



ULPGC

**Universidad de
Las Palmas de
Gran Canaria**

**Escuela de
Ingeniería Informática**



Asistente autogestor de equipaje

TITULACIÓN: GRADO EN INGENIERÍA INFORMÁTICA

AUTOR: MIGUEL ÁNGEL BERCIANO RODRÍGUEZ

TUTORIZADO POR: ZENÓN JOSÉ HERNÁNDEZ FIGUEROA

julio de 2021



ULPGC
Universidad de
Las Palmas de
Gran Canaria

Escuela de
Ingeniería Informática



TFT04

SOLICITUD DE DEFENSA DE TRABAJO DE FIN DE TÍTULO

D/D^a MIGUEL ÁNGEL BERCIANO RODRÍGUEZ, autor/a del Trabajo de Fin de Título ASISTENTE AUTOGESTOR DE EQUIPAJE

correspondiente a la titulación GRADO EN INGENIERÍA INFORMÁTICA (PLAN 2010), en colaboración con la empresa/proyecto (indicar en su caso)

SOLICITA

que se inicie el procedimiento de defensa del mismo, para lo que se adjunta la documentación requerida, haciendo constar que

☒ se autoriza / ☐ no se autoriza, la grabación en audio de la exposición y turno de preguntas.

Asimismo, con respecto al registro de la propiedad intelectual/industrial del TFT, declara que:

☐ Se ha iniciado o hay intención de iniciarlo (defensa no pública).
☒ No está previsto.

Y para que así conste firma la presente. (fecha en firma electrónica)

El/la estudiante

BERCIANO
RODRIGUEZ
MIGUEL ANGEL -

Fdo.: 44748718X

Firmado digitalmente por
BERCIANO RODRIGUEZ
MIGUEL ANGEL - 44748718X
Fecha: 2021.07.13 16:08:12
+01'00'

A rellenar y firmar **obligatoriamente** por el/la/los/las tutor/a/es/as.
En relación a la presente solicitud, se informa: (firmar donde corresponda)

Positivamente

Fdo.: _____

Negativamente (la justificación en caso de informe negativo deberá incluirse en el TFT05)

Fdo.: _____

DIRECTORA DE LA ESCUELA DE INGENIERÍA INFORMÁTICA

Ver.16B

4/2021



ULPGC
Universidad de
Las Palmas de
Gran Canaria

Escuela de
Ingeniería Informática



RESUMEN

A la hora planificar un viaje hay que tener en cuenta varios aspectos de éste como son el destino, los integrantes, el transporte, pero sobre todas estas cosas, la más importante es el equipaje. Para viajar con mayor comodidad y seguridad es importante planificarlo bien y dedicar parte del tiempo a organizarlo y revisarlo.

El objetivo de este proyecto es desarrollar la aplicación *JustInCase* para dispositivos Android cuyo propósito es ser un asistente de gestión de maletas para ayudar al usuario a controlar y organizar el equipaje a la hora de realizar un viaje.

Con esta aplicación, aparte de monitorizar las diferentes maletas que diseñe el usuario, se puede solicitar sugerencias de equipaje en función de las actividades que se vayan a realizar, diseñar plantillas de maletas y crear grupos de viajes con los que poder visualizar las maletas de sus integrantes.

ABSTRACT

When traveling, the users have to take care about some parts of the travel, such as the destination, the members or the transport, but, above all, the most important thing is the luggage. It is important to plan the travel well and spend time organizing and checking to ensure a comfortable and safe trip.

The objective of this project is to develop the application called *JustInCase* for Android devices. It's purpose is to be a luggage management assistant to help users to control and organize luggage when travelling.

In addition to monitoring the different suitcases that the user designs, this application allows the user to request luggage suggestions based on the activities, design suitcase templates and create travel groups for the user to view the member's suitcases.



ÍNDICE GENERAL

| | |
|---|----|
| 1 Introducción | 7 |
| 1.1 El turismo en la actualidad | 8 |
| 1.2 El problema con las maletas | 11 |
| 1.3 Objetivos del proyecto | 12 |
| 2 Metodología | 13 |
| 2.1 Tecnologías utilizadas..... | 15 |
| 2.2 Planificación | 23 |
| 3: Desarrollo | 27 |
| 3.1 Arquitectura | 40 |
| 3.2 Diseño..... | 46 |
| 3.3 Inconvenientes en el desarrollo | 52 |
| 3.4 Testeo..... | 54 |
| 4 Trabajo futuro | 56 |
| 5 Conclusiones..... | 58 |
| Bibliografía | 59 |
| ANEXO I: MANUAL DE USUARIO | 64 |

ÍNDICE DE FIGURAS

| | |
|---|----|
| Figura 1 Llegada de turistas internacionales (millones) e ingresos por turismo internacional (miles de millones de usd)..... | 8 |
| Figura 2: llegada de turistas internacionales a España entre 2001 y 2019..... | 9 |
| Figura 3: fragmento del cuadro de diálogo en la creación de un proyecto extraída del ide android studio | 16 |
| Figura 4: captura de pantalla del recurso xml fragment_create_trip.xml..... | 17 |
| Figura 5: teléfono samsung a50 (a la izquierda) por delante y (a la derecha) por detrás | 18 |
| Figura 6: imagen comparativa entre dos teléfonos de misma dimensiones, pero con diferente densidad de pantalla..... | 19 |
| Figura 7: imagen comparativa de densidades de pantalla..... | 20 |
| Figura 8: variación del escalado de imagen usando píxel estándar en diferentes resoluciones..... | 21 |
| Figura 9: variación del escalado de imagen usando píxel independiente de la densidad en diferentes resoluciones..... | 21 |
| Figura 10: diagrama de casos de uso del proyecto..... | 23 |
| Figura 11: diagrama de clases del proyecto | 24 |
| Figura 12: capturas de pantalla del prototipo del proyecto | 25 |
| Figura 13: capturas de pantalla de las interfaces de (a la izquierda) Suitcase lite, (al centro) PackKing y (a la derecha) MyTravelSuitcase..... | 26 |
| Figura 14: diferencia entre una actividad y un fragmento | 27 |
| Figura 15: ciclos de vida (a la izquierda) de un fragmento y (a la derecha) de una actividad | 28 |
| Figura 16: gráfico de navegación de la aplicación..... | 29 |
| Figura 17: representación del comportamiento de la pila de actividades..... | 30 |
| Figura 18 representación de la creación de múltiples instancia de una misma actividad..... | 30 |
| Figura 19: captura de pantalla del archivo drawer_menu.xml, a la izquierda el código xml y a la derecha el diseño resultante | 31 |
| Figura 20: captura de pantalla de los ficheros (a la izquierda) nav_graph.xml y (a la derecha) drawer_menu.xml del proyecto..... | 31 |
| Figura 21: captura de pantalla del método onOptionsItemSelected() de la clase MainActivity..... | 32 |
| Figura 22: drawer menu de la aplicación, a la izquierda sin desplegar y a la derecha desplegado..... | 32 |
| Figura 23: diagrama de flujo de trabajo para declarar y solicitar permisos en tiempo de ejecución de una aplicación Android..... | 33 |
| Figura 24: captura de pantalla del método bindCamera() en la implementación de CameraX..... | 34 |
| Figura 25: captura de pantalla del método takephoto() en la implementación de camerax | 35 |
| Figura 26: declaración de la clase Adapter_Item..... | 36 |
| Figura 27: implementación de los métodos onCreateViewHolder(), onBindViewHolder() y getItemCount() de la clase Adapter_Item..... | 37 |
| Figura 28: captura de pantalla de (arriba) la clase AdapterViewHolder para Adapter_Item y (abajo) del recurso xml card_view_item.xml..... | 37 |
| Figura 29: captura de pantalla del recurso recyclerview_addbutton.xml..... | 38 |
| Figura 30: extracto del código de la clase Fragment_ShowItems..... | 38 |
| Figura 31: capturas de pantalla de la aplicación en ejecución | 39 |
| Figura 32: captura de pantalla de (a la izquierda) la lista de ítems y de (a la derecha) la lista de ítems filtrada de la aplicación en ejecución | 39 |
| Figura 33: representación de los patrones (a la izquierda) MVC y (a la derecha) MVP | 40 |
| Figura 34: captura de pantalla de (a la izquierda) la lista de ítems y (a la derecha) la lista de ítems separada por categorías..... | 42 |
| Figura 35: extracto del código de la clase Fragment_ShowBaggageByItem | 42 |
| Figura 36: esquema de la base de datos de la aplicación..... | 43 |
| Figura 37: Diagrama de la arquitectura de Room | 44 |
| Figura 38: declaración de la clase Item | 45 |
| Figura 39: captura de pantalla de la clase ItemDao | 45 |

| | |
|--|----|
| Figura 40: captura de pantalla del archivo themes.xml..... | 46 |
| Figura 41: captura de pantalla del archivo AndroidManifest.xml | 46 |
| Figura 42: captura de pantalla del archivo colors.xml..... | 47 |
| Figura 43: captura de pantalla de la herramienta para la creación de paletas de colores de la página web Design Material | 48 |
| Figura 44: análisis de capturas de pantalla de la aplicación en ejecución, a la izquierda la pantalla “inventario” sin ítems y, a la derecha, con ítems..... | 49 |
| Figura 45: capturas de pantalla de (a la izquierda) “Equipaje” de un viaje, (al centro) “categorías” y (a la derecha) “Mis maletas” | 50 |
| Figura 46: captura de pantalla de (a la izquierda) “Mis viajes”, (al centro) “agregar ítems” a una maleta y (a la derecha) “generar maleta” | 51 |
| Figura 47: extracto de código del proyecto de la clase Adapter_Item..... | 53 |
| Figura 48: captura de pantalla del fichero build.gradle..... | 55 |
| Figura 49: captura de pantalla de la ventana principal de la aplicación | 64 |
| Figura 50: presentación del paso 1 y 2 para crear un nuevo viaje..... | 65 |
| Figura 51: presentación del paso 3 y 4 para crear un nuevo viaje..... | 65 |
| Figura 52: presentación del paso 5 para crear un nuevo viaje | 65 |
| Figura 53: presentación del paso 5.b y 6 para crear un nuevo viaje..... | 65 |
| Figura 54: representación de “Hacer check-in” de una maleta | 65 |
| Figura 55: presentación del paso 1 y 2 para crear un nuevo ítem | 65 |
| Figura 56: presentación del paso 3 y 4 para crear un nuevo ítem | 65 |
| Figura 57: cuadros de diálogo para agregar una nueva categoría, plantilla y maleta..... | 65 |
| Figura 58: presentación del paso 5 crear un nuevo ítem..... | 65 |
| Figura 59: captura de pantalla de la lista de ítems, a la izquierda, con solo un ítem creado y, a la derecha, con varios ítems creados | 65 |
| Figura 60: representación del procedimiento para editar un ítem | 65 |
| Figura 61: representación del procedimiento para eliminar un ítem..... | 65 |
| Figura 62: representación del paso 1 para agregar un equipaje a una maleta | 65 |
| Figura 63: pantalla de “equipaje” de una maleta de un viaje..... | 65 |
| Figura 64: representación del paso 2.a para agregar un equipaje a una maleta | 65 |
| Figura 65: representación del paso 2.a para agregar un equipaje a una maleta | 65 |
| Figura 66: representación del paso 2.a para agregar un equipaje a una maleta | 65 |

1 Introducción

Viajar y hacer turismo son dos actividades que realizan miles de personas cada año, e involucran a varios sectores y que suponen un gran impacto en la economía de varios países. La Real Academia de la lengua española define viajar como: *“Trasladarse de un lugar a otro, generalmente distante, por cualquier medio de locomoción”* [1], por lo que cualquier desplazamiento, ya sea a otra ciudad u otro país, supone hacer un viaje. Por otro lado, según la **Organización Mundial del Turismo** (en adelante, OMT) el turismo es un fenómeno social, cultural y económico que supone el desplazamiento; es decir, la realización de un viaje, de personas, denominadas viajeros, a países o lugares fuera de su entorno habitual, ya sea por motivos personales, profesionales o de negocios. Aparte, el turismo abarca las actividades que se llevan a cabo, algunas de las cuales pueden suponer un gasto turístico [2]. El turismo ha ido creciendo de forma continuada a la par que se iba diversificando, siendo uno de los sectores económicos que mayor rapidez crece en el mundo. Por otro lado, el turismo mundial guarda una estrecha relación con el desarrollo por lo que, con el paso del tiempo van aumentando el número de nuevos destinos, convirtiéndose en un motor clave del progreso socioeconómico [3].

Debido a ese crecimiento, también han aumentado las empresas involucradas tales como las aerolíneas y los servicios de alojamiento como los hoteles y casas rurales, generando más oferta y teniendo que poner precios más competitivos. Durante años, las agencias de viaje tomaron un papel fundamental, actuando como intermediarios entre clientes y empresas. Para reservar y planificar un viaje, lo más cómodo era acudir a una de esas agencias, ya que incluía la posibilidad de organizar viajes completos (vuelo, estancia y actividades), además de disponer de un amplio abanico de ofertas. Sin embargo, con la aparición de internet, todo este sector sufrió una revolución iniciando una mudanza a las plataformas digitales, lo que permitió acercar, aún más, la organización y realización de viajes a los propios turistas, quienes pueden realizarlos desde cualquier dispositivo y de manera personalizada.

Estos cambios también se aplican a las diferentes páginas webs de las aerolíneas, que permiten tanto visualizar las previsiones de vuelos, como poder comprar un billete directamente. También encontramos los llamados “motores de búsqueda de vuelos y hoteles”, que sirven como tablón de exposición de diferentes ofertas. Junto con las plataformas digitales, también llegaron las aplicaciones móviles; algunas similares a los buscadores y otras, como Airbnb, que cambiaron el sector de la hostelería permitiendo a cualquier particular transformar una propiedad privada en una vivienda de alquiler vacacional.

A la hora de realizar un viaje hace falta planificarlo previamente: destino, fechas, transporte, alojamiento... un sinfín de cosas que preparar antes de salir. Organizarse antes y durante un viaje es fundamental para evitar contratiempos y disgustos. Todo esto repercute en el equipaje que llevemos pues, dependiendo del lugar, el clima y las actividades a realizar se necesitará, o no, llevar ciertas cosas. Esto se complica si se viaja en grupo, ya sea con amigos, en familia o si llevamos mascotas.

Al no existir una guía o normativa acerca de “*qué se debe llevar en un viaje*”, es muy frecuente acabar cargando con maletas innecesariamente grandes que son más difíciles de transportar y manejar. Aun teniendo en cuenta ciertas restricciones como el peso máximo, el propio tamaño de la maleta, los días o el clima, es muy común aplicar el “por si acaso” y muchas veces se acabe por llevar más equipaje del que se necesita. Para los viajes en avión, esto puede encarecer el viaje, ya que cada vez son más las aerolíneas que ofertan billetes más baratos a cambio de cobrar más por las maletas grandes, e incluso, si la maleta sobrepasa el peso permitido, pagar más por cada kilo de exceso.

