaciones sobre mantenimiento de vehículos	Informe de análisis comparativo	21 de enero	9 de febrero
2	Elaboración del documento de especificación de requisitos de software	Documento de especificación de requisitos	25 de enero	10 de marzo
3	Validación de los requisitos	—	30 de enero	15 de marzo
4	Selección de plataforma/-lenguajes para el desarrollo	Criterios y justificación de la elección	1 de febrero	10 de febrero
5	Realización de un wireframe de la aplicación	Wireframe	7 de febrero	11 de febrero
6	Desarrollo de la aplicación	Aplicación funcional	12 de febrero	6 de junio
7	Redacción de la memoria	Memoria del proyecto	21 de enero	1 de julio
8	Preparación de la presentación	Presentación en formato PPT	1 de julio	6 de julio

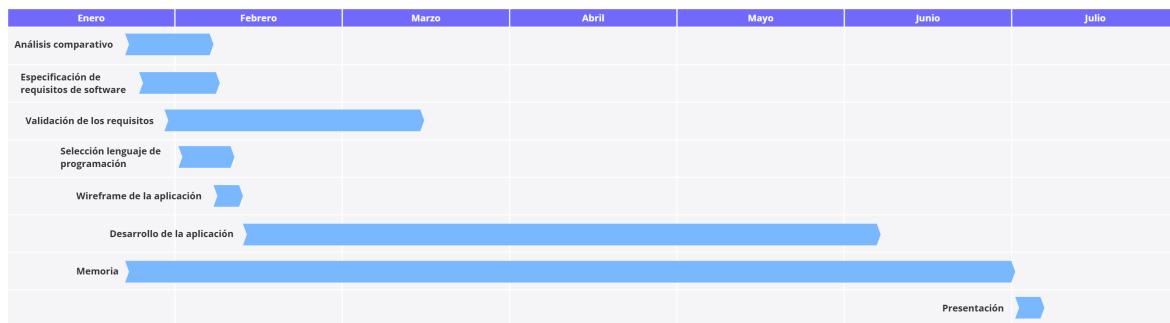
**Tabla 5.1:** Listado de tareas a realizar.

La tabla detalla tanto las tareas específicas a realizar como los entregables que se generarán al finalizar cada etapa. Esto permite un seguimiento claro del progreso del proyecto y facilita la identificación de las actividades en curso.

## 5.1. Diagrama de Gantt

Como complemento a la tabla anterior, se ha elaborado un **diagrama de Gantt** [5] (**Figura 5.1**), que ofrece una representación visual de la planificación. Este diagrama permite observar con claridad la duración estimada de cada actividad, así como la posible superposición entre tareas.

El diagrama destaca por su capacidad para reflejar la **secuencia lógica de las actividades**, mostrando qué tareas se realizan de forma secuencial y cuáles se desarrollan en paralelo, lo que permite una optimización del tiempo y una gestión eficiente de los recursos. Además, facilita la identificación de **hitos clave**, como la finalización del análisis, el diseño del *wireframe* o el desarrollo de la aplicación final.



**Figura 5.1:** Diagrama de Gantt.

Ambas herramientas, la **tabla** y el **diagrama de Gantt**, han sido fundamentales para garantizar que el desarrollo de la aplicación avance de manera ordenada y que se cumplan todos los plazos establecidos hasta la entrega final del proyecto.

## 5.2. Impacto de la planificación

La planificación ha sido una herramienta clave para garantizar el cumplimiento de los plazos propuestos y una gestión eficaz del proyecto. Gracias a una estructura de tareas bien definida, ha sido posible adaptar el cronograma ante retrasos puntuales, minimizando su impacto sobre las fases posteriores y asegurando una ejecución ordenada hasta la entrega final.

# 6. Marco Tecnológico

## 6.1. Criterio de selección de la tecnología

La elección de la tecnología fue una decisión clave del proyecto. Para ello, se analizaron distintas alternativas actuales, teniendo en cuenta factores como la modernidad, proyección futura, popularidad, curva de aprendizaje y compatibilidad multiplataforma.

Tras comparar tecnologías como **Kotlin**, **Swift**, **React Native** y **Flutter**, la opción seleccionada fue **Flutter** [6], un *framework* desarrollado por Google que permite construir aplicaciones nativas para Android e iOS a partir de un único código base. Esta característica lo hace especialmente adecuado para proyectos con recursos limitados y objetivos de cobertura amplia. Los criterios principales para su elección son:

- **Popularidad:** Cuenta con una **comunidad activa y en crecimiento**, lo que garantiza una amplia disponibilidad de recursos, tutoriales y soporte.
- **Potencial a futuro:** Al ser un *framework* respaldado por Google y ampliamente adoptado, tiene una proyección sólida para seguir evolucionando como una tecnología relevante en el desarrollo de aplicaciones móviles.
- **Curva de aprendizaje:** Comparado con las demás tecnologías, presenta una curva de aprendizaje razonable. Su lenguaje principal, **Dart**, es moderno y fácil de asimilar para quienes ya tienen experiencia previa en programación orientada a objetos [7].
- **Multiplataforma:** Flutter ofrece la posibilidad de desarrollar una única aplicación que funcione de manera nativa tanto en **Android** como en **iOS**, lo que reduce significativamente el tiempo y el esfuerzo necesario para implementar y mantener el proyecto.
- **Herramientas de desarrollo:** Su herramienta **Hot Reload** permite visualizar cambios en tiempo real, facilitando el proceso iterativo de diseño y desarrollo.

## 6.2. Análisis comparativo de tecnologías

En cuanto a las alternativas evaluadas, se analizaron diversas opciones ampliamente utilizadas en el desarrollo de aplicaciones móviles [8] [9]. A continuación, se presenta la **Tabla 6.1**, que resume las características clave de cada una de estas tecnologías.

Características	Flutter	React Native	Swift	Kotlin
Desarrollador	Google	Facebook	Apple	JetBrains
Lenguaje	Dart	JavaScript	Swift	Java
Plataformas	iOS, Android, Web, Desktop	iOS, Android, Web	iOS	Android
Curva de aprendizaje	Moderada, fácil para quienes dominan POO	Moderada, más fácil para quienes conocen JavaScript	Alta, requiere conocimiento de iOS	Alta, requiere experiencia en Android
Soporte	Comunidad activa y respaldo de Google	Grande y activa, con amplio ecosistema	Muy grande, solo para iOS	Grande, pero solo para Android
Costo de desarrollo	Menor (base de código compartida)	Menor (base de código compartida)	Mayor (desarrollo independiente para iOS)	Mayor (desarrollo independiente para Android)
Apps existentes	Google Ads, Alibaba, BMW	Facebook, Instagram, Skype	Lyft, Airbnb, Twitter	Pinterest, Trello

**Tabla 6.1:** Análisis comparativo de las tecnologías existentes.

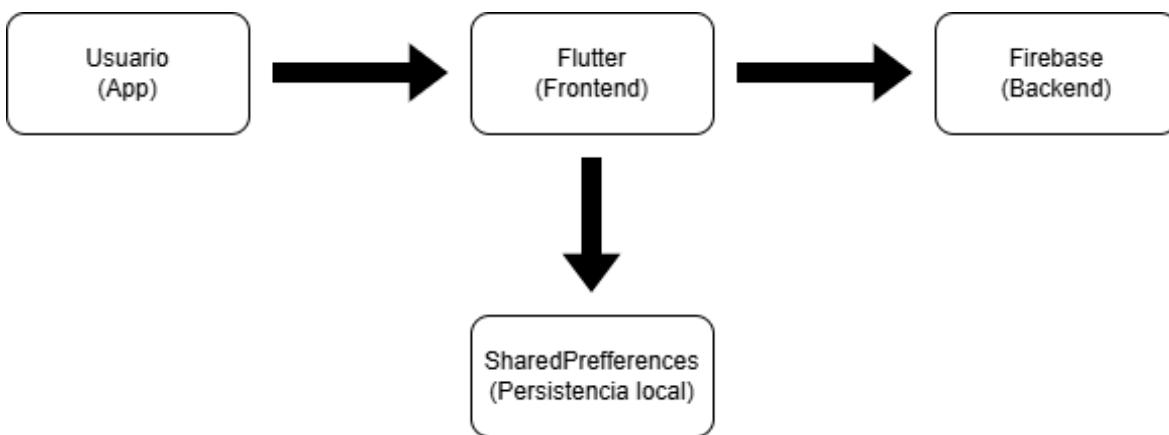
Como se observa en la tabla, **Flutter** destaca por su capacidad multiplataforma, su comunidad activa y su menor coste relativo de desarrollo. Una vez analizadas las principales características de cada tecnología, procederé a justificar el descarte de cada una en comparación con la alternativa seleccionada:

- **Kotlin:** Es un **lenguaje nativo para Android** que ofrece un control total sobre el sistema. Sin embargo, desarrollar una aplicación exclusiva para Android no se ajustaba al alcance del proyecto, ya que excluiría a usuarios de iOS. Además, desarrollar versiones separadas implicaría mayor tiempo y complejidad.
- **Swift:** Aunque ideal para **aplicaciones nativas en iOS**, su falta de compatibilidad multiplataforma también fue un factor decisivo para descartarlo. Al no contar con un dispositivo iOS, el proceso de pruebas hubiese sido limitado y menos eficiente.
- **React Native:** A pesar de ser **multiplataforma y muy popular**, el desconocimiento previo de JavaScript llevó a descartar esta opción.

En conclusión, la elección de **Flutter** no solo permite cumplir con los objetivos técnicos del proyecto, sino que también se ajusta a los recursos, conocimientos y plazos disponibles. Su lenguaje **Dart** —más cercano a **Java**— resultó más accesible gracias a mi experiencia previa en programación orientada a objetos. Su arquitectura basada en un único código base, la posibilidad de desarrollo multiplataforma nativo, y el respaldo continuo de Google, convierten a **Flutter** en la opción más adecuada para el desarrollo de la aplicación.

### 6.3. Arquitectura general de la aplicación

La aplicación está diseñada bajo un modelo de arquitectura cliente-servidor, en el cual el cliente es la aplicación móvil desarrollada con **Flutter** y el servidor está representado por los servicios gestionados en **Firebase**. Esta elección facilita la escalabilidad, el mantenimiento y la integración con múltiples servicios en la nube, eliminando la necesidad de gestionar un *backend* propio. A continuación en la **Figura 6.1** podemos ver los elementos principales de la aplicación.



**Figura 6.1:** Arquitectura general.

- **Frontend:** **Flutter** se encarga de la interfaz de usuario y la lógica de presentación. Su arquitectura reactiva y el sistema de *widgets* permiten construir una experiencia visual coherente y dinámica que se adapta a diferentes dispositivos y plataformas. Coordina además la interacción con el *backend* y la gestión local de datos, garantizando una comunicación eficiente y segura con **Firebase**.
- **Backend:** **Firebase** provee funcionalidades esenciales para la aplicación, como autenticación de usuarios, almacenamiento y sincronización de datos en tiempo real mediante **Firestore**. Esta plataforma en la nube permite manejar la mayor parte de la lógica de servidor sin necesidad de implementar un *backend* tradicional, acelerando el desarrollo y reduciendo la complejidad.

- **Persistencia local:** Para mejorar la experiencia de usuario y asegurar la disponibilidad de ciertos datos sin conexión a internet, se utiliza `SharedPreferences` para almacenar configuraciones, preferencias y otros datos ligeros de forma local en el dispositivo. Esto permite que la aplicación funcione de forma rápida y eficiente, incluso en situaciones de conectividad limitada.

