Trabajo Fin de Grado

Fuel Price

Jesús Rodríguez Layos
Grado de Ingeniería Informática
Desarrollo de Aplicaciones para dispositivos móviles (Android)
06/2024

David Escuer Latorre
Jordi Almirall López

FICHA DEL TRABAJO FINAL

Título del trabajo:	Fuel Price
Nombre del autor:	Jesús Rodríguez Layos
Nombre del consultor/a:	David Escuder Latorre Jordi Almirall López
Nombre del PRA:	Carles Garrigues Olivella
Fecha de entrega (mm/aaaa):	07/2024
Titulación::	Grado Ingeniería Informática
Área del Trabajo Final:	Desarrollo de aplicaciones dispositivos móviles (Android)
Idioma del trabajo:	Castellano
Palabras clave	Kotlin, Hilt y Jetpack Compose

Resumen del Trabajo (máximo 250 palabras): *Con la finalidad, contexto de aplicación, metodología, resultados y conclusiones del trabajo.*

El propósito de este Trabajo de Fin de Grado consiste en exponer de manera integral el flujo de trabajo desde la toma de requisitos hasta la publicación de la aplicación. El proyecto pretende ayudar a aquellos usuarios que tengan la necesidad de conocer el precio del combustible en aquellos lugares que más transitan.

El proyecto comienza con la definición del DCU (Diseño Centrado en el Usuario), mediante la realización de un cuestionario a varias personas para determinar las características claves del mismo; y un análisis de *benchmarking* para destacar los elementos positivos y negativos de la competencia.

Posteriormente, se realiza el diseño conceptual a partir de los cuestionarios realizados, continuando con la conceptualización de estos en el prototipo final, el mismo que será utilizado para la codificación de las interfaces de usuario.

Luego, se elabora el diseño técnico, el encargado de definir los diagramas que serán utilizados por el programa, así como la arquitectura. También, se identifican los casos de uso que tendrá la aplicación.

Como solución de *software*, se presenta una aplicación para dispositivos *Android* donde se muestra un listado de estaciones de servicio en las proximidades, con la posibilidad de visualizarlas en un mapa. Asimismo, cuenta con un buscador permite realizar búsquedas por municipio, mostrando las estaciones relativas a dicha localidad.

En conclusión, esta memoria detalla el recorrido realizado para desarrollar una aplicación de *Android*, desde la realización de técnicas de Diseño Centrado en el Usuario, pasando por el diseño de las interfaces, hasta la codificación del *software*.

Abstract (in English, 250 words or less):

The purpose of this Bachelor's Thesis is to comprehensively expose the workflow from the requirements gathering to the publication of the application. The project aims to help those users who need to know the price of fuel in the places where they travel the most.

The project begins with the definition of the UCD (User-Centered Design), by carrying out a questionnaire to several people to determine the key characteristics of the same; and a benchmarking analysis to highlight the positive and negative elements of the competition.

Subsequently, the conceptual design is made from the questionnaires carried out, continuing with the conceptualisation of these in the final prototype, which will be used for the coding of the user interfaces.

Then, the technical design is elaborated, in charge of defining the diagrams that will be used by the programme, as well as the architecture. Also, the use cases that the application will have are identified.

As a software solution, an application for Android devices is presented, which shows a list of service stations in the vicinity, with the possibility of viewing them on a map. It also has a search engine that allows searches to be carried out by city, showing the stations related to that locality.

In conclusion, this report details the route taken to develop an Android application, from the implementation of User-Centered Design techniques, through the design of the interfaces, to the coding of the software.

Índice

Listado de figuras

1. Introducción

1.1. Contexto y justificación

En el contexto actual, la sociedad española ha experimentado una evidente dependencia a los dispositivos móviles inteligentes, los cuales se han convertido en objetos esenciales en la vida cotidiana, creciendo día tras día y sin signos de cambiar [1] [2].

Esta necesidad ha ido aumentando en aquellas tareas que carecían de carácter digital, como son: realización de compras a través de portales digitales, comunicación a través de mensajería instantánea y videollamadas, educación en línea, gestiones con entidades financieras, e incluso, comparar precios de varios productos a tan solo unos *clicks*.

Esta transición hacia una era cada vez más digital ha sido acompañada por la constante fluctuación de los precios, bien sea por la inflación repercutida anualmente en los precios de los distintos productos, o bien por conflictos nacionales e internacionales que han provocado un encarecimiento en el valor de los artículos.

Destaca especialmente la variación frecuente en el precio del combustible, donde se ha visto constantemente alterado por factores externos como tensiones geopolíticas. Unido a esto, está la fluctuación diaria condicionada por la cotización en bolsa del petróleo crudo.

Los continuos cambios, junto con la creciente influencia de la transformación digital, han incitado a los consumidores a recurrir a comparadores de precios de combustible con el objetivo de encontrar la estación de servicio que ofrezca el carburante con la mejor relación calidad-precio.

Por las razones descritas anteriormente, el desarrollo de una aplicación móvil para dispositivos móviles Android ofrece una herramienta más a los usuarios para abordar el problema derivado por la inestabilidad de los precios de los carburantes.

Esta herramienta ofrece los precios en tiempo real, mostrándolos en un listado agrupados por estación de servicio, además de permitir al usuario la posibilidad de filtrar por sus preferencias como el combustible que utiliza, o incluso también, aquellas marcas en las que más confían.

Del mismo modo, cada estación de servicio se visualiza de forma geolocalizada en un mapa, ayudando al usuario a visualizar de una manera más cómoda y concisa aquella que se sitúe más cerca de su posición actual.

Finalmente, hay gran variedad de aplicaciones cumpliendo con las expectativas que cada usuario espera de este tipo de herramientas. Es aquí donde el proyecto busca irrumpir en el mercado y aportar una solución que ofrezca aspectos diferenciales respecto a la competencia.

Entre las aplicaciones más populares con estas características, se encuentran: Precio de Gasolina al Instante [3], Gasall [4] y Gasolina y Diesel España [5]. Estas herramientas cuentan con similitudes como mostrar un listado todos los precios de cada estación de servicio, un mapa explícito donde visualizarlas y la posibilidad de navegar hasta el lugar.

Sin embargo, existen diferencias significativas que generan una experiencia de usuario más enriquecedora. Algunas de estas características son: añadir estaciones de servicio como favoritos, crear alertas de precio por estación de servicio, mostrar precios a partir de una ruta entre dos puntos y aplicar descuentos de forma automática.

Precio de Gasolina al Instante

La aplicación presenta una interfaz simple y llana mostrando la información que el usuario espera recibir: el precio del combustible.

Del mismo modo, utiliza un mapa geolocalizando cada estación de servicio usando como marcador el precio, empleando colores para identificar si se trata de un precio por encima de la media, en la media, o por debajo.

Finalmente, permite al usuario añadir alertas de precio para aquellas estaciones de servicio que han sido marcadas como favoritas, recibiendo de esta forma una notificación cuando esta condición se cumpla.

Disponible en: https://play.google.com/store/apps/details?id=de.mwwebwork.benzinpreisblitz

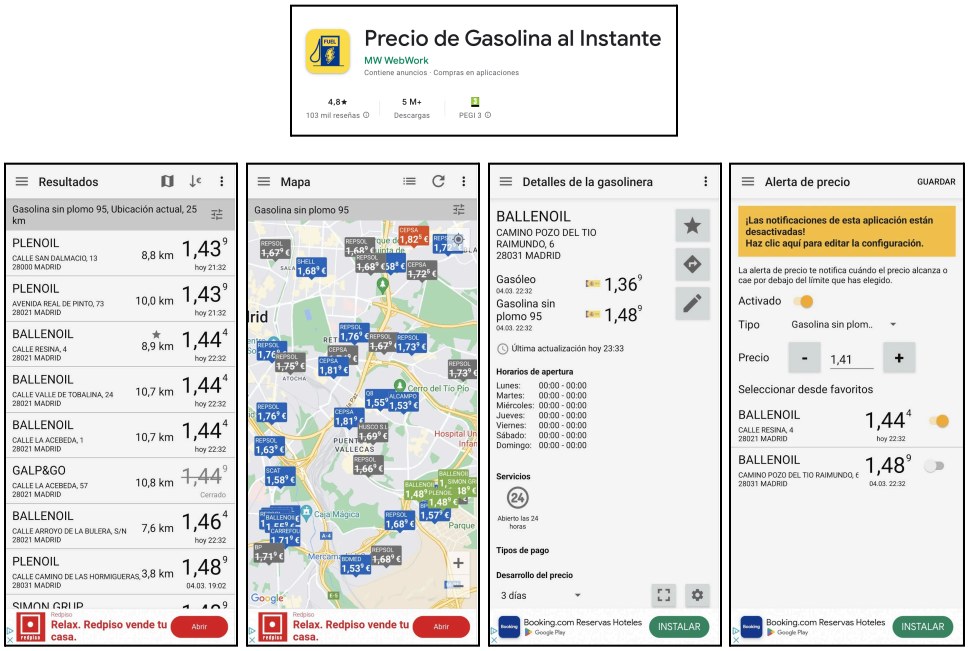

Figura 1: Aplicación Precio de Gasolina al Instante

Gasall

De igual forma que la aplicación anterior, la pantalla principal responde a un listado de precios ordenados ascendentemente, incluso se puede observar el color que se usa para clasificarlos en tres rangos: bajo, medio y alto.

Igualmente, presenta un mapa geolocalizando cada estación de servicio y usando como marcador el logo de la marca. Sin embargo, no se aprecia el precio, obligando al usuario a pulsar individualmente sobre cada marcador o regresando al listado previo.

Adicionalmente, el usuario puede añadir información de su vehículo que más tarde se utilizará para mostrar el coste total del depósito si fuese llenado por completo.

Disponible en:
https://play.google.com/store/apps/details?id=com.gasall

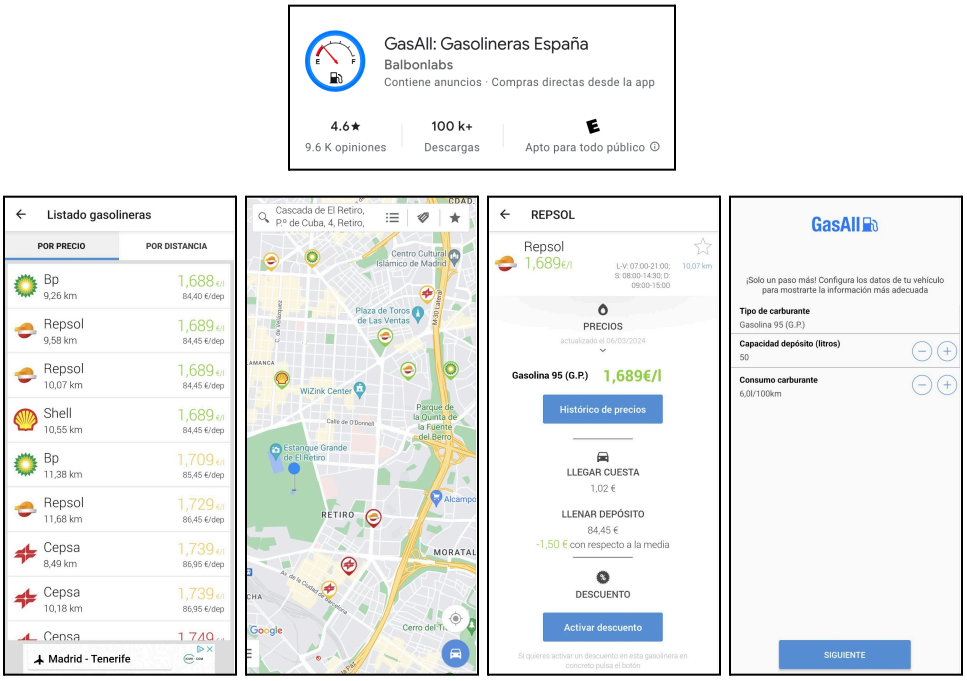

Figura 2: Aplicación Gasall

Gasolina y Diesel España

Siguiendo el patrón de las aplicaciones anteriores, se presenta como pantalla principal el listado de precios, aportando más información como la última vez que fueron actualizados, la dirección de la estación de servicio y la distancia actual desde la localización actual del usuario.

Una vez más, se puede visualizar en un mapa las estaciones de servicio utilizando un marcador con varios rangos de colores: rojo, amarillo y verde. También se puede observar el precio encima de cada marcador.

Una característica particular es la posibilidad de visualizar a partir de una ruta las diferentes estaciones de servicio. Incluso permite añadir descuentos realizados por cada marca que serán aplicados de forma directa durante la representación del precio.

Disponible en: https://play.google.com/store/apps/details?id=com.quadbits.smartrefueling

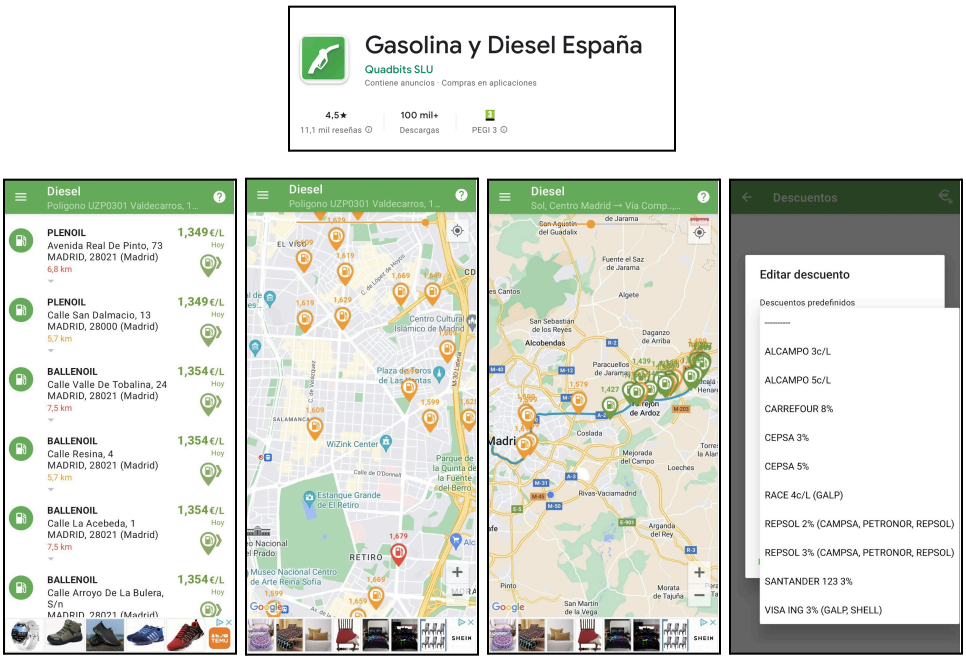

Figura 3: Aplicación Gasolina y Diesel España

1.2. Objetivos

Como estudiante del Grado de Ingeniería Informática, el proyecto persigue como principal objetivo plasmar los conocimientos adquiridos en las asignaturas cursadas. Asimismo, busca ofrecer una solución real y viable para todos aquellos usuarios que estén interesados en encontrar la mejor relación calidad-precio del carburante para sus vehículos.

Estos objetivos se dividen en funcionales y no funcionales. Por un lado, los objetivos funcionales son aquellas funcionalidades que se esperan ver en la aplicación cuando el usuario interactúe con ella. Por otro lado, los objetivos no funcionales definirán la experiencia de usuario, como la usabilidad, el comportamiento de cada funcionalidad y el rendimiento que ésta ofrece.

1.2.1. Objetivos funcionales

Son los descritos a continuación:

- **OF1: Visualizar el precio a partir de un combustible**
  En función de un área delimitada, presentar un listado con los precios ofrecidos de cada estación de servicio por orden descendente.

- **OF2: Visualizar precios en un mapa**
  Dada una zona geolocalizada, mostrar los precios ubicados geográficamente como puntos en un mapa, proporcionando así una visión más clara y concisa de la posición de cada estación de servicio.

- **OF3: Navegar hasta la estación de servicio seleccionada**
  Permitir realizar una ruta de navegación siendo el punto de origen la posición actual, y el destino la ubicación de la estación de servicio seleccionada, utilizando cualquier aplicación de terceros que permita realizar rutas dado dos puntos geolocalizados, como Google Maps, Waze o Sygic, entre otras.

- **OF4: Añadir información del vehículo**
  Agregar datos relevantes que son utilizados para otras acciones de la aplicación. Esta información será: el consumo medio que realiza el vehículo, el tipo de carburante que utiliza, y por último, la capacidad total que poseé el depósito.

- **OF5: Indicar el coste del trayecto hacia la estación de servicio elegida**
  Calcular el coste aproximado resultante de la navegación hacia el punto elegido utilizando como referencia la información del vehículo. Si no se disponen de estos datos, utilizar cifras estándar.

- **OF6: Indicar el coste total de llenar el depósito**
  Estimar el precio fruto del llenado completo del depósito con el precio del combustible en cuestión, utilizando los datos del vehículo registrado. En caso contrario, usar cifras estándar.

- **OF7: Administrar descuentos ofrecidos por cada marca de carburante**
  Seleccionar aquellos descuentos que serán aplicados de forma automática a la hora de visualizar los precios. Incluso, añadir descuentos no presentes en el listado, ofreciendo la posibilidad de aplicar dicho descuento de forma porcentual o monetaria.

- **OF8: Añadir estaciones de servicio como favoritas**
  Agregar en la sección de favoritos aquellas estaciones de servicio que se deseen. En consecuencia, dichas estaciones mostrarán el precio en la parte superior del listado de precios para evitar el desplazamiento a lo largo de dicho listado.

- **OF9: Añadir marcas de carburante como favoritas**
  De igual manera que en el punto anterior, mostrar todas las estaciones de servicio que formen parte de la marca del carburante seleccionada como favorita. Dicho listado está ordenado de forma descendente.

- **OF10: Notificar variaciones en los precios**
  Avisar de las disminuciones de los precios de las estaciones marcadas como favoritas. Del mismo modo, cabe la posibilidad de informar de los aumentos.

- **OF11: Mostrar puntuación por estación de servicio y marca de carburante**
  Añadir una calificación individual por estación de servicio, siendo ésta generada por el

promedio de las votaciones que realizan los usuarios. Además, permite valorar la marca del carburante.

1.2.2. Objetivos no funcionales

Acto seguido, se detallan los siguientes:

- **ONF1: Experiencia de usuario**
  Dotar de pantallas intuitivas, claras y concisas. Replicar patrones de comportamiento utilizados en herramientas similares, evitando de esta manera realizar un nuevo aprendizaje por parte del usuario. Informar en cada momento sobre las acciones que se están realizando en segundo plano, ofreciendo información relevante.

- **ONF2: Adaptable a dispositivos con tamaño de letra grande**
  Mostrar cada información de la aplicación ajustándose a las preferencias individuales de cada dispositivo utilizando tamaños relativos. Por lo tanto, cualquier usuario independientemente del tamaño de letra o de pantalla, puede leer todo el contenido sin producir un corte en los textos imposibilitando su correcta lectura.

- **ONF3: Rendimiento**
  Ofrecer un sistema que reacciona a las interacciones del usuario en un tiempo razonable y previniendo el drenaje masivo del uso de la batería, datos de red y almacenamiento de base de datos.

- **ONF4: Escalable y mantenible**
  Permitir ampliar las funcionalidades que ofrece la aplicación sin alterar las existentes y utilizando las bibliotecas presentes siempre que sea posible, adaptándose a las necesidades que se requieran en cada momento.

- **ONF5: Confiable**
  Brindar estabilidad y garantizar la solidez de la herramienta evitando interrupciones en el flujo de las interacciones del usuario generados por errores no controlados.

- **ONF6: Ético**
  Ofrecer prácticas honestas y justas mostrando precios libres de manipulación. Además, no favorecer a estaciones de servicio ni marcas de carburantes en la presentación de sus datos en la aplicación.

1.3. Enfoque y método seguido

Antes comenzar con el desarrollo de la aplicación, hay que prestar especial atención al tipo de enfoque y método de trabajo que marcarán las pautas en cada etapa del ciclo de vida del producto. En la actualidad, los enfoques más utilizados a día de hoy son: ágil y en cascada.

El enfoque ágil se define como una metodología centrada en la entrega continua de valor a través de *sprints*, haciendo así que el desarrollo del proyecto sea adaptable a los cambios que surjan. El método que más estrechamente trabaja con este enfoque es *Scrum*, orientado a

equipos de trabajo que colaboran conjuntamente para realizar y entregar funcionalidades en pequeños espacios de tiempo, siendo lo más habitual una semana [6] [7].

En cambio, está el enfoque tradicional en cascada, donde se definen las etapas que formarán parte del proyecto y éstas serán realizadas formando una cascada, es decir, no comenzará una etapa hasta que no finalice la anterior. Centrado en este enfoque, el ciclo de vida del proyecto se desarrolla principalmente en cinco fases: requisitos, diseño, implementación, verificación y mantenimiento [8].

Por lo tanto, dada la situación actual del proyecto, y conociendo las funcionalidades que ofrece la competencia, cabe pensar que el enfoque y método más acertado para desarrollar el proyecto es el desarrollo en cascada, presentando un producto sólido y funcional una vez acabado el trabajo.

Las etapas que formarán el proyecto son las siguientes:

- **Requisitos**
  Toma de requisitos en base a las necesidades de los diferentes *stakeholders*.

- **Diseño**
  Crear una interfaz de usuario que permita plasmar los requerimientos recopilados en la etapa anterior.

- **Desarrollo**
  En base a los requerimientos establecidos y el diseño elaborado, codificar el *software* para cumplir las especificaciones detalladas.

- **Pruebas**
  Verificar el correcto funcionamiento de la aplicación realizando diferentes tipos de tests.

- **Despliegue**
  Poner en funcionamiento en un entorno de producción accesible a los *stakeholders* identificados en la primera fase.

- **Mantenimiento**
  Solucionar errores a través de nuevas actualizaciones.

1.4. Planificación

Dada la elección del enfoque elegido, la planificación se divide en varias fases consecutivas, bloqueando así el comienzo de una si la anterior no ha finalizado aún. Cada fase está formada por múltiples tareas, las cuales agrupan a su vez un conjunto de subtareas.

En este sentido, todas las tareas serán realizadas por la misma persona, por lo tanto, el proyecto cuenta como recurso humano únicamente un desarrollador. Por consiguiente, la distribución de las horas de trabajo no serán repartidas uniformemente, siendo un rango comprendido entre 25 y 35 horas por semana, por ejemplo:

- De lunes a jueves, hasta 2,5 horas al día.

- Viernes, hasta 5 horas al día.
- Sábado, domingo y festivos, hasta 10 horas al día.

Siguiendo las guías del enfoque en cascada, las fases del proyecto son divididas en las siguientes:

- **Fase 1: Introducción**
  Descripción del contexto y justificación del proyecto, enlazando con los objetivos funcionales y no funcionales que se realizarán. Del mismo modo, elaboración de una planificación que plasma la hoja de ruta a seguir durante la realización de las tareas. Y finalmente, un breve resumen del resto de capítulos que se encuentran en esta memoria.

- **Fase 2: Diseño y arquitectura**
  Se divide en dos subfases:

  - **Fase 2.1 Diseño centrado en el usuario**
    Identificación de los perfiles de los usuarios, así como sus necesidades y objetivos a través de cualquier técnica de indagación. A continuación, realización del diseño conceptual a través de los escenarios de uso. Después, creación del prototipo, que más tarde servirá como base para la elaboración del diseño de alta fidelidad. Finalmente, se evaluará éste último junto con los usuarios.

  - **Fase 2.2 Diseño técnico**
    Definición de los casos de uso que representan las diferentes interacciones que el usuario puede llegar a hacer en la aplicación. Y por último, diseño de la arquitectura como base para la siguiente fase, que implica la definición de las tablas de base de datos, los modelos de datos, la organización del código dentro del proyecto, entre otras.

- **Fase 3: Implementación**
  Codificación de la aplicación utilizando los diseños de alta fidelidad elaborados en la anterior fase, así como los casos de uso definidos.

- **Fase 4: Verificación**
  Realización de pruebas para garantizar que el código cumple con los requisitos y conseguir el número de errores inesperados más bajo posible.

- **Fase 5: Entrega**
  Preparación de ficheros que serán entregados, entre ellos: memoria, manual de usuario, aplicación y vídeo presentando y explicando el proyecto.

A continuación, se muestra el resumen de las fases con las fechas de inicio y fin, y las horas de dedicación que serán utilizadas.

Nombre	Inicio	Fin	Horas
Plan de trabajo	**1 mar 2024**	**7 jun 2024**	**465h**
FASE 1: Introducción	1 mar 2024	9 mar 2024	**40h**
Contexto y justificación	1 mar 2024	2 mar 2024	10h
Objetivos	3 mar 2024	3 mar 2024	10h
Enfoque y método	4 mar 2024	4 mar 2024	5h
Planificación	5 mar 2024	6 mar 2024	10h
Resto capítulos	7 mar 2024	9 mar 2024	25h
FASE 2: Diseño y arquitectura	16 mar 2024	31 mar 2024	**105h**
FASE 2.1 Diseño centrado en el usuario	16 mar 2024	25 mar 2024	**55h**
Usuarios y contexto	16 mar 2024	17 mar 2024	10h
Diseño conceptual	18 mar 2024	18 mar 2024	5h
Prototipado	19 mar 2024	24 mar 2024	35h
Evaluación	25 mar 2024	25 mar 2024	5h
FASE 2.2 Diseño técnico	26 mar 2024	31 mar 2024	**50h**
Definición de los casos de uso	26 mar 2024	27 mar 2024	10h
Diseño de la arquitectura	28 mar 2024	31 mar 2024	40h
FASE 3: Implementación	10 abr 2024	5 may 2024	**150h**
Splash	10 abr 2024	11 abr 2024	5h
Home	12 abr 2024	12 abr 2024	5h
Listado	13 abr 2024	14 abr 2024	20h
Mapa	15 abr 2024	20 abr 2024	25h
Favoritos	21 abr 2024	23 abr 2024	15h
Vehículo	24 abr 2024	26 abr 2024	10h
Notificaciones	27 abr 2024	5 may 2024	70h
App	27 abr 2024	1 may 2024	35h
Back-end	2 may 2024	5 may 2024	35h
FASE 4: Verificación	6 may 2024	19 may 2024	**90h**
Pruebas unitarias	6 may 2024	11 may 2024	25h
Pruebas de integración	12 may 2024	17 may 2024	45h
Correcciones y mejoras	18 may 2024	19 may 2024	20h
FASE 5: Entrega	22 may 2024	7 jun 2024	**80h**
Memoria	22 may 2024	26 may 2024	30h

Manual de usuario	27 may 2024	1 jun 2024	25h
Vídeo	2 jun 2024	7 jun 2024	25h

Figura 4: Diagrama de Gantt - Planificación plan de proyecto

Del mismo modo, se ha elaborado un diagrama de Gantt. Se trata de una técnica muy utilizada y extendida en la gestión de proyectos, donde se refleja de forma gráfica un diagrama de barras que representan la longitud de las tareas. Unido al enfoque en cascada, se puede observar como una tarea comienza justo después de acabar su predecesora, de esta forma se garantiza que se está llevando a cabo de forma correcta [9].

En primer lugar, desde el 1 de marzo hasta el 9 de marzo, la primera fase: Introducción.

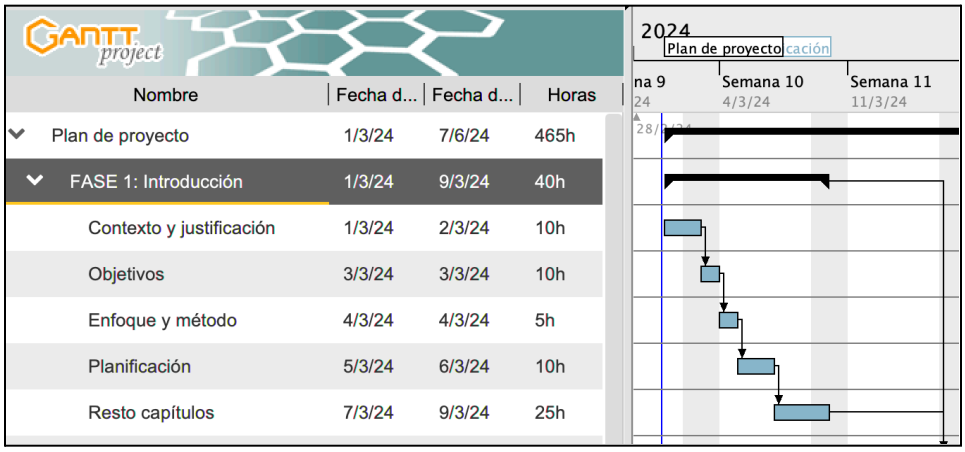

Figura 5: Diagrama de Gantt - Fase 1 Introducción

A continuación, la segunda fase, desde el 16 de marzo hasta el 31 de marzo: Diseño y arquitectura.