En definitiva, la maleta es tan importante como el viaje en sí mismo y, aunque planificar una maleta parezca sencillo, no son pocas las ocasiones en las que echamos de menos no habernos llevado algo concreto, o que, al contrario, nos sobra equipaje. Por si fuera poco, también están los casos en los que, al hacer la maleta para regresar del viaje, nos olvidamos cosas dejándolas como objetos perdidos.

1.1 El turismo en la actualidad

Aunque existe desde la antigüedad, no es hasta la mitad del siglo XX cuando comienza a ser más asequible para más personas a lo largo del mundo, sobre todo para la clase media. Esto se debe a varios factores como el aumento del tiempo libre y mejoras salariales, además del abaratamiento de los medios de desplazamiento destacando el transporte aéreo, pasando de ser un producto considerado de lujo debido a su alto coste a ser de los medios más utilizados.

En el informe “*Panorama del turismo internacional - Edición 2019*” desarrollado por la OMT, determina que 2018 fue el noveno año consecutivo de crecimiento sostenido del turismo internacional, generando más de 1.400 millones de dólares, siendo la tercera mayor categoría de exportaciones del mundo [4].

En la siguiente gráfica lineal (Figura 1), extraída del informe, están representados, año por año desde 1995 hasta el 2018, el total de turistas internacionales (en millones) y el total de ingresos que éstos generan (en miles de millones de dólares).

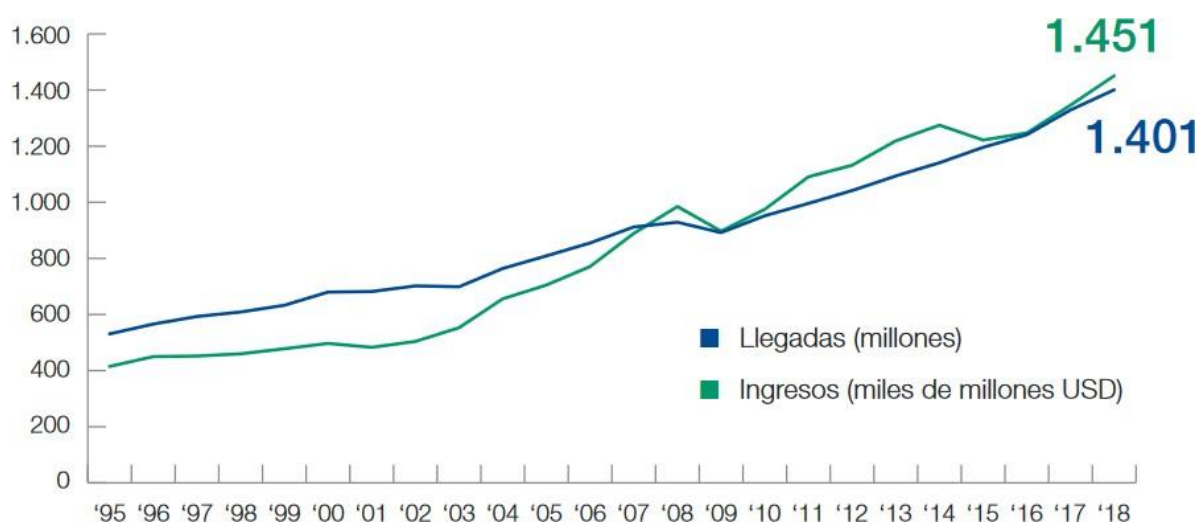


FIGURA 1 LLEGADA DE TURISTAS INTERNACIONALES (MILLONES) E INGRESOS POR TURISMO INTERNACIONAL (MILES DE MILLONES DE USD)

Analizando la gráfica, vemos que el número de turistas no deja de crecer nunca, salvando un segmento puntual entre 2008 y 2009, cuya causa podría asociarse a la crisis financiera de 2008. Aun así, se puede observar que desde el año 1995 hasta el 2018 casi se ha triplicado el tráfico de turistas. Es a partir del 2010 cuando este crecimiento adquiere una pendiente más pronunciada lo que significa que dicho aumento es cada vez más rápido.

Por otro lado, los ingresos por turismo guardan una relación lineal con los turistas pues, a mayor número, mayores son los ingresos; sin embargo, este valor se ve afectado por otras variables, pues se aprecian picos y valles que no se explican. Si bien la cantidad de turistas es importante, también influye mucho el tipo de turismo y otras características que hacen que un viaje sea más, o menos, caro.

A causa de la pandemia mundial del SARS-CV-2 [5], más conocida como COVID-19, el número de turistas ha bajado drásticamente durante los años 2020 y 2021, consecuencia de las cuarentenas y restricciones impuestas en los diferentes países. [6]. A pesar de ello, el sector turístico sigue siendo un principal motor de la economía mundial y, previsiblemente, en cuanto se haya solucionado completamente, se volverá a la normalidad, al igual que sucedió tras la crisis económica del 2008.

En 2018, España, junto con Estados Unidos y Francia, lidera el sector turístico a nivel mundial, siendo el segundo país que más turistas internacionales (cerca de 83 Millones) recibe del mundo y el segundo en la recepción de ingresos por gasto turístico (74 Millones de dólares aproximadamente) [4]. En el siguiente gráfico (Figura 2), proporcionado por la página *epdata* [7], se puede apreciar la evolución de turistas que llegan cada año a España desde el 2001 hasta el 2019. Al igual que con el turismo mundial, éste no deja de crecer salvo el segmento entre los años 2007 y 2009.



FIGURA 2: LLEGADA DE TURISTAS INTERNACIONALES A ESPAÑA ENTRE 2001 Y 2019

Por otro lado, cada año está dividido en temporadas, las cuales van en función de la afluencia de turistas. En las temporadas altas vienen más turistas y, generalmente, se corresponden con los periodos vacacionales, siendo estas fechas señas cuantos más viajes se realizan. También depende del atractivo turístico del lugar; por ejemplo, un sitio con un clima cálido en las vacaciones de navidad tenderá a un “turismo de sol y playa”, mientras que las estaciones de esquí tendrán más excursionistas y “turismo de nieve”.

Uno de los rasgos clave del turismo en España es la diversidad de destinos turísticos, los cuales podemos agrupar en cuatro: litoral, montaña, rural y urbano.

- **Turismo litoral:** coloquialmente conocido como “turismo de sol y playa” o “turismo tradicional”, se da en localidades costeras en las que se encuentran playas y la mayoría de tiempo, las condiciones climáticas son de tiempo soleado y temperaturas suaves.
- **Turismo rural:** las actividades se realizan en un espacio natural; pequeñas localidades o entornos no urbanos como bosques, praderas, campo, ríos o lagos.
- **Turismo urbano:** tiene lugar en las ciudades, las cuales ofrecen diferentes experiencias y productos culturales, arquitectónicos o sociales como son teatros, museos, catedrales monumentos, parques y edificios emblemáticos.
- **Turismo de montaña:** tiene lugar en un espacio geográfico definido como son las colinas o montañas y engloba un amplio espectro de actividades de ocio y deporte al aire libre.

Estos tipos de turismo no son excluyentes, de manera que se pueden combinar o ir variando a lo largo de un viaje. Aparte de los mencionados anteriormente, existen más tipos de turismo como el gastronómico, cultural, sostenible o el deportivo, los cuales no están sujetos a un entorno concreto como si ocurre con los anteriores.

Asimismo, España posee una desarrollada infraestructura en el sector servicios, un gran patrimonio artístico y cultural y una riqueza gastronómica. Todo esto convierte al país en uno de los países más visitados del mundo y el impacto turístico representa alrededor del 11% de su producto interior bruto (PIB). A pesar de ello, la mayoría de los turistas extranjeros provienen de países del centro y norte de Europa, cuya motivación principal es la realización del turismo litoral, el cual sigue predominando, siendo algunas zonas dependientes exclusivamente de él [8] [9].

1.2 El problema con las maletas

Como se ha comentado anteriormente, hacer el equipaje; es decir, planificar y organizar lo que nos queremos llevar en un viaje, puede resultar en una tarea bastante complicada. A más cosas se añadan, más grande y pesada se hace la maleta, incluso pudiendo llegar el caso de que se necesite más de una maleta para poder transportar todo lo que se quiera llevar en el viaje. Optimizar el espacio es clave para poder viajar con mayor comodidad.

Existe una gran variedad de maletas: de diferentes colores, formas, tamaño e incluso hechas de distintos materiales como plástico, tela o piel. A la hora de la verdad, la mejor maleta es aquella que pueda adaptarse mejor a las necesidades del viajero en el momento. Por ejemplo, para los excursionistas viene mejor una mochila para poder llevarla fácilmente a la espalda; para viajes de negocio de pocos días un maletín y para largas vacaciones la maleta tradicional.

En cualquier caso, la maleta tiene una capacidad máxima, por lo que no podemos llevarnos todos los objetos que nos gustaría e incluso pudiendo, sería difícil de transportar debido al peso, por lo que es necesario ir seleccionando que cosas queremos llevar y cuáles no. Esto implica que, dentro del amplio abanico de ítems disponibles, existen unos que tienen más prioridad que otros como, por ejemplo, documentación, medicina o ropa. Sin embargo, no hay peor sensación que la de darse cuenta de que se ha olvidado algo que sí se quería llevar, aunque no fuera imprescindible.

Si además se utiliza el avión como medio de transporte, facturar maletas grandes sale mucho más caro pues, como se ha comentado anteriormente, las compañías aéreas cobran más por las maletas grandes. Existe un movimiento cultural denominado “minimalismo viajero” dentro del concepto de “minimalismo” [10] que propone el reducir al máximo lo necesario para viajar de manera que, todo lo que se vaya a utilizar, puedas llevarlo contigo en una sola mochila.

Planificar una maleta es, en última instancia, preparar una lista de los ítems que uno quiera llevarse consigo y donde hay que tener en cuenta muchas variables como son: destino, clima, tiempo, grupo de personas, tipo de maleta, tamaño de la maleta, cantidad de maletas, etcétera. El problema está en que dicha lista no está vinculada a la maleta en sí; es decir, se puede hacer una lista de ítems para controlar que se lleva todo lo necesario y darse el caso de que la lista sea demasiado grande, lo que nos lleva al problema inicial. Al mismo tiempo, la manera en la que se coloquen los ítems influye en la cantidad que se pueda transportar. Existen consejos y formas de como colocarlos para optimizar el espacio disponible.

Ante este problema se han ido desarrollando diferentes soluciones informáticas como son las páginas webs, blog o revistas con consejos, experiencias y trucos. Sin embargo, son las aplicaciones móviles las que toman un papel más fundamental gracias a que son más accesibles, manejables y compatibles con el proceso de hacer una maleta.

Si hacemos una búsqueda en la “app Store” de Google [11], a día de hoy encontraremos varias aplicaciones relacionadas a la gestión de maletas. Aunque todas cumplen con el mismo propósito de ayudar hacer una maleta para un viaje, algunas están orientadas solo a viajes de avión, donde tienen un apartado para que se pueda guardar información acerca del vuelo o calcular aproximadamente el peso de la maleta en base a los ítems introducidos.

1.3 Objetivos del proyecto

Este proyecto consiste en el desarrollo de una aplicación móvil para dispositivos Android que el usuario pueda utilizar para gestionar, de forma organizada, la preparación del equipaje tanto en el momento de salir de viaje; es decir, ítems necesarios en función de las características del destino y duración del viaje, como al regreso, con comprobación de que no se está dejando ningún ítem.

Se podrán visualizar los ítems disponibles/necesarios/usados/pendientes en diferentes formatos y las recomendaciones apropiadas en función del viaje. Las listas de ítems podrán estar constituidas por un amplio abanico de información, incluyendo imágenes y fotografías. Se podrán crear grupos de listas para los viajes en grupo de manera que se pueda visualizar el estado del equipaje individual y compartido de todos sus miembros.

Para diferenciarse del resto de aplicaciones se construirá en base a tres ideas fundamentales:

1. **Simplificar y facilitar la creación de maletas:** mientras el resto de las aplicaciones, a nuestro parecer, ofrecen unas interfaces complejas, engorrosas y llenas de objetos; en este proyecto se optará por un diseño más limpio y simple, dejando espacio para que los diferentes elementos puedan respirar y separando las diferentes utilidades en sus respectivas ventanas.
2. **Adaptarse al usuario:** como se ha explicado, viajar es una actividad que involucra muchos aspectos y variables a tener en cuenta, además de que cada persona afronta la preparación y realización del viaje de una manera distinta. Con esto en mente, es importante permitir la mayor cantidad de posibilidades para hacer la maleta y que el usuario escoja la que más se le adecúe.
3. **Darle la responsabilidad al usuario:** al mismo tiempo que se presenta una gran versatilidad a la hora de utilizar la aplicación, es esencial tener en cuenta que es el usuario quien crea sus maletas, categorías e ítems, que las cree como él convenga, a diferencia del resto de aplicaciones que vienen con ítems ya preestablecidos. En algunas, incluso, no permiten crear sus propias categorías o ítems o para llevarlo a cabo hace falta suscribirse a la aplicación.

2 Metodología

Para llevar a cabo el proyecto se optó por utilizar una metodología desarrollo ágil inspirada en el método SCRUM; es decir, dentro del marco de trabajo SCRUM, se han seleccionado una serie de características para llevar a cabo un desarrollo iterativo e incremental. Para ello, se plantea la realización un producto mínimo viable, el cual consiste en una aplicación base que debe ser capaz de cumplir con los requisitos mínimos planteados, y, a partir de éste, se empieza a iterar. En cada iteración, se plantean, implementan y prueban nuevas funcionalidades. Este ciclo se va repitiendo de manera que la aplicación va consiguiendo mayor profundidad tanto a nivel de diseño como de lógica, pero conservando el núcleo; es decir, la base sobre la cual se empieza el desarrollo.

Cada vez que se termine un ciclo se realiza una retrospectiva de este, en la que se envía un correo al tutor indicando las características agregadas, problemas e inconvenientes que hayan surgido a lo largo del desarrollo, las nuevas funcionalidades a agregar en la siguiente iteración más una estimación del tiempo que llevarían llevarlas a cabo.