# 7. Desarrollo de la Aplicación

En este capítulo se detallan los aspectos principales del desarrollo de la aplicación móvil **Mi Garaje**. Se describe la organización del proyecto, las tecnologías y herramientas implementadas, así como las decisiones técnicas adoptadas durante el desarrollo. Además, se explican las funcionalidades implementadas y la forma en que se integran para ofrecer una experiencia de usuario coherente y eficiente.

La aplicación ha sido desarrollada utilizando **Flutter**. Se han empleado patrones modernos para la gestión del estado, la navegación y la interacción con los servicios en la nube. A continuación, se presenta la estructura general del proyecto y los componentes clave que conforman la aplicación.

## 7.1. Estructura del Proyecto

La aplicación se ha estructurado bajo un enfoque modular que facilita la separación de responsabilidades, la mantenibilidad y la escalabilidad. El código fuente se organiza en capas y carpetas que reflejan claramente su funcionalidad y ámbito. A continuación se describen las principales capas y la organización física del proyecto:

- **Modelo:** Define las entidades del dominio, tales como `Usuario`, `Vehículo`, `Actividad` y `Familia`. Estas clases representan la estructura de datos y encapsulan reglas básicas de validación. (La [sección 4.3](#) profundiza en este aspecto).
- **Servicios:** Contiene la lógica para la interacción con **Firebase**, incluyendo autenticación, almacenamiento y consultas a **Firestore**. Por ejemplo, el servicio `AuthService` gestiona la autenticación de usuarios, mientras que `GarageService` se encarga de la gestión de vehículos y actividades. Además, el archivo `GlobalTypesService` carga tipos predefinidos desde un archivo JSON.
- **Gestión de estado:** Se utiliza **Riverpod** como solución para la gestión del estado. Esta capa centraliza la lógica de negocio y sincroniza los datos con Firebase. Se emplean varios `StateNotifiers` o `AsyncNotifiers` (como `AuthNotifier`, `GarageNotifier`, `ActivityNotifier`) para manejar estados específicos de la aplicación.
- **Presentación:** Compuesta por *widgets* organizados en pantallas y componentes reutilizables. Esta capa presenta la información al usuario y responde dinámicamente a los cambios de estado.

### 7.1.1. Organización Física del Proyecto

La estructura de carpetas y archivos está diseñada para reflejar la modularidad y funcionalidad de la aplicación, facilitando la localización y el mantenimiento del código. A continuación, se describen las principales carpetas del proyecto:

- **lib/**: Carpeta raíz que contiene todo el código fuente en **Dart**.
- **models/**: Clases que representan las entidades del dominio.
- **services/**: Servicios que gestionan la comunicación con **Firebase** y el almacenamiento local mediante **SharedPreferences**.
- **notifiers/**: Gestores de estado implementados con **Riverpod**.
- **views/**: Contiene la interfaz de usuario.
- **screens/**: Pantallas principales de la aplicación.
- **widgets/**: Componentes UI reutilizables.
- **utils/**: Funciones auxiliares, constantes y *helpers* usados en varias partes del proyecto.
- **l10n/**: Archivos para la localización y traducción de la aplicación, permitiendo soporte multilenguaje mediante **Flutter Intl** u otros paquetes similares.
- **shared/**: Elementos compartidos en distintas partes de la aplicación, organizada en subcarpetas como:
  - **constants/**: Constantes globales usadas en toda la aplicación.
  - **routes/**: Definición y configuración del sistema de rutas y navegación.
  - **utils/**: Funciones auxiliares reutilizables.
  - **themes/**: Definición de temas visuales, incluyendo configuraciones para modo claro y oscuro.
- **assets/**: Recursos estáticos como imágenes y archivos JSON.

Esta estructura modular y bien definida facilita el desarrollo independiente de cada módulo, mejora la reutilización del código y contribuye a la escalabilidad de la aplicación. Asimismo, favorece una rápida localización de funcionalidades y mejora la legibilidad global del proyecto.

---

## 7.2. Backend con Firebase

En el desarrollo de la aplicación, se utilizó **Firebase Firestore** como sistema de gestión de base de datos en tiempo real, que permite almacenar y sincronizar los datos de manera eficiente. A través de **Firebase Authentication**, se gestionó el registro y la autenticación de los usuarios, garantizando la seguridad y facilidad de acceso.

### 7.2.1. Gestión de la Autenticación de Usuarios

Para gestionar la autenticación de los usuarios de manera segura y eficiente, se utilizó **Firebase Authentication**, que permite la integración de varios métodos de inicio de sesión. En este proyecto se implementaron tres métodos principales:

- **Autenticación con correo electrónico y contraseña:** Esta opción permite a los usuarios registrarse utilizando su dirección de correo electrónico y una contraseña segura. **Firebase Authentication** valida automáticamente los datos ingresados, asegurando que el correo electrónico no esté registrado previamente y que la contraseña cumpla con los requisitos de seguridad establecidos. Durante el proceso de registro, se solicita un nombre de usuario, que es un nombre personalizado asignado por el usuario, pero que no tiene valor para el inicio de sesión ni es único, por lo que varios usuarios pueden elegir el mismo nombre. Además, el proceso de inicio y cierre de sesión se maneja de forma sencilla y eficiente.
- **Autenticación anónima:** Para facilitar el acceso de usuarios que no deseen registrarse, se implementó la opción de autenticación anónima. Este método permite a los usuarios acceder a la aplicación sin necesidad de proporcionar datos personales como el correo electrónico o la contraseña. A pesar de ser un acceso anónimo, **Firebase** genera un identificador único para cada usuario, lo que permite asociar sus acciones dentro de la aplicación.
- **Autenticación con Google:** Esta opción permite a los usuarios iniciar sesión utilizando su cuenta de Google. Al integrar este método, los usuarios pueden acceder rápidamente a la aplicación sin necesidad de crear una cuenta manualmente. **Firebase Authentication** maneja de forma transparente la autenticación y la gestión de sesiones, lo que facilita la experiencia del usuario.

Cada uno de estos métodos de autenticación se integra de manera fluida en la aplicación, ofreciendo a los usuarios la opción que mejor se adapte a sus necesidades. Además, en cualquier momento, un usuario que haya iniciado sesión de manera anónima puede convertir su cuenta en una cuenta registrada con correo electrónico o con Google. De igual manera, los usuarios con una cuenta normal pueden transformarla en una cuenta de Google, proporcionando flexibilidad en la gestión de su acceso a la aplicación.

---

En todas las opciones de autenticación, los usuarios tienen la posibilidad de eliminar su cuenta, lo que también elimina todos los datos asociados a ella. Cabe destacar que solo las cuentas vinculadas a un correo electrónico y las de Google permiten realizar el cierre de sesión. En el caso de las cuentas anónimas, si el usuario cierra sesión, no podría volver a acceder sin credenciales, por tanto no se contempla esta opción.

### 7.2.2. Estructura y Almacenamiento de Datos

Para la gestión de datos de la aplicación se utilizó **Firebase Firestore**, una base de datos NoSQL flexible, escalable y en tiempo real. **Firestore organiza los datos en colecciones y documentos**, permitiendo un acceso eficiente y estructurado. Para este proyecto, se diseñó la siguiente estructura de base de datos:

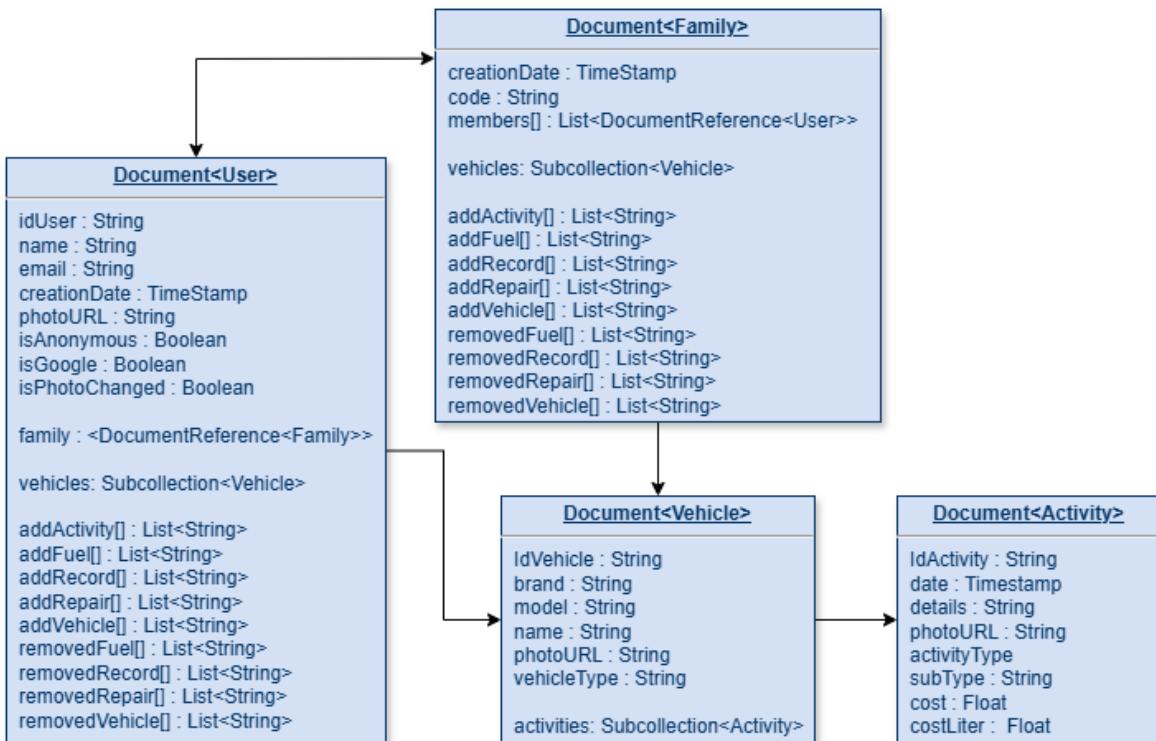
- **Colección de usuarios:** La colección principal de la base de datos está centrada en los usuarios. Cada documento representa un usuario único, identificado por el `userId` proporcionado por **Firebase Authentication**. En este documento se almacenan datos personales como el nombre, correo electrónico, fecha de creación, si es anónimo o autenticado con Google, y la URL de su foto de perfil. Además, cada usuario posee una subcolección de vehículos con los vehículos que ha registrado. En caso de que el usuario pertenezca a una familia, el documento incluye una referencia (`DocumentReference`) al documento correspondiente en la colección de familias.
- **Colección de familias:** Además de los usuarios individuales, se creó una **colección de familias**. Esta colección permite agrupar a varios usuarios bajo un mismo identificador de familia. Cada documento representa una familia, identificada por un código único, una fecha de creación, y una lista de referencias (`DocumentReference`) a los usuarios que pertenecen a ella. Las familias, al igual que los usuarios, también pueden tener una subcolección de vehículos y, por tanto, actividades relacionadas.
- **Subcolección de vehículos:** Tanto los usuarios como las familias pueden tener vehículos asociados. Cada vehículo está representado como un documento. Un vehículo contiene información como la marca, el modelo, un nombre personalizado y el tipo de vehículo.
- **Colección de actividades:** Cada vehículo tiene asociada una **subcolección de actividades**, donde se almacenan registros detallados de las actividades realizadas. Las actividades incluyen información como el tipo de actividad, el subtipo concreto, la fecha en la que se realizó, y el precio. Dependiendo del tipo de actividad varía la información almacenada.