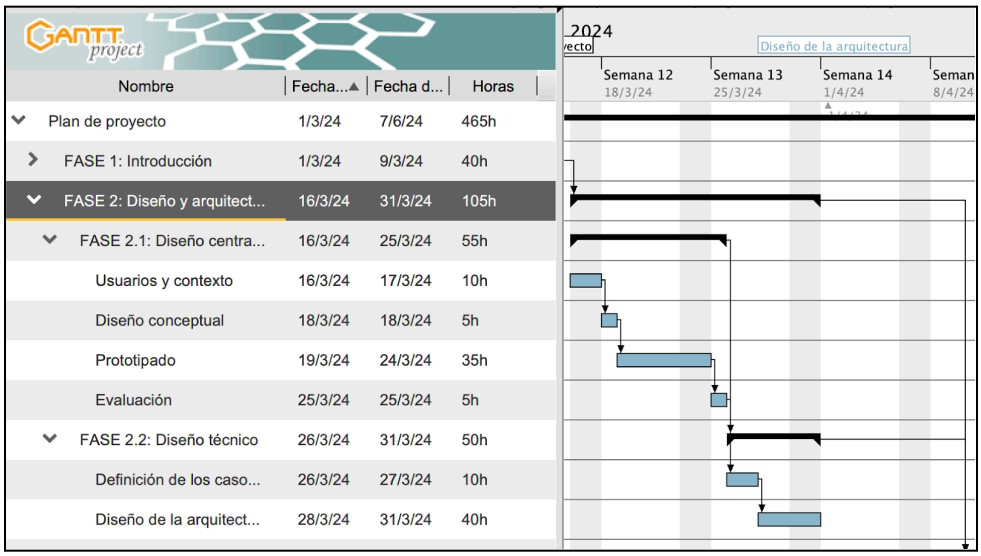

Figura 6: Diagrama de Gantt - Fase 2 Diseño y arquitectura

Luego, la tercera fase, con fechas comprendidas entre el 10 de abril y el 5 de mayo: Implementación.

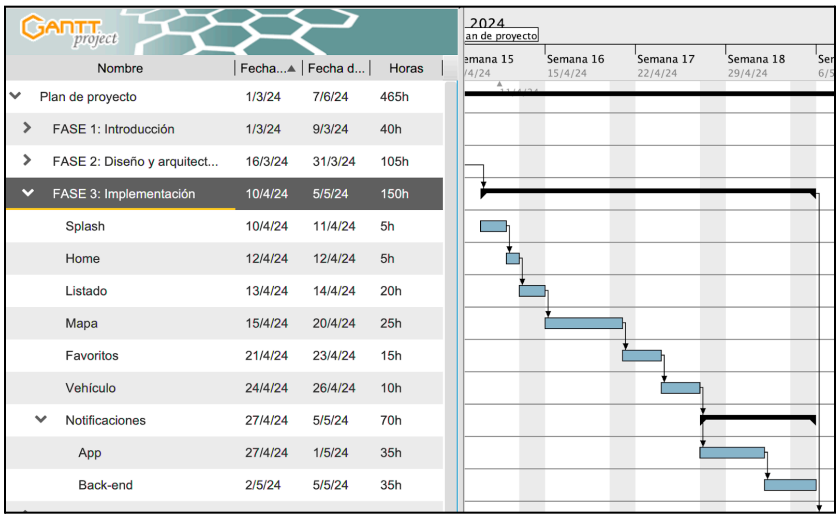

Figura 7: Diagrama de Gantt - Fase 3 Implementación

Más adelante, entre el 6 y el 19 de mayo, la cuarta fase: Verificación.

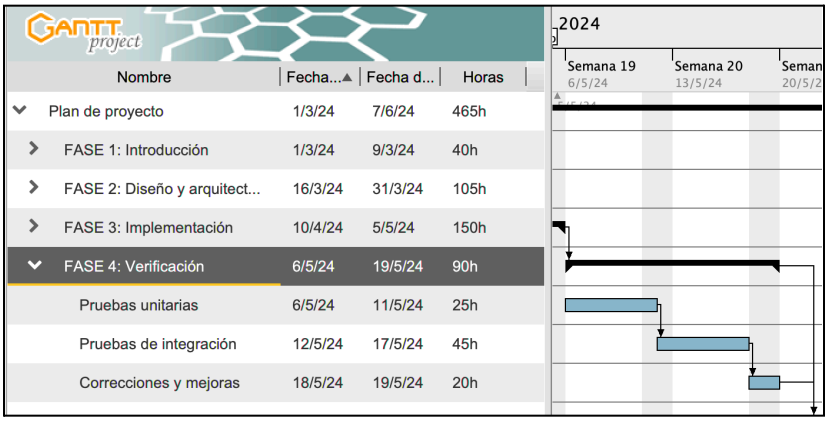

Figura 8: Diagrama de Gantt - Fase 4 Verificación

Y por último, la fase cinco, con fechas entre el 22 de mayo y el 7 de junio: Entrega.

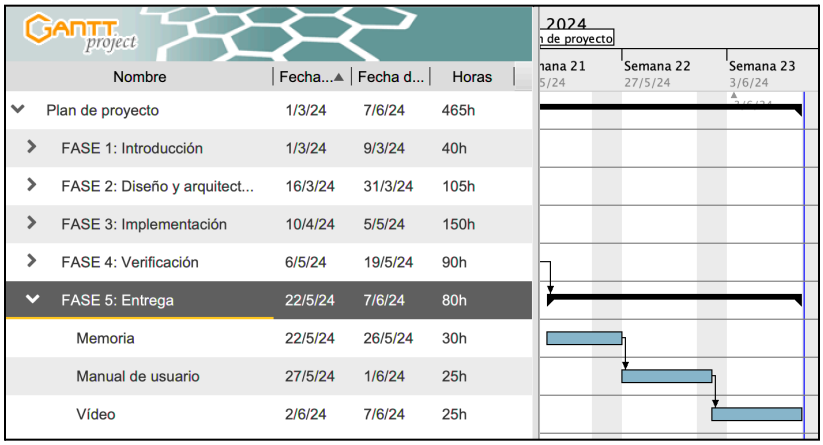

Figura 9: Diagrama de Gantt - Fase 5 Entrega

El diagrama de Gantt al completo se encuentra en el anexo 9.1.

1.5. Breve sumario de productos obtenidos

Una vez finalizado el proyecto, los productos obtenidos fundamentales para el entendimiento y comprensión del proyecto serán los indicados a continuación:

- **Memoria de trabajo**
  Este documento recoge de manera concisa y clara cada uno de los apartados que se han ido trabajando, ofreciendo de esta manera una visión completa del proyecto.

- **Manual de usuario**
  Este fichero en formato PDF será una guía que contiene instrucciones detalladas del funcionamiento de la aplicación, enseñando de este modo a los usuarios el correcto funcionamiento del *software*.

- **Código fuente de la aplicación**
  Contiene todos los ficheros necesarios para compilar el proyecto de *software* y ejecutarlo en cualquier dispositivo con sistema operativo Android.

- **Fichero ejecutable de la aplicación**
  El resultado de la compilación del código fuente dará lugar a un archivo ejecutable, que servirá para instalar y ejecutar la aplicación en cualquier *smartphone* con Android.

- **Presentación en formato vídeo**
  Un vídeo resumiendo los documentos escritos y el código fuente desarrollado, explicando los aspectos más relevantes del proyecto, así como el desarrollo realizado en las diferentes etapas.

1.6. Breve descripción de los otros capítulos

El restos de los capítulos que forman parte de la memoria son los enumerados a continuación:

- **2. Diseño Centrado en el Usuario**
  Este capítulo pone el foco en el diseño centrado al usuario, abordando la identificación de los diferentes perfiles y comprendiendo las necesidades y objetivos que poseen.

- **3. Arquitectura**
  Este apartado identifica los casos de uso y describe los componentes principales de la arquitectura elegida para el proyecto de *software*.

- **4. Implementación**
  Esta sección aborda la etapa de codificación materializando de esta forma el análisis realizado en la anterior fase. Detalla exhaustivamente los aspectos más esenciales como las herramientas y tecnologías utilizadas para el completo desarrollo del proyecto.

- **5. Verificación**
  Este apartado culmina con la realización de pruebas de código en el proyecto de *software* garantizando que la aplicación es libre de errores no controlados y cumpliendo con los objetivos, tanto funcionales como no.

- **6. Conclusiones**
  Este capítulo recoge las conclusiones elaboradas al finalizar el proyecto. Expone los resultados obtenidos y los retos superados, así como las posibles líneas futuras para incrementar el valor de la aplicación.

- **7. Glosario**
  Este apartado recopila todos los términos relacionados con el ámbito del desarrollo de *software* utilizados durante la redacción de esta memoria. Cada término va acompañado por una breve definición para asimilar de forma rápida el concepto.

- **8. Bibliografía**
  Esta sección lista todas las fuentes de información consultadas durante la realización de este documento, tales como páginas web, libros, blogs, entre otros. Las referencias bibliográficas están citadas siguiendo el estilo Harvard.

- **9. Anexos**
  Este capítulo incluye todos los documentos adicionales que permiten ampliar la información de la memoria.

2. Diseño Centrado en el Usuario

El Diseño Centrado en el Usuario (DCU) se trata de un enfoque de diseño que tiene como objetivo recopilar las necesidades de los perfiles de usuario, así como sus comportamientos utilizando la aplicación. De esta forma, al conocer todas estas necesidades, el diseño del *software* puede crear una interfaz que permita satisfacer a todos los usuarios con el mínimo esfuerzo de trabajo [10].

Esta técnica se divide en tres fases:

- **Análisis**
  Esta fase consiste en analizar los diferentes perfiles de usuario que conforman el grupo de usuarios de la aplicación. A raíz de este análisis, se obtiene información esencial para comprender las necesidades que poseen los usuarios y comportamientos que realizan sobre las funcionalidades de la aplicación.

- **Diseño**
  Se divide en dos etapas:

  - **Diseño conceptual**
    Esta etapa define el diseño conceptual inicial de la aplicación, un diseño básico que sirve como punto de partida para la siguiente etapa y donde su cometido principal es mostrar las funcionalidades principales del *software*.

  - **Diseño de alta fidelidad**
    La etapa posterior consiste en trasladar el diseño conceptual a un diseño de alta fidelidad, en él se observará minuciosamente cada componente y pantalla del diseño de la interfaz de usuario.

- **Evaluación**
  Finalmente, la última fase consiste en evaluar el diseño de alta fidelidad con los diferentes perfiles de usuario definidos previamente. De este modo, permite identificar aquellas funcionalidades y características que no se han diseñado tal y como se esperaba que el perfil de usuario pudiese utilizar, y por ende, mejorar a posteriori la experiencia del usuario.

2.1. Análisis

El análisis comienza con la ejecución de tres técnicas de indagación: cuestionario, perfiles de usuario, y finalmente, benchmarking.

Al comenzar, se realizará una encuesta con una serie de preguntas específicas sobre el producto en cuestión. Este cuestionario permitirá recabar información para descubrir más detalles de las funcionalidades y características de la aplicación.

Después, se definen los perfiles de usuario que determinan en gran medida las diferentes personas que interactúan con el producto, detallando sus comportamientos y necesidades.

Por último, la técnica del *benchmarking* permite elaborar de forma detallada una comparación sobre tres aplicaciones de la competencia utilizando como base las características y funcionalidades definidas como criterios previamente.

2.1.1 Cuestionario

El cuestionario permite obtener información detallada sobre el producto que se desea desarrollar. La encuesta se realiza acotando los perfiles de usuario, siendo personas con acceso a un vehículo, ya sea propio o no, y por ende, mayores de 18 años. Además, las

preguntas serán relacionadas con sus comportamientos y la utilización de aplicaciones en dispositivos móviles.

En primer lugar, se realizan dos preguntas demográficas, siendo sexo y edad. A continuación, se pregunta sobre sus comportamientos a través de la frecuencia de uso del vehículo durante la semana y la preferencia de desplazarse para realizar un repostaje. Y por último, dos preguntas más sobre el uso y características de aplicaciones de tipología similar al proyecto en desarrollo.

2.1.1.1. Resultados

El cuestionario ha sido respondido por 18 personas y la información recogida es la siguiente:

- **Sexo**
  El 55,6% de los encuestados son hombres, mientras que el 44,4% restante mujeres, dando lugar a 0% a la opción “Prefiero no decirlo”.

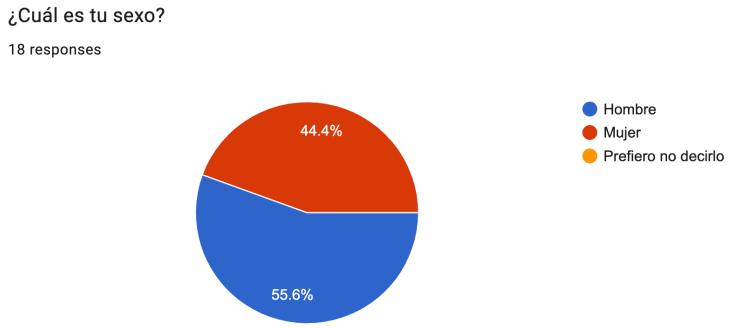

Figura 10: Resultados encuesta sexo

- **Edad**
  La mayor parte de los encuestados tienen un rango de edad entre 36 y 50 años, representando el 55,6%, le sigue el rango entre 26 y 35 años con un 22,2%, luego 66 años en adelante con 16,7%, y finalmente, entre 18 y 25 años con 5,6%.

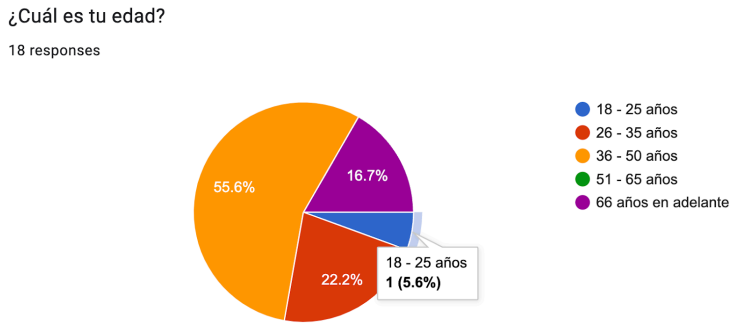

Figura 11: Resultados encuesta edad

- **Frecuencia de uso del vehículo**
  La mayoría de los encuestados utilizan el vehículo de 6 a 7 días a la semana, representando el 38,9%, a continuación de 0 a 1 días el 27,8%, después de 2 a 3 días con el 22,2%, y por último, de 4 a 5 días el 11,1%.

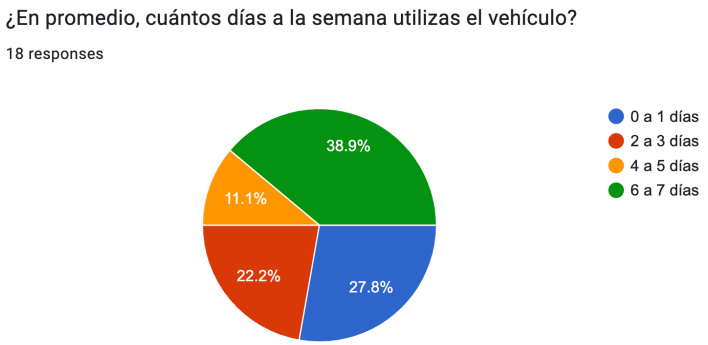

Figura 12: Resultados encuesta frecuencia de uso del vehículo

- **Preferencia a la hora de repostar**

  Los usuarios prefieren repostar en estaciones de servicio aledañas a la ubicación del encuestado, siendo el 55,6%, mientras que al 27,8% no le importa trasladarse hasta la estación más económica. Finalmente, el 16,7% decide en función de la situación del momento.

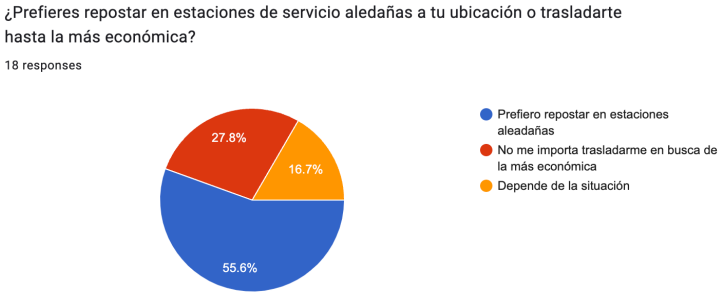

Figura 13: Resultados encuesta preferencia a la hora de repostar

- **Uso de aplicaciones similares**

  La gran mayoría ha respondido “No”, siendo el 55,6%, le sigue “Sí” con el 38,9%, y finaliza “No lo recuerdo” con el 5,6%.

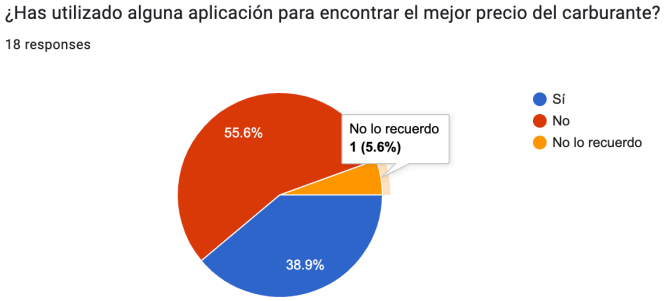

Figura 14: Resultados encuesta uso de aplicaciones similares

- **Características y funcionalidades**

  Las opciones más destacadas son “Listado de precios”, “Mapa” y “Añadir descuentos”, mientras que “Alertas de precio”, “Favoritos” y “Valorar marca y/o estación” son las menos valoradas.

  Nota: los resultados se han ordenado para mejorar la visualización, en ningún caso éste ha sido el orden utilizado en la encuesta.

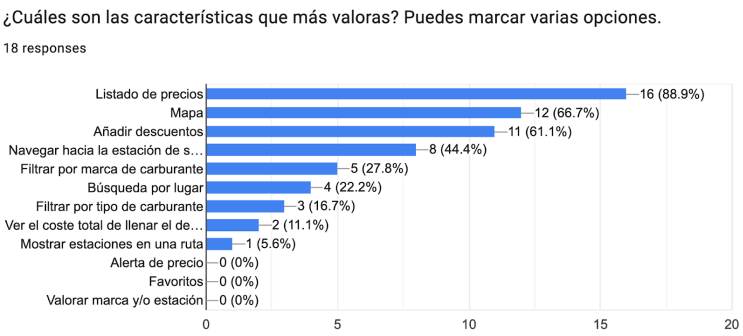

Figura 15: Resultados encuesta valoración características y funcionalidades

2.1.1.2. Conclusiones del cuestionario

Tras la realización de la encuesta, se obtienen unas conclusiones que brindan más detalles sobre las características y funcionalidades que los usuarios esperan utilizar en una aplicación para visualizar precios de combustibles.

Las principales conclusiones se basan en la frecuencia de uso de los vehículos, siendo entre 0 y 1 días, y, 6 y 7 días, las opciones más predominantes. Además, la opción más elegida en cuanto la preferencia de repostar fue la de repostar en estaciones aledañas.

Estos datos dan lugar a pensar que hay dos tipos de perfiles de usuario muy claramente diferenciados:

- Usuarios que utilizan el vehículo con poca frecuencia, probablemente para realizar viajes sobre su zona, como por ejemplo realizar compras, ocio o similar.
- Usuarios que realizan el trayecto del trabajo en coche, y además, los fines de semana lo utilizan para realizar compras, ocio o actividades similares.

Ambos perfiles concluyen en su preferencia por acudir a estaciones aledañas a su ubicación, seguramente ubicadas próximas a sus domicilios o zonas de trabajo.

Después, se observa que gran parte de los encuestados no utilizan aplicaciones para visualizar precios de combustible. Puede estar relacionado con la experiencia propia de los usuarios al repostar en estaciones de servicio aledañas a su ubicación y ser conocedores de los precios, en gran medida, pues conocen cuales son más económicas.

Y finalmente, las características más valoradas han sido las relacionadas con el listado de precio, el mapa y los descuentos. Dejando en último lugar las opciones de alertar de precio, favoritos y valorar marca y/o estación de servicio.

Por lo tanto, se puede concluir que los usuarios tienen como foco principal repostar en estaciones de servicio próximas a su ubicación, de ahí la necesidad de ver estas estaciones en un mapa.

Además, mostrar un listado de precio les permite visualizar de forma compacta las estaciones ubicadas en los alrededores, observando el precio con sus descuentos más usados ya aplicados, pues si son clientes recurrentes es probable que dispongan de ciertas ventajas a la hora de repostar.

2.1.2. *Benchmarking*

Se trata de una metodología de indagación con el objetivo de comparar el producto que se va a desarrollar con aquellos más relevantes en el mercado. De esta manera, se analiza las funcionalidades principales de los competidores, utilizando estos datos en una tabla comparativa para visualizar de una forma clara y concisa los aspectos fuertes y débiles de los productos de la competencia.

2.1.2.1. Definición de los criterios

Para realizar el *benchmarking* de forma correcta, es fundamental comenzar estableciendo los criterios que se utilizarán para el análisis de los productos de la competencia. Estos criterios se usarán para evaluar y analizar las funcionalidades y características que ofrece cada producto.

Requerimientos	Criterios	Sub-criterios
Consultar el precio de la gasolina en un listado	1. Información del precio sobre el listado	a. Ofrecer el precio más barato b. Mostrar el nombre de la estación de servicio c. Indicar el tipo de combustible d. Visualizar el coste total resultante de llenar el depósito e. Observar el gasto que supondría navegar hasta la estación seleccionada f. Mostrar la dirección de la estación de servicio g. Visualizar la distancia de la estación de servicio h. Observar la fecha de última actualización de precio i. Mostrar el icono de navegación
Consultar el precio de la gasolina en un mapa	2. Información del precio sobre el mapa	a. Visualizar los marcadores geolocalizados en un mapa b. Mostrar los marcadores con su precio respectivo c. Indicar el logo de la marca del combustible d. Observar el icono de navegación

Consultar los favoritos	3. Información sobre los elementos favoritos	a. Añadir favoritos b. Eliminar favoritos c. Visualizar los favoritos en el listado de precios d. Observar los favoritos en el mapa
Consultar alertas de precio	4. Información sobre las alertas de precio	a. Crear alertas de precio b. Eliminar alertas de precio
Consultar información del vehículo	5. Información sobre el vehículo	a. Añadir información del vehículo b. Eliminar información del vehículo
Consultar valoraciones de las estaciones de servicio	6. Información sobre la puntuación	a. Ver valoraciones por estación de servicio b. Visualizar valoraciones por marca c. Votar una estación de servicio d. Calificar una marca
Consultar valoraciones de las estaciones de servicio	7. Información sobre los descuentos	e. Añadir descuentos f. Eliminar descuentos

2.1.2.2. Tabla de comparación

Este apartado evaluará de forma individual cada criterio sobre los productos de la competencia. Se utilizan tres valores para indicar el grado de funcionalidad, siendo:

- **Verde**: el criterio se puede utilizar perfectamente [11].
- **Amarillo**: el criterio presenta carencias en la evaluación [12].
- **Rojo**: el criterio no se observa en ninguna funcionalidad [13].

1. Información del precio sobre el listado

En primer lugar, se evaluarán los criterios relacionados con la información del precio visualizados en un listado. Esta pantalla permite al usuario observar de una manera clara y concisa los precios de su alrededor ordenados de forma ascendente.

Criterios de análisis	 Precio de Gasolina al Instante	 Gasall	 Gasolina y Diesel España
1. Información del precio sobre el listado			
a. Ofrecer el precio más barato	✓	✓	✓
b. Mostrar el nombre de la estación de servicio	✓	✓	✓
c. Indicar el tipo de combustible	✓	✗	✓
d. Visualizar el coste total resultante de llenar el depósito	✗	✓	✗
e. Observar el gasto que supondría navegar hasta la estación seleccionada	✗	✗	✗
f. Mostrar la dirección de la estación de servicio	✓	✗	✓
g. Visualizar la distancia de la estación de servicio	✓	✓	✓
h. Observar la fecha de última actualización de precio	✓	✗	✓
i. Mostrar el icono de navegación	✗	✗	✗

A continuación, se muestra gráficamente la pantalla que se ha comparado en cada producto:

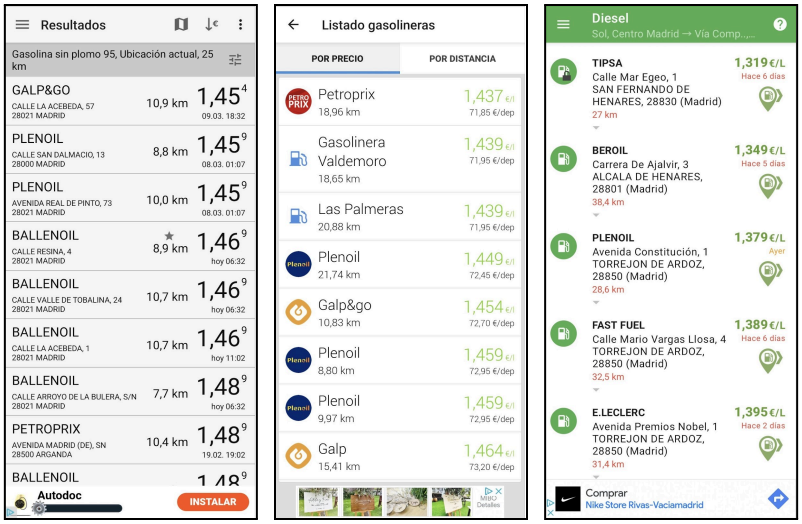

Figura 16: Comparativa listado de precios

2. Información del precio sobre el mapa

En segundo lugar, se analizarán los criterios relacionados con la información del precio vistos en un mapa. Esto permite al usuario visualizar cada precio geolocalizado en un mapa.

Criterios de análisis	Precio de Gasolina al Instante	Gasall	Gasolina y Diesel España
2. Información del precio sobre el mapa			
a. Visualizar los marcadores geolocalizados en un mapa	✓	✓	✓
b. Mostrar los marcadores con su precio respectivo	✓	⚠	✓
c. Indicar el logo de la marca del combustible	⚠	✓	✗
d. Observar el icono de navegación	✗	✗	✓

A continuación, se muestra gráficamente la pantalla que se ha comparado en cada producto:

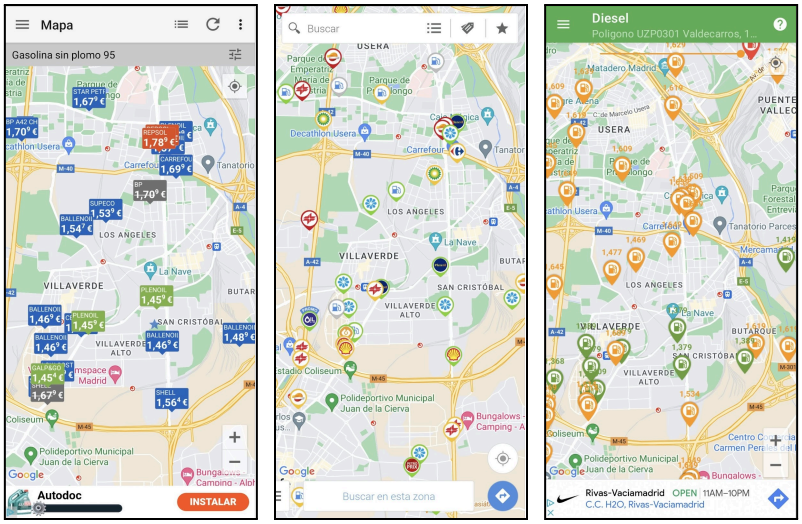

Figura 17: Comparativa mapa

3. Información sobre los elementos favoritos

En tercer lugar, se evalúan los criterios relacionados con la información de los elementos favoritos resultantes de la interacción del usuario. Usar favoritos en la aplicación permite al usuario resaltar aquellos elementos que considera más relevantes por encima del resto.