Para plantear cada actualización, se utilizó un sistema similar las historias de usuarios [12] [13] en el que se elaboró una lista de funcionalidades que debe tener la aplicación y a cada una se le asignó su descripción, nivel de complejidad, prioridad en el desarrollo y una estimación de tiempo. Además, para la redacción de esta lista se siguió la regla *INVEST* [14], la cual se trata de un acrónimo que recoge una serie de criterios que sirven para comprobar la calidad de una historia de usuario. Estos criterios son:

- **Independent:** cada historia de usuario debe ser independiente de las otras.
- **Negotiable:** las historias deben ser negociables por lo que se debe evitar historias de usuario con demasiados detalles para garantizar su flexibilidad.
- **Valuable:** todas las historias de usuario deben aportar algo de valor, ya sea para el cliente o para el usuario final.
- **Estimable:** las historias de usuario deben ser estimables.
- **Small:** las historias de usuario deben ser pequeñas, de manera que su producción no abarque varias iteraciones de trabajo.
- **Testable:** todas las historias deben ser verificables.

Gracias a este procedimiento, se simplifica la creación y selección de las diferentes funcionalidades que se vayan a añadir al proyecto, con lo que se obtiene un mayor control en lo que se está trabajando y, además, permite generar un histórico de la aplicación, donde se puede observar el progreso de esta en cada una de las iteraciones.

La idea, como se ha mencionado anteriormente, es seguir una serie de pautas recogidas dentro del marco de trabajo SCRUM, pero sin llegar a utilizar la metodología al completo, donde se utilizan conceptos como la pila de producto, las retrospectivas y los sprints y se adaptan al desarrollo de este proyecto.

Por otro lado, previamente al inicio de este proyecto se ha trabajado en un “*Sprint 0*” donde el objetivo es desarrollar un producto simple, en este caso, en Android, en un corto periodo de tiempo con el objetivo de familiarizarnos con el entorno y la tecnología. Para este *Sprint 0* se planteó el desarrollo de una calculadora simple y de esta manera tener una toma de contacto con el diseño de interfaces y la lógica de las aplicaciones móviles.

Respecto al proyecto a llevar a cabo, éste se dividió en tres etapas diferenciadas, siendo la primera etapa la fase de planificación (Sección 2.2); en ella se plantea la idea a llevar cabo y el procedimiento a seguir. Una vez obtenida la idea general de producto, se esboza un prototipo para obtener el diseño, las funcionalidades y su interacción con el usuario, además de plantear los casos de uso, el diagrama de clases, el diagrama de flujo y las diferentes entidades que intervienen en el proceso.

Después de la fase de planificación comienza la fase de desarrollo (Sección 3). Para este proyecto hay que tener en cuenta que se va a trabajar tanto en un lenguaje como un entorno menos conocido, puesto que no existe ninguna asignatura en la carrera en la que se imparta desarrollo de aplicaciones para dispositivos móviles, por lo que, conjuntamente a la creación del producto, se irá realizando un aprendizaje.

El primer objetivo de esta fase es obtener el producto mínimo viable: una aplicación que reúna los requisitos principales para ser funcional y que sirva como base sobre la que construir el resto. En este caso que se trata de la creación y gestión de maletas, el propósito será crear un programa que tenga la capacidad de crear un viaje, al cual se le pueda asociar una maleta y una lista de ítems.

Una vez se ha obtenido, se pasa al siguiente paso: agregar nuevos métodos, como son poder asociar fechas a un viaje, permitir agrupar ítems por medio de categorías, agregar fotos de ítems, etc. La forma en la que se van agregando es totalmente modular, de manera que el núcleo del proyecto sigue siendo el mismo en cada versión.

Al mismo tiempo, para cada una de las entregas, hay un proceso de revisión continua con el objetivo simplificar y ajustar de cara a mantener un código limpio, optimizado y ordenado siguiendo diferentes pautas, entre ellas, las descritas en el manual *clean code* [15] el cual agrupa una serie de principios que ayudan a construir el código más intuitivo y, en consecuencia, más entendible y sencillo de modificar.

Una vez se han agregado todas las funcionalidades se pasa a la tercera y última fase de desarrollo que consiste en probar y corregir el código y ajustar detalles de estilo (Sección 3.3). En esta última fase, el objetivo es comprobar que la aplicación funciona correctamente tanto a nivel lógico (que no haya errores o excepciones) como a nivel funcional.

2.1 Tecnologías utilizadas

A lo largo del desarrollo de este proyecto se han utilizado varios entornos y herramientas para poder llevar a cabo las diferentes tareas como son la programación de la aplicación, su testeo, la comunicación con el tutor o la realización de la memoria.

Para la redacción de documentación se ha hecho uso del programa de procesamiento de textos de la compañía *Microsoft* llamado *Microsoft Word* [16]. Se ha escogido este entorno frente a otros por su sencillez, versatilidad y experiencia en el mismo, siendo además compatible con otras herramientas utilizadas, como el servicio de correo electrónico *Outlook* [17]. Este portal también perteneciente a la compañía *Microsoft*, es el utilizado para la correspondencia con el tutor. Ambos poseen la capacidad de guardar archivos de forma automática en la nube, un espacio virtual asociado a la cuenta *Microsoft* del estudiante, proporcionada por la universidad. En caso de pérdida de datos a nivel local, se podrán recuperar de una copia sin inconvenientes.

Para la creación de prototipos se utilizó la herramienta *Balsamiq* [18]. Se trata de una aplicación que permite la creación de prototipos y con la que, gracias a sus múltiples funciones, es posible diseñar interfaces de usuario con mucho detalle permitiendo conseguir una experiencia cercana a la interacción con una aplicación real. Es bastante flexible, sencilla y cómoda de usar y, además, es posible exportar los proyectos de forma automática a formato PDF, facilitando el poder compartirlo y exponerlo en cualquier dispositivo manteniendo los enlaces interactivos. Por otro lado, existe una amplia variedad de tutoriales y documentación que ayudan al uso de esta herramienta. Todo esto hace que se haya escogido *Balsamiq* frente a otras alternativas.

Para la creación de diagramas se usó *StartUML* [19] de *MKLab*, siendo una herramienta que trabaja con el lenguaje unificado de modelado UML [20]. *StartUML* permite la creación de varios tipos de diagramas entre los que encontramos diagramas de caso de uso, de actividades, de clases y de secuencia. Es una aplicación sencilla que se ha utilizado en varias asignaturas de la carrera por lo que ya se tienen conocimientos de su funcionamiento, razón adicional por haberla seleccionado.

En lo referente al desarrollo del código, al ser una aplicación móvil para dispositivos Android, se ha hecho uso del entorno Android Studio [21]. Basado en *IntelliJ IDEA*, Android Studio es el entorno de desarrollo integrado o IDE por sus siglas en inglés Integrated Development Environment, para el desarrollo de productos Android. Al ser una herramienta oficial y gratuita, cuenta con una amplia variedad de versiones y librerías que se adaptan en función del desarrollo que se planea llevar a cabo.

Para programar una aplicación móvil hace falta definir que lenguaje se va a utilizar pudiendo elegir entre *Java*, *C++* o *Kotlin*. En este caso se procederá con *Java*, dado a que es el lenguaje que más práctica se tiene y por el cual se han impartido varias asignaturas en la carrera, abarcando diferentes áreas del conocimiento en programación.

También hay que tener en cuenta la versión de Android con la que se va a trabajar, pues si se escoge una versión muy nueva la aplicación no podrá ser instalada en móviles más viejos. Por lo tanto, a menor versión de Android utilicemos, más dispositivos móviles podrán acceder a la aplicación, pero a costa de no poder utilizar funciones y librerías más nuevas. Sin embargo, las versiones de Android son diferentes a las versiones de la API Android. La *Application Programming Interface* o API es la encargada de la comunicación entre los diferentes productos y servicios, sin necesidad de saber cómo están implementados.

En el caso de Android, la API conecta el Sistema Operativo del dispositivo con cualquier aplicación ya sea interna o externa, simplificando en gran medida el desarrollo y otorgando gran flexibilidad. De esta manera, el nivel de API es un valor entero que identifica el marco de trabajo o *framework*; es decir, el conjunto de paquetes, librerías, elementos, atributos xml, permisos y recursos disponibles que ofrece una determinada versión de Android.

Un detalle importante que destacar sobre las actualizaciones de la API de Android, es que están diseñadas para que continúen siendo compatible con versiones anteriores. Esto es porque la mayoría de los cambios en la API es para agregar nuevas funcionalidades o reemplazar viejas, pero sin dejarlas obsoletas; es decir, se indica que se deben utilizar de la forma más actualizada, pero dejan un rango de tiempo y/o versiones donde se siga aceptando la forma antigua.

En la siguiente imagen (Figura 3), capturada del IDE de Android Studio a la hora de crear un nuevo proyecto, se muestra las diferentes versiones con las que se puede trabajar en un producto Android. Actualmente se puede utilizar desde la versión 15 de la API, que se corresponde a la versión 4.0 de Android, hasta la versión 19, que se corresponde a la versión 10.0. Como podemos observar, en la tercera columna se muestran los porcentajes en los que la aplicación se ejecutará en aproximadamente de los dispositivos de cada versión.

| ANDROID PLATFORM VERSION | API LEVEL | CUMULATIVE DISTRIBUTION |
|--------------------------|-----------|-------------------------|
| 4.0 Ice Cream Sandwich | 15 | |
| 4.1 Jelly Bean | 16 | 99,8% |
| 4.2 Jelly Bean | 17 | 99,2% |
| 4.3 Jelly Bean | 18 | 98,4% |
| 4.4 KitKat | 19 | 98,1% |
| 5.0 Lollipop | 21 | 94,1% |
| 5.1 Lollipop | 22 | 92,3% |
| 6.0 Marshmallow | 23 | 84,9% |
| 7.0 Nougat | 24 | 73,7% |
| 7.1 Nougat | 25 | 66,2% |
| 8.0 Oreo | 26 | 60,8% |
| 8.1 Oreo | 27 | 53,5% |
| 9.0 Pie | 28 | 39,5% |
| 10. Android 10 | 29 | 8,2% |

FIGURA 3: FRAGMENTO DEL CUADRO DE DIÁLOGO EN LA CREACIÓN DE UN PROYECTO EXTRAÍDA DEL IDE ANDROID STUDIO

Dado a que este proyecto queremos que sea lo más accesible posible la versión de la API escogida ha sido la 21, que se corresponde al 5.0 de Android y, como vemos en la imagen, podrá ser utilizada por, aproximadamente, un 94,1% de los dispositivos Android existentes. Se ha elegido esta versión frente a otras porque se trata de la versión mínima posible en la que se cuentan con todas las funcionalidades, librerías y métodos que queremos utilizar en la creación del producto. Aun así, la mayoría de los dispositivos Android que tengan una versión posterior podrán ejecutar la aplicación sin inconvenientes. Una de las causas por las que el porcentaje disminuye a medida que se utiliza una versión más nueva es porque los dispositivos más antiguos no poseen las características necesarias para poder actualizarse, quedándose obsoletos.

Otra característica de *Android Studio* es que está dotado de cierta inteligencia, pudiendo detectar errores específicos de Android y realizar sugerencias que se adaptan al contexto, además de revelar problemas de rendimiento y de compatibilidad de versiones, entre otros. Esto último se consigue gracias a que todo proyecto Android se construye bajo *Gradle* [22]. Este sistema se usa para la construcción de forma automática del código, permitiendo *preprocesar*, compilar y enlazar varios archivos en un orden preestablecido. Además, se encarga de administrar el proceso de compilación de manera que *Android Studio* no necesite un generador de código interno, sino que delega todas las tareas de compilación al sistema *Gradle*, el cual se declara en un fichero aparte dentro del proyecto. Es en el archivo *build.gradle* donde se indica la versión de la aplicación, versión mínima de la *SDK*, opciones de compilación, entre otros y es donde se deben agregar las dependencias y librerías que se quieran utilizar.

Un *Software Development Kit* o *SDK* es un conjunto de herramienta que permiten la creación de aplicaciones para una plataforma específica. Por tanto, el “*SDK* de Android” es el que se tiene que instalar para poder desarrollar un proyecto para dispositivos Android. Este *SDK* incluye el compilador, el depurador, una máquina virtual y la API de Android, entre otras cosas; sin embargo, no incluye el *Java Development Kit* o *JDK*, que es quien facilita las utilidades necesarias para el desarrollo de programas en Java, por lo que es necesario instalarlo aparte. Como el *SDK* de Android trae consigo la API de Android cuando se especifica la versión del *SDK* en el *build.gradle* ya se está estableciendo la versión de la API con la que se va a trabajar.

Otra cualidad de Android Studio es que permite de dividir la pantalla en dos o más partes, muy útil cuando se trabaja en el apartado gráfico pues se puede mostrar en una ventana el código y en la otra, la interfaz que genera dicho código. También es posible realizar cambios interactuando con la propia interfaz que luego se reflejan en la programación. En la siguiente figura (Figura 4), obtenida del código del proyecto, se puede observar, a la izquierda, el código *xml* que construye los diferentes elementos y, a la derecha, cómo se generarían. Además, se puede ver que se ha seleccionado el botón de color azul, de manera que se muestran unos puntos que permitirían modificar las dimensiones del propio botón, así como su posición, en función si se hace clic o se arrastra.

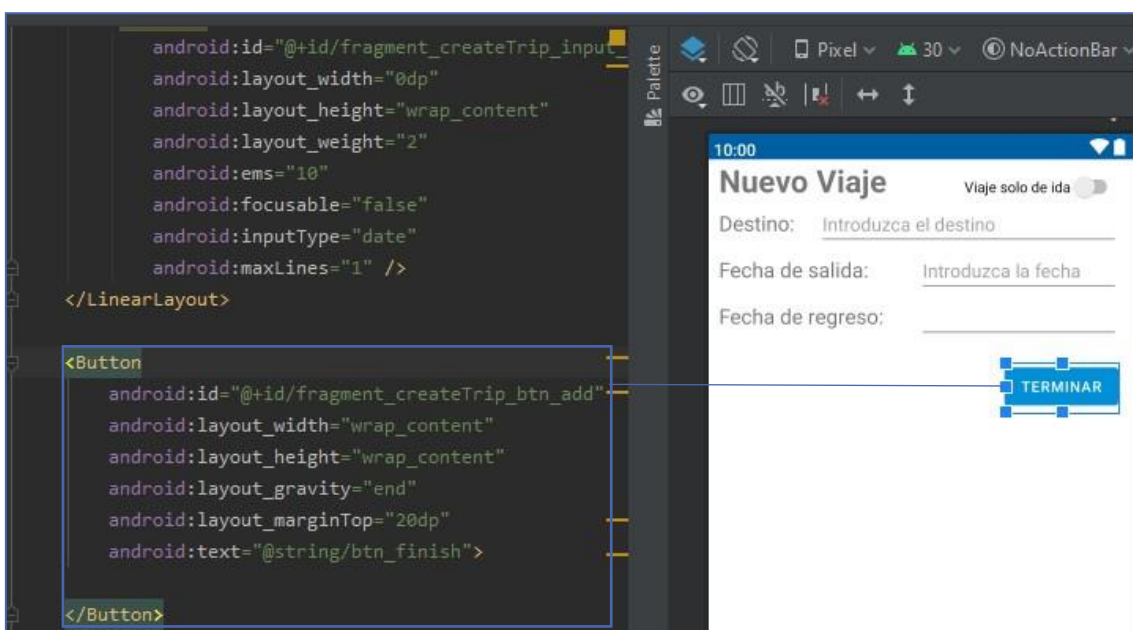


FIGURA 4: CAPTURA DE PANTALLA DEL RECURSO XML *FRAGMENT_CREATE_TRIP.XML*

A la hora de hacer pruebas de código, Android Studio permite simular un dispositivo móvil a través de una máquina virtual; sin embargo, este método resulta muy costoso y poco práctico porque, por un lado, simular un dispositivo móvil entero consume una gran cantidad de recursos y por otro, debido a que las pruebas se realizan a través de un ordenador, no se conseguiría la interacción deseada con la aplicación pues se realiza mediante el ratón.