Por diseño, el sistema cuenta con dos colecciones principales: *users* y *families*. Estas funcionan como puntos de entrada al resto de la estructura de datos, permitiendo

---

escalar de manera independiente los perfiles individuales y los grupos familiares dentro de la aplicación.

La **Figura 7.1** ilustra la estructura de la base de datos en **Firestore**, mostrando cómo se organizan y estructuran los datos mediante colecciones, subcolecciones y referencias entre documentos.



**Figura 7.1:** Diagrama de colecciones en Firebase Firestore.

### 7.2.3. Reglas de Seguridad en Firebase Firestore

Para garantizar la seguridad y la privacidad de los datos, se implementaron **reglas de seguridad en Firestore**. Estas reglas regulan el acceso y las operaciones permitidas sobre las colecciones de **Usuarios**, **Vehículos**, **Actividades** y **Familias**, asegurando que los datos solo sean accesibles por los usuarios autorizados y evitando accesos no deseados.

Las reglas de seguridad implementadas están estructuradas bajo el principio de **mínimos privilegios**, lo que significa que los usuarios solo tienen acceso a los datos necesarios para realizar sus funciones dentro de la aplicación, y se evita que un usuario modifique información que no le pertenece. Esta estrategia protege tanto la privacidad de los usuarios como la integridad de los datos.

Sin embargo, en el caso de la lectura de los datos de los usuarios, se ha decidido no restringir el acceso en base al `userID` del usuario autenticado. Esto se debe a la necesidad de permitir que los miembros de una misma familia puedan acceder a la información básica de los demás, como su nombre y foto de perfil.

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    // Reglas para la colección "users"
    match /users/{userId} {
      allow read: if request.auth != null;
      allow write: if request.auth != null && request.auth.uid == userId;

      match /vehicles/{vehicleID} {
        allow read, write: if request.auth != null && request.auth.uid == userId;

        match /activities/{activityId} {
          allow read, write: if request.auth != null && request.auth.uid == userId;
        }
      }
    }

    // Reglas para la colección "families"
    match /families/{familyId} {
      allow write, read: if request.auth != null && request.auth.uid in resource.data.members;

      match /vehicles/{vehicleID} {
        allow read, write: if request.auth != null && request.auth.uid in get(/databases/${database}/documents/families/${familyId}).data.members;

        match /activities/{activityId} {
          allow read, write: if request.auth != null && request.auth.uid in get(/databases/${database}/documents/families/${familyId}).data.members;
        }
      }
    }
  }
}
```

**Figura 7.2:** Reglas de seguridad en Firebase Firestore.

## 7.3. Persistencia local y preferencias del usuario

Además de utilizar **Firebase** como fuente principal de datos remotos, la aplicación implementa mecanismos de **persistencia local**. Esta persistencia se utiliza tanto para guardar preferencias del usuario como para precargar datos estáticos necesarios para la lógica de la aplicación.

### 7.3.1. SharedPreferences

Para gestionar configuraciones que deben conservarse entre sesiones, se emplea la librería **SharedPreferences**, una solución basada en almacenamiento **clave-valor** en el dispositivo. Esta herramienta permite guardar y recuperar información local de forma sencilla y eficiente. Entre las configuraciones persistidas se encuentran:

- **Tema de la aplicación:** Se almacena si el usuario prefiere el modo claro u oscuro. Esta configuración se recupera al iniciar la *app* y se aplica automáticamente, asegurando coherencia visual sin necesidad de que el usuario la configure cada vez.
- **Idioma seleccionado:** Se guarda la preferencia de idioma para ofrecer una experiencia localizada persistente. Aunque **Flutter** puede usar la configuración del

sistema, se permite que el usuario elija manualmente otro idioma dentro de la *app*.

### 7.3.2. Precarga de datos desde JSON local

Además del almacenamiento de preferencias, la aplicación también realiza una precarga de datos estáticos necesarios para el funcionamiento de ciertas funcionalidades. Para ello, se emplea un archivo JSON localizado en la carpeta `/assets/json`, que contiene listas predefinidas de tipos utilizados en distintos formularios y vistas de la *app*. Estos tipos incluyen, entre otros:

- **Tipos de combustible** (e.g., Gasolina, Diésel, Eléctrico).
- **Tipos de reparación** (e.g., Cambio de aceite, Frenos, Revisión general).
- **Tipos de actividades** (e.g., Mantenimiento, Repostaje, Documento).
- **Tipos de vehículos** (e.g., Coche, Moto, Camión).

Esto permite desacoplar los datos estáticos de la lógica de *backend*, ofreciendo una forma flexible de actualizar o extender la información sin necesidad de modificar **Firestore**.

## 7.4. Gestión del estado

La gestión del estado es un componente esencial en el desarrollo de aplicaciones modernas, ya que permite mantener sincronizados los datos de la lógica de negocio con la interfaz de usuario de forma reactiva. En este proyecto se ha utilizado **Riverpod**, una solución escalable y moderna que promueve una arquitectura limpia, modular y fácilmente testeable.

**Riverpod** fue seleccionado frente a otras opciones como **Provider** o **Bloc**, debido a sus ventajas técnicas:

- No depende del **BuildContext**, lo que facilita el testeo y la reutilización de lógica fuera del árbol de *widgets*.
- Permite declarar estados de forma modular, segura y predecible.
- Es compatible con los principios de programación reactiva y funcional.
- Dispone de una sintaxis limpia y potente, con una comunidad activa y documentación bien mantenida.

### 7.4.1. Arquitectura de estado

Se han implementado múltiples **StateNotifier** y **AsyncNotifier**, responsables de manejar el estado asociado a distintos aspectos de la aplicación. A continuación se detalla la funcionalidad de cada uno:

- **AuthNotifier:** Controla la autenticación de usuarios y la lógica asociada a familias.
- **GarageNotifier:** Administra los vehículos registrados por el usuario o familia.
- **ActivityNotifier:** Maneja las actividades asociadas al vehículo seleccionado.
- **GlobalTypesNotifier:** Carga y gestiona los tipos personalizados de vehículos, actividades y subactividades.
- **LocaleNotifier y ThemeNotifier:** Permiten al usuario personalizar el idioma y el tema visual de la aplicación.

Cada uno de estos notifiers se encuentra encapsulado dentro de su propio **Provider**, permitiendo así un consumo reactivo desde cualquier parte de la aplicación.

### 7.4.2. Ejemplo: AuthNotifier

A continuación se presenta un ejemplo detallado de uno de los gestores de estado implementados: **AuthNotifier**. Este patrón es representativo del resto de notifiers utilizados en la aplicación, ya que todos comparten una estructura y filosofía similares basadas en **AsyncNotifier** o **StateNotifier**, dependiendo del caso.

**AuthNotifier** es un gestor de estado avanzado que extiende la clase **AsyncNotifier<AuthState>**, lo que permite manejar operaciones asíncronas relacionadas con la autenticación del usuario y su posible vinculación a una familia dentro de la aplicación. Su funcionamiento se articula en torno a dos componentes principales.

El estado gestionado se modela mediante la clase **AuthState** (ver **Código 7.1**), que encapsula al usuario autenticado. Además, incluye diversas propiedades computadas que permiten simplificar la lógica condicional en la interfaz (como **isUser**, **type**, **isGoogle**, entre otras), así como un método **copyWith** para mantener la inmutabilidad del estado. Estas utilidades permiten a la UI reaccionar con mayor claridad y menos lógica redundante.

---

Código 7.1: Clase AuthState.

```
1 class AuthState {  
2     final User? user;  
3  
4     AuthState({this.user});  
5  
6     bool get isUser => user != null;  
7     bool get isFamily => user!.hasFamily;  
8  
9     String get id => user?.family?.id ?? user!.id!;  
10  
11    String get type => isFamily ? "families" : "users";  
12  
13    bool get isGoogle => user?.isGoogle ?? false;  
14    bool get isPhotoURL => user!.photoURL != null;  
15  
16    bool get isLastMember => user!.family!.members!.length == 1;  
17  
18    AuthState copyWith({User? user}) {  
19        return AuthState(  
20            user: user ?? this.user,  
21        );  
22    }  
23}
```

Por su parte, **AuthNotifier** gestiona todas las operaciones relacionadas con la autenticación y el contexto familiar, como el inicio de sesión, registro, unión a una familia, actualización de perfil o eliminación de cuenta. Para ello, se apoya en los siguientes servicios:

- **AuthService:** Encargado de la autenticación y de recuperar la información del usuario y su familia desde **Firebase Auth** y **Firestore**.
- **GarageService:** Utilizado en procesos como la conversión a familia o la eliminación de cuenta, donde puede ser necesario actualizar o eliminar vehículos.
- **UserTypesService:** Gestiona los tipos personalizados, según el contexto del usuario (individual o familiar).

El método `build()` se ejecuta automáticamente al inicializar el Notifier y tiene como objetivo construir el estado inicial del usuario autenticado. Tal como se muestra en la **Código 7.2**, este método obtiene el usuario actualmente autenticado desde **Firebase** mediante `AuthService` y devuelve una instancia de `AuthState` que encapsula

al usuario. Este enfoque permite iniciar el estado de forma asíncrona y centralizada, aprovechando las capacidades reactivas de **AsyncNotifier**.

Código 7.2: Método build() de la clase AuthNotifier.

```

1  @override
2  FutureOr<AuthState> build() async {
3      try {
4          return AuthState(user: await _authService.currentUser);
5      } catch (e, stackTrace) {
6          throw AsyncError(e, stackTrace);
7      }
8 }
```

Código 7.3: Clase AuthWrapper.