Criterios de análisis	Precio de Gasolina al Instante	Gasall	Gasolina y Diesel España
3. Información sobre los elementos favoritos			
a. Añadir favoritos	✓	✓	✗
b. Eliminar favoritos	✓	✓	✗
c. Visualizar los favoritos en el listado de precios	✓	✗	✗
d. Observar los favoritos en el mapa	✓	✗	✗

A continuación, se muestra gráficamente la pantalla que se ha comparado en cada producto:

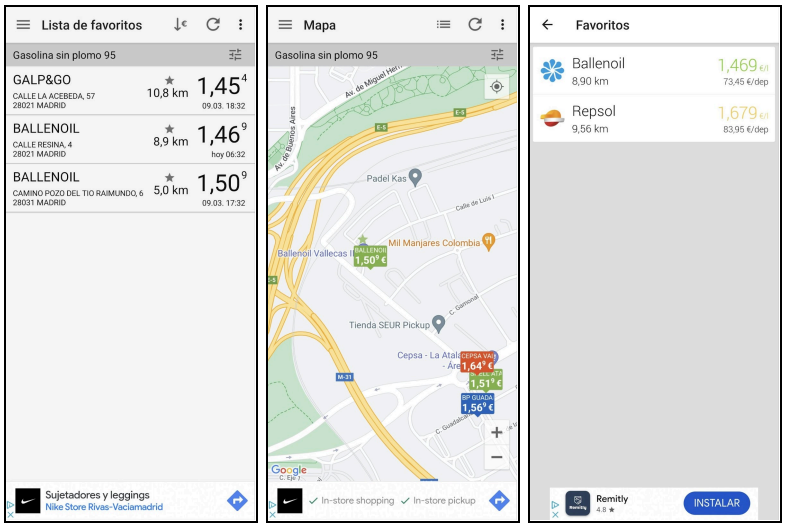

Figura 18: Comparativa favoritos

4. Información sobre las alertas de precio

En cuarto lugar, se analizarán los criterios relacionados con las alertas de precio. Una notificación de esta característica alerta al usuario de una bajada de precio en el combustible seleccionado.

Criterios de análisis	Precio de Gasolina al Instante	Gasall	Gasolina y Diesel España
4. Información sobre las alertas de precio			
a. Se pueden añadir alertas de precio	✓	✗	✗
b. Se pueden eliminar alertas	✓	✗	✗

A continuación, se muestra gráficamente la pantalla que se ha comparado en cada producto:

Figura 19: Comparativa alertas

5. Información sobre el vehículo

En quinto lugar, se analizarán los criterios relacionados con la información del vehículo que el usuario puede proporcionar. Esta información puede emplearse para indicar el coste total del coste por llenar el depósito al completo y mostrar el importe de navegar hacia una estación de servicio a partir del consumo que realiza el vehículo.

Criterios de análisis	Precio de Gasolina al Instante	Gasall	Gasolina y Diesel España
5. Información sobre el vehículo			
a. Añadir información del vehículo	✗	✓	✗
b. Eliminar información del vehículo	✗	✓	✗

A continuación, se muestra gráficamente la pantalla que se ha comparado en cada producto:

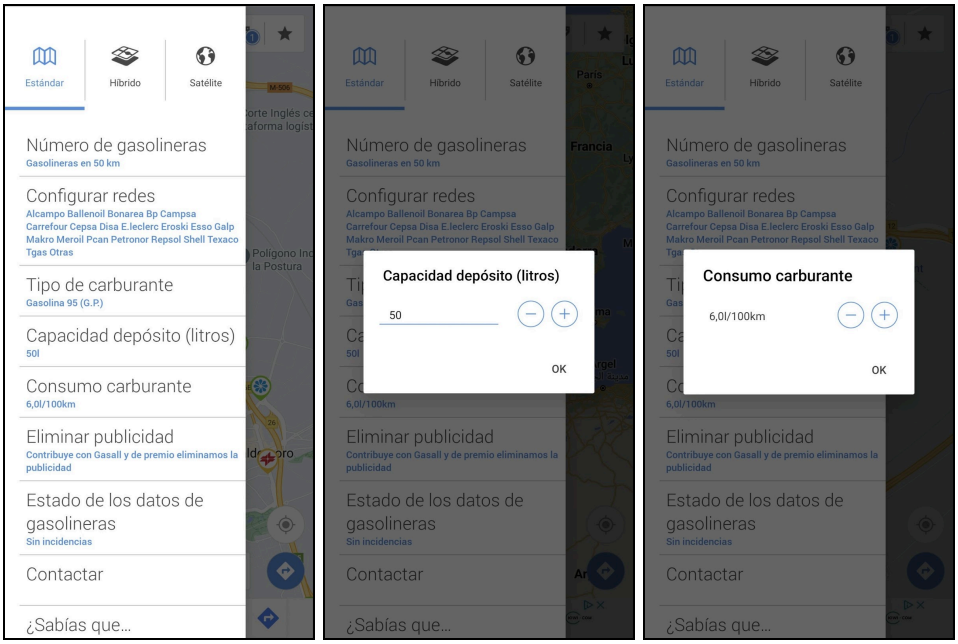

Figura 20: Comparativa agregar vehículo

6. Información sobre la puntuación

En sexto lugar, se analizarán los criterios relacionados con las valoraciones que pueden visualizar y realizar los usuarios en la aplicación. A través de las valoraciones los usuarios podrán saber qué estaciones de servicio ofrecen un mejor servicio y qué marcas rinden mejor.

Criterios de análisis	Precio de Gasolina al Instante	Gasall	Gasolina y Diesel España
6. Información sobre la puntuación			
a. Ver valoraciones por estación de servicio	✗	✗	✗
b. Visualizar valoraciones por marca	✗	✗	✗
c. Votar una estación de servicio	✗	✗	✗

d. Calificar una marca	✗	✗	✗

7. Información sobre los descuentos

Finalmente, en séptimo lugar, se analizarán los criterios relacionados con los descuentos que ofrece cada marca y estación de servicio. De esta forma, los descuentos serán aplicados directamente cuando el usuario visualiza el listado de precios.

Criterios de análisis	Precio de Gasolina al Instante	Gasall	Gasolina y Diesel España
7. Información sobre los descuentos			
a. Añadir descuentos	✗	✓	✓
b. Eliminar descuentos	✗	✓	✓

A continuación, se muestra gráficamente la pantalla que se ha comparado en cada producto:

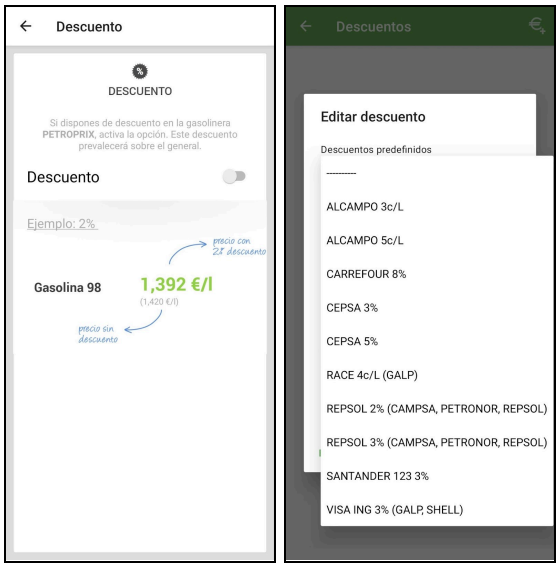

Figura 21: Comparativa añadir descuentos

2.1.2.3. Elementos clave positivos

Después del análisis, se destacan aquellos elementos positivos más relevantes sobre los productos de la competencia. Estos elementos serán claves para imitarlos en la aplicación que va a ser desarrollada con el fin de ofrecer la mejor experiencia de usuario usando las mejores características posibles. Los elementos a destacar son los enumerados a continuación:

1. **Mostrar como pantalla principal el listado de precios ordenados ascendentemente**
   Al abrir la aplicación la primera pantalla es el listado de precios. Esto permite al usuario visualizar aquella estación de servicio que ofrece el mejor precio.

2. **Indicar el tipo de combustible por el cuál se están filtrando los precios**
   Es importante mostrar en todo momento el tipo de combustible que se está utilizando como filtro para mostrar los precios, para que en caso de no desear el usuario ese tipo de combustible, poder cambiarlo fácilmente.

3. **Mostrar la dirección y distancia de la estación de servicio**
   El usuario puede ver útil observar la dirección completa de la estación de servicio, así como la distancia desde su ubicación actual.

4. **Ofrecer la fecha de última actualización**
   Debido a posibles problemas técnicos al actualizar los precios oficiales de cada estación de servicio y marca, es conveniente mostrar al usuario la fecha de la última actualización.

5. **Presentar los marcadores geolocalizados en un mapa con precio y marca**
   Ayuda a visualizar de forma clara y concisa el precio de cada estación de servicio y la marca que trabaja.

6. **Mostrar icono de navegación al pulsar un marcador**
   Ayudaría al usuario a no pulsar numerosas veces hasta encontrar dicha opción.

7. **Ofrecer los favoritos en el listado de precios y mapa**
   Destacar los elementos favoritos por encima del resto en el listado de precios y en el mapa ayuda al usuario a observar más rápido este tipo de elementos.

8. **Poder activar o desactivar alertas previamente creadas.**
   Crear y eliminar alertas facilita al usuario conocer cuando el precio del combustible elegido disminuye. Asimismo, poder activar o desactivar estas alertas ofrece flexibilidad al usuario.

9. **Agregar información del vehículo básica, como consumo y capacidad del depósito**

   No es necesario conocer demasiados detalles del vehículo, solo los prescindibles para mostrar el coste total por llenar el depósito y el importe que supondría navegar hasta la estación de servicio seleccionada.

10. **Añadir descuentos utilizando porcentaje o cantidad monetaria**

Esta opción permite aplicar descuentos en los precios de forma automática. De esta forma el listado de precios sería reordenado en función de los descuentos del usuario.

2.1.2.4. Elementos clave a evitar

Después de enumerar los elementos claves a imitar, es conveniente indicar aquellos que se han observado durante la comparación de los productos y que ayudarían a los usuarios a obtener una mejor experiencia de usuario durante el uso de la aplicación. Los elementos clave a evitar son los siguientes:

1. **No mostrar el icono de navegación en el listado de precios**
   No disponer de un icono para navegar hacia la estación de servicio seleccionada provoca que el usuario deba realizar más pasos hasta encontrar esta opción.

2. **No mostrar icono de navegación en el mapa**
   Al igual que en el punto anterior, los usuarios deben realizar varios pasos hasta lograr navegar a la estación seleccionada.

3. **No mostrar el icono de la marca del carburante**
   Puede generar desconcierto si en el mapa no se diferencian las marcas.

4. **No mostrar precios en el mapa**
   Utilizar una escala de colores no es suficiente para mostrar un marcador en el mapa. El usuario no comprende en base a qué se define esa escala de colores. Por lo tanto, lo más apropiado es mostrar el precio situado cerca del marcador.

5. **No poder añadir favoritos**
   La posibilidad de añadir favoritos en la aplicación permite mostrar los precios afectados resaltados por encima del resto, facilitando la visualización por parte de los usuarios.

6. **No valorar estaciones de servicio y marcas**
   Los usuarios pueden observar aquellas estaciones de servicio y marcas con mejor valoración a partir de las aportaciones del resto de usuarios.

2.1.3. Perfiles de usuario

2.1.3.1. Contexto

La definición de los perfiles de usuario abarca una amplia variedad de personas, por ello se ha atendido a los tres conceptos básicos que conciernen el diseño de un perfil. Estos son: comportamientos, demografía, y, necesidades y objetivos. Cada perfil ha sido minuciosamente seleccionado en base a sus características individuales cubriendo un gran número de perfiles similares.

En primer lugar, se encuentra Natalia, una joven que recientemente ha comenzado su carrera profesional. En su día a día utiliza el vehículo para realizar el trayecto desde su casa hacia el trabajo, y viceversa. Natalia busca el precio más económico cuando se trata de repostar. Suele

utilizar varias aplicaciones para encontrar el más bajo, aunque a veces piensa que sería más económico acudir a una estación de servicio menor ahorro en el precio del combustible por litro pero más cercana a su ubicación, debido al ahorro que obtendría al realizar un viaje de menor distancia.

En segundo lugar, se presenta Lorena, una mujer de mediana edad con una hija y trabajo estable. Utiliza el coche esporádicamente, ya que se mueve en transporte público para sus desplazamientos diarios. Le gusta repostar en aquellas marcas que más confianza le sugiere. Además, al contratar otro tipo de servicios bajo la misma marca ha obtenido un descuento que aprovecha siempre que tiene oportunidad. A veces encuentra grandes diferencias de precio entre estaciones de servicio bajo la misma marca.

En tercer lugar, está Diego, un hombre de mediana edad con dos hijos y trabajo estable. Hace uso del vehículo todos los días, y prácticamente, trayectos cortos, aunque ocasionalmente realiza viajes largos. Suele repostar en aquellas estaciones de servicio que se encuentren aledañas en el recorrido de sus trayectos. No tiene preferencias por las marcas, aunque admite no repostar en aquellas más desconocidas.

2.1.3.2. Definición

Para realizar la definición de los perfiles, se ha utilizado imágenes de personas que no existen a través del sitio web https://this-person-does-not-exist.com/es [14].

Natalia	**Comportamientos**
	- Usa con frecuencia su smartphone - Utiliza el coche de 4 a 5 días a la semana - Siempre reposta en estaciones de servicio ubicadas en su zona de ca casa y trabajo
Demografía	**Necesidades y objetivos**
- 23 años - Soltera - Vive en un piso compartido en Getafe - Trabaja en Torrejón de Ardoz - Utiliza el coche para ir a trabajar	- Ahorrar dinero cuando reposta combustible - Repostar en estaciones más cercanas a su ubicación

Lorena	Comportamientos
	- Usa el coche ocasionalmente, de 0 a 1 días a la semana - Se desplaza a estaciones de servicio aledañas a su casa, aunque a veces realiza un desplazamiento mayor si el ahorra es importante - Utiliza descuentos
Demografía	**Necesidades y objetivos**
- 37 años - Casada, 1 hija - Vive en Leganés - Tiene trabajo estable en Madrid	- Utilizar descuentos para ahorrar en cada repostaje - Repostar cerca de su casa, o si merece la pena desplazarse a una estación lejana pero más económica - Reducir el gasto de combustible en su economía

Marcos	Comportamientos
	- Utiliza de 6 a 7 días a la semana - Siempre realiza trayectos de media distancia para ir al trabajo y cortos en en fin de semana - Ocasionalmente realiza viajes largos - Reposta en estaciones cercanas en su ruta
Demografía	**Necesidades y objetivos**
- 45 años - Viudo, 2 hijos - Vive y trabaja en Madrid	- Busca estaciones cercanas en su ruta - Ahorrar dinero en cada repostaje

2.1.3.4. Conclusiones

Tras la comparativa realizada, se llega a la conclusión de mostrar como pantalla principal el listado de precios ordenados ascendentes. Esta característica es la que el usuario puede llegar a esperar cuando abre la aplicación en busca del mejor precio en su zona.

Asimismo, una funcionalidad muy acusada es el uso del mapa para geolocalizar cada estación de servicio. Es de gran relevancia exhibir el precio próximo al marcador, aportando mayor cantidad de información al usuario. Incluso mostrar el icono de la marca sobre el marcador facilita la rápida identificación.

Sin embargo, la posibilidad de crear alertas cabe la posibilidad de ser una funcionalidad que carezca de uso según el cuestionario realizado y el *benchmarking* analizado. Esta característica evita consultar constantemente la aplicación en aquellos usuarios que buscan el mejor momento para repostar carburante en sus vehículos.

Sumado a lo anterior, ninguna aplicación utilizada en la comparación ha permitido calificar ni la estación de servicio ni la marca del carburante. Esta característica sirve a los usuarios a catalogar a aquellas estaciones de servicio que ofrezcan una mejor calidad de servicio, así como la calidad referente a los carburantes que ofrece. Pero la carencia en las aplicaciones de la comparativa está justificada por la ausencia de votos que ha obtenido en el cuestionario realizado.

Finalmente, la opción de añadir en favoritos las estaciones de servicio, agiliza la visualización en el listado de los precios resaltando este tipo de elementos por encima del resto. Esta característica es útil para aquellos usuarios que deseen repostar en determinadas estaciones de servicio. Aún así, esta funcionalidad no ha sido seleccionada en el cuestionario, por lo tanto, no se descarta que ningún usuario la utilice.

2.2. Diseño conceptual

2.2.1. *Problem statements*

El *problem statement* (también conocido como definición del problema), trata de una técnica que busca identificar y solucionar en el proyecto los problemas a los que se enfrentan los usuarios. Se presenta como un nexo entre el problema concebido por los usuarios y la solución aportada por el proyecto [15].

Por lo tanto, en base a la información recopilada en las fases anteriores, los *problem statements* elaborados son los enunciados a continuación:

- **PS1**
  Los conductores que repostan en estaciones de servicio próximas a su ubicación necesitan una forma de visualizar el precio más bajo posible porque les ayuda a ahorrar en cada repostaje.

- **PS2**
  Los conductores que tienen como preferencia acudir a las estaciones de servicio aledañas necesitan una forma de ver qué estaciones de servicio se encuentran en su zona a través de un mapa.

- **PS3**
  Los conductores que tienen descuentos en determinadas estaciones de servicio necesitan una forma de ver esos descuentos aplicados en el precio que están visualizando en el listado.

- **PS4**
  Los conductores que hayan decidido acudir a una estación de servicio visualizada en el listado o el mapa necesitan una forma de navegar hasta ella utilizando como punto de inicio la ubicación del usuario.

2.2.2. Flujos de interacción

Los flujos de interacción representan gráficamente los diversos caminos que puede recorrer el usuario mientras interactúa con una interfaz gráfica. Estos flujos sirven para comprender el comportamiento del software y mejorar la experiencia de usuario añadiendo, modificando, o eliminando, cualquier interacción que se considere innecesaria para el usuario.

Para su realización se ha utilizado el *software* facilitado por LucidChart, página especializada en cualquier tipo de diagramas [16]. A continuación, se expone los flujos de interacción de la aplicación:

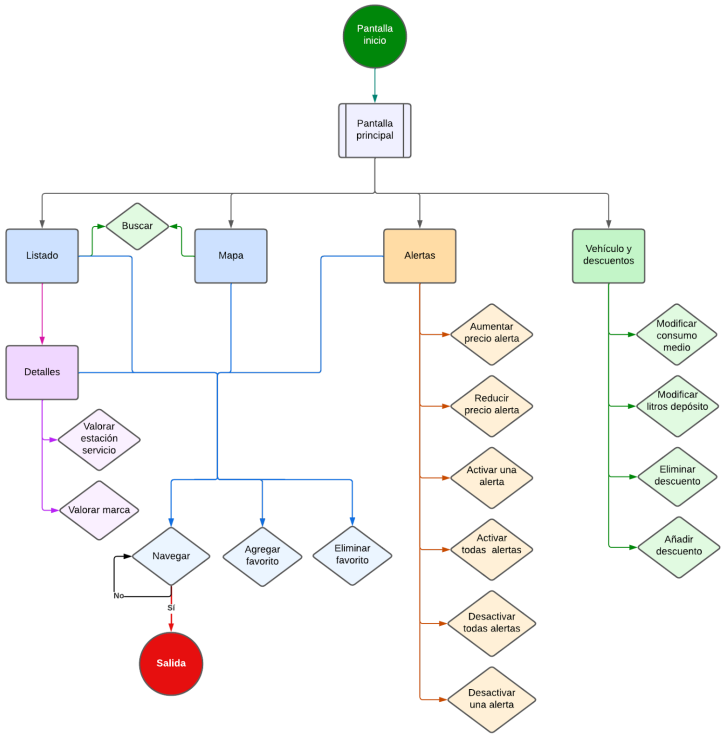

Figura 22: Flujos de interacción

El usuario comienza visualizando la pantalla de inicio, también conocida como *splash*. Después de una breve espera se muestra la pantalla principal, donde se puede observar un menú con cuatro opciones de interacción:

- **Listado**
  En esta pantalla, el usuario puede desplazarse horizontalmente a lo largo de la interfaz. Del mismo modo, tiene la posibilidad de buscar un destino, marcar o desmarcar cualquier elemento como favorito, y por último, navegar a las coordenadas de una estación de servicio a través de una aplicación de terceros.

- **Mapa**
  Del mismo modo que en la pantalla anterior, el usuario puede realizar las acciones de buscar, agregar y quitar elementos de favoritos, y navegar externamente. Asimismo, puede interactuar tanto en los ejes verticales como horizontales con la interfaz del mapa integrado, incluso pellizcar para aumentar o disminuir el zoom.

- **Alertas**
  El usuario interactúa con las mismas acciones ya descritas en las pantallas anteriores: marcar y desmarcar como favoritos, y navegar. Además, puede aumentar o disminuir el precio por el cual será activada la alerta, activar o desactivar todas las alertas, y por último, activar o desactivar individualmente una alerta.

- **Vehículo y descuentos**
  En esta pantalla el usuario puede modificar el consumo medio de su vehículo, así como la capacidad total de su depósito. Del mismo modo, tiene la opción de añadir y eliminar descuentos.

Finalmente, la pantalla de detalles, permite al usuario visualizar más datos asociados con el elemento en cuestión. Se puede acceder desde el listado de precios y desde el mapa. En esta pantalla el usuario interactúa con las cuatro opciones presentes en otras pantallas: filtrar, marcar y desmarcar favoritos, y navegar. De igual modo, puede valorar tanto una estación de servicio como una marca de carburante.

2.3. Prototipado

2.3.1 Sketches

Los dibujos a mano alzada, también conocidos como sketches, permiten materializar las características y funcionalidades que los usuarios utilizarán en la aplicación. Realizar esta técnica al principio facilita la posibilidad de concentrar el esfuerzo en los aspectos más útiles para el usuario, dejando para la siguiente fase las definiciones más técnicas del diseño de la interfaz de usuario. En el punto 9.2. De los anexos se amplia información sobre el proceso realizado.

2.3.2 Prototipo horizontal de alta fidelidad

Finalmente, después de la elaboración de los flujos de interacción y el diseño a mano alzada de las pantallas de la aplicación, se realiza el prototipo horizontal de alta fidelidad. Con este prototipo se pretende plasmar todas las características y funcionalidades con las que el usuario va a interactuar. Además, se proporciona una guía de estilos que será utilizada para realizar el diseño del prototipo, y más tarde, para la implementación de las pantallas a través del código.

Para la realización del prototipo se ha utilizado el *software* de diseño proporcionado por Figma [17], y como base la plantilla *Now In Android Case Study* desarrollado por el equipo de Material Design [18].

Para visualizar al completo el prototipo desarrollado se debe visitar el siguiente enlace: https://www.figma.com/file/JFO3sADXeedMNBhBxyr2Wf/TFG__Fuel_Price__Jesus_Rodriguez_Layos?type=design&mode=design&t=ZW9MYjgUSHPVkKdE-1.

2.3.2.1. Tipografía

Es importante elegir una tipografía adecuada para el usuario. Debe permitir leer sin problemas los textos representados en la pantalla, sin necesitar ningún tipo de esfuerzo por parte del usuario.

Asimismo, esta tipografía debe tener una consistencia a lo largo de la interfaz, evitando utilizar diferentes tamaños para elementos con características visuales similares. Esta uniformidad dotará a la interfaz de un mismo componente tipográfico, ayudando al usuario a comprender mejor los elementos visuales que está contemplando.

La tipografía elegida para esta aplicación es Montserrat. Se trata de una tipografía muy utilizada en diferentes ámbitos, lo cual hace que sea más familiar para el usuario. Tiene una interpretación moderna, ideal para proyectos minimalistas [10]. Se puede interactuar con la tipografía a través de Google Fonts:
https://fonts.google.com/specimen/Montserrat [20].

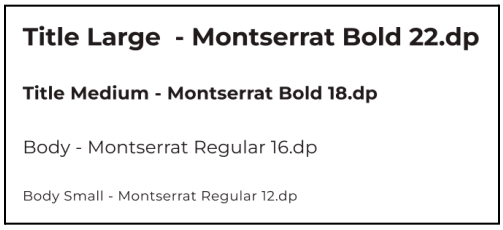

Figura 23: Tipografía

En total son cuatro los diferentes tamaños que se utilizan en el diseño del prototipo. Se pueden catalogar en dos grupos: encabezados y cuerpos.

Por un lado, para los encabezados se ha elegido un estilo con un mayor peso visual con el objetivo de resaltar el texto y hacerlo más llamativo sobre el resto de elementos. Del mismo modo, el tamaño es mayor con el fin de alcanzar dicho propósito.

Por otro lado, para el resto de textos, principalmente cuerpos, se usa un estilo estándar, sin cargar en exceso el elemento gráfico y ayudando al usuario una fácil diferenciación entre aquellos elementos que se desean destacar, y los que no.

2.3.2.2. Colores

La elección de los colores es una fase muy importante porque servirá para identificar la aplicación frente al resto de competidores. Además, es fundamental elegir los colores más adecuados para el tipo de aplicación que se va a desarrollar, evitando aquellos que puedan crear problemas visuales a los usuarios por no ser legibles los elementos que se muestran por pantalla.

En este sentido, la aplicación tiene tres colores principales: azul, blanco y negro. Se ha dejado el color rojo, naranja y verde para indicar el color del precio en el mapa, o cuando una notificación está activada o desactivada.

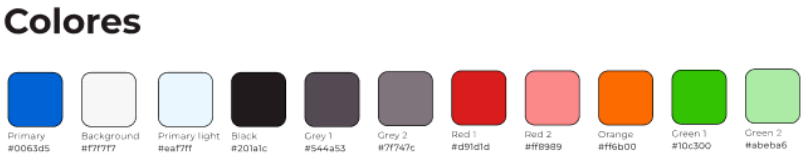

Figura 24: Colores

2.4.2.3. Logo

Se ha elegido un icono que inmediatamente transmita al usuario la temática de la aplicación. En este caso, se trata de una manguera de repostaje, un objeto más que habitual para los usuarios que acudan a las estaciones de servicio.

Figura 25: Logo

2.3.2.4. Pantallas

2.3.2.4.1. Splash

Figura 26: Prototipo *splash*

Se ha utilizado como icono de la aplicación un dispensador de combustible para ayudar a los usuarios a identificar más fácilmente la aplicación. Siguiendo un estilo minimalista, se ha decido usar como fondo gris claro, haciendo que resalte aún más el icono con tinte negro y sombras en el interior.

2.3.2.4.2. Listado de precios

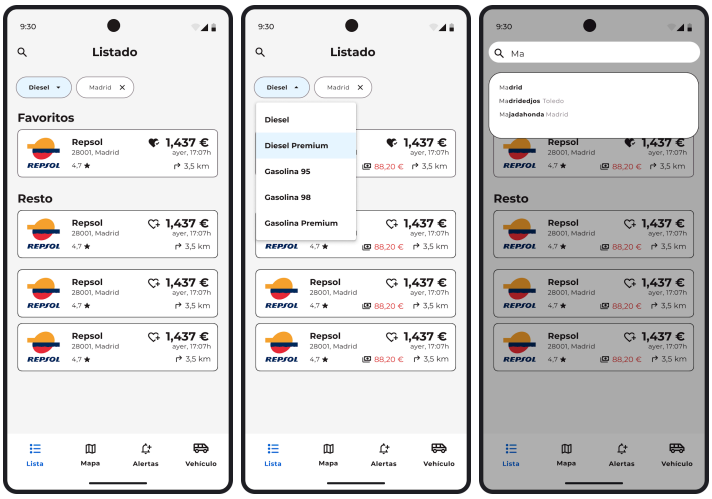

Figura 27: Prototipo listado de precios

Siguiendo la línea anterior, el fondo es rellenado con gris claro, resaltando muy pocos elementos. Estos elementos son el tipo de combustible por el cual se están filtrando los precios, el logo de la marca del carburante, el coste que tendría llenar el depósito al completo, y finalmente, el precio.

2.3.2.4.3. Detalles

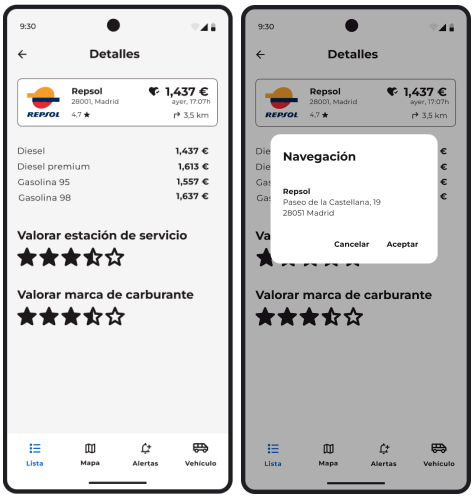

Figura 28: Prototipo detalles

Se accede pulsa cualquier elemento del listado. En esta pantalla se visualizan las mismas acciones como marcar o desmarcar favoritos y navegar a la estación de servicio. Se ha

ocultado la opción que muestra el coste por llenar el depósito porque se considera que en esta pantalla no es una opción que tenga relevancia.