Una alternativa a la máquina virtual es utilizar un teléfono móvil real, siendo este método más eficiente pues solo hay que instalar la aplicación en el dispositivo y ya éste se encargará de ejecutarla. Esto se puede hacer mediante una conexión directa con el ordenador, por ejemplo, mediante un cable USB, o mediante un archivo *.apk*, por sus siglas *Android Application Package*. Mediante este archivo que es capaz de generar el propio entorno de Android Studio, es posible exportar la aplicación para que pueda ser usada en cualquier teléfono, pues un archivo *APK* contiene todo el código del programa, sus recursos y el archivo de manifiesto.

Por tanto, la opción escogida para ir probando el código ha sido la de utilizar un dispositivo físico, en concreto un Android A50 cuyas características principales son:

- Dimensiones: 158,5 × 74,7 × 7,7 mm
- Memoria RAM: 4 GB
- Cámara: una cámara delantera de 25 megapíxeles, full frame y f/2.0 y tres traseras:
 - Principal: 25 megapíxeles, enfoque automático y f/1.7
 - Secundarias: 5 megapíxeles, full frame y f/2.2 y 8 megapíxeles, full frame y f/2.2
- Pantalla: 6,4 pulgadas
- Versión de Android: 11

Nota sobre las cámaras: La *apertura f* es un valor que afecta a dos variables: la distancia focal y la cantidad de luz que es capaz de absorber. El *full frame*, formato completo o sensor de 35 mm indica que posee el mismo tamaño análogo que las cámaras analógicas.

En la siguiente fotografía (Figura 5), obtenida de la página oficial de Samsung, se muestra un teléfono Samsung A50. Se puede destacar que posee tres cámaras traseras diferentes, siendo la cámara principal la de 25 megapíxeles, la cual será la que se utilice a la hora de realizar fotos.



FIGURA 5: TELÉFONO SAMSUNG A50 (A LA IZQUIERDA) POR DELANTE Y (A LA DERECHA) POR DETRÁS

Sin embargo, hay que tener en cuenta que, aunque se utilice este teléfono para la realización de las pruebas, el diseño y construcción de la interfaz se llevará a cabo utilizando herramientas que se adapten a la densidad de pantalla, permitiendo que las imágenes, textos e iconos se escalen correctamente. Para ello, se utilizarán *píxeles independientes de la densidad* como unidad de medida en sus dimensiones en vez de los píxeles convencionales. Los píxeles, del inglés *picture element*, son los componentes de una imagen donde cada píxel es un punto que emite un solo color. Las imágenes se forman como una sucesión de píxeles.

Entre los distintos dispositivos Android hay de diferentes tamaños de pantallas como, por ejemplo, entre los teléfonos, *tablets* y televisiones. Si nos centramos en un solo conjunto, por ejemplo, el de los teléfonos móviles, también encontramos variedad de tamaños pues existen múltiples marcas y modelos. Aun así, incluso si tomamos dos teléfonos con las mismas dimensiones, puede que sus pantallas tengan píxeles de distintos tamaños.

La siguiente imagen (Figura 6), extraída de la página para desarrolladores de Android [23], se muestra una comparación entre dos dispositivos que tienen las mismas dimensiones, pero el dispositivo de la izquierda tiene menos píxeles por pantalla que el de la derecha, de manera que la misma imagen, en este caso una letra 'a', se renderiza de forma distinta mostrándose más nítida en la que tiene más cantidad de píxeles; es decir, el teléfono de la derecha.

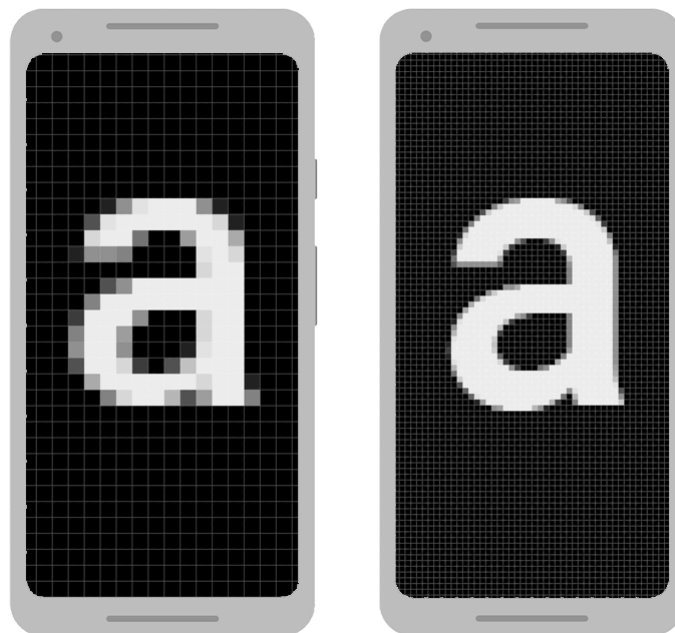


FIGURA 6: IMAGEN COMPARATIVA ENTRE DOS TELÉFONOS DE MISMA DIMENSIONES, PERO CON DIFERENTE DENSIDAD DE PANTALLA

Las pantallas de los dispositivos digitales se miden normalmente en pulgadas¹, concretamente, el valor que se utiliza es de la diagonal de la pantalla en pulgadas. Como se puede observar en la figura anterior, entre más píxeles tiene una pantalla mejor será la calidad de imagen. El número de píxeles que puede ser mostrado en la pantalla se denomina *resolución de pantalla*, de manera que se puede tener dos dispositivos del mismo tamaño y de resoluciones de pantalla diferentes.

¹ una pulgada equivale a 2.54 cm

A la hora de diseñar una interfaz de usuario, es importante tener en cuenta tanto la resolución de la pantalla como el tamaño de los píxeles. Si no se tienen en cuenta estas variaciones el escalado de imágenes, texto e iconos puede fallar y generar elementos borrosos o de tamaño incorrecto. Para ello se debe trabajar con la *densidad de pantalla*, que es la cantidad de píxeles que caben dentro de una pulgada y, como se aventuró antes, utilizando *píxeles independientes de la densidad (dp)* como unidad de medida ofrece mejor ratio en el escalado de cualquier elemento frente al píxel convencional. La densidad de pantalla se mide en píxeles por pulgada, en inglés *pixels per inch (dpi)* y se clasifican desde densidad baja a densidad extremadamente alta [23].

En la siguiente tabla se muestran las principales categorías de densidad de pantalla destacando la densidad media de 160 *dpi* siendo ésta la densidad de referencia a la hora de realizar el escalado de objetos.

| Calificador de densidad | | Descripción |
|-------------------------|--------------------------|---|
| ldpi | (low) | Densidad baja (120 dpi) |
| mdpi | (medium) | Densidad media (160 dpi) ¡densidad de referencia! |
| hdpi | (high) | Densidad alta (240 dpi) |
| xhdpi | (extra-high) | Densidad muy alta (320 dpi) |
| xxhdpi | (extra-extra high) | Densidad muy, muy alta (480 dpi) |
| xxxhdpi | (extra-extra-extra-high) | Densidad extremadamente alta (640 dpi) |

En la siguiente imagen (Figura 7), extraída del blog *prototypr* [24], se muestra una comparativa de cuatro densidades diferentes, partiendo de la densidad de referencia o base de 160 *dpi* hasta la densidad muy, muy alta de 480 *dpi*. Al igual que se vio anteriormente, a mayor densidad más nitidez en la imagen puesto que, con la misma resolución, en este caso de una pulgada de alto por una pulgada de ancho, hay mayor número de píxeles para definir la imagen.



FIGURA 7: IMAGEN COMPARATIVA DE DENSIDADES DE PANTALLA

Un *píxel independiente de la densidad* es un píxel estándar en un dispositivo de densidad media; es decir, de 160 *dpi*. Se puede calcular multiplicando el ancho en píxeles por 160 y dividiendo el resultado por la densidad de la pantalla. Un objeto que utilice *dp* recalcula su tamaño partiendo de esa densidad. En la siguiente representación (Figura 8), obtenida de la página *altova* [25], se muestra una comparación del uso del píxel estándar (*px*) en tres resoluciones distintas de cuatro pulgadas de alto y tres de ancho. Como se puede ver, el cuadrado rojo varía en cada resolución mientras que sus dimensiones se conservan, pues ocupa dos píxeles de alto y dos píxeles de ancho en cada pantalla, pero el tamaño varía, en este caso haciéndose más pequeño conforme se aumenta la resolución. Esto es porque el píxel no cambia con la resolución de pantalla.

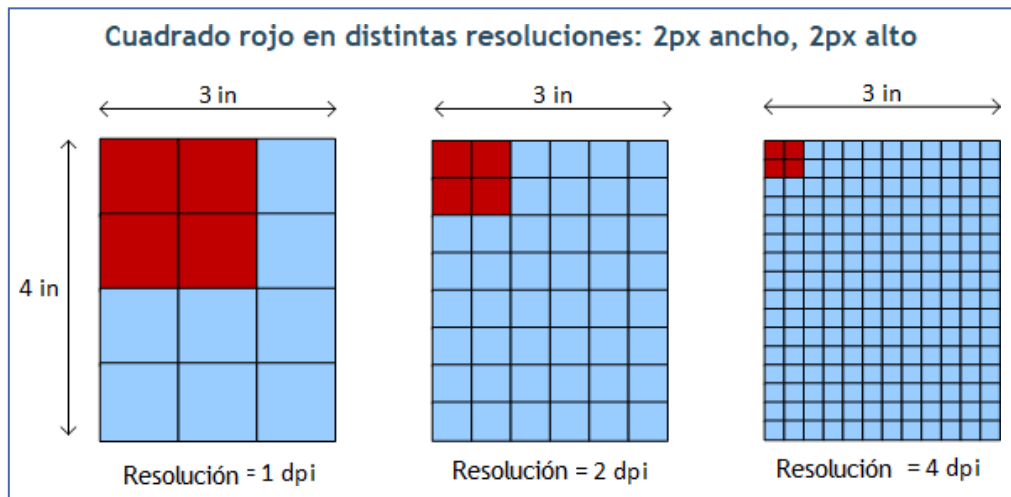


FIGURA 8: VARIACIÓN DEL ESCALADO DE IMAGEN USANDO PÍXEL ESTÁNDAR EN DIFERENTES RESOLUCIONES

Pasando a la siguiente ilustración (Figura 9), extraída de la misma página que la anterior, se puede observar cómo, utilizando píxel dependiente de la densidad, si se va escalando la imagen conservando en gran medida su tamaño. En todas las resoluciones ocupa un píxel dependiente de la densidad, pero sus dimensiones en píxeles estándar si varían.

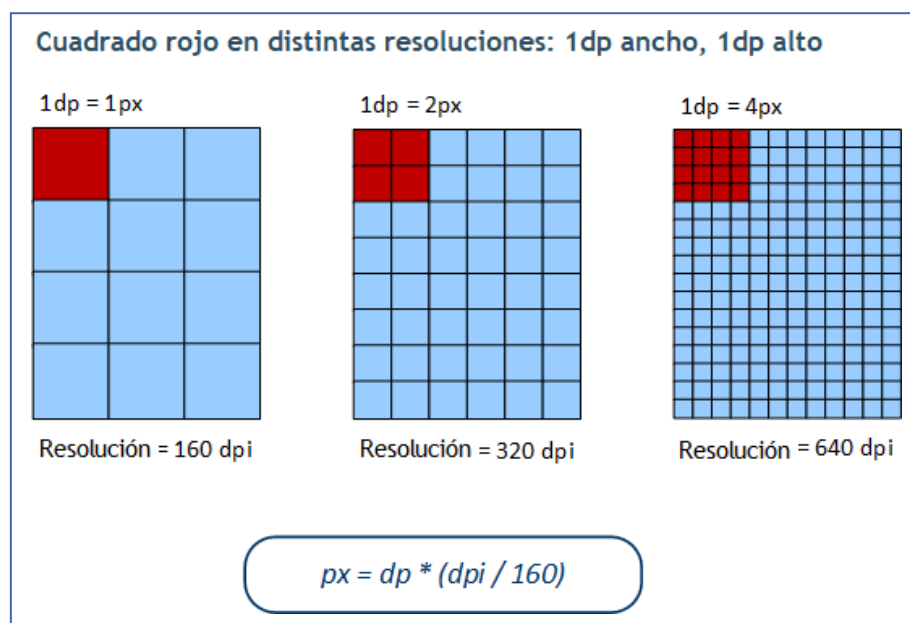


FIGURA 9: VARIACIÓN DEL ESCALADO DE IMAGEN USANDO PÍXEL INDEPENDIENTE DE LA DENSIDAD EN DIFERENTES RESOLUCIONES

Aparte del escalado de imágenes, también hay que tener en cuenta los textos. Aunque sería posible utilizar píxeles independientes de la densidad, existe otra unidad de medida bastante similar y enfocada para esta tarea. Se trata de los píxeles escalables, del inglés *scaleable pixels* o *sp* y tienen la misma función que los *dp*, pero aplicada a textos. Los píxeles escalables agregan un factor de escala adicional basado en el tamaño de fuente que el usuario haya configurado.

Utilizando *sp*, cuando se transforma un texto a diferentes resoluciones se preservarán los ajustes del usuario respecto a la fuente de manera que, los usuarios que tengan ajustes para agrandar o empequeñecer el texto verán que concuerda con dichas preferencias. Con esto, Android consigue dar homogeneidad entre aplicaciones evitando que cada una tenga fuentes distintas.

Como complemento al IDE de Android Studio, también está la página web oficial para desarrolladores Android [26] gestionada por Google y que contiene documentos, tutoriales y ejemplos de múltiples campos en el desarrollo móvil. Este servicio web se actualiza continuamente y posee un histórico de versiones, de manera que se pueden realizar consultas desde funciones ya obsoletas, hasta métodos que están aún en la fase de pruebas. Aparte, existe una biblioteca que contiene la API de Android de manera que se pueden consultar clases, librerías y métodos específicos, su jerarquía, su descripción, parámetros entre muchas otras características y que es prácticamente fundamental a la hora de desarrollar cualquier aplicación.