```

1  class AuthWrapper extends ConsumerStatefulWidget {
2      const AuthWrapper({super.key});
3
4      @override
5      ConsumerState<AuthWrapper> createState() => _AuthWrapperState();
6  }
7
8 class _AuthWrapperState extends ConsumerState<AuthWrapper> {
9     @override
10    void initState() {
11        super.initState();
12    }
13
14    @override
15    Widget build(BuildContext context) {
16        final authState = ref.watch(authProvider);
17
18        return authState.when(
19            data: (state) => state.isUser ? HomeView() : LoginView(),
20            loading: () => const SplashScreen(),
21            error: (e, _) => ErrorScreen(errorMessage: e.toString()),
22        );
23    }
24}
```

Gracias al uso de **AsyncNotifier**, la interfaz puede reaccionar ante los distintos estados del flujo asíncrono —cargando, error o datos disponibles— mediante el patrón

`when(...)`. Esto se observa claramente en la **Código 7.3**, donde la clase `AuthWrapper` utiliza el `authProvider` para determinar qué vista mostrar en función del estado actual:

- Mientras se cargan los datos, se muestra una pantalla de carga (`SplashScreen`).
- Si ocurre un error, se presenta una pantalla de error con el mensaje correspondiente.
- Cuando los datos están disponibles, se navega a la vista principal (`HomeView`) o a la vista de inicio de sesión (`LoginView`) según si el usuario está autenticado o no.

Además, dado que el estado se expone a través de un `NotifierProvider`, puede ser consumido de forma reactiva desde cualquier *widget* de la aplicación usando `ref.watch(...)`, como se ejemplifica en esa misma figura.

## 7.5. Diseño de la interfaz y prototipado

Para el diseño de la interfaz de usuario, se ha utilizado **Figma**, una herramienta de prototipado ampliamente empleada en el desarrollo de aplicaciones. Esta plataforma permite crear prototipos visuales e interactivos de alta fidelidad, facilitando la validación temprana de la experiencia de usuario.

Además, la aplicación contempla **dos modos de visualización: modo claro y modo oscuro**, con el objetivo de adaptarse a las preferencias del usuario y ofrecer una experiencia personalizada en distintos entornos de iluminación.

### 7.5.1. Prototipo de interfaz

El prototipo ha sido diseñado bajo principios de **usabilidad, simplicidad y accesibilidad**, priorizando una navegación intuitiva y una disposición clara de los elementos visuales. Las pantallas principales están concebidas para proporcionar acceso rápido a las funcionalidades esenciales de la aplicación.

---

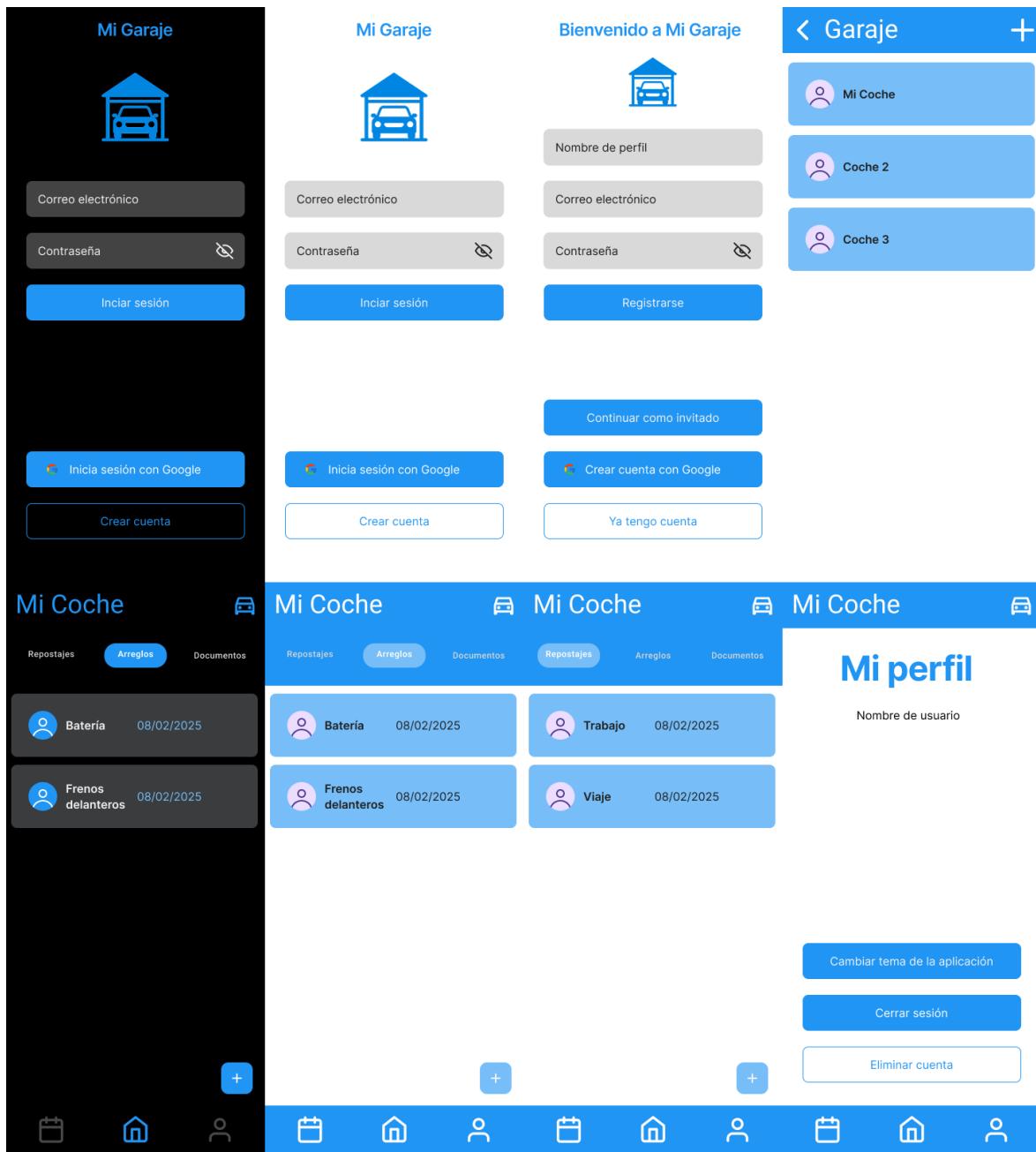


Figura 7.3: Diseño prototipado inicial.

- **Modo claro:** Esta versión utiliza una paleta cromática basada en tonalidades azules. El AppBar y la barra de navegación inferior se presentan en azul, con texto e iconos en blanco para garantizar un buen contraste. Las tarjetas de contenido (*cards*) emplean un azul más claro, con texto en tonos oscuros que favorecen la legibilidad en entornos bien iluminados.

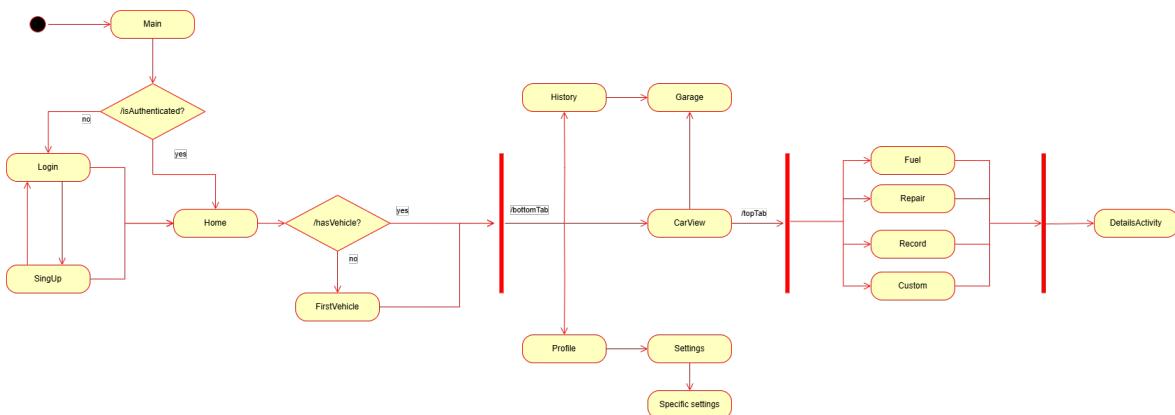
- **Modo oscuro:** Pensado para su uso en condiciones de baja iluminación, este modo adopta un fondo general oscuro con texto claro, reduciendo el esfuerzo visual del usuario. Se mantienen las tonalidades azules como color principal de la interfaz. Las tarjetas presentan un fondo ligeramente más claro que el general, generando el contraste necesario sin comprometer la estética.

Este diseño corresponde a un **prototipo inicial**, que sirvió como base de referencia para la implementación de la aplicación. Sin embargo, durante el desarrollo final, se fueron realizando ajustes en función de pruebas, limitaciones técnicas y decisiones de usabilidad. En el **Anexo B** se incluye la **versión final de la interfaz de la aplicación**, tal como fue desarrollada e implementada en **Flutter**.

### 7.5.2. Mapa de navegación

Para complementar el prototipo visual, se ha desarrollado un **mapa de navegación** que representa de forma esquemática la estructura y el flujo de pantallas de la aplicación. Esta herramienta permite comprender la lógica de navegación del usuario, así como identificar posibles mejoras en la arquitectura de interacción.

El mapa se presenta a continuación en la **Figura 7.4**, y ofrece una visión clara de cómo se enlazan las distintas secciones de la *app*, sirviendo como guía para asegurar una experiencia de usuario coherente durante el desarrollo.



**Figura 7.4:** Mapa de navegación.

El mapa refleja un flujo de navegación jerárquico en el que se distinguen claramente las pantallas principales: **Login**, **SignUp** y **Home**, junto con sus rutas secundarias asociadas. Al iniciar la aplicación, se realiza una comprobación del estado de autenticación del usuario. En caso afirmativo, se accede directamente a la pantalla **Home**; si no, se redirige al usuario a la pantalla de **Login**, desde la cual puede iniciar sesión o registrarse mediante la pantalla **SignUp**.

Una vez registrado o autenticado correctamente, se verifica si el usuario dispone de vehículos asociados. Si no es así, se presenta una pantalla inicial donde se le guía para crear su primer vehículo o unirse a una familia existente. Si ya existen vehículos, se accede directamente a **Home**, desde donde comienza el flujo principal de la aplicación.

Dentro de la pantalla **Home**, el usuario puede acceder a distintas secciones clave:

- **History**, donde se muestran estadísticas relacionadas con los vehículos.
- **Vehicle**, donde se presenta un listado categorizado de actividades del vehículo seleccionado, incluyendo también la posibilidad de acceder a la configuración de dicho vehículo.
- **Profile**, desde donde el usuario puede consultar información sobre los miembros de su familia, sus vehículos y acceder a la sección **Settings**, destinada a la personalización de la aplicación y la gestión de la cuenta.

Este esquema de navegación no solo estructura el acceso a las distintas funcionalidades de manera lógica y jerárquica, sino que también asegura una experiencia de usuario fluida, adaptándose dinámicamente al estado de la sesión y al contenido disponible. Gracias a este enfoque, la aplicación guía al usuario de forma natural a lo largo del recorrido funcional, reduciendo la fricción en la interacción y mejorando la usabilidad general del sistema.

## 7.6. Sistema de rutas

La navegación se gestiona a través de un sistema de rutas centralizado, definido en la clase **AppRoutes** (ver **Código 7.4**). Este sistema permite controlar la transición entre pantallas y facilitar la navegación con argumentos, mejorando la organización del flujo de la interfaz de usuario. La aplicación combina dos mecanismos principales para el enrutamiento:

- **Mapa de rutas estáticas (`routes`)**: Asociaciones directas entre una cadena de texto (ruta) y un *widget*. Estas se utilizan para las pantallas más básicas y no requieren argumentos.
  - **Rutas generadas dinámicamente (`onGenerateRoute`)**: Este método se utiliza para manejar rutas que requieren argumentos o transiciones personalizadas. Permite una mayor flexibilidad y control sobre la navegación.
-

Código 7.4: Clase AppRoutes.