Asimismo, se muestra información adicional de la estación de servicio como los diferentes precios de los carburantes que se pueden repostar. Y finalmente, se puede valorar tanto la estación de servicio como la marca del carburante.

2.3.2.4.4. Mapa

En esta pantalla se muestran los marcadores utilizando el logo de la marca del carburante, de esta forma el usuario puede identificarlos fácilmente. Del mismo modo, en la parte superior del marcador se visualiza el precio del combustible, y a su izquierda, el símbolo para marcar o desmarcar favorito.

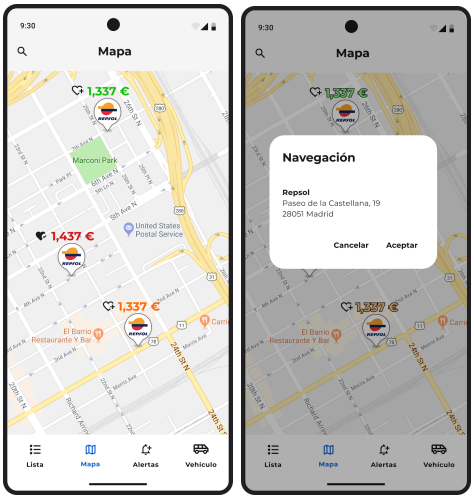

Figura 29: Prototipo mapa

2.3.2.4.5. Alertas

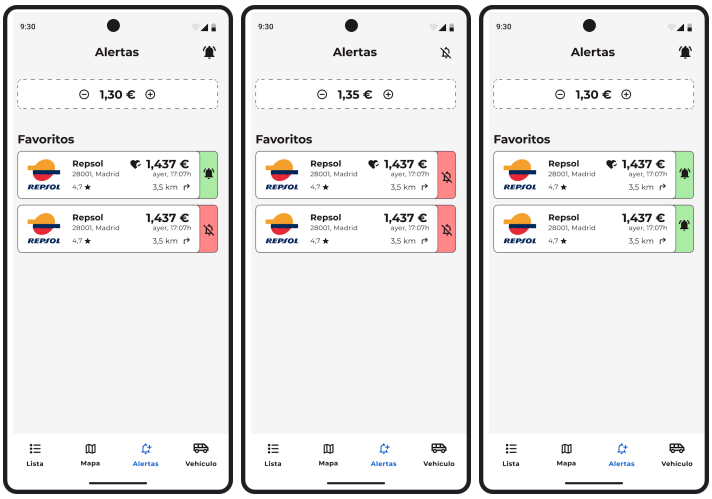

Figura 30: Prototipo alertas

El icono elegido para esta pantalla ubicado en la barra de navegación inferior, ha sido con la intención de indicar al usuario de una forma visual que en esa pantalla puede crear alertas.

En esta pantalla, se puede modificar el precio de la alerta utilizando dos iconos ubicados a los lados de la etiqueta. Asimismo, se pueden activar o desactivar todas las alertas pulsando el icono de la campana ubicado en la parte superior. Y además, se pueden realizar estas desactivaciones de forma individual.

2.3.2.4.6. Vehículo y descuentos

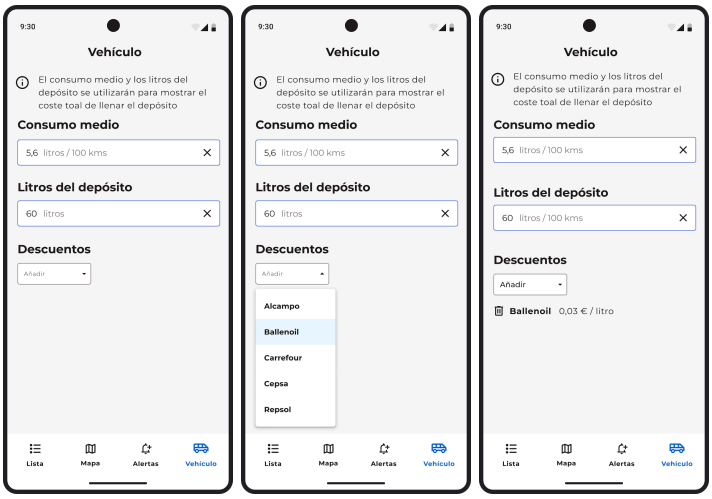

Figura 31: Prototipo vehículo y descuentos

Desde esta pantalla se puede indicar el consumo medio del vehículo, así como la capacidad total del depósito. Estos datos se utilizarán para calcular el precio total de llenar el depósito al completo.

De la misma manera, se pueden añadir descuentos proporcionados por las diferentes marcas de carburantes y las estaciones de servicio. Una vez agregado un descuento, éste puede ser eliminado fácilmente pulsando sobre el icono de la papelera.

2.3.2.4.7. Diálogo navegación

Finalmente, cuando el usuario pulsa sobre la característica de navegación sobre cualquier estación de servicio, se muestra un diálogo pidiendo la confirmación al usuario. En caso de aceptar, se abrirá una vista nativa del sistema operativo donde el usuario deberá seleccionar la aplicación que deseé usar para la navegación.

Figura 32: Prototipo diálogo de navegación

2.4. Evaluación

Esta fase tiene un papel importante en el Diseño Centrado en el Usuario (abreviado como DCU) por los datos que se recopilan tras el análisis de los diseños desarrollados. La evaluación se realiza a través de test de usuarios que consiste en un cuestionario que se realiza a un grupo de usuarios seleccionados a conciencia en base a sus características [21] [22].

La ejecución de esta fase consiste en definir en primer lugar el alcance que tendrá. A continuación, definir el perfil de cada usuario que participará en el cuestionario. Después, realizar el guión que se utilizará, y finalmente, analizar los resultados y elaborar las conclusiones.

El cuestionario se realizará a través de una videollamada, sin realizar grabación. Al inicio de la evaluación se pedirá el consentimiento para registrar las respuestas de los participantes. Y por último, al finalizar, se agradecerá el tiempo dedicado a la participación.

2.4.1. Definición del alcance de la evaluación

El objetivo de la evaluación es analizar a través de un cuestionario los diseños realizados en un prototipo de alta fidelidad. Por lo tanto, el alcance de la evaluación se delimitará exclusivamente en estos diseños, siendo los descritos a continuación:.

- **Pantalla de inicio**
  En esta pantalla se evaluará el diseño elegido.

- **Pantalla principal**
  La evaluación consiste en observar los elementos que forman la barra de navegación inferior, desde donde se accede al resto de pantallas.

- **Listado de precios**
  En primer lugar se debe observar el icono de búsqueda, ubicado en la parte superior

izquierda. A continuación, el título de la pantalla. Más abajo se encuentra la etiqueta que identifica el tipo de combustible por el cual el usuario está visualizando los precios.

En el propio listado se debe observar las características de cada elemento, siendo: logo de la marca del carburante, nombre de la estación de servicio, código postal y municipio, valoración de la estación de servicio, icono de favoritos, precio, última fecha de actualización, coste total por llenar el depósito, y finalmente, icono de navegación junto con la distancia en la que se encuentra.

- **Detalles**
  En esta pantalla, se debe evaluar el icono para regresar a la pantalla anterior ubicado en la parte superior derecha, y también, el título.

  Luego, se observa un elemento muy similar al visto en la pantalla anterior, por lo tanto las características a evaluar son las mismas salvo el coste total por llenar el depósito al completo.

  Por último, se evaluarán dos elementos para valorar a la estación de servicio y a la marca del carburante.

- **Mapa**
  Se analizará el marcador elegido para cada estación de servicio, así como los colores escogidos para los precios. De igual modo, en la parte superior se observará el icono de búsqueda y el título de la página.

- **Alertas**
  Igual que se ha observado en listado de precios y detalles, se evaluará el elemento en común para las tres, siendo: la vista de precio por estación de servicio.

  Además, se debe evaluar dos botones ubicados a los lados de la etiqueta que indica el precio de la alerta, y la posibilidad de activar o desactivar todas las alertas, y, activar o desactivar de forma individual.

- **Vehículo y descuentos**
  En esta pantalla se analizarán los dos elementos que hacen referencia al vehículo: el consumo medio y la capacidad total del depósito. Del mismo modo, se evaluará la posibilidad de añadir y eliminar descuentos.

2.4.2. Definición del perfil de los participantes y captación

Los participantes seleccionados para la participación cuentan con unas características muy similares a los usuarios descritos anteriormente en el punto *2.2. Perfiles de usuario*. El total de participantes es de tres personas, siendo un participante por cada perfil de usuario definido.

Para preservar su anonimato, se han utilizado imágenes de personas no reales disponibles en la página https://this-person-does-not-exist.com/es [5].

Los participantes son los descritos a continuación:

Marta	Comportamientos
	- Uso intensivo de aplicaciones - Utiliza su vehículo de 4 a 5 días en la semana - Reposta en estaciones de servicio cercas de su zona, aunque no descarta desplazarse a una más lejana si supone un ahorro
Demografía	**Necesidades y objetivos**
- 27 años - Soltera - Vive en un piso compartido en Parla - Trabaja en Alcobendas - Utiliza el coche en cualquier desplazamiento que realiza, siendo el principal el trabajo	- Encontrar el precio más bajo de carburante para ahorrar la mayor cantidad de dinero en cada repostaje - Evitar desplazamientos largos para repostar si no supone un ahorro significativo

Alba	Comportamientos
	- Utiliza el coche de 0 a 1 días en la semana - Solo reposta en estaciones de servicio cercanas a su casa - Siempre utiliza descuentos
Demografía	**Necesidades y objetivos**
- 41 años - Casada, 2 hijos - Vive a las afueras de Madrid - Tiene trabajo estable en el centro de Madrid	- Repostar en estaciones de servicio cercanas a su ubicación - Utilizar descuentos - Ahorrar en cada repostaje

Roberto	Comportamientos
	- Uso de 6 a 7 días a la semana si vehículo - Solo realiza trayectos cortos - Suele repostar en estaciones de servicio que se encuentran cercanas en sus trayectos

Demografía	Necesidades y objetivos
- 53 años - Viudo, 1 hija - Vive y trabaja en Madrid	- Repostar en estaciones de servicio ubicadas cercanamente en las rutas que realiza - Ahorrar dinero cada vez que reposta

2.4.3. Definición del guión de las sesiones

El objetivo de definir el guion de las sesiones es recopilar la mayor información posible sobre las evaluaciones realizadas por los participantes. Esto ayudará a refinar el prototipo de alta fidelidad y lograr obtener un diseño más acertado para los usuarios que van a interactuar con la aplicación.

Las preguntas que se realizarán son las enumeradas a continuación:

1. Información general

- ¿Usas con frecuencia aplicaciones para encontrar precios económicos de cualquier producto?
- ¿Utilizas aplicaciones para encontrar el mejor precio del combustible en tu zona?

2. Pantalla de inicio

- ¿Cómo visualizas la pantalla de inicio?

3. Pantalla de inicio

- ¿Entiendes y comprendes la barra de navegación inferior?

4. Listado de precios

- ¿Encuentras el botón de buscar? ¿Qué funcionalidad piensas que tiene?
- ¿Puedes determinar qué combustible se está utilizando para filtrar los resultados? ¿Sabrías cambiar de combustible?
- ¿Comprendes cada elemento que se lista en la pantalla?
- ¿Puedes encontrar el icono para marcar o desmarcar favoritos?
- ¿Y el de navegar?
- ¿Sabrías entrar en los detalles de una estación de servicio?

5. Detalles

- ¿Puedes encontrar las opciones de marcar como favorito? ¿Y navegar?
- ¿Comprendes qué significa valorar la estación de servicio y la marca del carburante?

6. Mapa

- ¿Visualizas correctamente los marcadores en el mapa?
- ¿Y los precios?

7. Alertas

- ¿Entiendes qué significan los iconos ubicados a los lados de la etiqueta del precio?
- ¿Sabrías activar o desactivar todas las alertas?

- ¿Y de forma individual?

8. Vehículo y descuentos

- ¿Por qué piensas que se pide el consumo medio y la capacidad total del depósito?
- ¿Sabrías añadir y eliminar un descuento?

9. Conclusión y mejoras

- ¿Qué te parecen los diseños? ¿Los ves claros y concisos?
- ¿Qué añadirías?
- ¿Qué cambiarías?
- ¿Qué eliminarías?
- ¿Utilizarías la aplicación?

2.4.4. Realización de las sesiones

Después de definir el cuestionario de la sesión, se realizan las preguntas a los usuarios seleccionados. Las sesiones serán realizadas a través de una videollamada sin realizar la grabación de la misma, únicamente se registrarán de forma escrita sus respuestas con el fin de poder analizarlas posteriormente. Al inicio de la sesión se pide el consentimiento para tal fin, y al finalizar, se agradece textualmente por el tiempo dedicado en su participación.

En los anexos se pueden encontrar las entrevistas realizadas, punto 9.3.

2.4.5. Análisis de los resultados

A continuación, se expone el análisis realizado a partir de los resultados recopilados de la realización del cuestionario a los participantes seleccionados. Este análisis engloba las características principales de la aplicación, con el fin de visualizar de una forma clara y rápida qué aspectos positivos y negativos se han recogido.

	Usuario 1: Marta	
	Solo reposta en marcas de precios bajos	
	Aspectos positivos	**Aspectos negativos**
Pantalla inicial	Buen diseño	-
Pantalla principal	Iconos bien elegidos	-
Listado de precios	Icono buscar, logo, nombre estación de servicio, precio, icono de favoritos e icono navegación	Coste por llenar el depósito al completo
Detalles	Todo	-
Mapa	Logo y precio en cada marcador, y	Color verde, difícil visibilidad

	colores naranja y rojo	
Alertas	Todo	-
Vehículo y descuentos	Descuentos	Desconocimiento por qué se pide consumo medio y capacidad del depósito

	Usuario 2: Alba	
	Reposta en marcas de confianza y siempre utiliza descuentos	
	Aspectos positivos	**Aspectos negativos**
Pantalla inicial	Buen diseño	-
Pantalla principal	Iconos y textos bien elegidos	-
Listado de precios	Todo salvo un icono	Coste depósito
Detalles	Todo	-
Mapa	Logo, colores naranja y rojo	Color verde, no se aprecia
Alertas	Todo	-
Vehículo y descuentos	Descuentos	Desconocimiento por qué se pide consumo medio y capacidad del depósito

	Usuario 3: Roberto	
	Tiene inclinación por marcas de confianza aunque siempre busca repostar en las ubicadas en su ruta	
	Aspectos positivos	**Aspectos negativos**
Pantalla inicial	Buen diseño	-
Pantalla principal	Iconos y textos	-
Listado de precios	Todo	-
Detalles	Todo	-
Mapa	Logo, colores naranja y rojo	Color verde, no se ve bien
Alertas	Todo	-

Vehículo y descuentos	Descuentos	Desconocimiento por qué se pide consumo medio y capacidad del depósito

2.4.6. Conclusiones

Tras realizar los cuestionarios a los usuarios seleccionados, y una vez analizada la información recopilada, se puede concluir lo siguiente:

- **Diseño**

  Se utiliza una línea minimalista donde los usuarios lo aprecian y lo agradecen. No existe exceso de colores, consiguiendo así evitar sobrecargar la interfaz con diferentes puntos de atención, y centrando toda la atención al dato más relevante: el precio.

- **Pantallas**

  En general se entiende toda la información que se plasma en cada diseño, ayudando en ocasiones por iconografía que en la mayoría de los casos facilita la comprensión por parte del usuario. Cabe destacar el elemento que indica el coste total por llenar el combustible, dos participantes no han comprendido su significado, por lo tanto se debe realizar una valoración para mejorarlo.

  Asimismo, se debe incorporar algún elemento que ayude al usuario a comprender aún más sobre esta funcionalidad. En la pantalla del vehículo se solicita información acerca del consumo medio y la capacidad total del combustible. En dos ocasiones, los usuarios no han sabido identificar el motivo por el cual se pregunta esto. Por ello, se debe modificar esta característica para evitar esta confusión en el futuro.

  Por último, el color escogido para representar el precio en verde sobre el mapa no es del todo visible, como afirman los participantes. Elegir un tono más oscuro facilita la visibilidad del mismo. Otra opción sería crear un borde sobre el precio, mejorando incluso aún más la visibilidad sobre zonas del mapa que predomine el verde.

- **Mejoras**

  Los participantes han sugerido mejoras sobre la aplicación. Una de ellas consiste en añadir un filtro para mostrar solo aquellas marcas de carburantes que el usuario esté interesado en ver. Y otra, consiste en mostrar las estaciones de servicio aledañas a partir de una ruta.

  Ambas mejoras mejorarían notablemente la experiencia de usuario, pero actualmente no es posible realizarlas por los tiempos que conlleva su ejecución. Sin embargo, se tendrán en consideración en posibles actualizaciones.

2.4.7. Mejoras aplicadas

Tras el análisis de las mejoras detectadas durante la evaluación, se han aplicado aquellas que no afecten en exceso el transcurso de la planificación del proyecto, y además mejoran notablemente la experiencia del usuario.

Las mejoras aplicadas son las siguientes:

- Mejorar la visibilidad de la etiqueta precio cuando es de color verde en el mapa

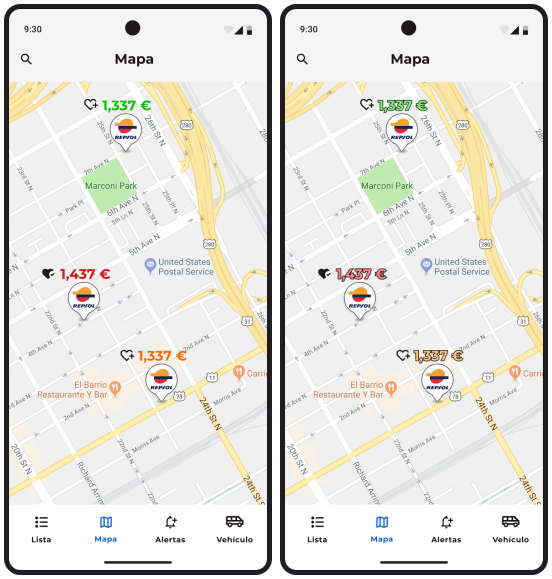

Figura 33: Prototipo mejorado sobre listado de precios, antes y después

- Mostrar información del motivo por el cual se solicita el consumo medio y la capacidad del depósito

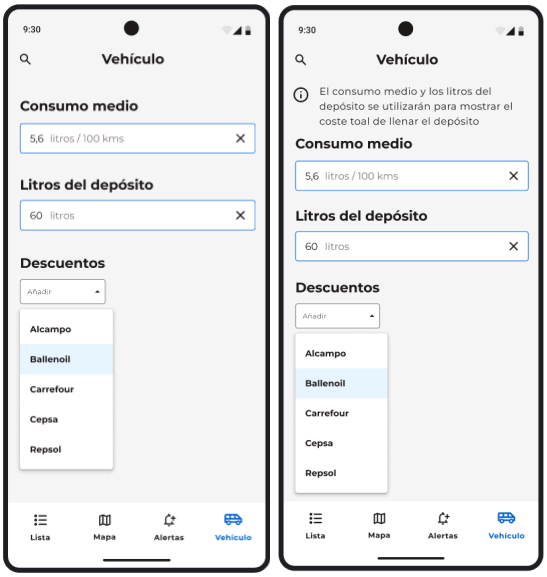

Figura 34: Prototipo mejorado sobre vehículo y descuentos, antes y después

Tras aplicar estas mejoras, los colores han sido modificados. A continuación, se exponen las modificaciones:

- ***Red1 (#d91d1d)***

  El color ya no se usa, por lo tanto se elimina de la guía de estilos y automáticamente el color *Red2 (#ff8989)* pasaría a ser *Red*.

- ***Orange (#ff6b00)***

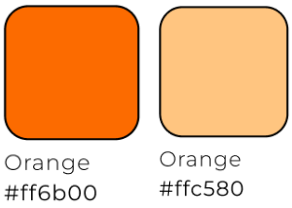

Figura 35: Prototipo mejorado sobre color *Orange*, antes y después

El color *Orange* ha modificado su valor *#ff6b00* por un tono más claro, para garantizar una mejor visibilidad sobre el mapa. El nuevo valor es *#ffc580*.

- ***Green1 (#10c300)***

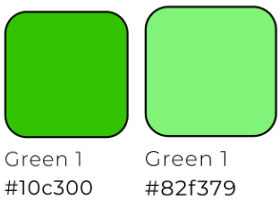

Figura 36: Prototipo mejorado sobre color *Green1*, antes y después

Para facilitar su visualización sobre el mapa, se ha elegido un tono más claro, que ayudado por el borde de la letra, mejora considerablemente la percepción de la etiqueta. El color actual tiene el valor *#82f379*.

3. Diseño técnico

El diseño técnico es la etapa durante el desarrollo del proyecto que define la estructura del sistema a través de diagramas, en particular para en esta ocasión son cuatro, siendo: de actores, de clases, de base de datos, de colecciones en Firebae.

Además, se especifica la arquitectura del sistema, donde se delimita el ámbito de acción de cada clase y la visibilidad que cada capa tendrá. La arquitectura es la pieza inicial del sistema que servirá para sustentar todas las clases y establecer comunicaciones entre los diferentes componentes.

Finalmente, se presentan los casos de uso que formarán parte del sistema y determinará qué acciones llevarán a cabo cada funcionalidad y características expuesta en etapas anteriores.

3.1. Diagramas estructurales

Para la realización de los siguientes diagramas se ha utilizado un tipo de lenguaje muy común en éste ámbito, el lenguaje *UML* (Lenguaje Unificado de Modelado, conocido del inglés *Unified Modeling Language*) [23]. Asimismo, se ha utilizado la herramienta *LucidChart* [24] para la realización de los diagramas.

3.1.1. Diagrama de clases

El diagrama de clases del proyecto es el descrito a continuación:

- ***GasStation***
  Contiene la información de las estaciones de servicio.

- ***Brand***
  Hace referencia a la marca del carburante. Esta clase está contenida en la clase *GasStation*.

- ***SaleType***
  Se trata de un enumerado que indica el tipo de venta que realiza la estación de servicio. Puede ser de venta pública (*PUBLIC*) o privada *(PRIVATE*). Este atributo lo contiene la clase *GasStation*.

- ***Place***
  Incluye datos de la ubicación de la estación del servicio. Al contener las coordenadas, no es posible crear una relación entre *Place* y *GasStation* 1:N.

- ***Price***
  Comprende todos los precios que pueden haber en una estación de servicio.

- ***Discount***
  En esta clase se encuentra la información relacionada con los descuentos que proporcionan cada marca de carburante y estación de servicio.

- ***DiscountType***
  Se trata de un enumerado donde se indica cómo se aplica el descuento sobre el precio del combustible. Puede ser en céntimos (*CENTS*) o en porcentaje *(PERCENTAGE*).

- ***Brand***
  Información de la marca del carburante.

- **Alert**
  Contienen los datos necesarios para generar una alerta.

- ***FuelType***
  Es un enumerado que hace referencia a los tipos de combustible. Este enumerado se utiliza en la clase *Alert* para identificar el tipo de carburante del que el usuario desea ser notificado.

- ***Rating***
  Se utiliza para valorar una marca (*BrandRating*) o una estación de servicio

(*GasStationRating*). Es una clase abstracta para cumplir con el Principio de Abierto-Cerrado, también conocido como *Open-Closed Principle* [25].

Este principio define que un objeto debe estar abierto a extensión y cerrado a modificación. La alternativa hubiese sido trabajar con un enumerado, lo que implicaría modificar todos los fragmentos de código cuando éste aumentase.

- ***BrandRating***
  Clase que hereda de *Rating*.

- ***GasStationRating***
  Igual que la clase anterior, hereda de *Rating*.

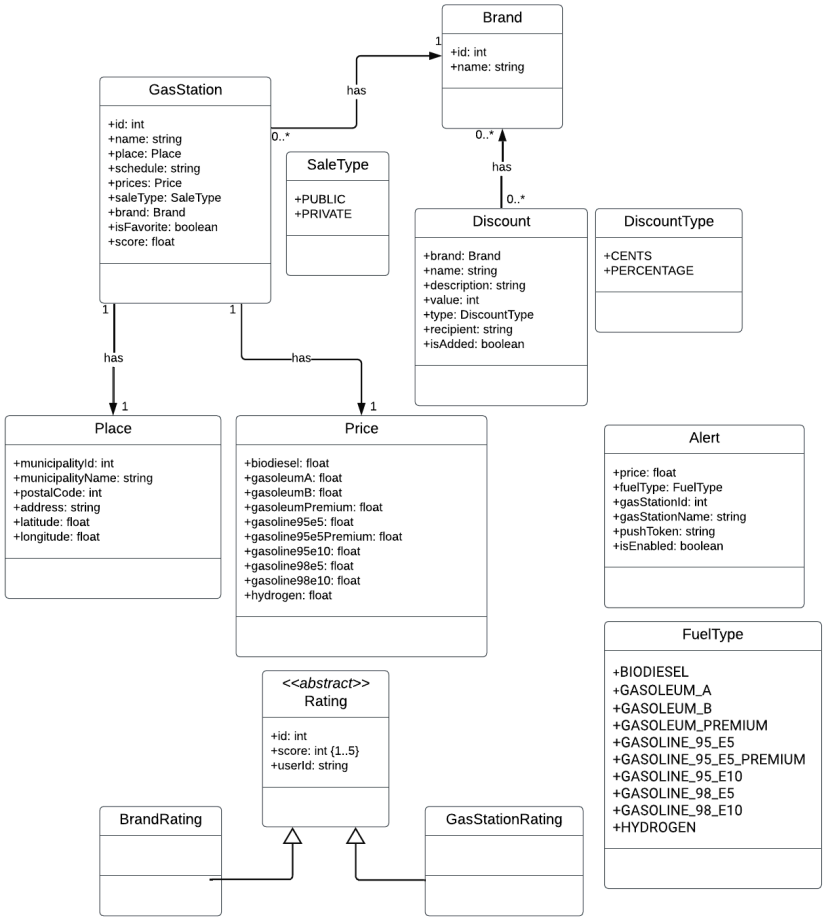

Figura 37: Diagrama de clases

3.1.2. Diagrama de base de datos

A continuación, se detalla el diagrama utilizado en la base de datos, donde se han reemplazado los enumerados por valores de tipo primitivo.

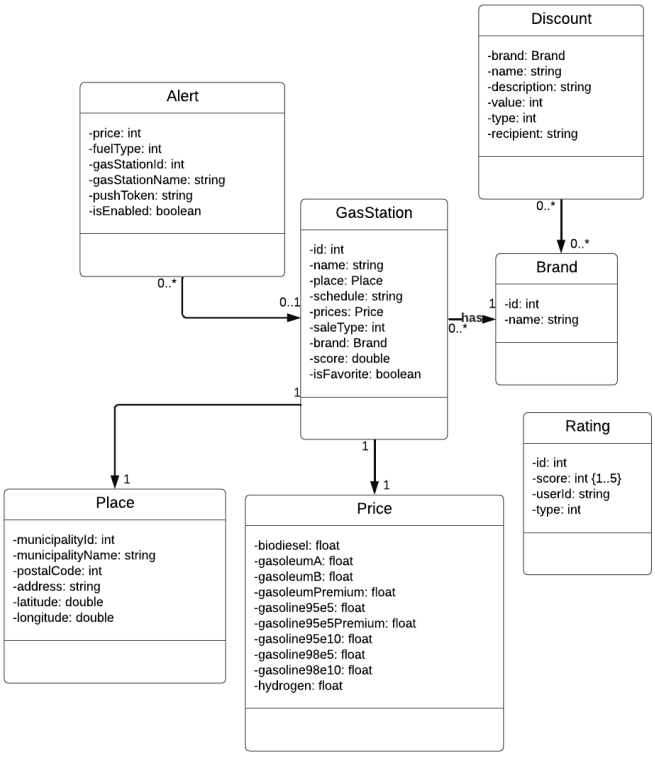

Figura 38: Diagrama de base de datos

3.1.3. Diagrama de colecciones en *Firebase*

Firebase es una plataforma en la nube que ofrece un conjunto de herramientas a los desarrolladores, entre ellas se encuentra *Cloud Firestore*, una base de datos no relacional. Esto significa que los datos serán almacenados bajo colecciones de datos [26] [27].

En vista a esta característica, en *Firebase* se almacenará la información relevante para realizar las notificaciones desde un sistema en la nube hacia el dispositivo del usuario.