Un aspecto importante de todo proyecto informático es gestionar el flujo de trabajo y llevar a cabo un control de versiones que permita ir guardando los avances en el código y, en caso de que se produzca un fallo, poder regresar a una versión anterior. Para llevar a cabo esta tarea se ha hecho uso del software de control de versiones *git* y la plataforma web *github*. De esta manera, se creó un repositorio local *git* del proyecto el cual se iba actualizando con cada modificación del código para, posteriormente, subirlo al repositorio online de la cuenta de *github*. Para manejar el repositorio tanto local como remoto, se ha hecho uso de la aplicación de escritorio *SourceTree* [27]. Es una herramienta sencilla, muy visual y gratuita que permite viajar entre cada versión sin problemas. Permite utilizar comandos *git* a través de su interfaz y visualizar las diferentes ramas del proyecto.

2.2 Planificación

Partiendo de la idea de “desarrollar un producto que ayude a la preparación de maletas” se decidió optar por llevar a cabo una aplicación móvil en vez de otras alternativas como un servicio web o de escritorio. La razón principal es a la flexibilidad y comodidad a la hora de usar dispositivos móviles, siendo además un objeto esencial para las personas, que llevan a todos lados gracias a que es sencillo de transportar y utilizar.

A nivel académico, llevar a cabo un proyecto de desarrollo móvil supone un nuevo reto pues, en el Grado de Ingeniería Informática (Plan 2010) de la Universidad de Las Palmas de Gran Canaria, no existe una asignatura en la que se imparta este tipo de tecnologías por lo que es una buena manera, no solo de aplicar los conocimientos adquiridos en la carrera a un proyecto de cierta envergadura, sino además de poner en práctica la capacidad de aprender y trabajar en otro entorno de desarrollo.

Como se ha mencionado anteriormente, antes de empezar a desarrollar, es imprescindible realizar una fase de planificación y diseño cuyo objetivo es detectar las funcionalidades de la que dispondrá la aplicación, el diseño de la interfaz, así como la forma en que el usuario interactuará con esta. Para identificar los diferentes aspectos fundamentales del proyecto se utilizan diagramas, en concreto uno de casos de uso y otro de actividades.

Un diagrama de caso de uso muestra un conjunto de actividades o procesos interaccionando con un sistema y sirve para especificar su comunicación y su comportamiento, tanto con usuarios como con otros sistemas. Es a partir del diagrama de casos de uso que se pueden obtener qué funcionalidades deberá tener la aplicación. En la siguiente imagen (Figura 10) se muestra el diagrama de casos de uso de diseño para este proyecto. En él se pueden destacar las dos funcionalidades principales: *Gestionar Viajes* y *CRUD Plantilla*.

Gestionar Viajes es una generalización; es decir, los casos de uso vinculados a éste, *Crear Viaje*, *Modificar Viaje*, *Eliminar Viaje* y *Hacer Maleta*, son abstracciones de *Gestionar viaje*.

Por otro lado, *CRUD plantilla* es una agrupación de cuatro casos de uso que son *Create*, *Rename*, *Update* y *Delete* que se aplican a una plantilla. Se denota de esta manera para simplificar el diagrama.

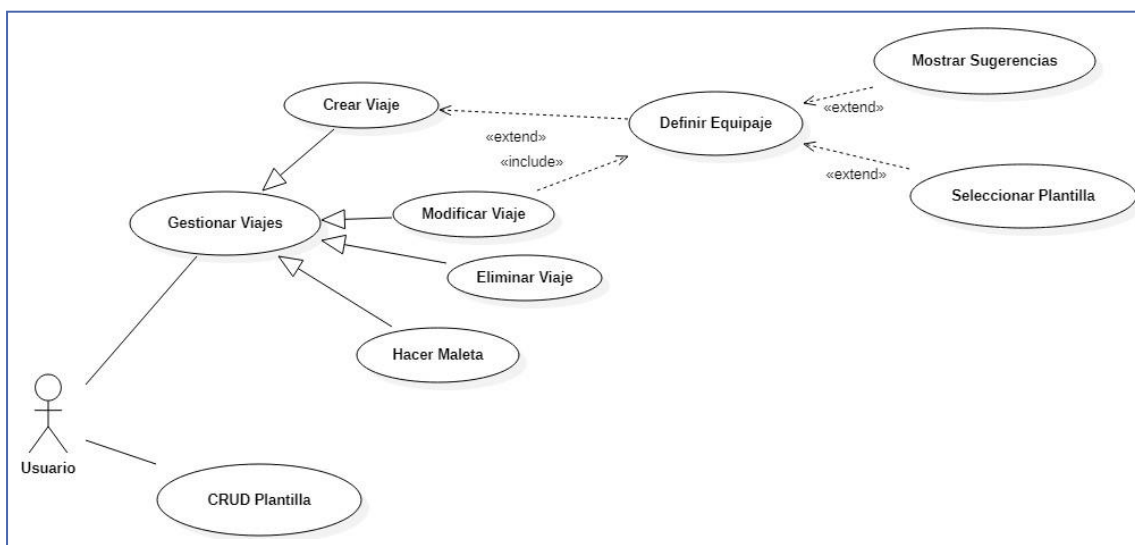


FIGURA 10: DIAGRAMA DE CASOS DE USO DEL PROYECTO

Es importante tener en cuenta que a partir del diagrama de casos de uso se obtienen las funcionalidades, pero no el cómo se van a implementar ni plantear y, a medida que se vaya avanzando en el proyecto, es posible que surjan nuevas utilidades no contempladas en un principio como, por ejemplo, la implementación de una cámara o poder ordenar entidades.

Por otro lado, los diagramas de clase describen la estructura que debe tener un determinado sistema mostrando las clases que lo componen, sus atributos, funciones, y las relaciones entre los objetos. En el siguiente esquema (Figura 11) se presenta el diagrama de clases inicial planteado para este proyecto donde se puede contar hasta cuatro clases (o entidades) diferentes.

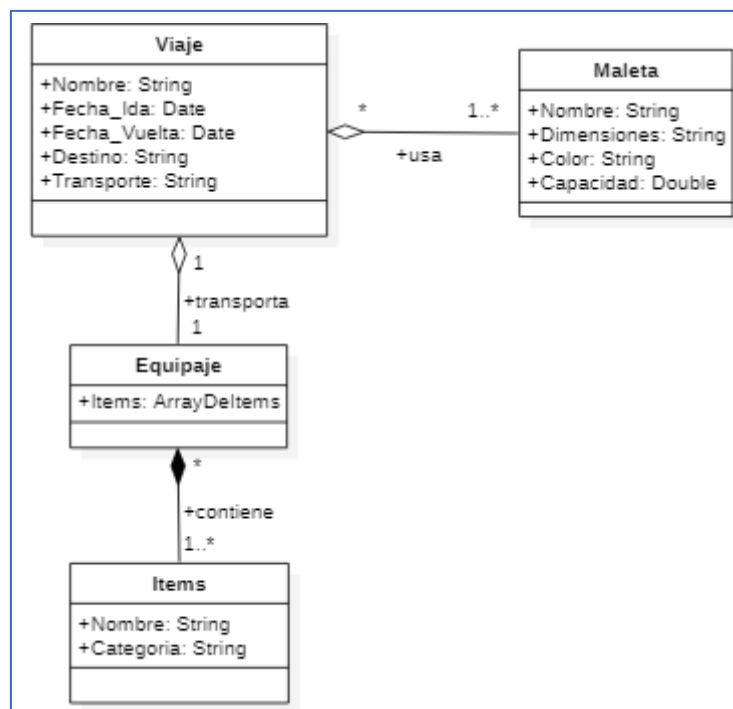


FIGURA 11: DIAGRAMA DE CLASES DEL PROYECTO

Mientras el diagrama de caso de usos nos indica “*que debe hacer la aplicación*” y el diagrama de clases, “*cuáles son los componentes del sistema y su relación*”, es posible utilizar otros diagramas como son el de actividad o el de secuencia para especificar los pasos a seguir a la hora de hacer un determinado procedimiento, así como el estado de la aplicación antes, durante y después de su ejecución. Gracias a los diagramas se consigue plasmar la idea de producto en algo más concreto con lo que poder trabajar; sin embargo, antes de pasar a la programación, es fundamental plantear también el diseño que se quiere llevar a cabo de la aplicación, para lo cual se elaborará un prototipo.

Un prototipo es un primer modelo interactivo que simula un sistema permitiendo verificar el diseño y comprobar que posee las características específicas. Entre más detallado y profundo sea, mayores características sobre el diseño se pueden obtener. Para evaluar un prototipo hay que tener en cuenta si cumple, o no, con especificaciones propuestas y si la solución que se plantea es, o no, la más adecuada; sin embargo, todas las conclusiones que se obtengan deben tomarse como algo orientativo puesto que, a lo largo del desarrollo de un proyecto, pueden producirse cambios debido a diferentes causas.

En las siguientes imágenes (Figura 12), obtenidas de las primeras versiones del prototipo de este proyecto, se muestran tres pantallas de diferentes procesos, donde aparece una pantalla principal con la que se accedería al resto de funciones, el formulario para la creación de un viaje y el procedimiento de crear una nueva maleta junto con la agregación de ítems.

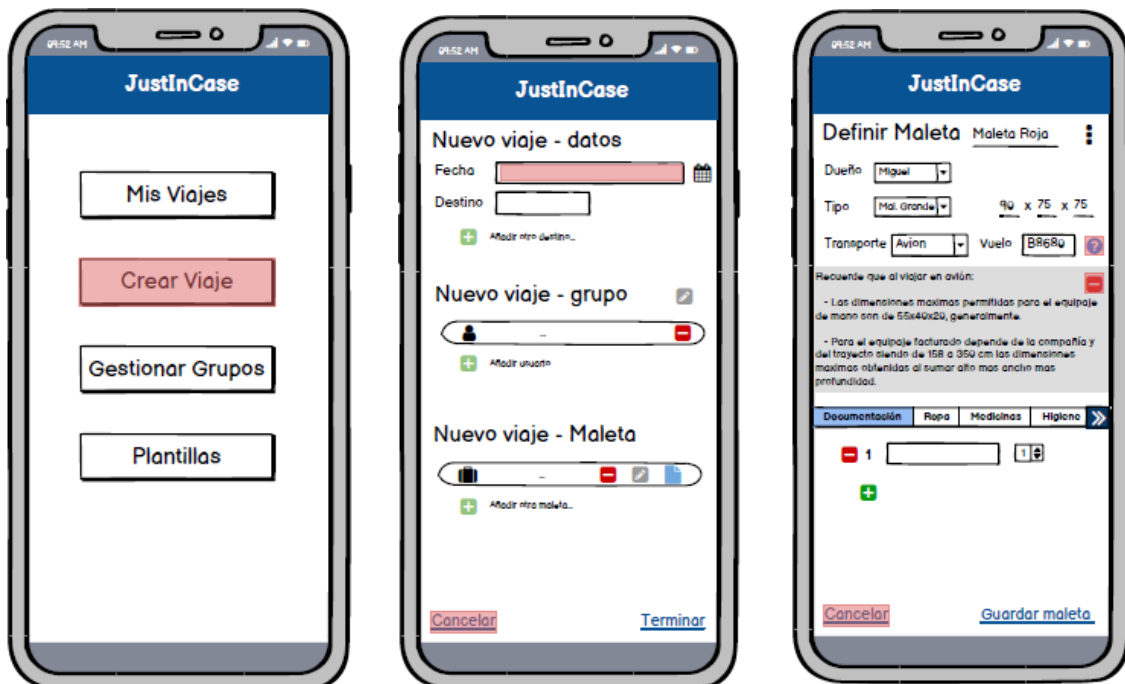


FIGURA 12: CAPTURAS DE PANTALLA DEL PROTOTIPO DEL PROYECTO

En este caso, analizando las imágenes del prototipo, podemos identificar cómo se debe plantear el diseño, descartando aquellos que son más complejos o poco funcionales de aquellos que se adaptan mejor al contexto de la aplicación. Por ejemplo, de la primera imagen se puede extraer que será necesario la implementación de un “menú de aplicación”. De la imagen del centro se puede anotar que, a la hora de crear un nuevo viaje, puede ser engorroso realizar todo el procedimiento que envuelve esta entidad en una misma pantalla y, por último, la tercera imagen indica que para organizar los ítems dentro de una maleta debe hacerse en una ventana diferente y utilizando algún tipo de estructura como una lista de elementos frente a la barra de navegación.

Todos estos fallos que se detectan se producen en gran medida por el poco conocimiento en el sector del desarrollo móvil el cual requiere de un planteamiento tanto técnico como de diseño muy diferentes al de las aplicaciones convencionales. La interacción con un dispositivo móvil es radicalmente distinta al de un producto web o de escritorio puesto que en un ordenador se dispone de periféricos extras como son el ratón y el teclado que facilitan en gran medida el manejo del usuario en cualquier entorno gráfico.

Al mismo que se desarrollaban estos diagramas, y para darle más entidad al proyecto, se pensó en un título para la aplicación. Tras varias opciones se optó por el nombre “JustInCase” siendo este un juego de palabras en inglés pues “Just in case” se traduciría como “por si acaso” y al mismo tiempo “case” está relacionado con “suitcase” que significa maleta. Dado a que el objetivo principal de la aplicación es evitar al usuario que se olvide de llevarse ítems tanto para la ida como la vuelta de un viaje, “JustInCase” es un título que recoge muy bien este concepto.

Aparte del prototipo, es importante fijarse en productos ya disponibles orientados a resolver el mismo problema que se plantea. Para ello, realizamos una búsqueda en la tienda de aplicaciones de Google [11], más conocida como “*app Store*”, buscando aplicaciones que, de alguna manera, permitan la creación de una lista de ítems y esté relacionada con maletas y/o viajes. A continuación, las instalamos en nuestro dispositivo móvil para visualizar sus interfaces y características con el objetivo de extraer ideas, sensaciones y conceptos.

La lista de aplicaciones instaladas es la siguiente: *PackTeo*, *TravelChecklist*, *PackKing*, *MyTravelSuitcase*, *PackPoint*, *SuitcaseLite* y *PackingList*. En las siguientes imágenes (Figura 13), obtenidas mediante capturas de pantalla de las aplicaciones citadas anteriormente, se muestran tres implementaciones de interfaces, en concreto de *SuitcaseLite*, *PackKing* y *MyTravelSuitcase*.