```
1 class AppRoutes {
2     static Map<String, WidgetBuilder> routes = {
3         RouteNames.login: (context) => LoginView(),
4         RouteNames.signup: (context) => SignupView(),
5         RouteNames.home: (context) => HomeView(),
6     };
7
8     static Route<dynamic> onGenerateRoute(RouteSettings settings) {
9         final args = settings.arguments != null
10            ? settings.arguments as Map<String, dynamic>
11            : {};
12         switch (settings.name) {
13             case RouteNames.loading:
14                 return MaterialPageRoute(
15                     builder: (context) => SplashScreen(onInit: args['onInit']),
16                 );
17
18             case RouteNames.firstCar:
19                 return _slideTransition(
20                     FirstCar());
21
22             case RouteNames.garage:
23                 return _slideTransition(const GarageView());
24
25             case RouteNames.activity:
26                 return _slideTransition(ActivityView(
27                     activity: args['activity'],
28                 ));
29
30             case RouteNames.settings:
31                 return _slideTransition(SettingsView());
32
33             case RouteNames.types:
34                 return _slideTransition(TypesView(type: args['type']));
35         }
36     }
}
```

Algunas pantallas utilizan una animación de transición personalizada basada en deslizamiento lateral (*slide*). Esta transición se aplica mediante la función `_slideTransition` (ver **Código 7.5**), que retorna un `PageRouteBuilder` con una animación de entrada desde la derecha. Este tipo de transición se aplica, por ejemplo, a pantallas como `FirstCar`, `GarageView`, `ActivityView` o `SettingsView`, ofreciendo una navegación

más intuitiva y coherente.

Código 7.5: Método `_slideTransition` de la clase `AppRoutes`.

```
1 static PageRouteBuilder _slideTransition(Widget page) {
2     return PageRouteBuilder(
3         pageBuilder: (context, animation, secondaryAnimation) => page,
4         transitionsBuilder: (context, animation, secondaryAnimation, child) ←
5             ↗ {
6                 const begin = Offset(1.0, 0.0);
7                 const end = Offset.zero;
8                 const curve = Curves.easeInOut;
9                 var tween =
10                     Tween(begin: begin, end: end).chain(CurveTween(curve: curve));
11                 var offsetAnimation = animation.drive(tween);
12                 return SlideTransition(position: offsetAnimation, child: child);
13             },
14     );
}
```

## 7.7. Capa de presentación

La interfaz de usuario de la aplicación está construida a partir de *widgets* organizados en una arquitectura clara que separa responsabilidades, facilitando tanto el mantenimiento como la escalabilidad del código. Esta capa está estructurada principalmente en dos carpetas:

- **views/screens/**: Contiene los *widgets* que representan pantallas completas, cada una asociada a una funcionalidad principal de la aplicación (como inicio, perfil, garage, etc.). Estas vistas constituyen los puntos de entrada del usuario dentro del flujo de navegación.
- **views/widgets/**: Agrupa componentes reutilizables de interfaz que encapsulan elementos comunes, como tarjetas de actividad, botones personalizados, diálogos, formularios, entre otros. Esta división promueve la reutilización, la coherencia visual y una gestión eficiente del código fuente.

Todos los componentes siguen un enfoque declarativo y reactivo mediante **Riverpod**, utilizando `ConsumerWidget` y `ConsumerStatefulWidget` para suscribirse a los `Providers` necesarios. Esto permite que la interfaz reaccione automáticamente ante cambios de estado, eliminando la necesidad de una gestión manual del ciclo de vida del *widget*.

---

A continuación, se describen tres componentes clave que ejemplifican el enfoque modular, reutilizable y reactivo adoptado en la capa de presentación. El primero corresponde a una pantalla completa que permite visualizar y gestionar las actividades asociadas a un vehículo específico. El segundo es un *widget* personalizado que representa cada actividad mediante una tarjeta. Finalmente, el tercero es un diálogo modal que facilita la creación o edición de actividades, integrando formularios dinámicos y funciones de validación.

## Ejemplo: CarTabView

La clase **CarTabView** (**Figura 7.5**) representa una pantalla completa dentro de la aplicación que permite al usuario visualizar y gestionar las distintas actividades asociadas a un vehículo específico. Estas actividades están categorizadas por tipo —como repostaje, mantenimiento, documentos o personalizadas— y se organizan mediante un componente **TabBar**, lo que facilita la navegación por categorías.

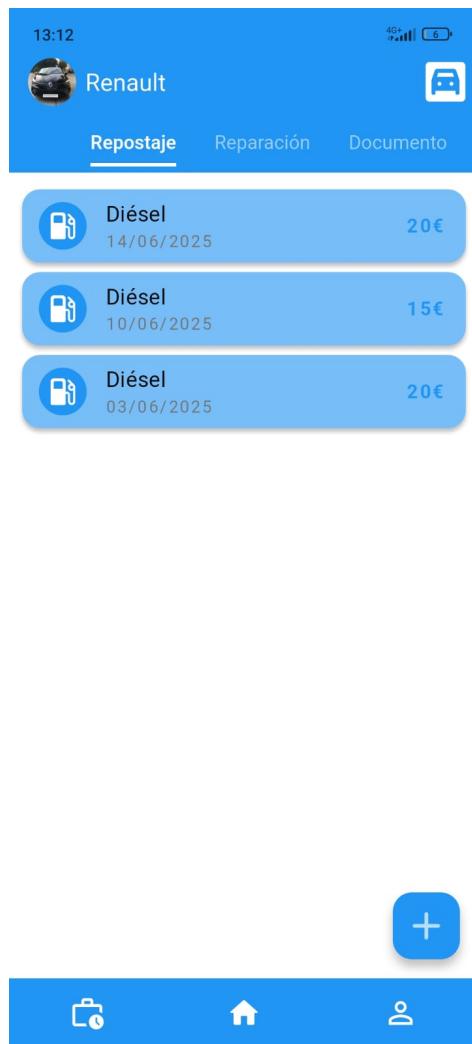
Este *widget* extiende de **Consumer StatefulWidget**, lo que le permite integrarse con el modelo reactivo de **Riverpod**. De esta forma, puede suscribirse a múltiples **providers** para obtener y actualizar datos en tiempo real. Entre los **providers** utilizados se encuentran:

- **garageProvider**: gestiona la información del vehículo seleccionado.
- **activityProvider**: proporciona y actualiza las actividades del vehículo.
- **tabStateProvider**: mantiene el estado y el índice activo de la pestaña.
- **globalTypesProvider**: define y actualiza los tipos de actividades globales disponibles.

Gracias a esta arquitectura, la interfaz de usuario se actualiza automáticamente ante cualquier cambio en el estado de la aplicación, garantizando así una visualización siempre coherente y actualizada.

La navegación entre las pestañas se gestiona mediante un **TabController**, el cual se inicializa de forma dinámica en función de los tipos de actividad disponibles. Este controlador se sincroniza con el **tabStateProvider**, lo que permite mantener el estado del índice activo incluso tras reconstrucciones del *widget*. Además, la clase implementa correctamente el ciclo de vida, liberando recursos a través del método **dispose** para evitar fugas de memoria.

En la parte superior de la interfaz (**AppBar**), se muestra el nombre y la imagen del vehículo seleccionado en el lado izquierdo, y un botón de acceso directo a la pantalla del garaje en el lado derecho. Inmediatamente debajo se encuentra el **TabBar**, que refleja dinámicamente las categorías de actividad disponibles, las cuales pueden variar según la configuración del usuario.



**Figura 7.5:** Interfaz de la clase CarTabView.

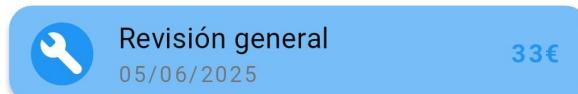
El contenido de cada pestaña se genera de forma reactiva mediante el método `_buildTabContent`, que utiliza el `activityProvider` para obtener una lista filtrada de actividades según la categoría seleccionada, ordenadas por fecha en orden descendente. Cada una de estas actividades se representa mediante el *widget ActivityCard*, que encapsula su presentación y comportamiento.

En la esquina inferior derecha, se encuentra un `FloatingActionButton` que permite al usuario añadir nuevas actividades. Al pulsarlo, se muestra un diálogo personalizado (`DialogAddActivity`) que facilita la creación de nuevas entradas. Asimismo, el contenido de cada pestaña incluye un `RefreshIndicator` que permite al usuario actualizar manualmente la lista de actividades deslizando hacia abajo. Esta acción invalida el `activityProvider`, forzando la recarga de los datos más recientes.

---

## Ejemplo: ActivityCard

La clase **ActivityCard** (**Figura 7.6**) es un *widget* personalizado que representa de forma visual y compacta una actividad específica asociada a un vehículo. Hereda de **ConsumerWidget**, lo que le permite integrarse de manera reactiva al sistema de estado gestionado por **Riverpod**, garantizando que los datos presentados estén siempre actualizados según el contexto de la aplicación.



**Figura 7.6:** Interfaz de la clase ActivityCard.

Este *widget* recibe como parámetro una instancia del modelo **Activity**, que contiene todos los datos relevantes para su representación. Su principal objetivo es ofrecer una presentación clara, accesible y coherente de cada actividad, así como permitir la interacción directa por parte del usuario.

### Estructura visual

La interfaz del *widget* está compuesta por un **Card** que contiene una fila horizontal (**Row**), organizada en tres elementos clave:

- **Ícono representativo:** En el extremo izquierdo, se muestra un **CircleAvatar** que incluye un ícono visualmente asociado al tipo principal de actividad (**fuel**, **repair**, **record**, **custom**). Para ello, se utilizan íconos de la librería **Material Icons**, facilitando así la identificación rápida del tipo de actividad.
- **Información textual:** En el centro de la tarjeta se encuentra una columna que presenta el subtipo de la actividad (como **Diesel**, **Cambio de aceite**, etc.), el cual es localizado automáticamente al idioma configurado mediante **AppLocalizations**. También se muestra la fecha de la actividad, formateada en el estilo dd/MM/yyyy, lo que mejora su legibilidad y comprensión.
- **Precio asociado:** En el lado derecho de la tarjeta, si la actividad contiene un coste definido, se visualiza el importe correspondiente en euros.

### Interacciones

**ActivityCard** permite al usuario ejecutar dos tipos de interacción sobre cada elemento:

- **Tap corto:** Al tocar la tarjeta, se navega hacia una pantalla de detalle para visualizar o editar la información de la actividad seleccionada. Esta navegación

se realiza mediante `Navigator.pushNamed`, pasando la instancia completa del objeto `Activity` como argumento para su consulta.

- **Tap largo:** Al mantener presionada la tarjeta, se activa un diálogo modal de confirmación para eliminar la actividad actual. En caso de que el usuario confirme la acción, se invoca el método `deleteActivity` del `activityProvider`. Si la operación se realiza con éxito, se muestra un mensaje de confirmación mediante un *toast*. En caso de error, se informa al usuario mediante un mensaje de fallo apropiado.

Esta combinación de presentación clara, integración con el estado global y soporte para múltiples gestos de interacción convierte a **ActivityCard** en un componente esencial de la interfaz de usuario.

## Ejemplo: DialogAddActivity

La clase **DialogAddActivity** (**Figura 7.7**) es un *widget* modal de **Flutter** que facilita al usuario la creación o edición de actividades asociadas a un vehículo. Este componente está diseñado con un enfoque reactivo, utilizando el paquete `image_picker` para la integración con la galería del dispositivo, permitiendo al usuario adjuntar imágenes a sus actividades.

### Estructura visual

El *widget* se presenta como un diálogo modal que contiene un formulario dinámico, adaptado según el tipo de actividad que el usuario desea añadir o modificar. Los campos del formulario incluyen:

- **Tipo de actividad:** Desplegable que presenta los subtipos disponibles, suministrando listas según el tipo principal seleccionado (por ejemplo, `repostaje`, `mantenimiento`, `documentos`, `actividades personalizadas`).
- **Fecha de la actividad:** Selector de fecha que permite establecer cuándo se realizó la actividad.
- **Coste asociado:** Campo numérico para ingresar el coste económico de la actividad.
- **Precio por litro:** Campo exclusivo para actividades de repostaje, donde se puede ingresar el precio unitario del combustible.
- **Detalles adicionales:** Campo opcional para añadir observaciones o comentarios relacionados con la actividad.

En caso de actividades personalizadas, el usuario dispone de un campo de texto para introducir manualmente el tipo de actividad, brindando flexibilidad y personalización.



**Figura 7.7:** Interfaz de la clase DialogAddActivity.

## Gestión de imágenes

El diálogo incorpora la funcionalidad para seleccionar una imagen desde la galería del dispositivo a través del paquete `image_picker`. La imagen seleccionada se muestra en una vista previa dentro del formulario, y el usuario tiene la opción de eliminarla antes de guardar, asegurando un control completo sobre el contenido multimedia asociado.

## Validación y guardado

Antes de guardar, el *widget* realiza una validación exhaustiva para asegurar que todos los campos obligatorios estén correctamente completados y que los datos sean coherentes. Una vez validada la información, se crea una instancia del modelo `Activity`, adaptada al tipo específico seleccionado.

Posteriormente, se invocan los métodos correspondientes del `provider` para añadir una nueva actividad o actualizar una existente dentro del estado global de la aplicación. Durante este proceso, se ofrecen mensajes informativos mediante `toasts` que confirman el éxito de la operación o alertan al usuario en caso de error, garantizando una experiencia clara y fluida.

## 7.8. Personalización del usuario

La personalización de la experiencia de usuario ha sido una prioridad en el desarrollo de la aplicación. Para ello, se ha implementado un sistema que permite al usuario seleccionar tanto el idioma de la interfaz como el tema visual, de forma dinámica y persistente.

Tal como se ha detallado en apartados anteriores, las preferencias del usuario se almacenan de forma persistente a nivel local. Aunque este detalle técnico ya fue tratado en el punto correspondiente sobre persistencia local, cabe señalar que el sistema garantiza que las preferencias de idioma y tema se mantengan incluso tras cerrar y volver a abrir la aplicación.

### 7.8.1. Cambio de tema

La aplicación permite alternar entre **modo claro** y **modo oscuro** desde el apartado de configuración. Este cambio se aplica de forma inmediata a toda la interfaz, gracias al sistema de *theming* de **Flutter** y su integración con el árbol de *widgets*.

El estado del tema se gestiona a través de un **ThemeNotifier**, que expone el modo activo mediante un **Provider** de **Riverpod**. Este **Notifier** ofrece métodos para alternar entre `ThemeMode.light` y `ThemeMode.dark`, y su valor es observado directamente desde el *widget* raíz de la aplicación (`MaterialApp.router`), mediante la instrucción:

```
1 themeMode: ref.watch(themeProvider)
```

### 7.8.2. Soporte multilingüe

La aplicación también permite al usuario seleccionar manualmente el idioma, independientemente del configurado en el sistema del dispositivo. Para ello, se utiliza el sistema de localización de **Flutter** junto con archivos `.arb` generados automáticamente y almacenados en el directorio `110n/`.

El idioma activo se gestiona mediante un **LocaleNotifier**, cuyo estado también es observado desde `MaterialApp.router` mediante la propiedad:

```
1 locale: ref.watch(localeProvider)
```

Esto asegura que al cambiar el idioma, toda la interfaz se actualiza automáticamente sin necesidad de reiniciar la aplicación ni reconstruir *widgets* manualmente.

## 8. Conclusiones y vías futuras

El desarrollo de la aplicación **Mi Garaje** ha permitido alcanzar el objetivo principal de este proyecto: crear una herramienta móvil multiplataforma, gratuita y funcional que facilite a los usuarios la gestión eficiente del mantenimiento de sus vehículos. A lo largo del proceso se ha seguido una planificación estructurada que ha favorecido un desarrollo iterativo y una validación progresiva del sistema.

Entre los principales logros del proyecto destacan:

- La implementación de una interfaz intuitiva, adaptable a diferentes idiomas y modos de visualización.
- El uso de tecnologías modernas como **Flutter** y **Firebase**, que han aportado flexibilidad, escalabilidad y facilidad de mantenimiento.
- La posibilidad de gestionar múltiples vehículos y usuarios dentro de una misma familia, facilitando la colaboración y el seguimiento compartido.
- La personalización de categorías, tipos de vehículos y actividades, lo cual amplía la aplicabilidad de la herramienta a diferentes contextos.

Además, me ha permitido aplicar de forma integrada y práctica varios conocimientos clave adquiridos durante el grado, entre ellos:

- **Gestión de proyectos de desarrollo de software:** las técnicas de especificación de requisitos y la planificación mediante **diagramas de Gantt** fueron fundamentales para estructurar el trabajo y redactar el **SRS**.
- **Interfaces de usuario:** los principios de usabilidad y accesibilidad guiaron el prototipado de la interfaz y la validación con usuarios.
- **Programación Orientada a Objetos:** resultó clave para organizar el código, aplicando conceptos fundamentales como herencia, encapsulación, polimorfismo y composición, lo que permitió un diseño modular, reutilizable y fácil de mantener.
- **Aplicaciones Distribuidas:** resultó de utilidad para diseñar servicios reutilizables dentro de la arquitectura de la aplicación, así como para implementar la lectura e interpretación de archivos **JSON**.

Además de afianzar conocimientos previos, este proyecto me ha permitido adquirir nuevas competencias técnicas y metodológicas. He aprendido a trabajar con **Firebase** de forma más profunda, explorando aspectos como la seguridad con reglas personalizadas, la estructura óptima de datos en **Firestore**. También he desarrollado experiencia práctica con **Flutter** en un entorno real, enfrentándome a desafíos como la gestión del estado, la persistencia local y la navegación dinámica entre pantallas. A nivel metodológico, he mejorado mi capacidad para documentar el desarrollo de forma rigurosa, aplicar principios de desarrollo ágil en un entorno individual y realizar pruebas de usabilidad con usuarios reales, lo que ha Enriquecido mi comprensión del ciclo completo de vida de una aplicación móvil.

No obstante, el proyecto ha revelado diversas oportunidades de evolución que podrían abordarse en desarrollos futuros, tanto para enriquecer la experiencia del usuario como para incrementar la robustez técnica y funcional del sistema. Entre las principales líneas de mejora destacan:

- **Incorporación de notificaciones *push*,** para recordar al usuario próximas tareas de mantenimiento o vencimiento de documentos.
- **Ampliación del sistema de familias,** incluyendo funcionalidades como un chat interno o un historial de acciones realizadas por los distintos miembros.
- **Sistema de copia de seguridad y restauración en la nube,** configurable por el usuario, que facilite la migración entre dispositivos y la recuperación de datos.
- **Gestión de mantenimientos predictivos basada en inteligencia artificial,** que analice el historial de actividades y patrones de uso para estimar cuándo será necesario un próximo cambio de aceite, frenos, revisión, etc.
- **Asistente de configuración inicial,** que guíe al usuario paso a paso en su primera vez configurando vehículos, actividades y preferencias, reduciendo la curva de aprendizaje.
- **Internacionalización completa,** con soporte para más idiomas, unidades de medida, formatos de fecha y moneda, para facilitar su adopción en otros países.

En definitiva, **Mi Garaje** representa una solución sólida y extensible, preparada para seguir evolucionando con nuevas funcionalidades y adaptándose a las necesidades cambiantes de sus usuarios, a la vez que ha supuesto para mí una valiosa experiencia profesional y académica.

---

# Glosario

**Android** Sistema operativo desarrollado por Google, diseñado principalmente para dispositivos móviles como smartphones y tablets.

**Aplicación móvil (App)** Software diseñado para ser ejecutado en dispositivos móviles como teléfonos inteligentes o tabletas.

**Backend** Parte del sistema que se ejecuta en el servidor y gestiona la lógica, base de datos, autenticación y procesamiento de datos.

**Base de datos NoSQL** Tipo de base de datos que no utiliza el modelo relacional tradicional. Puede manejar estructuras como documentos, pares clave-valor, grafos o columnas. Ejemplos populares incluyen MongoDB, Firebase y Cassandra.

**Dart** Lenguaje de programación desarrollado por Google, utilizado en Flutter para crear aplicaciones multiplataforma.

**Desktop** Aplicaciones diseñadas para sistemas operativos de escritorio como Windows o macOS.

**Diagrama de Gantt** Herramienta gráfica utilizada en la gestión de proyectos para representar el cronograma de tareas, mostrando su duración, secuencia y posibles solapamientos.

**Especificación de Requisitos de Software (SRS)** Documento que detalla las funcionalidades, características y limitaciones técnicas que debe cumplir una aplicación.

**Flutter** Framework de Google para crear apps multiplataforma con un solo código.

**Framework** Conjunto de herramientas y reglas para facilitar el desarrollo de software.

**Freemium** Modelo de negocio que ofrece funcionalidades básicas de forma gratuita, con la posibilidad de desbloquear características avanzadas mediante pago.

**Frontend** Parte de una aplicación o sitio web con la que interactúan directamente los usuarios. Incluye el diseño, la estructura visual y la experiencia de usuario.

**GitHub** Plataforma para gestionar proyectos de software usando Git, un sistema de control de versiones.

**Hot Reload** Funcionalidad de Flutter que permite a los desarrolladores ver en tiempo real los cambios realizados en el código sin reiniciar la aplicación.

**Interfaz de usuario (UI)** Elemento visual y operativo de una aplicación o sistema a través del cual los usuarios interactúan para realizar tareas.

**iOS** Sistema operativo desarrollado por Apple, utilizado exclusivamente en dispositivos como el iPhone, iPad y iPod Touch.

**Java** Lenguaje de programación orientado a objetos y de propósito general.

**JavaScript** Lenguaje de programación ampliamente utilizado para el desarrollo web, que permite crear interactividad y funcionalidades dinámicas en páginas web.

**Kotlin** Lenguaje de programación moderno desarrollado por JetBrains, oficialmente soportado para el desarrollo de aplicaciones Android.

**Multiplataforma** Capacidad de una aplicación o sistema para ejecutarse en diferentes sistemas operativos, como Android e iOS, ampliando su accesibilidad.

**React Native** Framework multiplataforma creado por Meta (anteriormente Facebook) que utiliza JavaScript para desarrollar aplicaciones móviles.

**Swift** Lenguaje de programación creado por Apple, utilizado para desarrollar aplicaciones nativas en iOS y macOS.

**Widget** Elemento básico de la interfaz de usuario en Flutter. Todo en Flutter es un widget: desde los botones, textos y campos de entrada, hasta el diseño estructural de la aplicación.

**Wireframe** Representación visual esquemática de la estructura y diseño de una aplicación o sitio web, utilizada para planificar su funcionalidad y disposición antes del desarrollo.

---

# Bibliografía

- [1] Libro de mantenimiento de coches. <https://play.google.com/store/apps/details?id=com.cars.android.carapps.carnotes&hl=es>, 2025. Fecha de último acceso: 23/01/2025.
- [2] Drivvo. <https://www.drivvo.com/es-ES>, 2025. Fecha de último acceso: 23/01/2025.