Las colecciones almacenadas en *Cloud Firestore* son las plasmadas a continuación:

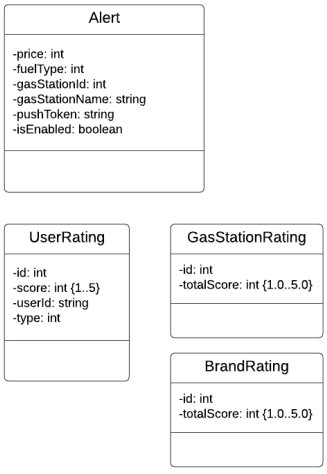

Figura 39: Diagrama de *Cloud Firestore*

3.1.4. Diagrama de almacenamiento en *Cloud Firestore*

A través de las bibliotecas que proporciona *Firebase* se almacenan los datos en documentos, formando colecciones dentro de la herramienta *Cloud Firestore*. La aplicación es la encargada de preparar el contenido y la estructura que tendrá dicho documento. A continuación, haciendo uso de la biblioteca de *Firebase* se envían los datos que más tarde serán procesados y almacenados por *Cloud Firestore*.

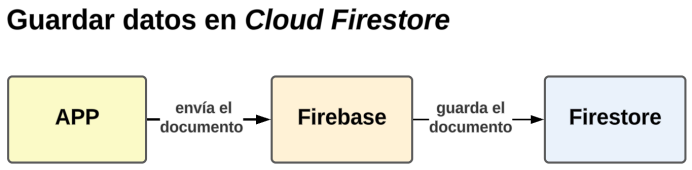

Figura 40: Diagrama para guardar datos en *Cloud Firestore*

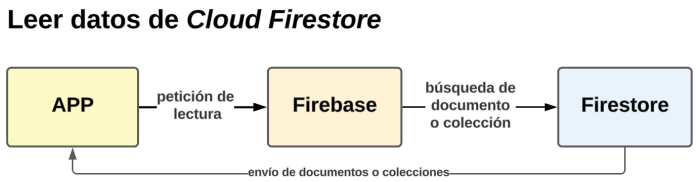

Figura 41: Diagrama para leer datos en *Cloud Firestore*

3.1.5. Diagrama para enviar las notificaciones

Los usuarios tienen la posibilidad de crear alertas en la aplicación para ser avisados cuando el precio del combustible de una determinada estación de servicio sea igual o menor al precio indicado.

Para lograrlo, se utiliza la herramienta que proporciona *Firebase* llamada *Cloud Functions* [28]. Esta herramienta permite ejecutar código de *backend* sin necesidad de configurar un servidor, son servicios que se conocen como *Serverless computing* [29]. Entre los lenguajes de programación aceptados está Python [30], el cual se usará para programar la lógica del procesado de datos y el envío de las notificaciones.

El funcionamiento es el siguiente:

1. Cada 6 horas se activará este proceso llamando a la función creada para este proceso. Esto es posible gracias a la configuración de un *cron*, que consiste en ejecutar una tarea en un determinado tiempo especificado previamente.
2. Se llama a la *API* de “Precio de carburantes en las gasolineras españolas“ [31], y se procesan los datos recibidos.
3. Se obtienen todas las alertas activas almacenadas en *Cloud Firestore*.
4. Se itera por cada alerta y se comprueba si el precio de la gasolinera es igual o menor.
    a. Si es verdadero: se llama a la función de envío de notificaciones.
    b. Si es falso: se pasa a la siguiente iteración.

5. De haber sido verdadero, a partir del *token push* almacenado en el documento de la alerta, se genera una notificación en *Firebase Cloud Messaging* (también conocido por sus siglas *FCM*, es una plataforma de envío masivo de notificaciones) [32].
6. Si el dispositivo destinatario aún mantiene el *push token* asignado por primera vez al abrir la aplicación, recibirá la notificación. En caso de no ser posible el envío a un destinatario, la notificación se termina eliminando del servidor *FCM* pasado 7 días.

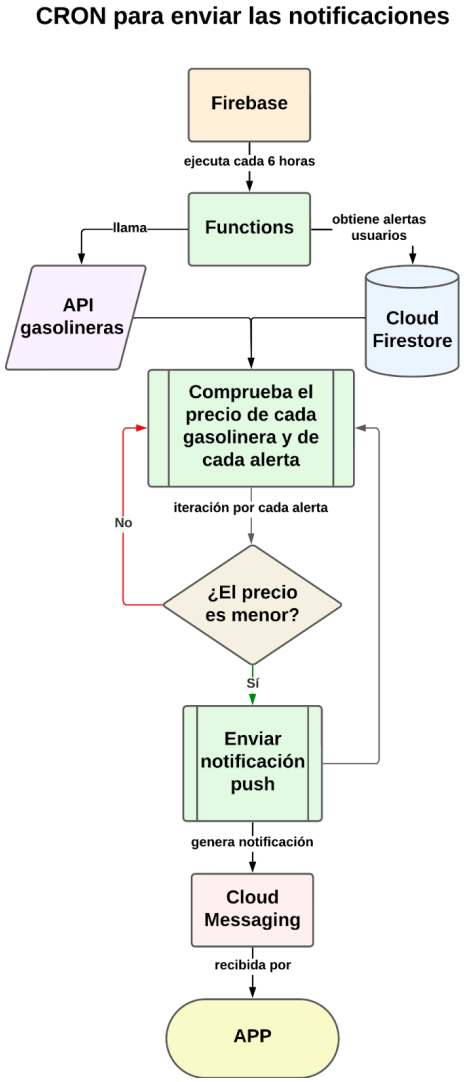

Figura 6: Diagrama para enviar notificaciones

3.2. Arquitectura

Se ha elegido una arquitectura basada en capas porque permite separar cada componente en ámbitos de actuación, incluso favoreciendo en el futuro la modularización del proyecto. Asimismo, la capa central, la capa de dominio, no tendrá visibilidad hacia ninguna de las capas adyacentes, siendo presentación y datos. Del mismo modo, la capa de presentación y la capa de datos únicamente tendrán visibilidad hacia la capa de datos. En el siguiente esquema se puede comprender mejor:

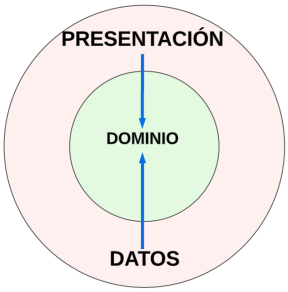

Figura 43: Arquitectura en capas simplificado

Las tres capas que la componen son:

- **Presentación**

  También conocida como *UI Layer* [33], contiene todos los elementos que representan información en la pantalla, y por ende, trabajan de forma directa con el contexto de la aplicación.

  Asimismo, se utiliza el paradigma MVI (*Model View Intent*) para presentar los datos en las clases *activity* o *fragment* [34].

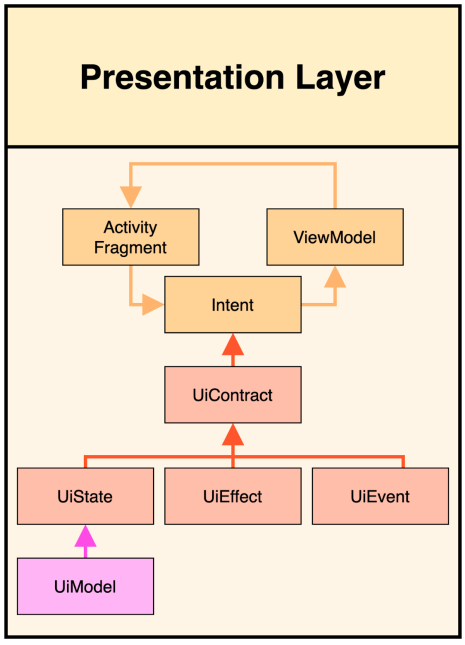

Figura 44: Arquitectura *Presentation Layer*

- **Dominio**

  En esta capa, llamada también *Domain Layer* of Business-*Logic Layer* [35], se encuentra la lógica de negocio, y por lo tanto, también se halla la clase *DataModel* y *Exception*. La primera hace referencia a cualquier modelo de datos, y la segunda, a las excepciones que puedan generarse durante el tratamiento de los datos.

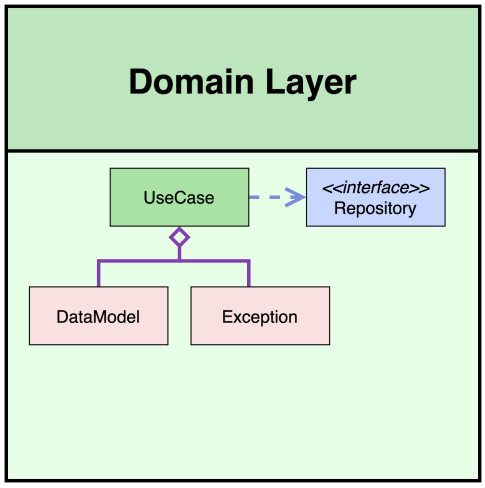

Figura 45: Arquitectura *Domain Layer*

- **Datos**

  También es conocida como *Data Layer* [36]. Se encarga de acceder a las diferentes fuentes de datos que utiliza el sistema, como puede ser el acceso a datos en red a través de un *API Rest* [37] [38], el acceso a una base de datos en local usando una biblioteca *ORM* [39], o incluso, obtener datos del contenedor de Android como es el caso de *SharedPreferences* [40].

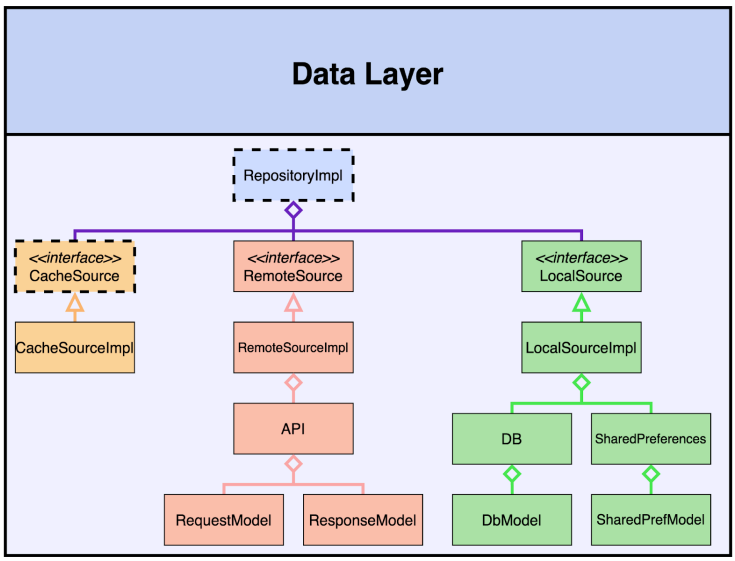

Figura 46: Arquitectura *Data Layer*

La imagen al completo de la arquitectura es la siguiente:

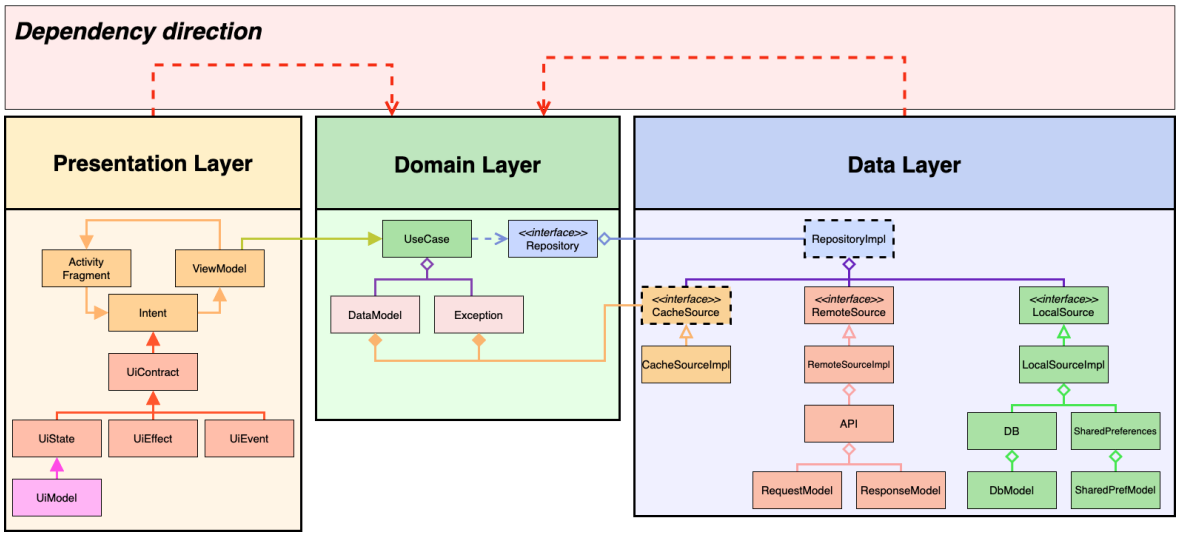

Figura 47: Arquitectura

3.3. Paradigma MVI

El patrón de presentación *MVI* (*Model-View-Intent*, también conocido como Modelo-Vista-Intención) es una derivación del patrón *MVC* (*Model-View-Controller*, también conocido por Modelo-Vista-Controlador) que plantea el envío de datos entre los diferentes componentes de manera unidireccional, separando la responsabilidad de cada elemento que compone este patrón [41].

Los componentes que forman este patrón de presentación son:

- ***Model* (Modelo)**
  Contiene el estado que se está representando por pantalla. Residen modelos de negocio o modelos adaptados a la interfaz de usuario.

  Un posible estado es “cargando”, donde se muestra un elemento visual que indica al usuario que se está cargando la pantalla, siendo éste un modelo de interfaz de usuario.

  En cambio, un estado que contiene un modelo de negocio sería representar un elemento visual con el nombre y el precio de la estación de servicio.

- ***View* (Vista)**
  Es la encargada de representar los diferentes estados por pantalla. Esta vista puede ser una *activity* (también conocida como actividad), *fragment* (como fragmento) o una vista *composable* (como componible).

- ***Intent* (Intención)**
  Contiene todas las iteraciones que realiza el usuario (o el sistema) sobre la vista, o viceversa, aquellas iteraciones que el usuario visualizará por pantalla después de una determinada acción.

Estas iteraciones, conocidas como intenciones, se catalogan en dos categorías:

- **Eventos (*Event*)**
  Por un lado, los eventos son aquellas intenciones que el sistema envía a la vista, donde el usuario verá un cambio por pantalla, por ejemplo, navegar a otra página.

- **Acciones (*Actions*)**
  Por otro lado, las acciones son intenciones que realiza el usuario sobre la interfaz, por ejemplo, pulsar un botón.

La representación gráfica del patrón MVI es:

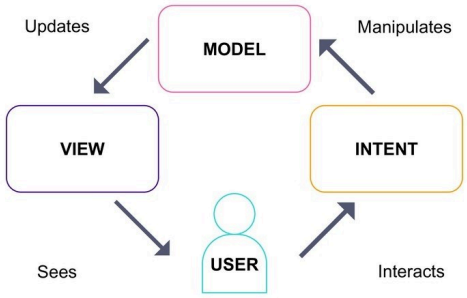

Figura 48: Patrón de presentación MVI,
https://www.linkedin.com/pulse/mastering-mvi-architecture-revolutionizing-android-app-dhule/

3.4. Listado de casos de uso

Los casos de uso proporcionan un listado detallado de todas las funcionalidades y características que ofrece la aplicación. En la definición se encuentran atributos importantes como el identificador único, la prioridad, los actores, las precondiciones y postcondiciones, entre otros.

Esta definición se usará más adelante para el desarrollo de cada funcionalidad en la propia aplicación, tomando como referencia los detalles indicados en cada descripción.

3.4.1 Diagrama de actores

En la aplicación solo se distinguen dos tipos de actores: usuario y usuario con elementos como favoritos. El segundo hereda todas las actividades que puede realizar el primero.

Las diferentes funcionalidades que puede realizar cada actor son las enumeradas a continuación:

- **Usuario**
  - Visualizar listado de precios
  - Visualizar en el mapa los precios
  - Buscar destinos
  - Cambiar combustible

- Marcar favorito
- Navegar a estación de servicio
- Modificar consumo medio
- Modificar capacidad del depósito
- Añadir descuento
- Eliminar descuento
- Puntuar estación de servicio
- Puntuar marca

- **Usuario con favoritos**
  - Desmarcar favorito
  - Activar todas las notificaciones
  - Desactivar todas las notificaciones
  - Activar notificación individual
  - Desactivar notificación individual

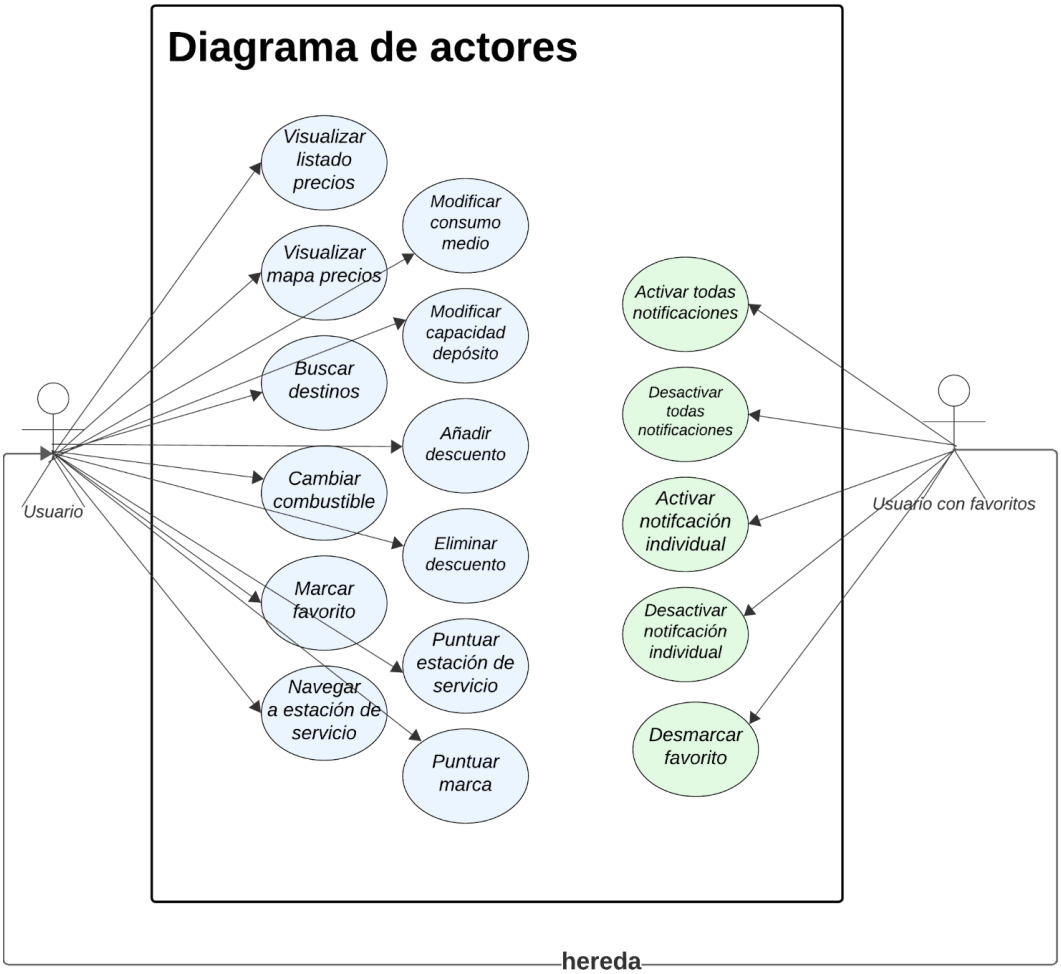

Figura 49: Diagrama de actores

3.4.2. Listado de casos de uso

Identificador	CU-001
Nombre	**Visualizar listado de precios**
Prioridad	Alta
Descripción	Visualizar el listado de precios ordenados de forma ascendente en base a la ubicación del usuario.
Actores	Usuario
Precondiciones	- Acceso a internet - Permiso de ubicación concedido
Iniciado por	Usuario
Flujo	1. Abrir aplicación 2. Ver pantalla de inicio 3. Ver pantalla principal 4. Ver listado de precios
Postcondiciones	- Se visualizan los precios a razón de la ubicación del dispositivo
Notas	Se utiliza el acceso a internet para obtener el listado de precios de cada estación de servicio y la ubicación para determinar cuáles mostrar.

Identificador	CU-002
Nombre	**Cambiar tipo de carburante**
Prioridad	Alta
Descripción	Mostrar elemento visual para cambiar la selección del tipo de carburante.
Actores	Usuario
Precondiciones	- Visualizar listado de precios
Iniciado por	Usuario
Flujo	1. Visualizar listado de precios 2. Pulsar sobre el elemento visual de cambiar tipo de carburante 3. Ver las opciones disponibles

	4. Pulsar sobre cualquier opción
Postcondiciones	- Observar listado de precios modificado en base al tipo de carburante seleccionado
Notas	No todas las estaciones de servicio disponen de todos los tipos de carburantes. Es posible que dada una combinación entre ubicación y tipo de carburante no se muestre ningún resultado.

Identificador	**CU-003**
Nombre	**Buscar lugar**
Prioridad	Media
Descripción	Utilizar el buscador para mostrar estaciones de servicio en base al término de búsqueda sobre ciudades, municipios y comunidades autónomas.
Actores	Usuario
Precondiciones	- Visualizar listado de precios
Iniciado por	Usuario
Flujo	1. Visualizar listado de precios 2. Pulsar icono de la lupa 3. Escribir un término de búsqueda 4. Pulsar sobre cualquier opción que se sugiere
Postcondiciones	- Observar listado de precios modificado en base al término de búsqueda - Visualizar elemento visual indicando el término buscado
Notas	No se mostrarán resultados para aquellos términos de búsqueda que no existan.

Identificador	**CU-004**
Nombre	**Visualizar detalles de una estación de servicio**
Prioridad	Media
Descripción	Al pulsar sobre una estación de servicio se muestran los detalles de la misma.
Actores	Usuario
Precondiciones	- Visualizar listado de precios

Iniciado por	Usuario
Flujo	1. Visualizar listado de precios 2. Pulsar sobre un elemento
Postcondiciones	- Ver los detalles de la estación de servicio
Notas	Los detalles de la estación de servicio se muestran en una nueva pantalla.

Identificador	**CU-005**
Nombre	**Marcar/desmarcar favorito en listado de precios**
Prioridad	Baja
Descripción	Al marcar sobre el icono favorito, la estación de servicio se mostraría en la parte superior, siendo el escenario opuesto al desmarcar el favorito.
Actores	Usuario
Precondiciones	- Visualizar listado de precios
Iniciado por	Usuario
Flujo	1. Visualizar listado de precios 2. Marcar/desmarcar favoritos
Postcondiciones	- Si se ha marcado: ver en la sección de favoritos el elemento marcado - Si se ha desmarcado. dejar de ver en la sección de favoritos
Notas	Los elementos favoritos también se muestran en el listado de precios.

Identificador	**CU-006**
Nombre	Navegar hacia estación de servicio en listado de precios
Prioridad	Alta
Descripción	Mostrar un mensaje informando al usuario si desea realizar la navegación hacia la estación elegida.
Actores	Usuario
Precondiciones	- Visualizar listado de precios - Tener el permiso de ubicación

Iniciado por	Usuario
Flujo	1. Visualizar listado de precios 2. Pulsar icono navegar en cualquier elemento 3. Ver diálogo de confirmación 4. Aceptar 5. Observar aplicaciones sobre mapas de terceros 2. a. Pedir permiso de ubicación
Postcondiciones	- Mostrar diálogo de navegación, tras aceptar, visualizar aplicaciones de terceros sobre mapas
Notas	Se utilizarán las coordenadas para navegar, en caso de no estar disponibles se usará la dirección postal.

Identificador	**CU-007**
Similar	***CU-005***
Nombre	**Marcar/desmarcar favorito en la pantalla detalles**
Prioridad	Baja
Descripción	Al marcar sobre el icono favorito, únicamente cambiará el estado del icono. Escenario idéntico al desmarcar.
Actores	Usuario
Precondiciones	- Visualizar detalles estación de servicio
Iniciado por	Usuario
Flujo	1. Ver pantalla de detalles 2. Marcar/desmarcar favoritos
Postcondiciones	- El icono favorito ha cambiado de estado
Notas	

Identificador	**CU-008**
Similar	***CU-006***
Nombre	**Navegar hacia estación de servicio en pantalla detalles**
Prioridad	Alta

Descripción	Mostrar un mensaje informando al usuario si desea realizar la navegación hacia la estación elegida.
Actores	Usuario
Precondiciones	- Ver pantalla detalles - Tener el permiso de ubicación
Iniciado por	Usuario
Flujo	1. Observar pantalla detalles 2. Pulsar icono navegar en cualquier elemento 3. Ver diálogo de confirmación 4. Aceptar 5. Observar aplicaciones sobre mapas de terceros 3. a. Pedir permiso de ubicación
Postcondiciones	- Mostrar diálogo de navegación, tras aceptar, visualizar aplicaciones de terceros sobre mapas
Notas	Se utilizarán las coordenadas para navegar, en caso de no estar disponibles se usará la dirección postal.

Identificador	**CU-009**
Nombre	**Valorar estación de servicio**
Prioridad	Baja
Descripción	Mostrar cinco estrellas para valorar la estación de servicio.
Actores	Usuario
Precondiciones	- Ver pantalla detalles - No haber realizado la valoración de estación de servicio
Iniciado por	Usuario
Flujo	1. Observar pantalla detalles 2. Seleccionar el número de estrellas en la valoración de estación de servicio
Postcondiciones	- El valor seleccionado debe visualizarse como estrellas rellenadas indicando que se ha valorado
Notas	Solo se permite valorar una única vez.

Identificador	**CU-010**

Similar	***CU-009***
Nombre	**Valorar marca de carburante**
Prioridad	Baja
Descripción	Mostrar cinco estrellas para valorar la marca del carburante.
Actores	Usuario
Precondiciones	- Ver pantalla detalles - No haber realizado la valoración de marca de carburante
Iniciado por	Usuario
Flujo	1. Observar pantalla detalles 2. Seleccionar el número de estrellas en la valoración de marca de carburante
Postcondiciones	- El valor seleccionado debe visualizarse como estrellas rellenadas indicando que se ha valorado
Notas	Solo se permite valorar una única vez.

Identificador	**CU-011**
Nombre	**Visualizar marcadores en el mapa**
Prioridad	Alta
Descripción	Mostrar en un mapa las estaciones de servicio utilizando marcadores. Cada marcador tendrá asociado el precio del carburante y el icono favorito.
Actores	Usuario
Precondiciones	- Visualizar pantalla mapa - Tener permiso de ubicación
Iniciado por	Usuario
Flujo	1. Ver pantalla mapa
Postcondiciones	- Observar los marcadores ubicados en el mapa con el precio y el icono de favoritos
Notas	No en todas las zonas geográficas hay estaciones de servicio.

Identificador	CU-012
Similar	***CU-005***
Nombre	**Marcar/desmarcar favorito en la pantalla del mapa**
Prioridad	Baja
Descripción	Al marcar sobre el icono favorito, únicamente cambiará el estado del icono. Escenario idéntico al desmarcar.
Actores	Usuario
Precondiciones	- Visualizar mapa - Visualizar estaciones de servicio
Iniciado por	Usuario
Flujo	1. Ver pantalla mapa 2. Marcar/desmarcar favoritos
Postcondiciones	- El icono favorito ha cambiado de estado
Notas	

Identificador	CU-013
Similar	**CU-006**
Nombre	**Navegar hacia estación de servicio en la pantalla del mapa**
Prioridad	Alta
Descripción	Mostrar un mensaje informando al usuario cuando pulsa el marcador si desea realizar la navegación hacia la estación elegida.
Actores	Usuario
Precondiciones	- Ver pantalla mapa - Ver estaciones de servicio - Tener el permiso de ubicación
Iniciado por	Usuario
Flujo	1. Observar pantalla mapa 2. Pulsar cualquier marcador de las estaciones de servicio en el mapa 3. Ver diálogo de confirmación

	4. Aceptar 5. Observar aplicaciones sobre mapas de terceros 4. a. Pedir permiso de ubicación
Postcondiciones	- Mostrar diálogo de navegación, tras aceptar, visualizar aplicaciones de terceros sobre mapas
Notas	Se utilizarán las coordenadas para navegar, en caso de no estar disponibles se usará la dirección postal.