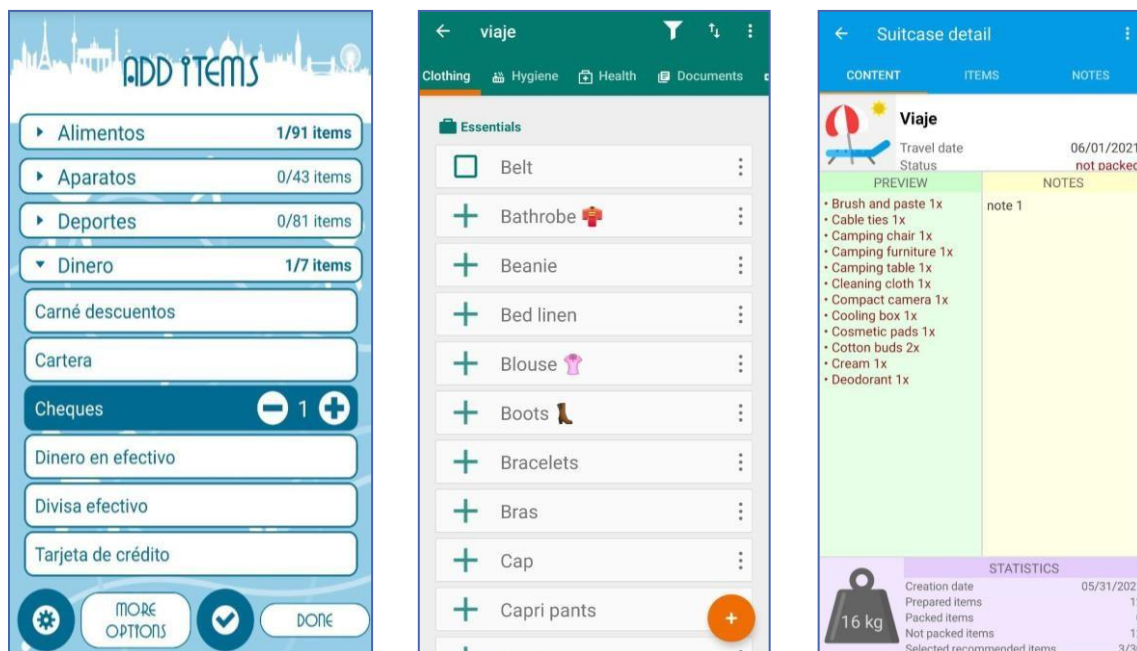


FIGURA 13: CAPTURAS DE PANTALLA DE LAS INTERFACES DE (A LA IZQUIERDA) SUITCASE LITE, (AL CENTRO) PACKKING Y (A LA DERECHA) MYTRAVELSUITCASE

Aunque visualmente sean bastante diferentes, comparten, a nuestro juicio, ciertas similitudes:

- **Interfaz engorrosa:** gran cantidad de elementos, opciones y menús por pantalla.
- **Baja modularidad:** múltiples operaciones solapadas.
- **Poco feedback al usuario:** falta de avisos o notificaciones tras realizar ciertas tareas.

Gracias a este análisis extra, podemos plantear un mejor producto, evitando las cosas que consideramos que se han hecho mal y aplicando las ideas que creemos que si han hecho bien.

3: Desarrollo

Este trabajo consiste en una aplicación para dispositivos Android el cual se desarrolla en el entorno Android Studio de la compañía Google. Un producto Android está dividido en dos partes diferenciadas:

- **Lógica:** se trata de la parte programática desarrollada principalmente en código Java, C++ o Kotlin y se encarga de implementar las funcionalidades de la aplicación; es decir, gestionan las interacciones y eventos que se producen en la interfaz, así como el tratamiento de datos y modificación de vista. Para este trabajo se utilizará solo Java.
- **Visual:** en ella se definen los componentes gráficos que formarán la interfaz de usuario con la que podrá interactuar con la aplicación. Estos elementos son las vistas (*View*) que son los diferentes textos, botones, imágenes, iconos, etc. y su disposición (*Layout*).

Debido a la naturaleza de las aplicaciones Android, los elementos de la interfaz de usuario deben ser invocados en la parte lógica. Este procedimiento se denomina inflar (*inflating*) y se trata de agregar una vista (.xml) a la actividad en tiempo de ejecución.

Las actividades (*Activities*) son cada una de las pantallas que se crean en una aplicación Android. Tienen tanto parte lógica como parte gráfica, las cuales están relacionadas entre sí. Es gracias a éstas, que la aplicación sabe que elementos debe, o no, mostrar al usuario y cuál es su comportamiento. A diferencia del resto de las aplicaciones, tanto de escritorio como web, donde requieren de un método principal (*main*) para iniciar su ejecución, todo sistema Android comienza en una actividad, la cual se define en el archivo manifiesto (*manifest*) [28].

La mayoría de las aplicaciones contienen varias pantallas, por lo que existen varias actividades relacionadas a cada una de ellas. En este caso, se especifica una como la actividad principal, que es la primera pantalla que aparece cuando el usuario inicia la aplicación.

Por otro lado, existen los fragmentos (*Fragment*): elementos que representan el comportamiento de una porción de la interfaz de usuario y deben estar asociados a una actividad. Es posible definir varios fragmentos dentro de una misma actividad, interaccionando entre ellos. Se pueden combinar actividades y fragmentos dentro un proyecto.

En la siguiente imagen (Figura 14), obtenida de la página de *academiaAndroid* [29], se muestran dos ilustraciones sobre cómo se mostraría dos contenidos diferentes: en la primera utiliza dos actividades para lo que se requieren dos pantallas diferentes y su correspondiente transición, mientras que en la segunda se utiliza solamente una actividad en la que se cargará el fragmento indicado, sin necesidad de cambiar de pantalla.

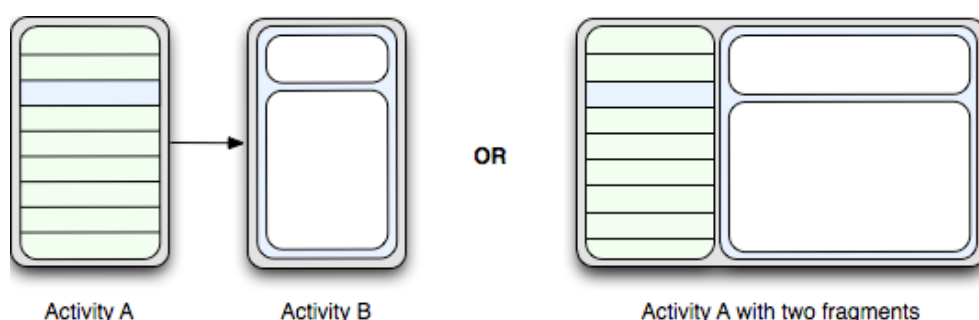


FIGURA 14: DIFERENCIA ENTRE UNA ACTIVIDAD Y UN FRAGMENTO

Otra diferencia fundamental entre las actividades y los fragmentos, son sus ciclos de vida: conformen se dejen de presentar, las actividades no se eliminan, sino que se almacenan en memoria, mientras que los fragmentos sí se van eliminando. En la siguiente imagen (Figura 15), construida a partir de dos representaciones de los ciclos de vida de una actividad [28] y un fragmento [30], ambas obtenidas de la página oficial de Android para desarrolladores, se puede observar que, cuando una actividad pasa a segundo plano (se deja de visualizar), pasa a estar a la espera (*onStop()*) pendiente ser llamada de nuevo (*onRestart()*) o de ser destruida (*onDestroy()*). En este estado sus elementos siguen cargados mientras que, en los fragmentos, una vez pasan a segundo plano, primero se limpia los recursos asociados con su Vista (*onDestroyView()*) y, si vuelen a ser invocados, se vuelven a crear (*onCreateView()*) o, en caso contrario, se descartan completamente y sin afectar al ciclo de vida de su actividad asociada.

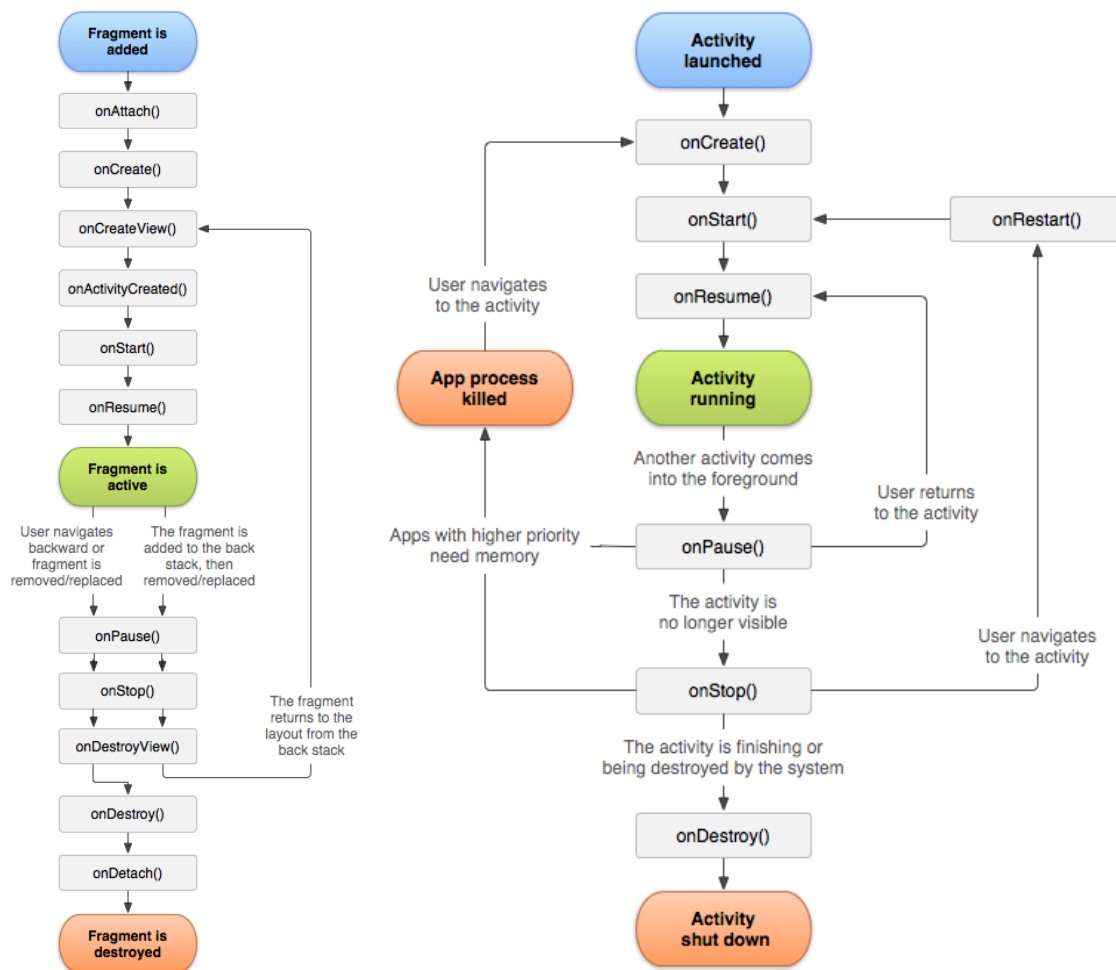


FIGURA 15: CICLOS DE VIDA (A LA IZQUIERDA) DE UN FRAGMENTO Y (A LA DERECHA) DE UNA ACTIVIDAD

Es a raíz de esta característica que, para este trabajo, se utilizó el enfoque “Single Activity” o “One-Activity-Multiple-Fragments” [31], donde cada pantalla es un fragmento que ocupa la ventana completa y todos estos fragmentos se encuentran dentro de una única actividad. Este enfoque no solo simplifica como se implementa la navegación, sino que también tiene un mejor rendimiento y, en consecuencia, mejora la experiencia de usuario.

Para la navegación entre fragmentos se ha utilizado el componente *Navigation* de la librería Android Jetpack [32] que facilita en gran medida su implementación. Este componente permite administrar transiciones de fragmentos, implementar y administrar vínculos directos y agiliza la creación del menú de aplicación. El componente *Navigation* consta de tres partes:

- *NavHost*: se trata del contenedor que se declara dentro de un Layout y sobre el que se mostrarán los diferentes destinos.
- *NavController*: controlador que gestiona la navegación dentro de un NavHost.
- *NavGraf*: recurso XML que contiene toda la información relacionada con la navegación y permite establecer los caminos que un usuario puede tomar. Este recurso no impide la creación de rutas a nivel de código utilizando el *NavController*.

En la siguiente imagen (Figura 16), obtenida del archivo *NavGraf.xml* de este proyecto, se puede visualizar el gráfico de navegación. Cada cuadro representa una ventana de la aplicación; es decir, los diferentes destinos que existen y las flechas son los vínculos que unen unos destinos con otros. El fragmento principal, situado abajo a la izquierda, se corresponde a la primera pantalla que se visualizaría al iniciar el programa.

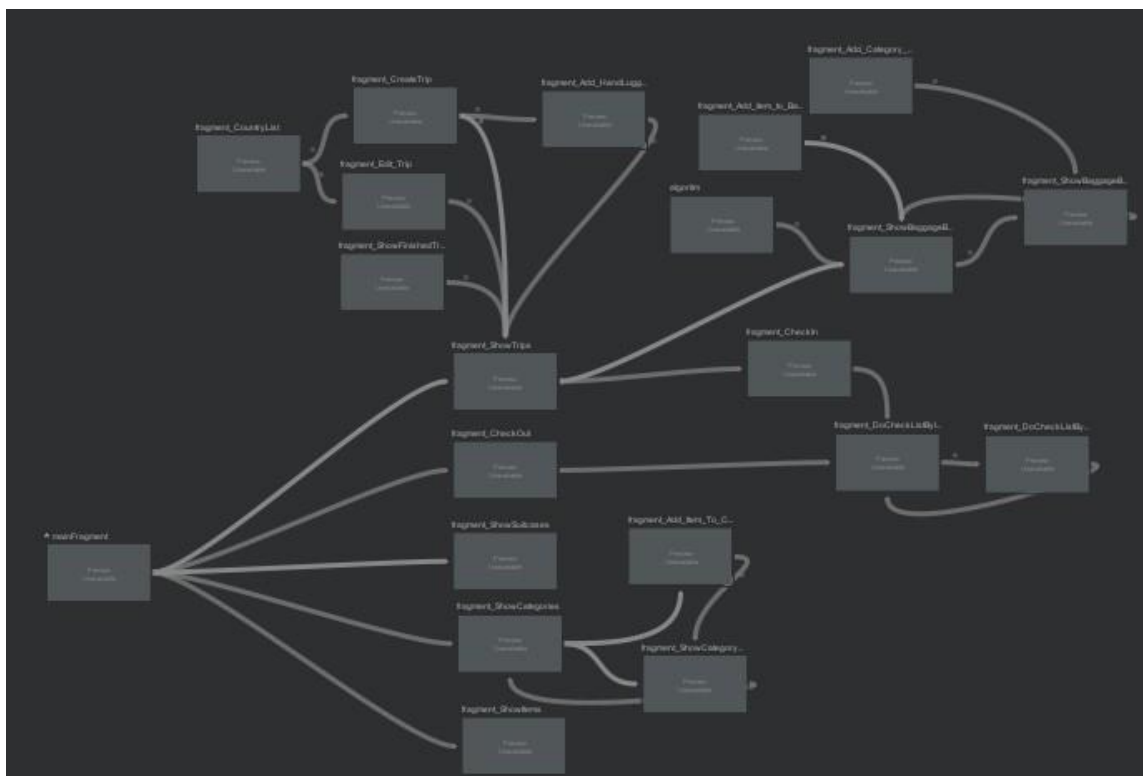


FIGURA 16: GRÁFICO DE NAVEGACIÓN DE LA APLICACIÓN

Las conexiones lógicas permiten agregar acciones que los usuarios pueden realizar para navegar de un destino al otro como son el añadir una animación en la transición, pasar argumentos al destino o modificar la pila de actividades.