- [3] Mis coches. <https://play.google.com/store/apps/details?id=com.aguirre.android.mycar.activity&hl=es>, 2025. Fecha de último acceso: 23/01/2025.
- [4] My car. <https://play.google.com/store/apps/details?id=com.kineapps.mycar&hl=es>, 2025. Fecha de último acceso: 23/01/2025.
- [5] Plantilla de diagrama de gantt de cacoo.com. <https://cacoo.com/diagrams/GdMIEIFT01Gj128B/8F5CE>, 2025. Fecha de último acceso: 15/03/2025.
- [6] Google. Flutter documentation. <https://docs.flutter.dev/>, 2025. Fecha de último acceso: 2025.
- [7] E. Windmill. *Flutter in Action*. Manning Publications, 2020.
- [8] Colan Infotech. React native vs. flutter vs. swift & kotlin for mobile app development. <https://colaninfotech.com/blog/react-native-vs-flutter-vs-swift-kotlin-for-mobile-app-development>, 2024. Fecha de último acceso: 07/02/2025.
- [9] G. Cohavy. Should i use react native, flutter, or native swift/-kotlin for mobile app? Medium, <https://medium.com/@cohavygal/should-i-use-react-native-flutter-or-native-swift-kotlin-for-mobile-app-f482fdjan-2024>. Fecha de último acceso: 07/02/2025.
- [10] Iso/iec 25010:2011. systems and software engineering – systems and software quality requirements and evaluation (square) – system and software quality models, 2011.
- [11] J. R. Molina Ríos, M. P. Zea Ordóñez, F. F. Redrován Castillo, M. R. Valarezo Pardo, J. A. Honores Tapia, R. F. Morocho Román, J. L. Armijos Carrión,

O. E. Cárdenas Villavicencio, and B. B. Romero Macharé. Metodología para el diseño y desarrollo de aplicaciones móviles. *Área de Innovación y Desarrollo, S.L.*, 2021. doi: 10.17993/IngyTec.2021.77.

# **A. Anexo I: Documento de Especificación de Requisitos de Software (SRS)**

## **Índice**

1. Introducción .....	56
1.1. Propósito .....	56
1.2. Alcance .....	56
1.3. Definiciones, acrónimos y abreviaturas .....	56
1.4. Referencias .....	57
1.5. Visión general .....	57
2. Descripción global .....	57
2.1. Perspectiva del producto .....	57
2.2. Funciones del producto .....	57
2.3. Características de los usuarios .....	58
2.4. Restricciones .....	58
2.5. Suposiciones y dependencias .....	59
2.6. Reparto de requisitos .....	59
3. Requisitos específicos .....	59
3.1. Requisitos de interfaces externas .....	59
3.1.1. Interfaces de usuario .....	59
3.1.2. Interfaces de hardware .....	60
3.1.3. Interfaces de software .....	60
3.2. Requisitos funcionales .....	60
3.2.1. Gestión de usuarios .....	60
3.2.2. Gestión de familias .....	61
3.2.3. Gestión de vehículos .....	61
3.2.4. Gestión de actividad .....	61
3.2.5. Estadísticas .....	61
3.2.6. Gestión de ajustes .....	62
3.2.7. Exportación de datos .....	62
3.3. Requisitos de rendimiento .....	62
3.4. Restricciones de diseño .....	62

3.5. Atributos del sistema software .....	62
3.5.1. Compatibilidad .....	62
3.5.2. Seguridad .....	62
3.5.3. Accesibilidad .....	63
3.5.4. Mantenibilidad .....	63
3.5.5. Usabilidad .....	63
3.6. Otros requisitos .....	63

## 1. Introducción

Este documento constituye la **Especificación de Requisitos de Software (SRS)** para el sistema de gestión de mantenimiento de vehículos móviles. Su estructura sigue las directrices del estándar **IEEE 830-1998**.

### 1.1. Propósito

El propósito de este documento es definir de forma detallada los requisitos funcionales y no funcionales necesarios para el desarrollo de una aplicación móvil multiplataforma, destinada a facilitar el control del mantenimiento de vehículos. Esta herramienta está dirigida a usuarios particulares, familias y pequeñas organizaciones que desean registrar actividades como repostajes, revisiones, y almacenar documentación asociada.

El presente documento servirá como referencia técnica para el equipo de desarrollo, diseñadores de experiencia de usuario (UX/UI), evaluadores de calidad y otras partes interesadas.

### 1.2. Alcance

Esta especificación está dirigida a todos los actores involucrados en el desarrollo y validación de la aplicación móvil. La solución propuesta permitirá gestionar de forma centralizada y digital diversos aspectos relacionados con los vehículos.

### 1.3. Definiciones, acrónimos y abreviaturas

- **SRS:** Software Requirements Specification
- **UX/UI:** Experiencia de Usuario / Interfaz de Usuario
- **App:** Aplicación móvil

## 1.4. Referencias

- **IEEE Std 830-1998:** IEEE Recommended Practice for Software Requirements Specifications.
- **ISO/IEC 25010:2011:** Quality models for system and software products.

## 1.5. Visión general

Este documento se organiza en tres secciones. En la primera sección proporciona una introducción general a la aplicación, el contexto del proyecto y las bases normativas empleadas. En la segunda sección describe el sistema de forma global, incluyendo su estructura modular, funcionalidades previstas, usuarios objetivo, restricciones y dependencias.

Por último, la tercera sección recoge los requisitos específicos que debe cumplir la aplicación, detallando tanto las funcionalidades como los atributos de calidad y requisitos técnicos complementarios.

# 2. Descripción global

## 2.1. Perspectiva del producto

La aplicación de gestión de mantenimiento de vehículos está diseñada como una solución móvil multiplataforma que permite a los usuarios controlar todas las actividades relacionadas con el mantenimiento de uno o varios vehículos. Funciona de forma independiente, pero con conectividad total a servicios en la nube para garantizar disponibilidad, respaldo y sincronización.

El sistema permitirá realizar funciones como: registro de repostajes, mantenimiento y documentos, estadísticas de los registros, registro de vehículos, gestión multiusuario. Estas funcionalidades estarán disponibles para cualquier usuario registrado en la aplicación, independientemente del dispositivo (Android o iOS), siempre que cuente con conexión activa a internet.

## 2.2. Funciones del producto

El sistema proporciona un conjunto completo de funcionalidades orientadas a la gestión eficiente del mantenimiento de vehículos, la personalización de la experiencia del usuario y la administración colaborativa de datos. Las funcionalidades principales incluyen:

- **Gestión del perfil de usuario:** Registro y eliminación de cuentas, inicio y cierre de sesión, actualización de datos personales y vinculación con otros perfiles si es necesario.
-

- **Gestión de vehículos y actividades:** Alta, modificación y eliminación de vehículos y actividades; visualización del historial de actividades asociadas (repostajes, mantenimientos, documentos, etc.).
- **Personalización de tipos de datos:** Administración de tipos predefinidos y personalizados para subtipos de actividad, categorías de vehículo y nuevas actividades adaptadas a las necesidades del usuario.
- **Gestión de familias:** Creación y administración de grupos familiares o colaborativos. Permite unirse mediante código o salir de una familia.
- **Estadísticas detalladas:** Visualización de informes analíticos sobre las actividades de mantenimiento y uso de vehículos, con datos históricos segmentados por tipo, fecha y coste.
- **Exportación de datos:** Generación de archivos en formato CSV para respaldo o análisis externo de las actividades registradas.
- **Personalización de la aplicación:** Configuración de temas visuales (modo claro/oscuro), idioma de la interfaz y preferencias generales de uso.
- **Retroalimentación al usuario:** Notificaciones instantáneas a través de componentes como “toasts” que informan sobre acciones exitosas, errores o advertencias en tiempo real.

### 2.3. Características de los usuarios

Usuarios sin conocimientos técnicos, que buscan una herramienta fácil y visual para el control del mantenimiento de sus vehículos. El sistema debe ser intuitivo y accesible.

### 2.4. Restricciones

- **Compatibilidad:** El sistema estará limitado inicialmente a las plataformas Android e iOS, dejando una futura expansión a versiones web o de escritorio como opción.
- **Conexión a internet:** El correcto funcionamiento de la aplicación dependerá de una conexión a internet activa. Sin conexión, el usuario no podrá acceder a los datos almacenados ni utilizar las funcionalidades de la aplicación.
- **Almacenamiento en la nube:** Todos los datos del usuario se almacenarán exclusivamente en la nube, lo que garantiza su seguridad y disponibilidad desde cualquier dispositivo compatible. No se admitirá almacenamiento local, lo que implica que no se podrá acceder a la aplicación en modo offline.

## 2.5. Suposiciones y dependencias

- Se asume que los usuarios tendrán acceso a un dispositivo Android o iOS actualizado.
- La aplicación dependerá de servicios en la nube para garantizar el respaldo de los datos.
- La correcta funcionalidad de la aplicación estará sujeta al cumplimiento de las políticas de las tiendas de aplicaciones (Google Play Store y App Store).