Identificador	**CU-014**
Nombre	**Disminuir/Aumentar precio de la alerta**
Prioridad	Baja
Descripción	Los botones aledaños al precio de alerta disminuyen y aumentan el precio de la alerta.
Actores	Usuario
Precondiciones	- Ver pantalla de las alertas
Iniciado por	Usuario
Flujo	1. Visualizar pantalla de las alertas 2. Pulsar botón disminuir o aumentar
Postcondiciones	- Observar si el precio ha disminuido o aumentado.
Notas	El precio disminuye o aumenta en 5 céntimos.

Identificador	**CU-015**
Nombre	**Activar/Desactivar todas las alertas**
Prioridad	Baja
Descripción	El botón con el icono de la campana ubicado en la parte superior derecha activa o desactiva todas las alertas.
Actores	Usuario
Precondiciones	- Visualizar la pantalla de alertas - Tener estaciones de servicio marcadas como favoritas - Tener permiso para mostrar notificaciones - Tener acceso a internet

Iniciado por	Usuario
Flujo	1. Ver las pantalla de las alertas 2. Pulsar el icono de la campana ubicado en la parte superior derecha 2. a. Solicitar permiso para notificaciones
Postcondiciones	- Todas las notificaciones se han activado si la campana está en modo activo (rellena) o se desactivan si es el caso opuesto.
Notas	Las alertas individuales automáticamente se activan o desactivan en función de la campana ubicada en la parte superior, es decir, si se activan todas las alertas entonces aparecerán todas de forma individual activadas, y viceversa.

Identificador	**CU-016**
Nombre	**Activar/Desactivar alertas de forma individual**
Prioridad	Baja
Descripción	El botón con el icono de la campana ubicado en la parte derecha de cada elemento favorito activa o desactiva la alerta de forma individual.
Actores	Usuario
Precondiciones	- Visualizar la pantalla de alertas - Tener estaciones de servicio marcadas como favoritas - Tener permiso para mostrar notificaciones
Iniciado por	Usuario
Flujo	1. Ver las pantalla de las alertas 2. Pulsar el icono de la campana ubicado en la parte superior derecha 2. a. Solicitar permiso para notificaciones
Postcondiciones	- La alerta individual se ha activado si la campana está en modo activo (rellena) o se ha desactivado si es el caso opuesto.
Notas	

Identificador	**CU-017**
Similar	***CU-005***
Nombre	**Desmarcar favorito en la pantalla de alertas**

Prioridad	Baja
Descripción	Al marcar sobre el icono favorito, la alerta desaparecerá automáticamente.
Actores	Usuario
Precondiciones	- Visualizar la pantalla de alertas - Visualizar elementos favoritos
Iniciado por	Usuario
Flujo	1. Ver la pantalla de alertas 2. Desmarcar favoritos
Postcondiciones	- El elemento favorito desaparece del listado
Notas	

Identificador	**CU-018**
Similar	***CU-006***
Nombre	**Navegar hacia estación de servicio en la pantalla de alertas**
Prioridad	Alta
Descripción	Mostrar un mensaje informando al usuario cuando pulsa el marcador si desea realizar la navegación hacia la estación elegida.
Actores	Usuario
Precondiciones	- Ver pantalla de alertas - Ver elementos favoritos - Tener el permiso de ubicación
Iniciado por	Usuario
Flujo	1. Observar la pantalla de alertas 2. Pulsar el icono de navegar en cualquier estación de servicio 3. Ver diálogo de confirmación 4. Aceptar 5. Observar aplicaciones sobre mapas de terceros 2. a. Pedir permiso de ubicación
Postcondiciones	- Mostrar diálogo de navegación, tras aceptar, visualizar aplicaciones de terceros sobre mapas
Notas	Se utilizarán las coordenadas para navegar, en caso de no estar disponibles se usará la dirección postal.

Identificador	CU-019
Nombre	**Modificar consumo medio**
Prioridad	Baja
Descripción	En una caja de texto el usuario puede escribir el consumo medio de su vehículo.
Actores	Usuario
Precondiciones	- Visualizar la pantalla vehículo
Iniciado por	Usuario
Flujo	1. Ver la pantalla vehículo 2. Modificar valor
Postcondiciones	- Observar el valor escrito en la caja de texto
Notas	Se utilizará la coma como separador decimal y se limita el campo de texto a 2 números enteros y 2 decimales.

Identificador	CU-020
Nombre	**Modificar litros del depósito**
Prioridad	Baja
Descripción	En una caja de texto el usuario puede escribir la capacidad total de su depósito de combustible.
Actores	Usuario
Precondiciones	- Visualizar la pantalla vehículo
Iniciado por	Usuario
Flujo	1. Ver la pantalla vehículo 2. Modificar valor
Postcondiciones	- Observar el valor escrito en la caja de texto
Notas	Solo se pueden escribir números enteros y como máximo se permiten 3 cifras.

Identificador	CU-021
Nombre	**Añadir descuento**
Prioridad	Alta
Descripción	Al pulsar el desplegable de descuentos se mostrarán los disponibles y al seleccionar uno se mostrará en pantalla.
Actores	Usuario
Precondiciones	- Visualizar la pantalla vehículo - Tener acceso a internet
Iniciado por	Usuario
Flujo	1. Ver la pantalla vehículo 2. Pulsar añadir descuentos 3. Seleccionar un descuento
Postcondiciones	- Observar el descuento elegido en pantalla - El precio que se muestra en cualquier pantalla ha sido modificado en base al descuento
Notas	

Identificador	CU-022
Nombre	**Eliminar descuento**
Prioridad	Alta
Descripción	Al pulsar el icono papelera al lado del descuento, éste se eliminará y dejará de aplicarse en todas aquellas pantallas que muestran precios.
Actores	Usuario
Precondiciones	- Visualizar la pantalla vehículo - Tener al menos un descuento aplicado
Iniciado por	Usuario
Flujo	1. Ver la pantalla vehículo 2. Pulsar eliminar descuento
Postcondiciones	- El descuento eliminado no se visualiza en la pantalla - El precio afectado por el descuento se ha restablecido
Notas	

Identificador	CU-023
Nombre	**Enviar notificación**
Prioridad	Baja
Descripción	Enviar una notificación *push* al dispositivo del usuario cuando se ha detectado que el precio del carburante seleccionado ha descendido a un precio igual o menor.
Actores	Usuario
Precondiciones	- Tener alertas activas
Iniciado por	Servidor
Flujo	1. Obtener listado de precios 2. Obtener listado de alertas 3. Comprobar alertas con listado de precios 4. Enviar notificación si se ha detectado un precio igual o menor
Postcondiciones	- Observar una notificación *push* en el dispositivo
Notas	Para el envío de notificaciones se utilizará el sistema *Firebase Cloud Messaging* [42].

4. Implementación

Después de diseñar el prototipo de alta fidelidad, definir los casos de uso y determinar la estructura que utilizará la aplicación, es necesario especificar qué herramientas se utilizarán para codificar los requisitos de usuario en una aplicación de móvil.

Para ello, se seguirán las buenas prácticas recomendadas en el desarrollo de *software*, haciendo énfasis en los principios *SOLID* [43] y Código Limpio (también conocido como *Clean Code*) [44]. Estos principios permitirán crear un código sólido, reusable y mantenible en el tiempo.

En consonancia con lo mencionado previamente, se utilizarán las bibliotecas más utilizadas en el desarrollo de aplicaciones *Android*, siendo las más conocidas las enumeradas a continuación: *Retrofit*, *OkHttp*, *Room*, *Hilt* y *JUnit*, entre otras.

Además, los lenguajes de programación que se emplearán a lo largo del desarrollo serán *Kotlin*, para el desarrollo de la aplicación; y *Python*, para la programación de las notificaciones en la herramienta *Cloud Functions*.

Del mismo modo, la versión mínima establecida para el desarrollo y ejecución de la aplicación es *API* 26, coincidiendo con Android 8.0. Se toma esta decisión teniendo en cuenta que es una

versión lanzada en 2017, y aproximadamente cubriría el 95,1% de los dispositivos disponibles [45].

4.1. Buenas prácticas

Como se ha mencionado anteriormente, se seguirán las recomendaciones de buenas prácticas a la hora de codificar el *software* del proyecto. Como base de estas buenas prácticas, se aplicarán los principios *SOLID* y la práctica *Clean Code*.

4.1.1. Principios SOLID

Estos principios son la base de la programación. Se introdujeron por primera vez por Robert C. Martin a principios de la década del 2000 [46]. Promueven la codificación de *software* de calidad aplicando cinco principios básicos que todo desarrollador debe conocer. Sus siglas representan cada principio, y son los siguientes:

- ***S**ingle responsability principle* (Principio de responsabilidad única)

  Este principio dictamina que un objeto únicamente debe tener una razón para cambiar, lo que se traduce a una única responsabilidad en todo el *software*.

- ***O**pen/closed principle* (Principio de abierto/cerrado)

  Se refiere al comportamiento que tienen las clases, métodos, variables, etc. Establece que cada elemento de *software* debe estar abierto a extensión, pero cerrado a modificación.

- ***L**iskov substitution principle* (Principio de sustitución de Liskov)

  Establece la posibilidad de reemplazar instancias que extiendan de la misma clase padre sin modificar el comportamiento del programa.

- ***I**nterface segregation principle* (Principio de segregación de la interfaz)

  Especifica la separación de funciones en varias interfaces, evitando de esta forma la utilización de interfaces en clases donde no se utilizan todos los métodos que incorpora, consiguiendo así la máxima dependencia entre clases e interfaces.

- ***D**ependency inversion principle* (Principio de inversión de dependencias)

  Este principio hace referencia a la dependencia existente de las clases de alto con las de bajo nivel. Para solucionar este problema se debe utilizar abstracciones, consiguiendo de esta forma que las clases de alto nivel no dependan de las de bajo nivel, además de evitar la dependencia de los detalles, ya que dependen de abstracciones.

4.1.2. Clean Code

Este concepto, al igual que el anterior, fue introducido por Robert C. Martin en su libro *Clean Code* [47]. En él habla de las técnicas recomendadas que todo desarrollador de *software*

independientemente del lenguaje de programación que utilice, debe realizar para que el código sea legible y mantenible a lo largo del tiempo, incluso por otros desarrolladores.

Algunas prácticas que recomienda son las enumeradas a continuación:

- **Simplicidad**
  La codificación del *software* debe ser simple, evitando la complejidad innecesaria, haciendo uso de otro principio llamado *KISS* [48].

- **Pruebas unitarias**
  El código debe estar escrito de forma aislada al resto de componentes, consiguiendo así la independencia de cada elemento. Realizando esto, el *software* tiene la facilidad de realizar pruebas unitarias de forma sencilla y eficiente.

- **Mantenibilidad**
  Las clases y las funciones deben ser fácilmente mantenibles en el futuro. Para lograrlo se debe realizar buenas prácticas de programación desde el inicio del proyecto para evitar el sobreesfuerzo cuando llegue la ocasión de modificar el código.

- **Claridad y legibilidad**
  Cualquier desarrollador ajeno al proyecto debe entender qué es lo que está leyendo. Esto se consigue escribiendo un código claro y legible. Una forma de conseguirlo es dar sentido a las clases, funciones y variables del proyecto, evitando nombres acortados o sin ningún sentido como por ejemplo *getCD()* en lugar de *getCarDetails()*.

- **Regla *boy scout***
  Esta regla especifica la necesidad de dejar el código igual o mejor de cómo se lo ha encontrado el desarrollador que lo ha modificado. Con esto no se evita que *software* no tenga la calidad suficiente, pero al menos sí evita aumentar la deuda técnica del proyecto.

- **Tratamiento de errores**
  Durante la codificación del proyecto hay que prestar mucha atención a los posibles errores que el *software* puede generar durante su ejecución. Controlarlos a tiempo es vital para el correcto funcionamiento del programa.

- **Evitar duplicar código**
  Una práctica muy recomendable es evitar duplicar código, también conocido por el principio *DRY* (conocido por *Don't Repeat Yourself*, No te repitas) especifica la necesidad de no repetir código en aquellas situaciones donde se puede llegar a evitar. Con esto se consigue no tener dos o más fragmentos de código con el mismo contenido, ayudando de esta manera a reducir el tiempo cuando llegue el momento de modificarlo.

[49]

4.2. Herramientas de trabajo

Para la correcta realización del proyecto se utilizarán principalmente tres herramientas clave que ayudarán de forma eficiente y ordenada la realización de las tareas de codificación del proyecto. Las herramientas en cuestión son las siguientes:

- ***Android Studio***
  Es el IDE por excelencia para el desarrollo de código nativo para el sistema operativo Android. Contiene diversas herramientas que permiten realizar una codificación más simple y eficiente respecto otras alternativas. Del mismo modo, cuenta con un sistema de depuración muy robusto y un visualizador de trazas muy completo [50].
- ***IntelliJ IDEA Ultimate***
  Con interfaz de usuario similar a Android Studio, pues ambos son desarrollados principalmente por *JetBrains*, es un *IDE* utilizado para diversos lenguajes de programación [51]. Para este proyecto se utilizará para codificar las notificaciones. Alternativamente se puede utilizar *PyCharm* [52].
- ***DB Browser for SQLite***
  Es un gestor visual de base de datos. Cuenta con una interfaz gráfica básica y simple, pero suficiente para el cometido que se desea realizar [53].

4.3. Bibliotecas

Las bibliotecas más relevantes que utilizará el proyecto son las descritas a continuación:

- ***Android SDK***
  Es el conjunto de herramientas utilizadas para el desarrollo de aplicaciones nativas en el sistema operativo *Android*. Cuenta con todas las bibliotecas necesarias para codificar una aplicación, además de tener disponible las *APIs* que acceden a las diferentes partes del sistema operativo [54].

- ***Jetpack Compose***
  Es la biblioteca que Google recomienda utilizar actualmente para el desarrollo de interfaces gráficas. Cada elemento visual es escrito en *Kotlin*, permitiendo desarrollar componentes gráficos más rápidamente y con mayor sencillez [55].

- ***Google Maps SDK***
  Conjunto de herramientas para renderizar por pantalla un mapa. Además, permite agregar puntos geolocalizados para aportar más información al usuario [56].

- ***Hilt***
  Se trata de una biblioteca para inyectar las dependencias de las clases del proyecto. Su sencillez *radica* con el objetivo que fue creado: reducir la curva de aprendizaje al utilizar *Dagger*. *Hilt* proporciona una capa simple y suficiente para la inyección de dependencias [57].

- ***OkHttp***
  Permite realizar llamadas HTTP en *Android*, además de manipular eficientemente las respuestas [58].

- ***Retrofit***
  Ligada a la anterior biblioteca, *Retrofit* es un cliente HTTP que permite definir las diferentes interfaces para consumir un servicio *REST*, utilizando como biblioteca para manejar las llamadas *OkHttp* [59].

- ***Gson***
  Permite la serialización y deserialización de los objetos en formato *JSON*. Para el caso de este proyecto se utilizará para los datos que se envían a través de una llamada HTTP, así como los que se reciben [60].

- ***Firebase***
  Es una herramienta en la nube que ofrece varias bibliotecas para utilizar los servicios que ofrecen. En el caso del proyecto se hará uso de *Cloud Firestore* para almacenar y leer datos alojados en la nube, y *FCM* para la generación del identificador único por usuario y dispositivo, además de la recepción de notificaciones [41] [42].

- ***Robolectric***
  Es un *framework* para la ejecución de código bajo la *JVM*. Esto permite realizar pruebas sobre aspectos visuales sin necesidad de realizarlas en un dispositivo real, reduciendo considerablemente el tiempo de ejecución de este tipo de pruebas [61].

- ***JUnit4***
  Es utilizado para realizar pruebas de *software*. En este caso, *Robolectric* solo funciona correctamente con la versión 4 de *JUnit* [62].

- ***JUnit5***
  Permite ejecutar pruebas de código utilizando *tests* parametrizados, consiguiendo de esta manera reducir el código escrito y aumentando la cobertura de código de una manera sencilla [63].

4.4. Organización del proyecto

La organización del proyecto es la tarea inicial del proyecto. Consiste en determinar qué tipo de organización se va a seguir para estructurar cada clase dentro del proyecto, garantizando de esta manera la legibilidad de cada fichero, así como la rápida localización de un archivo en concreto.

Como nomenclatura para las carpetas se utilizará *lowercase* (todas las letras en minúsculas, sin espacios, guiones o cualquier símbolo de separación). Utilizar otro tipo de nomenclatura no modificaría el comportamiento del proyecto, simplemente es estético, aunque recomienda su uso [64].

En este caso, se va a utilizar tres tipos de de organización, enumerados desde más alto a más bajo nivel a continuación:

- **Organización por capas**
  El primer nivel de la organización separa el código en las diferentes capas que forman la arquitectura elegida. Al poseer tres capas, las carpetas necesarias son: *data* (para el acceso a datos, ya sea local o remoto), *domain* (lógica de negocio) y *presentation* (representación de los datos por pantalla).

- **Organización por funcionalidades**
  El segundo nivel consta de las funcionalidades que posee el proyecto. Pueden ser acciones clave que el usuario puede utilizar en cualquier momento o un conjunto de ellas donde es recomendable que estén agrupadas en una misma carpeta.

- **Organización por tipología**
  El tercer nivel consiste en organizar los archivos por la tipología de los mismos. Por ejemplo, todas los casos de uso serán agrupados en una carpeta llamada *usecases*.

La representación visual de la organización descrita sería la siguiente:

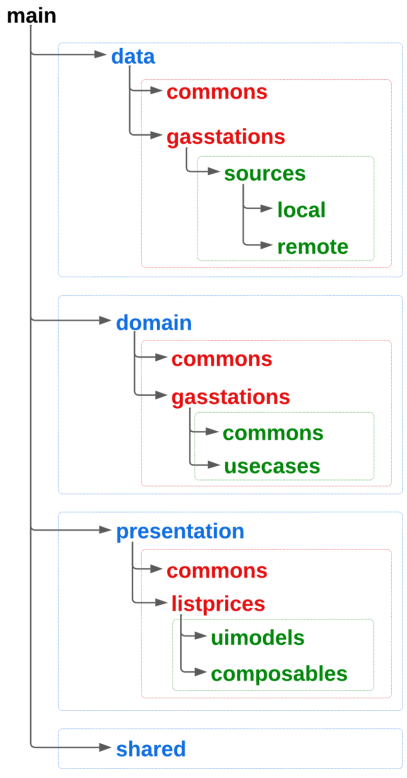

Figura 50: Organización carpetas

En color azul es representado el primer nivel, la organización por capas; en segundo nivel está el color rojo representando la organización por funcionalidades; y finalmente, el color verde hace referencia a la organización por tipología.

En todos los niveles se encuentra, o bien la carpeta *shared*, o la carpeta *commons*. Esta carpeta agrupa todas las clases que son comunes para dos o más carpetas del mismo nivel. Por ejemplo, en la carpeta *shared* se encuentra el archivo *Application*, siendo común para todos las capas.

4.5. Control de versiones

Es importante establecer una buena rutina para generar versiones del código fuente que se está desarrollando. Para esta ocasión, *Git* ofrece la posibilidad de gestionar de manera eficiente y sencilla el *software* del proyecto a través del sistema de control de versiones distribuido que ofrece [65].

Ligado a esto, se establece como modelo de trabajo *GitFlow*. Consiste en establecer dos ramas de trabajo, siendo *main* la que contendrá el código fuente que está en producción, y *develop* la rama base para la realización de nuevos desarrollos.

Del mismo modo, determina la utilización de las ramas *feature* para nuevas funcionalidades, *release* para el lanzamiento de cada versión de la aplicación, y finalmente, *hotfix* y *bugfix*, siendo el primero utilizado para la corrección de errores en producción, y el segundo en desarrollo.

A continuación, se ilustra gráficamente lo mencionado:

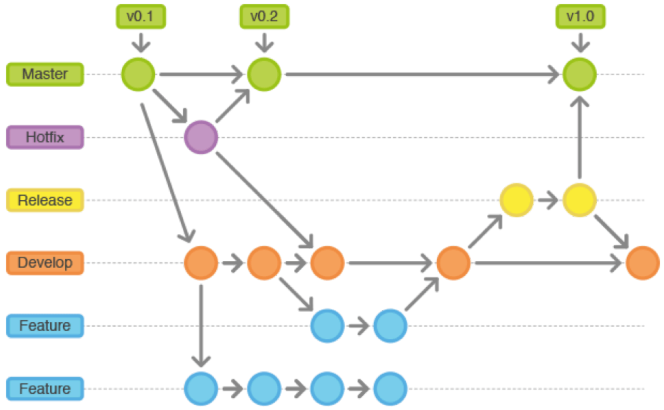

Figura 51: *GitFlow*, https://castor.com.co/integracion-continua-de-software-git-y-git-flow/

4.6. Codificación

4.6.1. Inyección de dependencias

Con la biblioteca *Hilt* se realiza la gestión e inyección de las dependencias de cada instancia del proyecto. Esta inyección permite realizar pruebas unitarias de código usando clases imitadas o simuladas, conocidas técnicamente como *mocks*.

Actualmente *Hilt* guarda relación con la biblioteca *Dagger2*. Esto se debe porque *Hilt* ha sido desarrollado sobre la propia *Dagger2*, creando de esta manera una *developer experience* más

óptima para aquellos desarrolladores que desean utilizar inyección de dependencias en sus proyectos.

Además, de igual manera que *Dagger2*, la inyección de dependencias se generan durante la compilación del código, logrando de esta manera el descubrimiento de un error antes de la ejecución de la aplicación.

Por lo tanto, *Hilt* se define como una biblioteca simplificada para inyectar dependencias en un proyecto de Android, consiguiendo una curva de aprendizaje muy poco pronunciada en comparación con *Dagger2*, logrando construir un proyecto de *software* más adaptable al cambio y con mayor facilidad a realizar pruebas unitarias.

La simplicidad de la biblioteca *Hilt* logra construir en las clases generadas durante la compilación las clases necesarias para la inyección de dependencias. Estas clases son generadas a través de las indicaciones que se proporcionan de las anotaciones. A continuación, se muestran varios ejemplos donde se ilustra el uso de las anotaciones resaltando las líneas afectadas.

Clase *Application*

```
1 @HiltAndroidApp
2 class FuelPriceApplication : Application() {
3
4     [ ... ]
5
6     private fun initPlaces() {
7         Places.initialize(this, BuildConfig.PLACES_API_KEY)
8     }
9 }
```

Figura 52: Inyección de la clase *Application*

Actividades

```
@AndroidEntryPoint
class HomeActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContent {
            FuelPriceTheme {
                HomeNavigation()
            }
        }
    }
}
```

Figura 53: Inyección de dependencias en actividades

View model inyectados durante la composición

```kotlin
@Composable
fun ListPricesScreen(
    showFullSearch: Boolean,
    onDismissSearchDialog: () → Unit,
    modifier: Modifier = Modifier,
    viewModel: ListPricesViewModel = hiltViewModel()
) {
```

Figura 54: Inyección de dependencias de *view models* durante la composición

Inyección de clases a través del constructor de la clase

```kotlin
class SearchPlacesUseCase @Inject constructor(
    private val repository: PlacesRepository,
) {
```

```kotlin
class PlacesRepositoryImpl @Inject constructor(
    private val remoteSource: PlacesRemoteSource,
) : PlacesRepository {
```

```kotlin
class PlacesRemoteSourceImpl @Inject constructor(
    private val api: PlacesClient,
) : PlacesRemoteSource {
```

Figura 55: Inyección de dependencias de clases

Provisión de clases que no se generan automáticamente en la compilación

```kotlin
@Module
@InstallIn(SingletonComponent::class)
object PreferencesManagerModule {

    @Provides
    fun providesPreferencesManager(
        @ApplicationContext context: Context,
    ): PreferencesManager = PreferencesManagerImpl(
        context = context
    )
}
```

```kotlin
@Module
@InstallIn(SingletonComponent::class)
object NetworkModule {

    @Provides
    @Singleton
    fun providesRetrofit(
        okHttpClient: OkHttpClient,
        gson: Gson,
    ): Retrofit {
        val baseUrl = "https://[ ... ]/PreciosCarburantes/"
        return Retrofit.Builder()
            .baseUrl(baseUrl)
            .addConverterFactory((GsonConverterFactory.create(gson)))
            .client(okHttpClient)
            .build()
    }
}
```

Figura 56: Provisión de clases a través de módulos

4.6.2. Presentación

En la capa de presentación se utiliza la biblioteca *Jetpack Compose* como representación de la interfaz gráfica. Además, se utiliza programación declarativa y el lenguaje de programación utilizado es *Kotlin*. Esto permite realizar interfaces de forma rápida y sencilla en comparación con la técnica tradicional de *XML*.

4.6.2.1. Estados

Los estados, denominados por el término inglés *states*, son composiciones inmutables de la representación gráfica de los objetos. Esto significa que el renderizado de los objetos que se representan por pantalla se mantienen presentes hasta que haya un cambio externo que los modifique.

En este sentido, a nivel general todas las pantallas poseen tres estados:

- ***Loading***
  Este estado se muestra al inicio de cualquier pantalla. También cuando los datos que se representan por pantalla necesitan ser procesados durante un tiempo más largo del habitual.

- ***Success***
  Una vez que los datos que se van a representar por pantalla han sido procesados de forma exitosa, se utiliza este estado.

- ***Error***
  En este estado se reflejan tanto los errores generados durante el procesamiento de los datos como los producidos a través de las interacciones del usuario sobre la interfaz gráfica.

Como se utiliza el patrón de representación *MVI*, cada estado es emitido de forma reactiva, por lo tanto, los diferentes objetos emitidos desde el *view model* son recolectados por la interfaz, como se muestra a continuación.

```
viewModel.uiStateFlow.collectAsState().value.let { state →
    when (state) {
        UiState.Loading → LoadingComposable()
        is UiState.Success → {
            ListPricesSuccessComposable(
                uiModel = state.data,
                onLoadMore = {
                    viewModel.submitAction(ListPricesUiAction.OnLoadMore(it))
                },
                onClickFavorite = {
                    viewModel.submitAction(ListPricesUiAction.OnClickFavorite(it))
                },
                onClickFuelType = {
                    viewModel.submitAction(ListPricesUiAction.OnClickFuelType(it))
                },
                onClickRadio = {
                    viewModel.submitAction(ListPricesUiAction.OnClickRadio(it))
                },
            )
        }
        is UiState.Error → ErrorCompossable(state.errorMessage)
    }
}
```

Figura 57: Estados en *Jetpack Compose*

4.6.2.2. Navegación

La biblioteca *Navigation Compose* se utiliza para realizar la navegación entre las distintas pantallas ubicadas bajo el contexto de una *activity*. Dicha navegación es gestionada a través de rutas, donde también se puede utilizar parámetros de tipo primitivo para pasar datos entre

pantallas. Esta organización basada en rutas permite la escalabilidad de la aplicación en función de las nuevas pantallas que se vayan necesitando.

A continuación, se ilustra con dos fragmentos de código la navegación que realiza la aplicación.

```
sealed class HomeScreens(
    val route: String,
    @StringRes val titleId: Int,
    @DrawableRes val iconId: Int,
) {
    data object List : HomeScreens(
        "list",
        R.string.home_list_prices_title,
        R.drawable.ic_list
    )
    data object Map : HomeScreens(
        "map",
        R.string.home_map_title,
        R.drawable.ic_map
    )
    data object Alerts : HomeScreens(
        "alerts",
        R.string.home_alerts_title,
        R.drawable.ic_notificacion_add
    )
    data object VehicleDiscounts : HomeScreens(
        "vehicle_discounts",
        R.string.home_vehicle_discount_title,
        R.drawable.ic_car)
}
```

Figura 58: Código HomeScreens

```
NavHost(
    navController = navController,
    startDestination = HomeScreens.List.route,
    modifier = Modifier.padding(innerPadding)
) {
    composable(HomeScreens.List.route) {
        ListPricesScreen(
            showFullSearch = showFullSearch,
            onDismissSearchDialog = { showFullSearch = false }
        )
    }
    composable(HomeScreens.Map.route) { MapScreen() }
    composable(HomeScreens.Alerts.route) { AlertScreen() }
    composable(HomeScreens.VehicleDiscounts.route) { VehicleDiscountsScreen() }
}
```

Figura 59: Código navegación entre pantallas

4.6.2.3. Geolocalización

La geolocalización es necesaria para mostrar la distancia entre el usuario y las diferentes estaciones de servicio. Para obtener la geoposición del usuario se debe solicitar permiso de ubicación, y una vez haya sido aceptado por el usuario, se utiliza la propia *API* que proporciona el *SDK* de Android para obtener las coordenadas actuales.