La pila de actividades es la encargada de guardar los destinos que se han ido recorriendo y se organizan usando un método *last in, first out* (LIFO). Cuando la actividad actual inicia otra, la nueva se coloca en la parte superior de la pila y toma el foco y la anterior permanece en la pila, pero se detiene reteniendo el estado actual de su interfaz de usuario.

Esto permite la navegación “hacia atrás”; pues cuando el usuario presiona el botón Atrás, se quita la actividad actual de la parte superior de la pila, eliminándola y se reanuda la actividad anterior.

En el siguiente esquema (Figura 17), extraído de la página oficial de Android para desarrolladores [33], se muestra una representación donde cada nueva actividad se va agregando a la pila, cuando el usuario presiona el botón *Atrás*, se elimina la actividad actual y se reanuda la anterior.

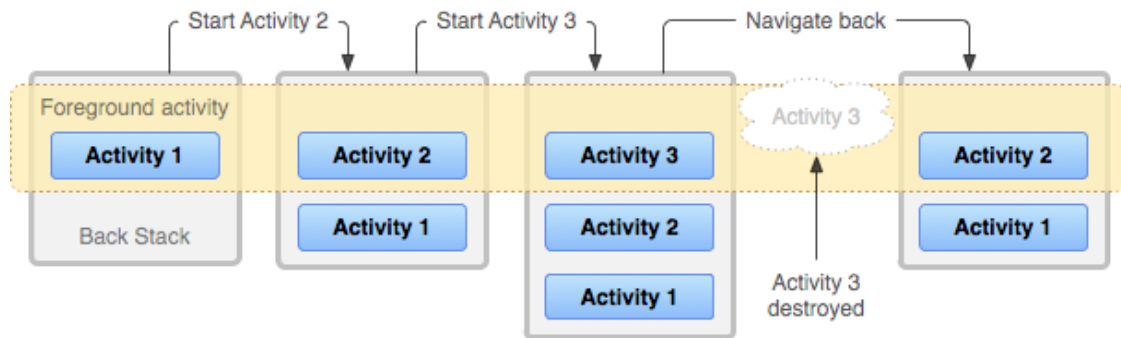


FIGURA 17: REPRESENTACIÓN DEL COMPORTAMIENTO DE LA PILA DE ACTIVIDADES

Hay que tener en cuenta que, si el usuario ejecuta muchas actividades a la vez, el sistema puede eliminar algunas a fin de recuperar memoria, por lo que se perdería el estado y podría dar errores. Por otro lado, debido a que la pila nunca se reordena puede ocurrir que, si se permite a los usuarios iniciar una actividad en particular desde cualquier punto, se creen nuevas instancias en lugar de utilizar una previa; es decir, existirían varias instancias de una misma actividad. La siguiente imagen (Figura 18), obtenida de la misma página [33], se puede observar una ilustración de la pila al ejecutar una actividad múltiples veces.

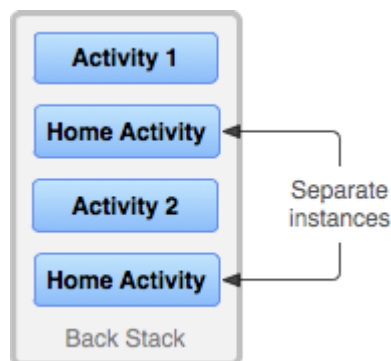


FIGURA 18 REPRESENTACIÓN DE LA CREACIÓN DE MÚLTIPLES INSTANCIA DE UNA MISMA ACTIVIDAD

Es posible controlar este comportamiento mediante el uso de los métodos *popUpTo()* y *popUpToInclusive()* manejados por el *NavController*. Se pueden ejecutar al transitar de un destino a otro y permiten indicar que, al viajar al destino indicado, se debe limpiar (hacer “pop up”) de la pila, en el caso de *popUpTo()* es hasta la actividad indicada que se queda mantenida mientras que *popUpToInclusive()* la saca también dejándola vacía.

A partir del componente *Navigation* se puede desarrollar también una barra de navegación, más conocida como “menú de la aplicación” o menú principal, el cual permite al usuario la capacidad de viajar a diferentes partes del programa de forma libre y sencilla. En este proyecto se optó por un menú deslizante, en inglés *DrawerMenu*. Para ello, se crea un recurso menú; un archivo *xml* que define los elementos del menú, su estructura, nombre e iconos. En la siguiente imagen (Figura 19), capturada del proyecto en Android Studio, se puede observar la construcción de dicho recurso y su diseño.

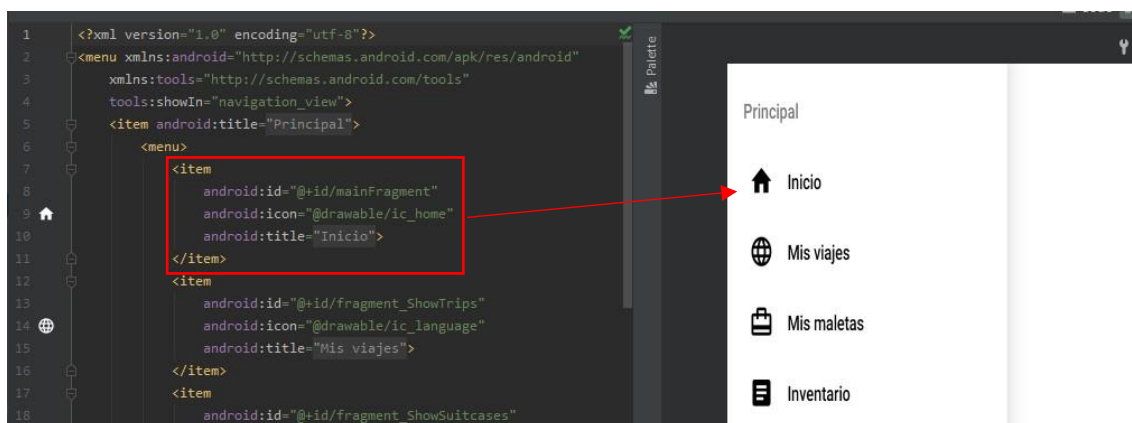


FIGURA 19: CAPTURA DE PANTALLA DEL ARCHIVO *DRAWER_MENU.XML*, A LA IZQUIERDA EL CÓDIGO XML Y A LA DERECHA EL DISEÑO RESULTANTE

A cada elemento se le asocia un identificador (*id*), el cual será idéntico a un destino ya definido en el gráfico de navegación. De esta manera, cuando se pulse una opción del menú, el controlador (el *NavController*) sabrá a qué fragmento se debe transitar sin necesidad de especificarlo en el gráfico o mediante una estructura condicional. En la siguiente captura de pantalla (Figura 20) extraída del proyecto, se puede observar como el destino que se corresponde al fragmento principal; es decir, la pantalla de inicio tiene el mismo identificador que el elemento de menú que se relaciona al botón de “inicio”.

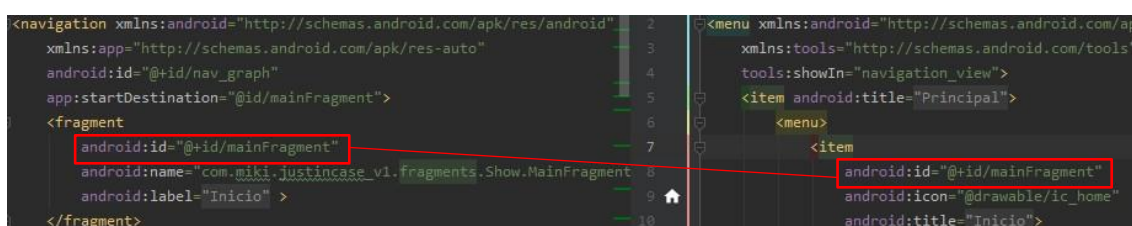


FIGURA 20: CAPTURA DE PANTALLA DE LOS FICHeros (A LA IZQUIERDA) *NAV_GRAPH.XML* Y (A LA DERECHA) *DRAWER_MENU.XML* DEL PROYECTO

En el momento que se pulsa un botón del menú deslizante, se invoca al método *onOptionsItemSelected()* perteneciente a la clase *Activity*, el cual recibe como parámetro el elemento del menú que ha sido pulsado. Al haber hecho la asociación de destino-elemento mediante *id*, éste parámetro se puede pasar al método *onNavDestinationSelected()*, de la clase *NavigationUI*, que, citando su descripción en la API de Android, “Intenta navegar al *NavDestination* asociado con el *MenuItem* dado” [34], de manera que lo puede realizar directamente.