- Se espera que los usuarios ingresen datos de manera manual en ciertas secciones, como repostajes, mantenimientos y facturas.

## 2.6. Reparto de requisitos

Las siguientes funcionalidades están previstas para versiones futuras con el fin de enriquecer la experiencia del usuario y ampliar las capacidades del sistema:

- **Notificaciones push:** Recordatorios sobre tareas de mantenimiento o vencimientos de documentos.
- **Ampliación del sistema de familias:** Incorporación de chat interno y registro de acciones realizadas por distintos usuarios.
- **Copia de seguridad y restauración en la nube:** Sistema configurable para facilitar la migración entre dispositivos y recuperación de datos.
- **Mantenimientos predictivos basados en IA:** Predicción inteligente de necesidades de servicio (ej. cambio de aceite, frenos) a partir del historial y uso del vehículo.
- **Asistente de configuración inicial:** Guía interactiva paso a paso para ayudar a los nuevos usuarios en la puesta en marcha de la aplicación.
- **Internacionalización completa:** Soporte para más idiomas, unidades de medida, formatos de fecha y moneda, orientado a una adopción global.

# 3. Requisitos específicos

## 3.1. Requisitos de interfaces externas

### 3.1.1. Interfaces de usuario

3.1.1.1 Pantalla de inicio de sesión y registro (email y Google).

---

3.1.1.2 Interfaz principal con navegación por secciones: vehículos, actividades, estadísticas, ajustes.

3.1.1.3 Formularios para registrar vehículos, actividades, subtipos, tipos personalizados.

3.1.1.4 Pantallas de administración de perfil y familias.

3.1.1.5 Vista gráfica de estadísticas (barras y circulares).

3.1.1.6 Configuración de idioma y tema visual.

### **3.1.2. Interfaces de hardware**

3.1.2.1 Acceso al almacenamiento para exportación de datos.

3.1.2.2 Acceso a cámara o almacenamiento para importar imágenes.

### **3.1.3. Interfaces de software**

3.1.3.1 Servicios de Firebase (Authentication, Firestore) para la gestión de usuarios, almacenamiento de datos, archivos y lógica del backend.

## **3.2. Requisitos funcionales**

### **3.2.1. Gestión de usuarios**

3.2.1.1 Como usuario anónimo, quiero **iniciar sesión como invitado** para usar la aplicación sin necesidad de registro.

3.2.1.2 Como usuario anónimo, quiero **vincular mi cuenta a un correo electrónico** para conservar mis datos como cuenta registrada.

3.2.1.3 Como usuario, quiero **registrarme e iniciar sesión con email y contraseña** para acceder de forma segura.

3.2.1.4 Como usuario, quiero **registrarme e iniciar sesión con mi cuenta de Google** para evitar crear nuevas credenciales.

3.2.1.5 Como usuario, quiero **vincular mi cuenta a mi cuenta de Google** para unificar accesos.

3.2.1.6 Como usuario, quiero **actualizar mi perfil** (nombre e imagen) para mantener mis datos actualizados.

3.2.1.7 Como usuario, quiero **cerrar sesión** para proteger mis datos en dispositivos compartidos.

3.2.1.8 Como usuario, quiero **eliminar mi cuenta** para asegurar el borrado de todos mis datos.

---

### 3.2.2. Gestión de familias

3.2.2.1 Como usuario, quiero **crear una familia** para facilitar que otros usuarios puedan gestionar mis vehículos.

3.2.2.2 Como usuario, quiero **unirme a una familia mediante un código** para gestionar vehículos compartidos con otras personas.

3.2.2.3 Como usuario, quiero **abandonar una familia** cuando ya no desee compartir el acceso.

### 3.2.3. Gestión de vehículos

3.2.3.1 Como usuario, quiero **añadir un nuevo vehículo** introduciendo información relevante (marca, modelo, nombre, etc.) para tenerlo registrado en el sistema.

3.2.3.2 Como usuario, quiero **editar los detalles de un vehículo** ya registrado para actualizar información en caso de cambios o correcciones.

3.2.3.3 Como usuario, quiero **eliminar un vehículo** de mi lista para descartar registros que ya no son necesarios.

3.2.3.4 Como usuario, quiero **ver todos mis vehículos** registrados para tener una visión general de mi flota o historial de vehículos.

### 3.2.4. Gestión de actividades

3.2.4.1 Como usuario, quiero **añadir una nueva actividad a un vehículo** específico, incluyendo subtipo, fecha y coste, para registrar el mantenimiento o uso del vehículo.

3.2.4.2 Como usuario, quiero **modificar los datos de una actividad** registrada para corregir errores o actualizar la información.

3.2.4.3 Como usuario, quiero **eliminar una actividad de un vehículo** si fue ingresada por error o ya no es relevante.

3.2.4.4 Como usuario, quiero **ver la lista de todas las actividades realizadas** por un vehículo para hacer un seguimiento histórico de su uso o mantenimiento.

### 3.2.5. Estadísticas

3.2.5.1 Como usuario, quiero **consultar el historial de actividades de un vehículo** para revisar todos los servicios, repostajes y documentos asociados.

3.2.5.2 Como usuario, quiero **visualizar un resumen** con el gasto total, número de actividades, gasto medio por actividad y gasto medio mensual.

3.2.5.3 Como usuario, quiero **visualizar un gráfico de barras que muestre el gasto mensual acumulado** durante un año seleccionado, para identificar tendencias en mantenimiento o consumo.

---

3.2.5.4 Como usuario, quiero **consultar un gráfico circular que muestre la proporción del número de actividades por tipo**, para entender qué tipo de servicios o gestiones son los más frecuentes.

### **3.2.6. Gestión de ajustes**

3.2.6.1 Como usuario, quiero **personalizar los subtipos de actividad** para adaptarlos a mis necesidades.

3.2.6.2 Como usuario, quiero **personalizar los tipos de vehículos** para incluir categorías específicas.

3.2.6.3 Como usuario, quiero **personalizar los tipos de actividad** para reflejar mejor los servicios que realizo.

### **3.2.7. Exportación de datos**

3.2.7.1 Como usuario, quiero **exportar los datos de mis vehículos en formato CSV** para análisis o respaldo externo.

## **3.3. Requisitos de rendimiento**

3.3.1 Tiempo de carga inferior a 2 segundos en operaciones estándar.

3.3.2 Capacidad para mostrar estadísticas con hasta 500 actividades sin pérdida de rendimiento.

## **3.4. Restricciones de diseño**

3.4.1 Uso de arquitectura cliente-servidor.

3.4.2 Diseño multiplataforma mediante Flutter o React Native.

3.4.3 Interfaz adaptativa para diferentes tamaños de pantalla.

3.4.4 Uso exclusivo de almacenamiento en la nube.

## **3.5. Atributos del sistema software**

### **3.5.1. Compatibilidad**

3.5.1.1 Compatible con Android (versión X o superior) e iOS (versión Y o superior).

### **3.5.2. Seguridad**

3.5.2.1 Cifrado de datos en tránsito y en reposo.

3.5.2.2 Autenticación mediante correo y contraseña segura.

---

**3.5.3. Accesibilidad**

- 3.5.3.1 Interfaz disponible en español e inglés.
- 3.5.3.2 Modo oscuro y claro configurable por el usuario.

**3.5.4. Mantenibilidad**

- 3.5.4.1 Arquitectura modular para futuras expansiones.

**3.5.5. Usabilidad**

- 3.5.5.1 Cumplimiento con ISO/IEC 25010:2011.
- 3.5.5.2 Interfaz intuitiva y accesible para usuarios sin conocimientos técnicos.

**3.6. Otros requisitos**

- 3.6.1 Notificaciones internas mediante “toasts” para informar al usuario sobre acciones y errores.
- 3.6.2 Posibilidad de extender el sistema a versiones web en el futuro.
- 3.6.3 Control de sesión segura y gestión eficiente de cuentas compartidas (familias).

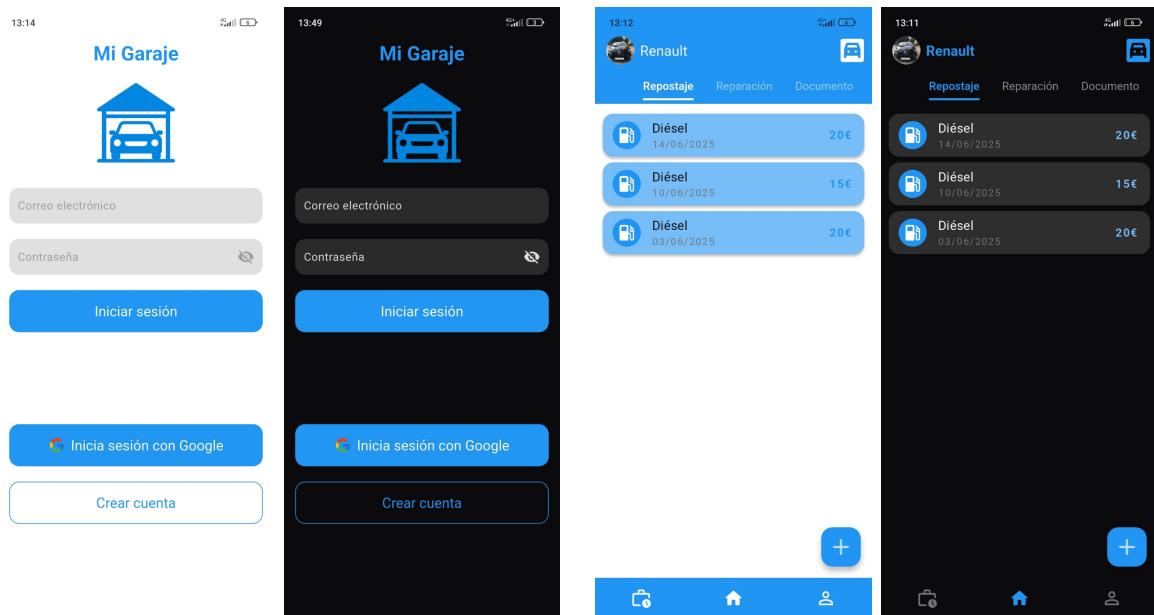


## B. Anexo II: Interfaz de usuario final de la aplicación

Este anexo presenta una recopilación de capturas de pantalla que muestran la interfaz gráfica final de la aplicación desarrollada. Las imágenes permiten observar las diferentes pantallas, menús y funcionalidades implementadas, y reflejan tanto el diseño como la experiencia de usuario que se ofrece.

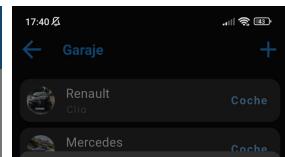
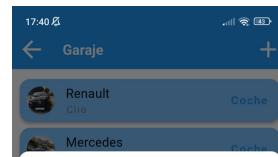
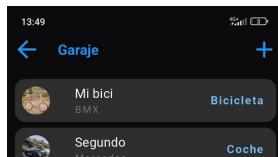
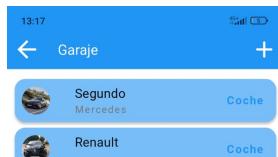
Para cada pantalla representativa, se incluyen tanto su versión en **modo claro** como en **modo oscuro**, a fin de mostrar la adaptabilidad visual de la aplicación a las preferencias del usuario.

Las capturas están organizadas en las páginas siguientes, distribuidas en filas de cuatro imágenes para facilitar su lectura y análisis visual.



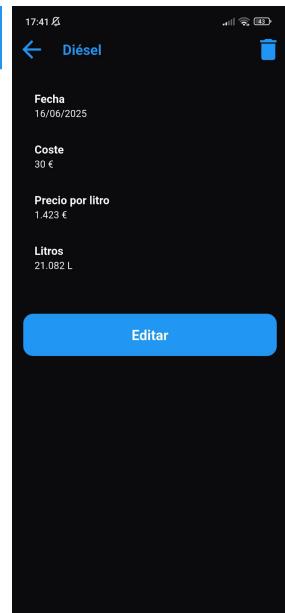
(a) Pantalla de inicio de sesión.

(b) Pantalla de inicio.



(c) Pantalla de garaje.

(d) Pantalla para añadir vehículo.



(e) Pantalla para añadir actividad.

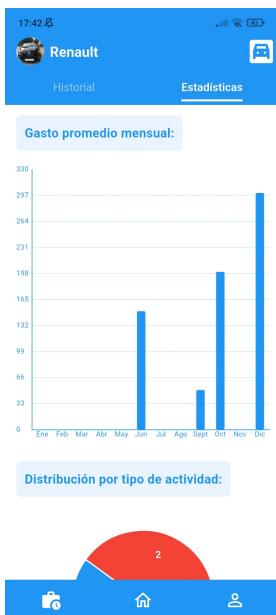
(f) Pantalla de una actividad.



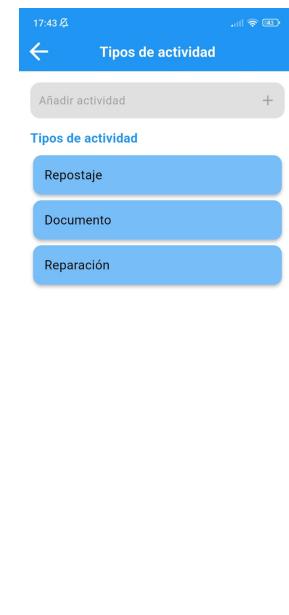
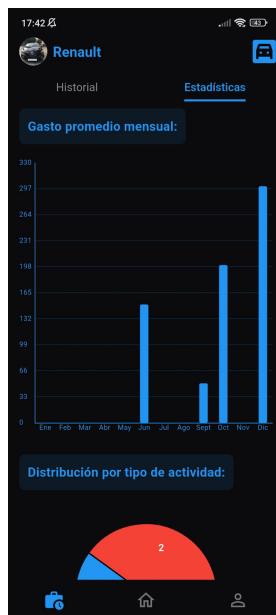
(g) Pantalla de historial.



(h) Pantalla de estadísticas 1.



(i) Pantalla de estadísticas 2.



(j) Pantalla de la sección de tipos.