En *Jetpack Compose* se solicita el permiso de ubicación utilizando *LaunchedEffect*, donde se lanza un efecto sobre la composición actual, en este caso el permiso de ubicación.

```kotlin
@Composable
fun RequestLocationPermission(
    onPermissionsGranted: () -> Unit,
    onPermissionsNotGranted: () -> Unit,
) {
    val permissionsState = rememberMultiplePermissionsState(
        permissions = listOf(
            Manifest.permission.ACCESS_COARSE_LOCATION,
            Manifest.permission.ACCESS_FINE_LOCATION
        )
    )

    val coroutineScope = rememberCoroutineScope()

    LaunchedEffect(permissionsState.allPermissionsGranted) {
        if (permissionsState.allPermissionsGranted) {
            onPermissionsGranted()
        } else {
            coroutineScope.launch {
                permissionsState.launchMultiplePermissionRequest()
                onPermissionsNotGranted()
            }
        }
    }
}
```

Figura 60: Código compactado del permiso de ubicación

Cuando el usuario acepta el permiso de ubicación, cambia el valor que previamente se ha recordado utilizando la función *remember*. Esto permite que cualquier función que dependa de un valor *remember* pueda ser ejecutada al instante cuando ese valor cambie su estado.

```kotlin
var permissionsGranted by remember { mutableStateOf(false) }

RequestLocationPermission(
    onPermissionsGranted = {
        permissionsGranted = true
    },
    onPermissionsNotGranted = {
        permissionsGranted = false
    },
)
```

Figura 61: Código reducido del permiso de ubicación

Y finalmente, cuando el código detecta que el valor *remember* asociado a la aceptación del permiso de ubicación cambia al estado verdadero, se ejecuta la función utilizada para obtener la geoposición del usuario.

```
if (permissionsGranted) {
    GetLocation(
        fusedLocationClient = fusedLocationClient,
        onLocation = {
            locationState = it
            viewModel.submitAction(ListPricesUiAction.OnLocation(it))
        }
    )
}
```

Figura 62: Código obtener ubicación del usuario

4.6.3. Dominio

En la capa de dominio se ubican los casos de uso que se comunican con los *view models* ubicados en la capa aledaña, a su vez reciben una instancia de la clase *repository* inyectada a través de inyección de dependencias.

4.6.3.1. Casos de uso

Conocidos con el término inglés *use cases*, se tratan de acciones que el programa realiza desde la capa de presentación hacia la capa de dominio, ya sea intencionadas por el usuario o por el sistema. En el proyecto actual, son las clases que se comunican con las interfaces *repository*, y son llamadas desde las clases *view models*.

Los términos que se utilizan para el método de invocación son:

- ***operator***
  Se usa para sobreescribir el funcionamiento de ciertas operaciones, en este caso, el uso de *invoke* [66].

- ***invoke***
  Permite llamar a las instancias de las clases que contengan este término como si fueran funciones. De esta manera, los casos de uso quedan más legibles en el código, pasando de *getDetails.execute()* a *getDetails()* [67].

- ***suspend***
  Indica la suspensión del fragmento de código afectado, ejecutándose en un hilo separado al principal, sin bloquearlo [68].

Por esta razón, la utilización de los dos primeros términos permiten que los casos de uso cumplan, en cierta manera, el Principio de Responsabilidad Única (*Single Responsibility Principle*), ya que solo se podrá tener una función en cada clase con estas características. A continuación, se ilustra gráficamente lo anteriormente comentado.

```
class GetLocationFromPlaceIdUseCase @Inject constructor(
    private val repository: PlacesRepository,
) {

    suspend operator fun invoke(
        placeId: String
    ): Result<LocationDomainModel> {
        return repository.getLocationByPlaceId(placeId = placeId)
    }
}
```

Figura 63: Código caso de uso

```
    private fun getLocationFromPlaceId(place: PlaceDomainModel) {
        launchIO {
            getLocationFromPlaceIdUseCase(place.id)
                .onSuccess {
                    location = it
                    loadGasStations()
                }
                .onFailure { }
        }
    }
```

Figura 64: Código invocación caso de uso en el *view model*

4.6.3.2. Repositorios

En la capa de dominio se encuentran las interfaces de los repositorios. Esto quiere decir que dicha capa no conocerá los detalles de los repositorios, ya que éstos serán encontrados en la capa adyacente, datos. Un ejemplo de lo hablado es el siguiente fragmento de código.

```
interface PlacesRepository {

    suspend fun searchPlaces(
        query: String
    ): Result<List<PlaceDomainModel>>

    suspend fun getLocationByPlaceId(
        placeId: String
    ): Result<LocationDomainModel>
}
```

Figura 65: Código interfaz *repository*

4.6.4. Datos

La capa de datos ubica los diferentes acceso a las fuentes de recursos, como puede ser una llamada a un servidor remoto, guardar datos en una base de datos, o simplemente, almacenarlos en memoria a través de un objeto o un listado de ellos.

4.6.4.1. Implementación del repositorio

En la capa de dominio, adyacente a la de datos, se encuentran las interfaces de los repositorios. En la capa de datos se codifican los detalles de estas interfaces, abstrayendo de esta forma la fuente de datos de la lógica de negocio.

El siguiente fragmento de código representa gráficamente el detalle de la implementación de un repositorio.

```
class DiscountsRepositoryImpl @Inject constructor(
    private val dbSource: DiscountsDbSource,
) : DiscountsRepository {

  override suspend fun getDiscounts(): Result<List<DiscountDomainModel>> =
        runCatching {
            dbSource.getDiscounts().mapNotNull { it.toDomain() }
        }

  override suspend fun updateDiscount(
      discountId: Int,
      isActive: Boolean,
  ) {
      dbSource.updateDiscount(discountId, isActive)
  }
}
```

Figura 66: Código implementación *repository*

4.6.4.2. Fuente de datos

Las diferentes fuentes de datos que se presentan en la aplicación son:

- ***API***
  Se trata de una interfaz para comunicarse entre dos sistemas, en este caso la aplicación con el servidor remoto que proporciona las estaciones de servicio junto con sus detalles.

- **Base de datos**
  Se utiliza para almacenar las estaciones de servicio que previamente se han recibido a través de una llamada *API*. Después de almacenarlos permitirán su rápido acceso sin necesidad de realizar nuevamente una llamada *API*.

- ***Firestore***
  Es el almacenamiento basado en colecciones almacenado en los servidores de *Firebase*. Se utiliza una biblioteca para realizar las llamadas de lectura y escritura.

- ***SharedPreferences***
  Es un almacenamiento clave-valor en un fichero privado en la aplicación. Esto quiere decir que no se podrá almacenar datos complejos, o de hacerlo tendrán que ser transformados a una cadena de texto. Es utilizado principalmente en las configuraciones de la aplicación.

Las diferentes fuentes de datos son ilustrados a continuación.

```
class GasStationRemoteSourceImpl @Inject constructor(
    private val api: GasStationApi,
) : GasStationRemoteSource {

    override suspend fun getStations(): Result<List<GasStationResponse>> {
        return api
            .getGasStations()
            .handleApiResponse()
            .mapCatching { it.gasStations }
    }
}
```

Figura 67: Código fuente de datos remoto

```
class DiscountsDbSourceImpl(
    private val dao: DiscountDao,
) : DiscountsDbSource {

    override fun getDiscounts(): List<DiscountsWithBrandDbModel> {
        return dao.getDiscounts()
    }

    override fun updateDiscount(
        discountId: Int,
        isActive: Boolean
    ) {
        dao.update(discountId, isActive)
    }
}
```

Figura 68: Código fuente de datos a través de base de datos

```kotlin
class RatingRemoteSourceImpl @Inject constructor(
    private val firestore: FirebaseFirestore,
) : RatingRemoteSource {

    override suspend fun getRatings(): Result<List<RatingResponse>> {
        return withContext(Dispatchers.IO) {
            try {
                val snapshot = firestore.collection(
                    RatingResponse.KEY_RATINGS_DOCUMENT
                ).get().await()
                val ratings = snapshot.documents.mapNotNull {
                    it.fromFirestoreToResponse()
                }
                Result.success(ratings)
            } catch (exception: Exception) {
                Result.failure(exception)
            }
        }
    }
}
```

Figura 69: Código fuente de datos a través de *Firestore*

```kotlin
class VehicleDetailsLocalSourceImpl @Inject constructor(
    private val preferencesManager: PreferencesManager,
) : VehicleDetailsLocalSource {

    companion object {
        const val KEY_CONSUMPTION = "consumption"
        const val CONSUMPTION_DEFAULT_VALUE = 56
        const val KEY_TANK = "tank"
        const val TANK_DEFAULT_VALUE = 50
    }

    override fun getVehicleDetails(): VehicleDetailsDomainModel {
        return VehicleDetailsDomainModel(
            consumption = getConsumption(),
            tank = getTank(),
        )
    }

    [ ... ]
}
```

Figura 70: Código fuente de datos a través de *SharedPreferences*

4.6.4.3. *Firestore* y alertas

Para almacenar las alertas de los usuarios se utiliza *Firestore*, el almacenamiento basado en colecciones de *Firebase*. Estas colecciones tienen asociadas un identificador, pudiendo ser aleatorio o generado previamente.

Para el caso de las alertas se ha decidido utilizar un identificador creado a partir de las características de la propia alerta, con el objetivo de realizar búsquedas más rápidas. Por lo tanto, la estructura del identificador sigue el patrón siguiente:

```
gasStationId_fuelType_pushToken
```

Este identificador sería único por estación de servicio, tipo de combustible (diesel, gasolina, hidrógeno, etc.) y *token* (valor único por usuario e instalación de aplicación).

A continuación, un ejemplo que ilustra lo relatado.

Figura 71: Ejemplo identificador en *Firestore*

4.6.5. Back-end

La gestión de las alertas es realizada a través de las bibliotecas que proporcionan las diferentes herramientas de *Firebase*. El lenguaje utilizado es Python por ser versátil y fácil de usar.

Las bibliotecas de *Firebase* que son necesarias para el tratamiento de las alertas son las descritas a continuación:

- ***Firestore***
  Permite leer los datos almacenados en las colecciones de *Firestore Database*.

- ***Functions***
  Es utilizada para el envío de notificaciones *push* a los dispositivos que previamente se han registrado a través de un *token*.

```python
from firebase_admin import firestore

def process_alerts(stations):
    db = firestore.client()
    alerts_ref = db.collection('alerts')
    alert_docs = alerts_ref.stream()

    for data in alert_docs:
        if data:
            alert: AlertModel = AlertModel.from_dict(data.to_dict())
            if (alert.is_valid() and alert.is_enabled and alert.price
                    >= get_price_by_gas_station_id(
                        stations,
                        alert.fuel_type,
                        alert.gas_station_id
                    )
            ):
                send_push_notification([...])
```

Figura 72: Código fuente lectura datos *Firestore Cloud*

```python
from firebase_admin import messaging

def send_push_notification(token, message_title, message_body):
    message = messaging.Message(
        notification=messaging.Notification(
            title=message_title,
            body=message_body,
        ),
        token=token,
    )

    try:
        response = messaging.send(message)
        print('Successfully sent message:', response)
    except Exception as e:
        print(f"Failed to send message: {e}")
```

Figura 73: Código fuente envío notificaciones con *Firebase Messaging*

5. Verificación

Las pruebas de *software* son fragmentos de código que verifican el correcto funcionamiento de la aplicación. Por este motivo, en este apartado se identifican los diferentes tipos de pruebas realizadas para asegurar que el proyecto cumple con los requisitos de negocio, además de indicar las bibliotecas utilizadas en cada tipo de prueba.

5.1. Tipos de pruebas

En el proyecto se han realizado tres tipos de pruebas, siendo las descritas a continuación:

- **Prueba unitaria**
  Este tipo comprueba la unidad de un fragmento de código.
- **Prueba de integración**
  Comprueban la conexión con las *APIs* definidas en la aplicación.
- **Prueba de interfaz**
  Verifican los elementos visuales que se muestran en una determinada pantalla.

5.2. Pruebas unitarias

Para la realización de las pruebas unitarias se ha utilizado como bibliotecas las enumeradas a continuación:

- ***JUnit5***: parametrizar las pruebas unitarias con diferentes estados, permitiendo ejecutar varias utilizando el mismo método [63].
- ***Turbine***: realizar pruebas con *flows* [69].
- ***MockK***: imitar clases durante la ejecución de las pruebas [70].
- ***Kotest Assertions***: verificar el valor recibido con el esperado a través de *assertions* (afirmaciones) [71].

Un ejemplo de código:

```
@ParameterizedTest(name = "GIVEN fuel {0} and location {1} WHEN get alerts THEN return {3} alerts")
@MethodSource("getArguments")
fun `USE CASE get alerts`(
    fuelType: FuelTypeDomainEnum,
    location: LocationDomainModel?,
    userId: String?,
    alertsSize: Int,
    alertsExpected: List<AlertGasStationDomainModel>,
) = runTest {
    // 🔵 Given //
    coEvery {
        mockMessagingRepository.getToken()
    } returns userId

    // 🟡 When //
    val flow = useCase.invoke([ ... ])

    // 🟢 Then //
    flow.test {
            awaitItem().shouldBe(alertsExpected)
            awaitComplete()
        }
    }
```

Figura 74: Código fuente prueba unitaria caso de uso *getAlerts*

5.3. Pruebas de integración

5.3.1. *API*

En este tipo de pruebas se ha utilizado las mismas de las pruebas unitarias, añadiendo la siguiente:

- ***MockWebServer***: permite imitar un servidor durante la ejecución de las pruebas [72].

```kotlin
@ParameterizedTest(name = "GIVEN gas stations with status code {0} WHEN call api THEN return {3}")
@MethodSource("getArguments")
fun `REMOTE get gas stations`(
    httpCode: Int,
    responseFileName: String,
    isSuccess: Boolean,
    alertsSize: Int,
) = runTest {
    // 🔵 Given //
    mockWebServer.enqueueResponse(responseFileName, httpCode)

    // 🟡 When //
    val result = remote.getStations()

    // 🟢 Then //
    assertEquals(expected = isSuccess, actual = result.isSuccess)
    assertEquals(expected = alertsSize, actual = result.getOrElse { emptyList() }.size)
}
```

Figura 75: Código fuente prueba integración *API*

5.3.2. Base de datos

Para verificar el estado inicial de la base de datos se han ejecutado las pruebas bajo un dispositivo real. Esto se debe al contexto utilizado por la biblioteca *Room*. Para este caso, la base de datos es inicializada a partir de la existente en *assets*.

Desde una clase abstracta se inicializa la base de datos. De esta manera, el resto de clases solo tendrán que extender de la clase *base*, permitiendo reducir la inicialización de la base de datos por cada prueba, logrando así evitar la duplicidad de código.

```kotlin
abstract class AbsDaoTest {

    protected lateinit var db: FuelPriceDb

    @Before
    open fun createDb() {
        val context = ApplicationProvider.getApplicationContext<Context>()
        db = Room
            .databaseBuilder(context, FuelPriceDb::class.java, "fuel_price.db")
            .createFromAsset("db/fuel_price.db")
            .build()
    }

    @After
    @Throws(IOException::class)
    fun closeDb() {
        db.close()
    }
}
```

Figura 76: Código fuente clase abstracta prueba integración base de datos

```kotlin
@RunWith(AndroidJUnit4::class)
class DiscountDaoTest : AbsDaoTest() {

    private lateinit var discountDao: DiscountDao

    @Before
    override fun createDb() {
        super.createDb()
        discountDao = db.discountDao()
    }

    @Test
    @Throws(Exception::class)
    fun testDiscountsDao() {
        // 🔵 Given //
        val expectedDiscounts = listOf([...])
        discountDao.update(1, true)     // update only the first one

        // 🟡 When //
        val discountsWithBrandDb = discountDao.getDiscounts()

        // 🟢 Then //
        discountsWithBrandDb.forEach { actual →
            actual.discount.shouldBe(expectedDiscounts.find { it.id == actual.discount.id })
        }
    }
}
```

Figura 77: Código fuente prueba integración base de datos

5.4. Prueba de interfaz

Para realizar las pruebas de interfaz se puede utilizar un dispositivo real o ser ejecutadas bajo la *JVM* (*Java Virtual Machine*, Máquina Virtual de Java), donde su ejecución será más rápida.

Para este caso, se ha decidido utilizar *Robolectric* como *framework* para ejecutar las pruebas de interfaz, bajo el contexto indicado anteriormente, ejecución de pruebas sin dispositivo real.

```
@RunWith(RobolectricTestRunner::class)
@GraphicsMode(GraphicsMode.Mode.NATIVE)
@Config(
    manifest = "AndroidManifest.xml",
    sdk = [
        [...]
    ],
)
open class BaseUiTest {

    protected lateinit var context: Context

    @get:Rule
    val composeTestRule = createAndroidComposeRule<ComponentActivity>()

    @Before
    @Throws(Exception::class)
    open fun setUp() {
        MockKAnnotations.init(this)
        context = InstrumentationRegistry.getInstrumentation().targetContext
        [...]
        Shadows.shadowOf(appContext.packageManager).addOrUpdateActivity(activityInfo)
    }

    @After
    open fun tearDown() {
        clearAllMocks()
        unmockkAll()
    }

    protected fun setContent(composable: @Composable () -> Unit) {
        composeTestRule.setContent(composable)
    }

    protected fun waitForIdle() {
        composeTestRule.waitForIdle()
        printLog()
    }

    [...]
}
```

Figura 77: Código fuente clase base prueba de interfaz

```kotlin
@Test
fun `GIVEN gas station details screen WHEN success THEN show expected layout`() {
    // 🔵 Given //
    val fakeGasStationName = "Fake Gas Station Name"
    val fakePriceDieselA = "1,459 €"
    val fakePriceGasoline = "1,599 €"
    setContent {
        GasStationDetailsSuccessComposable(
            uiModel = GasStationDetailsUiModel([ ... ]),
            onClickFavorite = {},
            onClickNavigateMaps = {},
            onRatingGasStation = {},
            onRatingBrand = {},
        )
    }

    // 🟡 When //
    waitForIdle()

    // 🟢 Then //
    composeTestRule.onNodeWithText(text = fakeGasStationName, useUnmergedTree = true).assertExists()
    composeTestRule.onNodeWithText(text = fakePriceDieselA, useUnmergedTree = true).assertExists()
    composeTestRule.onNodeWithText(text = fakePriceGasoline, useUnmergedTree = true).assertExists()
}
```

Figura 78: Código fuente prueba de interfaz detalles estación de servicio

6. Estado actual

El desarrollo de la aplicación ha supuesto una serie de retos no detectados durante la planificación de las tareas. Por este motivo, la tarea de realizar el listado de estaciones de servicio ha supuesto un sumatorio de horas extras, en total 40h frente a las 20h propuestas. Esto se debe a la codificación que se ha tenido que realizar para obtener los resultados de la fuente de datos, adaptar los datos a la lógica de negocio, y finalmente, representarlos.

En contraposición, las tareas Mapa, Favoritos y Notificaciones (en la aplicación) han resultado ser más eficientes, pues no ha sido necesario recurrir al monto total de horas estimadas. Del mismo modo que el Listado, esta reducción de horas se debe a la reutilización de la lógica de negocio y el consumo de la fuente de datos que previamente se había codificado, y durante la planificación no se tuvo en cuenta.

7. Conclusiones

La realización del proyecto desde la planificación y toma de requisitos hasta la entrega final de la aplicación, ha permitido explorar las diferentes áreas necesarias para el desarrollo del *software*.

La planificación se ha cumplido con ciertos cambios durante la realización del proyecto. Esto ayudará a realizar una planificación más concienzuda la próxima ocasión, teniendo en cuenta los errores cometidos.

Además, abordar el diseño de las interfaces a través de los datos recopilados por el Diseño Centrado al Usuario, ha sido vital para crear pantallas adecuadas a los usuarios que las iban a utilizar, evitando así elementos redundantes o requisitos innecesarios.

Desde el punto de vista más técnico, se han utilizado tecnologías que están en la vanguardia del desarrollo de aplicaciones *Android*, como son *Jetpack Compose* y *Flows*, entre otras. La exploración y el uso ha servido para seguir aprendiendo sobre estas tecnologías.

Finalmente, los próximos pasos a seguir en el proyecto son los enumerados a continuación:

- **Aumentar el número de pruebas**
  Actualmente solo se han escrito una prueba por cada tipo, aumentar este número permitirá tener un *software* más sólido. Además, no es descartable añadir pruebas de otro tipo, como estrés, interfaz en diferentes dispositivos, entre otras.

- **Automatización de los procesos**
  La ejecución de las pruebas actualmente es manual, igual que el proceso de publicación. Estas tareas se pueden automatizar con cualquier solución de *software* que ofrezca *CI/CD*, como *GitLab Ci/CD*.

- **Incorporación de herramientas de analítica**
  Los datos recopilados permitirán analizar con precisión el comportamiento de los usuarios, y realizar acciones que mejoren la experiencia durante la sesión de cada usuario. Una solución de *software* a utilizar es *Google Analytics*.

8. Glosario

A

Android: sistema operativo basado en Linux enfocado en dispositivos móviles.

API: *Application Programming Interface*, permite comunicar dos aplicaciones diferentes.

Aplicación: programa informático.

App: acrónimo inglés de *application*, traducción de aplicación.

Arquitectura: organización de las diferentes clases, modelos y datos que forman un sistema informático.

B

Back-end: parte de un programa informático que no es visible al usuario.

C

Casos de uso: descripción de la interacción del usuario con el programa informático.

Ciclo de vida: conjunto de las diferentes etapas que forman un proyecto, desde su planificación hasta su entrega.

Codificación: acto de trasladar los diseños de interfaz de usuario y los requisitos tomados a un programa de *software*.

Compilar: proceso que consiste en traducir el lenguaje de alto nivel del programa informático en otro de más bajo nivel, como lenguaje máquina.

D

Despliegue: acción de subir el programa a un entorno de producción para ser consumido por los usuarios finales.

F

Fichero ejecutable: programa informático preparado para ser iniciado en un sistema operativo.

H

Home: pantalla principal de un sistema informático.

L

Layer: capa.

O

ORM: *Object Relational Mapping*, transforma las tablas de la base de datos en objetos.

P

Prototipado: diseño preliminar al definitivo, utilizado para establecer las bases del diseño de la interfaz del programa informático.

Prueba de integración: verificación de la interacción entre dos o más módulos en un programa informático.

Prueba unitaria: verificación de una unidad de *software*, generalmente un método o una clase.

Push: notificación a partir de un identificador de dispositivo.

R

Requisito: especificación de una funcionalidad del programa informático.

S

Serverless: ejecución de código de servidor sin gestión directa.

SharedPreferences: sistema de almacenamiento de archivos en *Android* basado en ficheros *XML*.

Smartphone: dispositivo móvil inteligente.

Software: parte no física de un sistema informático, formados por instrucciones para ejecutar ciertas tareas por la parte física.

Splash: pantalla inicial de un programa informático.

Sprint: periodo corto de tiempo utilizado en las metodologías ágiles para realizar un determinado número de tareas, normalmente suele ser una o dos semanas.

Stakeholder: denominado así a la persona o conjunto de personas interesadas en una o más funcionalidades de un programa informático.

T

Test: prueba para verificar el correcto funcionamiento de una funcionalidad.

U

UML: *Unified Modeling Language*, lenguaje de modelado basado en diagramas para diseñar y modelar clases y arquitecturas de *software*.

9. Bibliografía

1. Navarro, R. (2023) *The average screen time and usage by country*, *ElectronicsHub*. Disponible en: https://www.electronicshub.org/the-average-screen-time-and-usage-by-country/ (Consultado: el 8 de marzo de 2024).
2. Cronista, E. (2023) "¿Adicción o necesidad? Un informe revela cuánto tiempo usan el teléfono los españoles: las cifras asustan", *El Cronista*, 16 mayo. Disponible en: https://www.cronista.com/espana/pc-movil/adiccion-o-necesidad-un-informe-revela-cuanto-tiempo-usan-el-telefono-los-espanoles-las-cifras-asustan/ (Consultado: el 8 de marzo de 2024).
3. *Precio de Gasolina al Instante* (sin fecha) *Google.com*. Disponible en: https://play.google.com/store/apps/details?id=de.mwwebwork.benzinpreisblitz (Consultado: el 8 de marzo de 2024).
4. *GasAll: Gas stations in Spain* (sin fecha) *Google.com*. Disponible en: https://play.google.com/store/apps/details?id=com.gasall (Consultado: el 8 de marzo de 2024).
5. *Gasolina y Diesel España* (sin fecha) *Google.com*. Disponible en: https://play.google.com/store/apps/details?id=com.quadbits.smartrefueling (Consultado: el 8 de marzo de 2024).
6. Colaboradores Wikipedia (sin fecha) *Desarrollo ágil de software*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Desarrollo_%C3%A1gil_de_software&oldid=157570052 (Consultado: el 8 de marzo de 2024).
7. Colaboradores Wikipedia (sin fecha) *Scrum (desarrollo de software)*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Scrum_(desarrollo_de_software)&oldid=158396196 (Consultado: el 8 de marzo de 2024).
8. Colaboradores Wikipedia (sin fecha) *Desarrollo en cascada*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Desarrollo_en_cascada&oldid=158529069 (Consultado: el 8 de marzo de 2024).
9. Colaboradores Wikipedia (sin fecha) *Diagrama de Gantt*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Diagrama_de_Gantt&oldid=153182377 (Consultado: el 8 de marzo de 2024).
10. Colaboradores Wikipedia (sin fecha) *Diseño centrado en el usuario*, *Wikipedia, The Free Encyclopedia*. Disponible en:

https://es.wikipedia.org/w/index.php?title=Dise%C3%B1o_centrado_en_el_usuario&oldid=126520734 (Consultado: el 16 de marzo de 2024).

11. *Check mark - OpenMoji 15.0* (sin fecha) *Emojipedia*. Disponible en: https://emojipedia.org/openmoji/15.0/check-mark (Consultado: el 16 de marzo de 2024).
12. *Cross mark - OpenMoji 15.0* (sin fecha) *Emojipedia*. Disponible en: https://emojipedia.org/openmoji/15.0/cross-mark (Consultado: el 16 de marzo de 2024).
13. *Warning - OpenMoji 15.0* (sin fecha) *Emojipedia*. Disponible en: https://emojipedia.org/openmoji/15.0/warning (Consultado: el 16 de marzo de 2024).
14. *ThisPersonDoesNotExist - Random AI Generado Fotos de Personas Falsas* (2021) *This-person-does-not-exist.com*. Google. Disponible en: https://this-person-does-not-exist.com/es (Consultado: el 16 de marzo de 2024).
15. *Problem statement - Design Toolkit* (sin fecha) *Uoc.edu*. Disponible en: https://design-toolkit.recursos.uoc.edu/es/problem-statement/ (Consultado: el 16 de marzo de 2024).
16. *Intelligent diagramming* (sin fecha) *Lucidchart*. Disponible en: https://www.lucidchart.com/ (Consultado: el 16 de marzo de 2024).
17. *Figma: The Collaborative Interface Design Tool* (sin fecha) *Figma*. Disponible en: https://www.figma.com/ (Consultado: el 16 de marzo de 2024).
18. *Now in Android case study* (sin fecha) *Figma*. Disponible en: https://www.figma.com/community/file/1164313362327941158 (Consultado: el 16 de marzo de 2024).
19. Wikipedia contributors (sin fecha) *Montserrat (tipo de letra)*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Montserrat_(tipo_de_letra)&oldid=154887689 (Consultado: el 16 de marzo de 2024).
20. *Montserrat* (sin fecha) *Google Fonts*. Disponible en: https://fonts.google.com/specimen/Montserrat (Consultado: el 16 de marzo de 2024).
21. *Test con usuarios (métodos) - Design Toolkit* (sin fecha) *Uoc.edu*. Disponible en: https://design-toolkit.recursos.uoc.edu/es/test-con-usuarios/ (Consultado: el 16 de marzo de 2024).
22. *Test con usuarios (guía) - Design Toolkit* (sin fecha) *Uoc.edu*. Disponible en: https://design-toolkit.recursos.uoc.edu/es/guia/test-con-usuarios/ (Consultado: el 16 de marzo de 2024).
23. Wikipedia contributors (sin fecha e) *Lenguaje unificado de modelado*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Lenguaje_unificado_de_modelado&oldid=157606665 (Consultado: el 19 de marzo de 2024).
24. Intelligent diagramming (sin fecha) Lucidchart. Disponible en: https://www.lucidchart.com/ (Consultado: el 19 de marzo de 2024).

25. Wikipedia contributors (2023) *Open–closed principle*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://en.wikipedia.org/w/index.php?title=Open%E2%80%93closed_principle&oldid=1187845413 (Consultado: el 19 de marzo de 2024).
26. Wikipedia contributors (sin fecha) *Firebase*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Firebase&oldid=156557745 (Consultado: el 19 de marzo de 2024).
27. *Firestore* (sin fecha) *Firebase*. Disponible en: https://firebase.google.com/docs/firestore (Consultado: el 19 de marzo de 2024).
28. *Cloud Functions for* (sin fecha) *Firebase*. Disponible en: https://firebase.google.com/docs/functions (Consultado: el 11 de abril de 2024).
29. Wikipedia contributors (sin fecha) *Serverless computing*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Serverless_computing&oldid=156087796 (Consultado: el 11 de abril de 2024).
30. Wikipedia contributors (sin fecha) *Python*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Python&oldid=159353053 (Consultado: el 11 de abril de 2024).
31. *Precio de carburantes en las gasolineras españolas - Conjunto de datos* (sin fecha) *Gob.es*. Disponible en: https://datos.gob.es/es/catalogo/e05068001-precio-de-carburantes-en-las-gasolineras-espanolas (Consultado: el 11 de abril de 2024).
32. *Firebase Cloud Messaging* (sin fecha) *Firebase*. Disponible en: https://firebase.google.com/docs/cloud-messaging (Consultado: el 11 de abril de 2024).
33. *UI layer* (sin fecha) *Android Developers*. Disponible en: https://developer.android.com/topic/architecture/ui-layer (Consultado: el 19 de marzo de 2024).
34. *Mastering the MVI Architecture: A Comprehensive Guide* (sin fecha) *Medium.com*. Disponible en: https://medium.com/@satriawaaan/mastering-the-mvi-architecture-a-comprehensive-guide-28b2cafee1c9 (Consultado: el 19 de marzo de 2024).
35. *Domain layer* (sin fecha) *Android Developers*. Disponible en: https://developer.android.com/topic/architecture/domain-layer (Consultado: el 19 de marzo de 2024).
36. *Data layer* (sin fecha) *Android Developers*. Disponible en: https://developer.android.com/topic/architecture/data-layer (Consultado: el 19 de marzo de 2024).

37. Wikipedia contributors (sin fecha) *API*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=API&oldid=158385052 (Consultado: el 20 de marzo de 2024).
38. Wikipedia contributors (2024) *REST*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://en.wikipedia.org/w/index.php?title=REST&oldid=1211950882 (Consultado: el 20 de marzo de 2024).
39. Wikipedia contributors (sin fecha) *Mapeo relacional de objetos*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Mapeo_relacional_de_objetos&oldid=155592130 (Consultado: el 21 de marzo de 2024).
40. *SharedPreferences* (sin fecha) *Android Developers*. Disponible en: https://developer.android.com/reference/android/content/SharedPreferences (Consultado: el 21 de marzo de 2024).
41. Dorfmann, H. (sin fecha) *Model-View-Intent on Android*, *Hannesdorfmann.com*. Disponible en: https://hannesdorfmann.com/android/model-view-intent/ (Consultado: el 21 de marzo de 2024).
42. *Firebase Cloud Messaging* (sin fecha) *Firebase*. Disponible en: https://firebase.google.com/docs/cloud-messaging (Consultado: el 22 de marzo de 2024).
43. Wikipedia contributors (sin fecha) *SOLID*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=SOLID&oldid=156873545 (Consultado: el 12 de abril de 2024).
44. *¿Qué es Clean Code o código limpio?* (2023) *Qindel: Consultoría IT. QINDEL.* Disponible en: https://www.qindel.com/que-es-clean-code-o-codigo-limpio/ (Consultado: el 12 de abril de 2024).
45. Belinski, E. (sin fecha) *Android API Levels*, *Apilevels.com*. Disponible en: https://apilevels.com/ (Consultado: el 12 de abril de 2024).
46. Wikipedia contributors (sin fecha) *Robert C. Martin*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Robert_C._Martin&oldid=155980620.
47. Martin, R. C. & Feathers, M. C. (2009) *Clean code : a handbook of agile software craftsmanship / Robert C. Martin ; [with] Michael C. Feathers ... [et al.].* 1st edition. Upper Saddle River, N.J: Prentice Hall (Consultado: el 12 de abril de 2024).
48. Wikipedia contributors (sin fecha) *Principio KISS*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Principio_KISS&oldid=134618076 (Consultado: el 12 de abril de 2024).

49. Redactor, P. (2024) *Qué es el código limpio o clean code*, *ÁreaF5*. Disponible en: https://areaf5.es/blog/que-es-el-codigo-limpio-o-clean-code (Consultado: el 12 de abril de 2024).

50. *Download Android Studio & app tools* (sin fecha) *Android Developers*. Disponible en: https://developer.android.com/studio (Consultado: el 12 de abril de 2024).
51. *IntelliJ IDEA – the leading Java and Kotlin IDE* (sin fecha) *JetBrains*. Disponible en: https://www.jetbrains.com/idea/ (Consultado: el 12 de abril de 2024).
52. *PyCharm: the Python IDE for data science and web development* (sin fecha) *JetBrains*. Disponible en: https://www.jetbrains.com/pycharm/ (Consultado: el 12 de abril de 2024).
53. *DB Browser for SQLite* (sin fecha) *Sqlitebrowser.org*. Disponible en: https://sqlitebrowser.org/ (Consultado: el 12 de abril de 2024).
54. Wikipedia contributors (sin fecha) *Android SDK*, *Wikipedia, The Free Encyclopedia*. Disponible en: https://es.wikipedia.org/w/index.php?title=Android_SDK&oldid=158099979 (Consultado: el 12 de abril de 2024).
55. *Jetpack Compose UI app development toolkit* (sin fecha) *Android Developers*. Disponible en: https://developer.android.com/develop/ui/compose (Consultado: el 12 de abril de 2024).
56. *Google maps platform documentation* (sin fecha) *Google for Developers*. Disponible en: https://developers.google.com/maps/documentation/android-sdk (Consultado: el 12 de abril de 2024).
57. *Dependency injection with hilt* (sin fecha) *Android Developers*. Disponible en: https://developer.android.com/training/dependency-injection/hilt-android (Consultado: el 12 de abril de 2024).
58. Square, Inc (sin fecha) *Overview - OkHttp*, *Github.io*. Disponible en: https://square.github.io/okhttp/ (Consultado: el 15 de mayo de 2024).
59. *Retrofit* (sin fecha) *Github.io*. Disponible en: https://square.github.io/retrofit/ (Consultado: el 15 de mayo de 2024).
60. *Gson: A Java serialization/deserialization library to convert Java Objects into JSON and back* (sin fecha). Disponible en: https://github.com/google/gson (Consultado: el 15 de mayo de 2024).
61. *Robolectric* (sin fecha) *Robolectric.org*. Disponible en: https://robolectric.org/ (Consultado: el 15 de mayo de 2024).
62. *junit4: A programmer-oriented testing framework for Java* (sin fecha). Disponible en: https://github.com/junit-team/junit4 (Consultado: el 15 de mayo de 2024).
63. *JUnit 5* (sin fecha) *Junit.org*. Disponible en: https://junit.org/junit5/ (Consultado: el 15 de mayo de 2024).

64. *Coding conventions* (sin fecha) *Kotlin Help*. Disponible en: https://kotlinlang.org/docs/coding-conventions.html (Consultado: el 15 de mayo de 2024).
65. *Git* (sin fecha) *Git-scm.com*. Disponible en: https://git-scm.com/ (Consultado: el 15 de mayo de 2024).
66. *Keywords and operators* (sin fecha) *Kotlin Help*. Disponible en: https://kotlinlang.org/docs/keyword-reference.html (Consultado: el 15 de mayo de 2024).
67. *Invoke* (sin fecha) *Kotlin*. Disponible en: https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/invoke.html (Consultado: el 15 de mayo de 2024).
68. *Coroutines basics* (sin fecha) *Kotlin Help*. Disponible en: https://kotlinlang.org/docs/coroutines-basics.html (Consultado: el 15 de mayo de 2024).
69. *Turbine: A small testing library for kotlinx.coroutines Flow* (sin fecha). Disponible en: https://github.com/cashapp/turbine (Consultado: el 15 de mayo).
70. *MockK* (sin fecha) *MockK*. Disponible en: https://mockk.io/ (Consultado: el 15 de mayo de 2024).
71. *Assertions* (sin fecha) *Kotest.io*. Disponible en: https://kotest.io/docs/assertions/assertions.html (Consultado: el 15 de mayo de 2024).
72. *Mockwebserver at master · square/okhttp* (sin fecha). Disponible en: https://github.com/square/okhttp/tree/master/mockwebserver (Consultado: el 15 de mayo de 2024).

10. Anexos

10.1. Diagrama de Gantt al completo

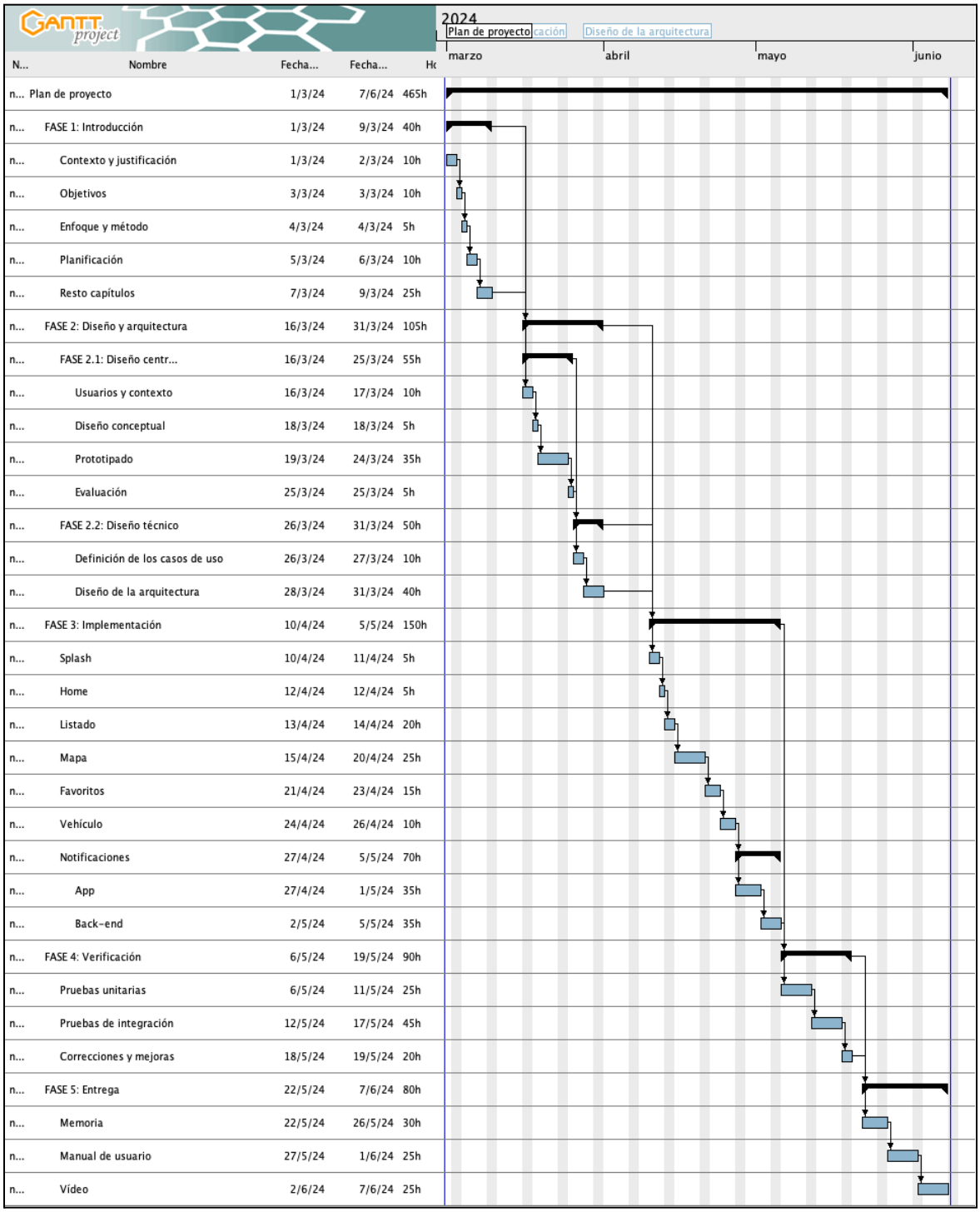

Figura 80: Diagrama de Gantt al completo

10.2. *Sketches*

10.2.1. *Splash*

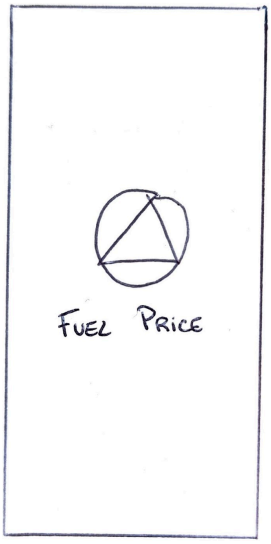

Figura 81: *Sketch* pantalla de inicio

Es la primera pantalla que visualiza el usuario al abrir la aplicación. Se muestra el logo y el nombre del programa. Debe durar un breve transcurso de tiempo, entre uno y tres segundos.

10.2.2. Pantalla principal y listado de precios

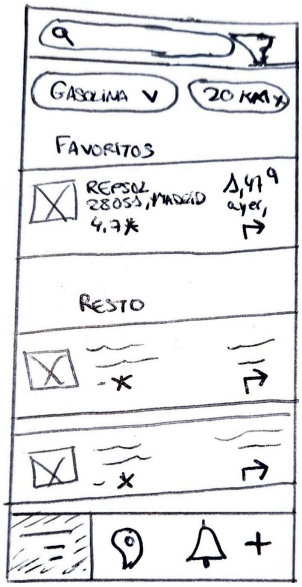

Figura 82: *Sketch* pantalla principal y listado de precios

Después de la pantalla de inicio, se muestra la pantalla principal, también conocida como *home*. En esta pantalla el usuario puede navegar a las diferentes áreas de la aplicación a través de la barra de navegación inferior, donde se encuentran los iconos correspondientes a cada pantalla, y son: listado, mapa, alertas y vehículo y descuentos.

Por defecto la primera pantalla que se visualiza es la del listado de precios. En ella se puede apreciar una barra de búsqueda en la parte superior, donde el usuario puede buscar por una dirección para mostrar otros resultados situados en el destino deseado. Más abajo, se encuentra una vista que permite desplegar el tipo de combustible que se está utilizando para filtrar los precios.

Finalmente, el usuario puede observar tanto los elementos marcados como favoritos como el resto. Dentro de cada elemento se distinguen varias características que aportan información relevante al usuario: el logo de la marca del carburante, el nombre, código postal y municipio de la estación de servicio, icono para marcar o desmarcar como favorito, el precio (tamaño grande para destacar sobre el resto), y el icono para navegar a través de una aplicación de terceros hacia el destino elegido.

10.2.3. Detalles

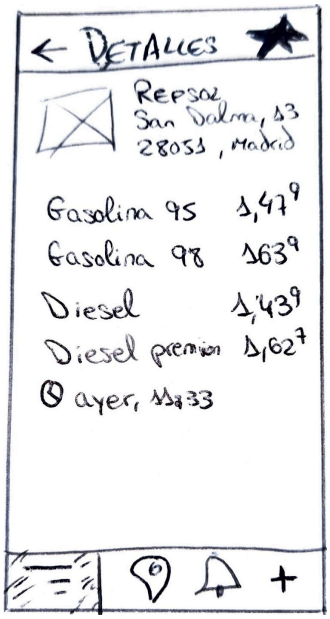

Figura 83: *Sketch* pantalla detalles

Para acceder a esta pantalla basta con pulsar sobre un elemento del listado o del mapa. De esta forma el usuario puede ampliar con más datos la información relacionada con la estación de servicio.

10.2.4. Mapa

Figura 84: *Sketch* pantalla mapa

En la segunda posición de la barra de navegación inferior se encuentra el mapa. En esta pantalla el usuario puede interaccionar tanto en los ejes verticales como horizontales con un mapa, del mismo modo, podrá pellizcar con los dedos para aumentar o disminuir el zoom.

Asimismo, se visualizarán todas las estaciones utilizando un marcador sobre el punto geolocalizado de cada una. Este marcador proporcionará información relevante como el nombre de la estación y el precio del combustible seleccionado.

10.2.5. Alertas

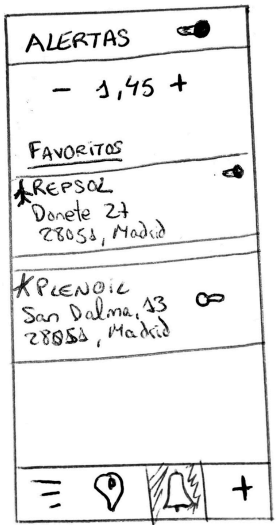

Figura 85: *Sketch* pantalla alertas

En esta pantalla el usuario puede crear alertas que se activarán bajo un precio seleccionado. Este precio puede ser aumentado y disminuido a través de dos iconos situados a los lados de la etiqueta.

El usuario puede activar o desactivar todas las alertas desde el botón de la parte superior, o hacerlo de forma individual en cada una de ellas.

10.2.6. Vehículos y descuentos

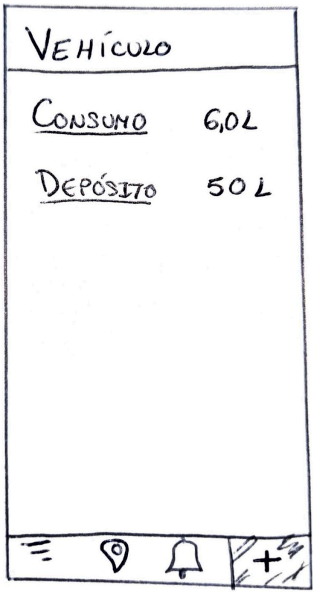

Figura 86: *Sketch* pantalla vehículo y descuentos

En la última pantalla accesible desde la barra de navegación inferior, se puede modificar el consumo medio que realiza el vehículo y la capacidad que poseé el depósito.

Asimismo, se puede añadir descuentos que son ofrecidos por las marcas de carburantes o las estaciones de servicio. Una vez añadidos, pueden ser eliminados fácilmente a través del icono que aparece al inicio de la descripción del descuento.

10.2.7. Diálogo de navegación

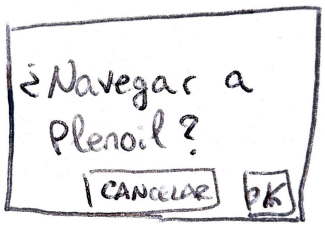

Figura 87: *Sketch* diálogo navegación

Finalmente, este diálogo pregunta al usuario si desea navegar hacia la estación de servicio que ha seleccionado. En caso afirmativo, se abrirá de forma automática un diálogo nativo del sistema operativo ofreciendo qué tipo de aplicación se desea usar para completar la acción.

10.3. Entrevistas de los participantes

10.3.1. Usuario 1: Marta

1. Información general

- ¿Usas con frecuencia aplicaciones para encontrar precios económicos de cualquier producto?

  Con mucha frecuencia no, solo a veces, cuando quiero un producto determinado pero pienso que tiene un precio excesivo, por lo tanto trato de buscarlo en aplicaciones que comparan el precio para encontrarlo más económico.

- ¿Utilizas aplicaciones para encontrar el mejor precio del combustible en tu zona?

  No, nunca. Siempre me he guiado por los precios de mi zona y los comentarios de familiares y amigos.

2. Pantalla de inicio

- ¿Cómo visualizas la pantalla de inicio?

  Bien.

3. Pantalla de inicio

- ¿Entiendes y comprendes la barra de navegación inferior?

  Sí. Veo cuatro iconos y supongo que corresponden a un listado, un mapa, notificaciones, y por último, información del vehículo.

4. Listado de precios

- ¿Encuentras el botón de buscar? ¿Qué funcionalidad piensas que tiene?

  Sí, arriba. Pienso que buscar sobre el listado.

- ¿Puedes determinar qué combustible se está utilizando para filtrar los resultados? ¿Sabrías cambiar de combustible?

  Diesel, se indica a través de un elemento con bordes redondeados. Pienso que al pulsar sobre él aparecerán otras opciones.

- ¿Comprendes cada elemento que se lista en la pantalla?

  Sí. Veo el logo, nombre, dirección, valoración, un corazón que supongo que será si es o no favorito, el precio, una fecha que entiendo que será cuando se ha actualizado, un coste en rojo que no sé muy bien qué es, y un icono para navegar ligado a la distancia de la estación de servicio.

- ¿Puedes encontrar el icono para marcar o desmarcar favoritos?

  Sí, al lado del precio.

- ¿Y el de navegar?

  Sí, debajo del precio.

- ¿Sabrías entrar en los detalles de una estación de servicio?

  Supongo que al pulsar sobre cualquier elemento, éste abrirá una pantalla con los detalles.

5. Detalles

- ¿Puedes encontrar las opciones de marcar como favorito? ¿Y navegar?

  Sí, el primero al lado del precio, el segundo debajo.

- ¿Comprendes qué significa valorar la estación de servicio y la marca del carburante?

  Sí.

6. Mapa

- ¿Visualizas correctamente los marcadores en el mapa?

  Sí. Se ven claros.

- ¿Y los precios?

  Sí, aunque el precio de color verde no lo veo muy resaltado y es posible confundirlo en el mapa.

7. Alertas

- ¿Entiendes qué significan los iconos ubicados a los lados de la etiqueta del precio?

  Sí, disminuir o aumentar el precio.

- ¿Sabrías activar o desactivar todas las alertas?

Sí, arriba a la derecha se encuentra el icono.

- ¿Y de forma individual?

  Sí, en la parte derecha de cada elemento está la campanita.

8. Vehículo y descuentos

- ¿Por qué piensas que se pide el consumo medio y la capacidad total del depósito?

  No lo sé.

- ¿Sabrías añadir y eliminar un descuento?

  Sí.

9. Conclusión y mejoras

- ¿Qué te parecen los diseños? ¿Los ves claros y concisos?

  Están muy bien. Me gustan los colores elegidos. Y sí, los veo claros y concisos.

- ¿Qué añadirías?

  No se me ocurre nada ahora mismo.

- ¿Qué cambiarías?

  El color verde que aparece en el mapa y la etiqueta de color rojo resaltada en el listado de precios.

- ¿Qué eliminarías?

  Nada.

- ¿Utilizarías la aplicación?

  Sí.

10.3.2. Usuario 2: Alba

1. Información general

- ¿Usas con frecuencia aplicaciones para encontrar precios económicos de cualquier producto?

  Bueno, con mucha frecuencia no, pero sí a menudo.

- ¿Utilizas aplicaciones para encontrar el mejor precio del combustible en tu zona?

  Alguna vez he utilizado alguna aplicación de este tipo, pero ha sido algo ocasional.

2. Pantalla de inicio

- ¿Cómo visualizas la pantalla de inicio?

  Básica pero clara.

3. Pantalla de inicio

- ¿Entiendes y comprendes la barra de navegación inferior?

Sí, para acceder a cada pantalla. Ayuda el texto ubicado en la parte inferior de cada icono.

4. Listado de precios

- ¿Encuentras el botón de buscar? ¿Qué funcionalidad piensas que tiene?

  Sí, al inicio de la pantalla. Supongo que sirve para buscar algún destino.

- ¿Puedes determinar qué combustible se está utilizando para filtrar los resultados? ¿Sabrías cambiar de combustible?

  Diesel. Justo debajo de la lupa se encuentra y pienso que al seleccionarlo podré cambiarlo a gasolina, por ejemplo.

- ¿Comprendes cada elemento que se lista en la pantalla?

  Comprendo todos salvo el resaltado de color rojo.

- ¿Puedes encontrar el icono para marcar o desmarcar favoritos?

  Sí, arriba al lado del precio,

- ¿Y el de navegar?

  Si, abajo a la derecha, indicado con una flecha.

- ¿Sabrías entrar en los detalles de una estación de servicio?

  Entiendo que pulsando sobre un elemento.

5. Detalles

- ¿Puedes encontrar las opciones de marcar como favorito? ¿Y navegar?

  Sí, ubicados en el mismo sitio que en la pantalla anterior.

- ¿Comprendes qué significa valorar la estación de servicio y la marca del carburante?

  Sí, está indicado claramente.

6. Mapa

- ¿Visualizas correctamente los marcadores en el mapa?

  Sí. Además el logo ayuda.

- ¿Y los precios?

  Veo todo bien, pero el marcado en verde podría verse mejor.

7. Alertas

- ¿Entiendes qué significan los iconos ubicados a los lados de la etiqueta del precio?

  Sí, para variar el precio de la alerta.

- ¿Sabrías activar o desactivar todas las alertas?

  Sí, arriba a la derecha.

- ¿Y de forma individual?

Sí, en cada elemento aparece una campanita.

8. Vehículo y descuentos

- ¿Por qué piensas que se pide el consumo medio y la capacidad total del depósito?

  Es posible que sea para calcular algo, aunque no estoy segura.

- ¿Sabrías añadir y eliminar un descuento?

  Sí, lo indica claramente.

9. Conclusión y mejoras

- ¿Qué te parecen los diseños? ¿Los ves claros y concisos?

  Los veo bien, me gustan.

- ¿Qué añadirías?

  Un filtro por marca, no me interesa ver los precios de todas las marcas.

- ¿Qué cambiarías?

  En el mapa, la etiqueta de color verde.

- ¿Qué eliminarías?

  Nada.

- ¿Utilizarías la aplicación?

  Sí, claro.

10.3.3. Usuario 3: Roberto

1. Información general

- ¿Usas con frecuencia aplicaciones para encontrar precios económicos de cualquier producto?

  Solo utilizo una aplicación de chollos y otra para ver los precios de los combustibles.

- ¿Utilizas aplicaciones para encontrar el mejor precio del combustible en tu zona?

  Sí, a menudo.

2. Pantalla de inicio

- ¿Cómo visualizas la pantalla de inicio?

  Está muy bien.

3. Pantalla de inicio

- ¿Entiendes y comprendes la barra de navegación inferior?

  Sí, para ir a cada pantalla. Los textos además lo clarifican.

4. Listado de precios

- ¿Encuentras el botón de buscar? ¿Qué funcionalidad piensas que tiene?

Sí, está arriba. Supongo que para buscar algún sitio o una estación de servicio.

- ¿Puedes determinar qué combustible se está utilizando para filtrar los resultados? ¿Sabrías cambiar de combustible?

  Diesel. Al pulsar sobre él, visualizaré otros tipos de combustibles.

- ¿Comprendes cada elemento que se lista en la pantalla?

  Sí. Logo, nombre, ubicación, valoración, favoritos, precio, fecha de actualización, precio por llenar el depósito y navegación.

- ¿Puedes encontrar el icono para marcar o desmarcar favoritos?

  Sí,

- ¿Y el de navegar?

  Si.

- ¿Sabrías entrar en los detalles de una estación de servicio?

  Sí, al pulsar sobre un elemento del listado.

5. Detalles

- ¿Puedes encontrar las opciones de marcar como favorito? ¿Y navegar?

  Sí.

- ¿Comprendes qué significa valorar la estación de servicio y la marca del carburante?

  Sí, igual que otros sistemas de valoración.

6. Mapa

- ¿Visualizas correctamente los marcadores en el mapa?

  Sí, el logo se ve claramente.

- ¿Y los precios?

  El verde no lo veo del todo bien, pero el resto sí.

7. Alertas

- ¿Entiendes qué significan los iconos ubicados a los lados de la etiqueta del precio?

  Sí, de esta forma disminuye o aumenta el precio por el que seré notificado.

- ¿Sabrías activar o desactivar todas las alertas?

  Sí, arriba.

- ¿Y de forma individual?

  Sí, en cada elemento veo una campana.

8. Vehículo y descuentos

- ¿Por qué piensas que se pide el consumo medio y la capacidad total del depósito?

  Supongo que para calcular el coste por llenar el depósito al completo.

- ¿Sabrías añadir y eliminar un descuento?

  Sí, veo un texto que indica añadir, y luego el icono de la papelera para eliminarlos.

9. Conclusión y mejoras

- ¿Qué te parecen los diseños? ¿Los ves claros y concisos?

  Me gusta el diseño escogido para la aplicación. Lo veo todo muy bien, claro y conciso.

- ¿Qué añadirías?

  Pues, principalmente añadiría un filtro para no visualizar todas las marcas. También la posibilidad de ver las estaciones de servicio a partir de una ruta.

- ¿Qué cambiarías?

  Nada.

- ¿Qué eliminarías?

  Nada.

- ¿Utilizarías la aplicación?

  Sí, por supuesto.