En el siguiente fragmento de código (Figura 21), extraído de la clase MainActivity, se muestra la implementación del método `onOptionsItemSelected()`. Primero se invoca al `NavController` y luego, mediante `onNavDestinationSelected()`, se viaja al destino especificado en el `MenuItem`.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    NavController navController = Navigation.findNavController( activity, this, R.id.fragment);
    return NavigationUI.onNavDestinationSelected(item, navController)
        || super.onOptionsItemSelected(item);
}
```

FIGURA 21: CAPTURA DE PANTALLA DEL MÉTODO `ONOPTIONSEITEMSELECTED()` DE LA CLASE `MAINACTIVITY`

El menú deslizante es bastante útil para cuando hay más de tres acciones posibles, pues usar otras alternativas como la barra de navegación inferior, en inglés *BottomMenu*, no es recomendable, dado que las opciones siempre están visibles y por tanto, o bien ocupan gran parte de la pantalla, o bien los elementos deben ser muy pequeños.

Otra ventaja del *DrawerMenu* es su capacidad de compactarse, de manera que no es necesario adaptar la interfaz, además de poder situarlo siempre en el mismo punto y ser utilizado en cualquier momento. En la siguiente figura (Figura 22) capturada a partir de la aplicación en ejecución, se puede observar cómo se pasa del menú compacto al menú desplegado. Se ha marcado con un círculo rojo el botón que se corresponde al menú.

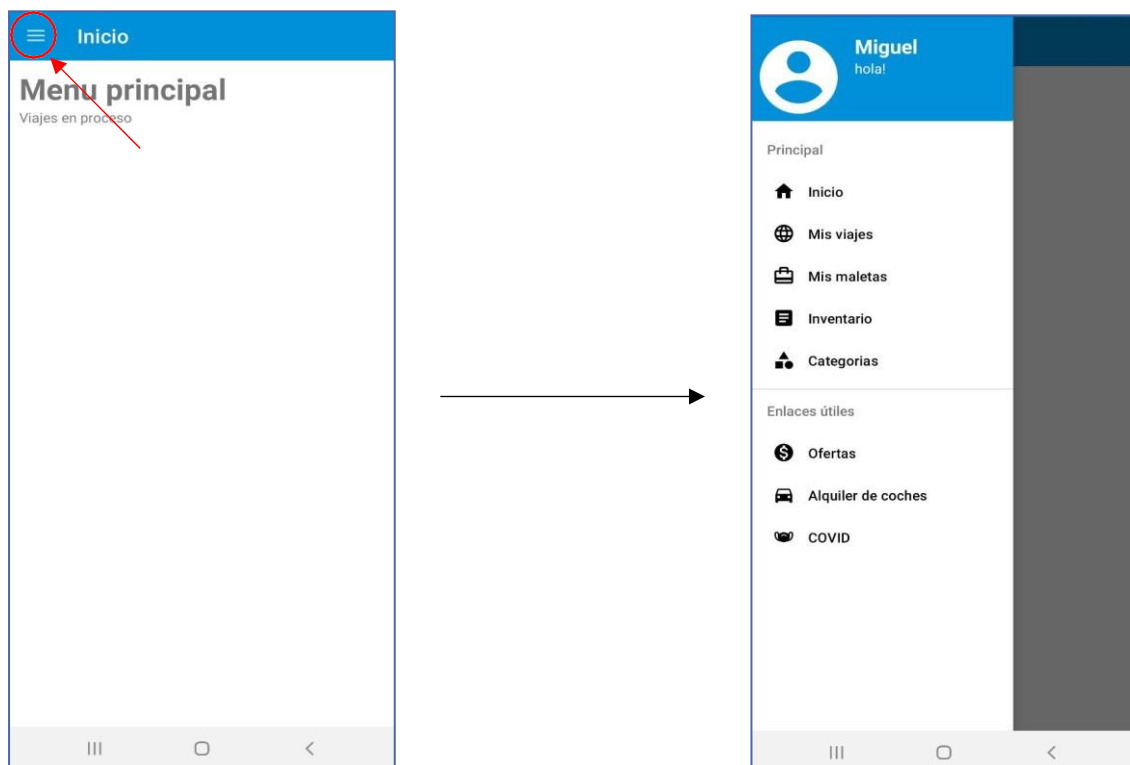


FIGURA 22: *DRAWER MENU* DE LA APLICACIÓN, A LA IZQUIERDA SIN DESPLEGAR Y A LA DERECHA DESPLEGADO

Una de las funcionalidades de la aplicación es poder asociar una imagen a un ítem. Para ello, se puede realizar de dos maneras diferentes: o bien cargar una imagen de la galería interna del dispositivo o bien sacar una foto a los diferentes ítems. En ambos casos es imprescindible que el usuario acepte una serie de permisos como son el de lectura y escritura en el almacenamiento interno del dispositivo y el acceso (si tiene) a la cámara del dispositivo. En el siguiente diagrama de flujo (figura 23), extraída de la página oficial para desarrolladores de Android [35], se explica el procedimiento para solicitar dichos permisos en tiempo de ejecución:

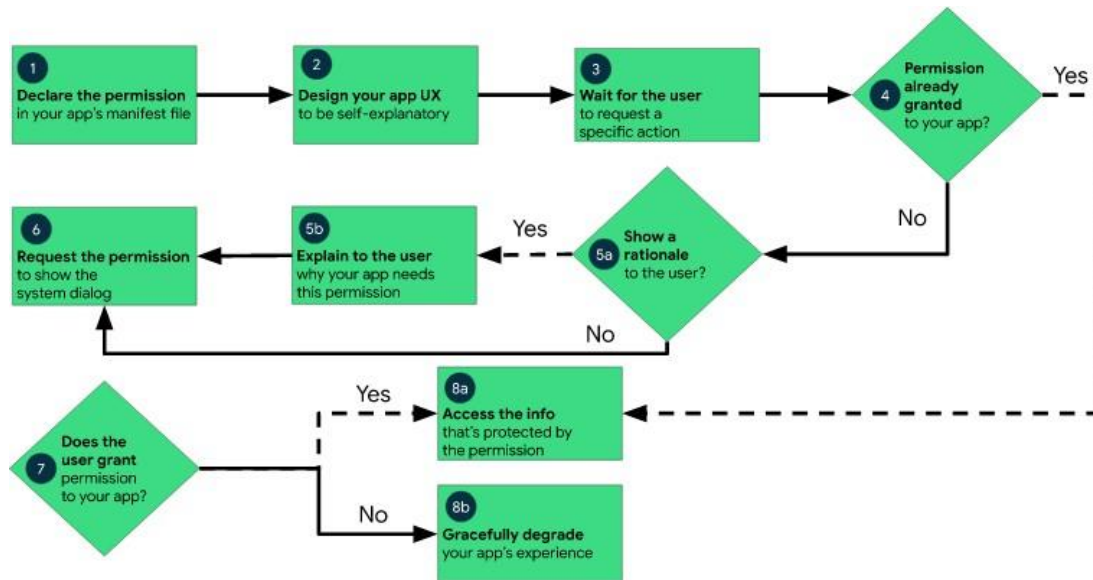


FIGURA 23: DIAGRAMA DE FLUJO DE TRABAJO PARA DECLARAR Y SOLICITAR PERMISOS EN TIEMPO DE EJECUCIÓN DE UNA APLICACIÓN ANDROID

1. Declarar los permisos en el archivo manifiesto.
2. Diseñar que acciones requieren dichos permisos e indicarlo al usuario.
3. Una vez el usuario invoque la tarea que requiere acceso, solicitar el permiso.
4. Comprobar si ya se otorgó el permiso: si no lo está, pedirlo.
5. Mostrar, si es necesario, una justificación al usuario en la que se explique la razón de requerir ese permiso.
6. Solicitar el permiso.
7. Verifica la respuesta del usuario.
8. Si el usuario otorgó el permiso se podrá acceder, en caso contrario, se debe reducir la experiencia con la aplicación, pero manteniendo la funcionalidad.

A la hora de declarar los permisos en el archivo manifiesto, se puede indicar ciertas características como la versión mínima de Android necesaria para ser ejecutado o que sea imprescindible que el dispositivo tenga una cámara para ser instalado. Se pueden combinar varios permisos sin problemas, incluso algunos abarcan ya otros como, por ejemplo, el permiso de escritura permite de manera implícita la lectura. Si se declaran ambos no se produce ningún fallo, aunque no es recomendable.

En el caso de asociar una imagen directamente a un ítem basta con leer de la galería interna del dispositivo; sin embargo, para hacer una foto hace falta implementar una función que permita acceder a la cámara del dispositivo, sacar la foto y guardarla en memoria, ya sea en un archivo interno o en la misma galería compartida.

Para llevarlo a cabo se ha hecho uso de la librería *CameraX* de Android Jetpack. Creada por los desarrolladores de Android, permite un enfoque más simple a la incorporación de una cámara, resolviendo los problemas de compatibilidad con otros dispositivos como son la relación de aspecto, el tamaño de la vista previa y el tamaño de la imagen de alta resolución. Esto lo consigue gracias al uso de un laboratorio de pruebas de diferentes dispositivos. Es importante que tener en cuenta que *CameraX* es aplicable a partir de Android 5.0 (nivel de API 21). [36]

Esta librería funciona a partir de la implementación de *casos de uso* los cuales indican al programa como debe interactuar con la cámara del dispositivo. Los casos de uso son:

- **Preview:** realiza una captura de la pantalla.
- **ImageCapture:** guarda imágenes de alta calidad.
- **ImageAnalysis:** proporciona una imagen a la que se puede aplicar algoritmos y técnicas de procesamiento.

Los casos de uso pueden combinarse y estar activos simultáneamente. De hecho, mientras que *Preview* y *ImageAnalysis* funcionan por sí solos, *ImageCapture* necesita de otro caso de uso. Para evitar un consumo excesivo de recursos, CameraX sigue una serie de pautas determinadas por un ciclo de vida (al igual que las actividades y fragmentos) para determinar cuándo abrir la cámara, realizar una captura, quedarse a la espera y apagarse.

Como el objetivo es hacer una foto y almacenarla, se tienen que implementar dos casos de uso. Por un lado, *Preview* que será quien obtenga la imagen a través de realizar una captura de pantalla de lo que está prevvisualizando la cámara y, por otro lado, *ImageCapture* será el encargado de almacenarla.

En el siguiente fragmento de código (Figura 24), extraído del proyecto, se muestra la implementación del método *bindCamera()*, en el cual se construyen los dos casos de uso (①) y se agregan al ciclo de vida de la cámara (②). Con *CameraSelector.DEFAULT_BACK_CAMERA* se indica al dispositivo móvil que use por defecto la cámara trasera (si la tiene).

```
private void bindCamera(ProcessCameraProvider cameraProvider) {
    ① {
        preview = new Preview.Builder().build();
        imageCapture = new ImageCapture.Builder().build();

        CameraSelector cameraSelector = CameraSelector.DEFAULT_BACK_CAMERA;
        cameraProvider.unbindAll();

        ② {
            cameraProvider.bindToLifecycle(
                lifecycleOwner: this,
                cameraSelector,
                preview,
                imageCapture);
        }
    }
}
```

FIGURA 24: CAPTURA DE PANTALLA DEL MÉTODO *BINDCAMERA()* EN LA IMPLEMENTACIÓN DE *CAMERAX*

Para usar el caso de uso Preview hace falta agregar a la interfaz un componente denominado *PreviewView*, que es el encargado de mostrar la previsualización de la cámara en la pantalla. Dicha pantalla se puede recortar, rotar y escalar para adaptarse a las dimensiones y orientación del dispositivo y se vincula al objeto *Preview* para manejarlo a nivel de código. Es importante tener en cuenta que *Preview* solamente se encarga de mostrar la captura de pantalla, por si solo no puede guardar la imagen. Por ello se utiliza *ImageCapture* que, una vez se tiene el objeto *Preview*, se llama al método *takePicture()* de esa clase, el cual guarda la imagen capturada en la ubicación proporcionada.

En la siguiente captura (Figura 25), extraída del código del proyecto, se muestra el método privado *takePhoto()* que es el encargado de realizar todo el procedimiento para guardar la captura de pantalla en un fichero, el cual se genera automáticamente cuando se va a realizar la operación (①). Es aquí donde se invoca a la función *takePicture()* del objeto *ImageCapture* (②). Si se produce algún error se recoge en *onError()* (④), en caso contrario se ejecutará el método *onImageSaved()* el cual guarda la ubicación en formato URI (③).

```
private void takePhoto() {
    ImageCapture rImageCapture = imageCapture;

    ① File photoFile = new File(outputDirectory, child: new SimpleDateFormat(FILENAME_FORMAT, Locale.US)
        .format(System.currentTimeMillis()) + ".jpg");

    ImageCapture.OutputFileOptions outputOptions = new ImageCapture.OutputFileOptions.Builder(photoFile).build();

    ② rImageCapture.takePicture(outputOptions, ContextCompat.getMainExecutor(requireContext()),
        new ImageCapture.OnImageSavedCallback() {
            @Override
            ③ public void onImageSaved(@NonNull ImageCapture.OutputFileResults outputFileResults) {
                saveUri = Uri.fromFile(photoFile);
                Toast.makeText(getContext(), msg, Toast.LENGTH_LONG).show();
                getNav().navigate(R.id.fragment_ShowItems);
                createItemDialog(saveUri.toString());
            }

            @Override
            ④ public void onError(@NonNull ImageCaptureException exception) {
```

FIGURA 25: CAPTURA DE PANTALLA DEL MÉTODO *TAKEPHOTO()* EN LA IMPLEMENTACIÓN DE *CAMERAX*

Una vez se ha realizado todo este proceso, se obtiene una foto la cual puede, o no, ser utilizada para asociarse a un ítem. Esa asignación se realiza a través de la *URI* de la imagen, la cual se almacena en la base de datos siendo la forma más eficiente. La Uniform Resource Identifier o *URI* es una cadena de caracteres que identifica un recurso y proporciona una ruta o *path* al mismo permitiendo ser accesible por cualquier aplicación o sistema. A la hora de mostrar la imagen del ítem asociado solamente se tiene que recuperar la *URI* asignada y cargar la imagen. Es posible configurar la cámara para que las fotos se tomen con opciones de *flash* y enfoque automático. También se puede agregar el uso de filtros y que se vayan aplicando a la previsualización de la imagen.

Una de las principales características de la aplicación es la posibilidad de hacer una lista de ítems. Para mostrar cualquier lista lo ideal es utilizar una estructura que permita cargar un mismo tipo de elemento varias veces y que, en caso de sobrepasar los límites de la pantalla, permita al usuario realizar un desplazamiento vertical para visualizar el resto. De esta forma, solo hace falta diseñar en un recurso *xml* que contenga cómo se quiere presentar la entidad y hacer que la estructura lo muestre tantas veces como se le indique. Existen dos estructuras que realizan esta función que son el *ListView* y el *RecyclerView*.

El *ListView* [37] es un contenedor de vistas que permite la creación de listas verticales de elementos en pantalla, donde cada elemento se coloca inmediatamente debajo del anterior y que, además, incorpora u *scroll* automático. Una característica importante es que la lista entera está cargada en todo momento.

El *RecyclerView* [38] es una versión más moderna del *ListView* y que, a diferencia de éste, solamente carga los elementos que se están visualizando, permitiendo una mayor eficacia a la hora de trabajar con entidades que varíen continuamente o con modificaciones en tiempo de ejecución de la propia lista, sin necesidad de tener que recargar la actividad o fragmento que lo contenga. Por estas razones, sumado a que la página oficial para desarrolladores Android recomienda usar este contenedor frente a *ListView*, se utilizará el *RecyclerView*.

Perteneciente a la librería Android Jetpack [39], el *RecyclerView* es un contenedor cuya función es la de *reciclar* una vista; es decir, cuando un elemento se desplaza fuera de la pantalla no destruye, sino que los datos que se dejan de ver se almacenan en otra lista secundaria, oculta al usuario, y el elemento *xml* que se estaba usando se reutiliza asignándole los nuevos datos a mostrar, lo que mejora en gran medida el rendimiento y el consumo de memoria de la aplicación; sin embargo, el uso de esta lista secundaria para guardar los datos puede dar problemas si éstos no están bien encapsulados en su posición correspondiente.

Para implementar el *RecyclerView* hace falta declararlo en la interfaz como un recurso *xml* e invocarlo en la actividad o fragmento correspondiente. Al mismo tiempo, se requiere de una clase *adaptadora* que extienda de *RecyclerView.Adapter* y que será la encargada de definir la forma en que se muestran los datos del *RecyclerView*. En la siguiente figura (Figura 26), extraída del código del proyecto, se muestra la declaración de la clase *Adapter_Item* que es la encargada de la lista de ítems de la aplicación. Se puede observar que extiende de *RecyclerView.Adapter*, implementa el método *OnLongClickListener* y la clase *Filterable*. *OnLongClickListener* sirve para agregar un evento que se desencadene cuando el usuario mantenga pulsado un elemento de la lista y gracias a *Filterable* se puede aplicar un filtro a la lista, modificándola en tiempo de ejecución. Este filtro se construye dentro de la clase *Adapter_Item*.

```
public class Adapter_Item extends RecyclerView.Adapter<Adapter_Item.ViewHolder>
    implements View.OnLongClickListener, Filterable {
```

FIGURA 26: DECLARACIÓN DE LA CLASE *ADAPTER_ITEM*

Para cualquier clase *adaptadora* es imprescindible definir los siguiente tres métodos

- *onCreateViewHolder()*: crea e inicializa el *ViewHolder*.
- *onBindViewHolder()*: asocia el *ViewHolder* con los datos proporcionados.
- *getItemCount()*: devuelve el tamaño del conjunto de datos.

El *ViewHolder* es una clase que se construye aparte y es la encargada de vincular la vista que se va a reciclar con el *RecyclerView*.

En la siguiente captura de pantalla (Figura 27), obtenida de la clase *Adapter_Item*, podemos ver la programación de los tres métodos mencionados. En la función *onCreateViewHolder()*, se asocia la vista va a reciclar (①) y cuando se recorre la lista de elementos que pueden mostrarse, se invoca *onBindViewHolder()* donde se obtiene el objeto de tipo *Item* del cual se extrae su nombre para asignarlo al *ViewHolder* (②).

```

@NonNull
@Override
public Adapter_Item.AdapterViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    ① View view = LayoutInflater.from(parent.getContext())
        .inflate(R.layout.card_view_item, parent, attachToRoot: false);
    view.setOnLongClickListener(this);
    return new AdapterViewHolder(view);
}

@Override
public void onBindViewHolder(@NonNull Adapter_Item.AdapterViewHolder holder, int position) {
    ② Item item = dataset.get(position);
    holder.elementNameTV.setText(item.getItemName());
}

@Override
public int getItemCount() { return dataset == null ? 0 : dataset.size(); }

```

FIGURA 27: IMPLEMENTACIÓN DE LOS MÉTODOS *ONCREATEVIEWHOLDER()*, *ONBINDVIEWHOLDER()* Y *GETITEMCOUNT()* DE LA CLASE *ADAPTER_ITEM*

En la siguiente figura (Figura 28), extraída de la clase *Adapter_Item*, se observa la clase *AdapterViewHolder* de *Adapter_Item* y, debajo, una captura de pantalla del elemento xml denominado *card_view_item*. Este elemento está compuesto por varias vistas, entre las que se ha resaltado un *TextView* que representa un cuadro de texto que se usará para guardar el nombre del ítem que, a través del componente *id*, se enlaza con la clase *AdapterViewHolder*. De esta manera, cuando se cargue la lista de ítems, el *RecyclerView* reciclará la vista *card_view_item* mientras que el *AdapterViewHolder* asignar los datos correspondientes.

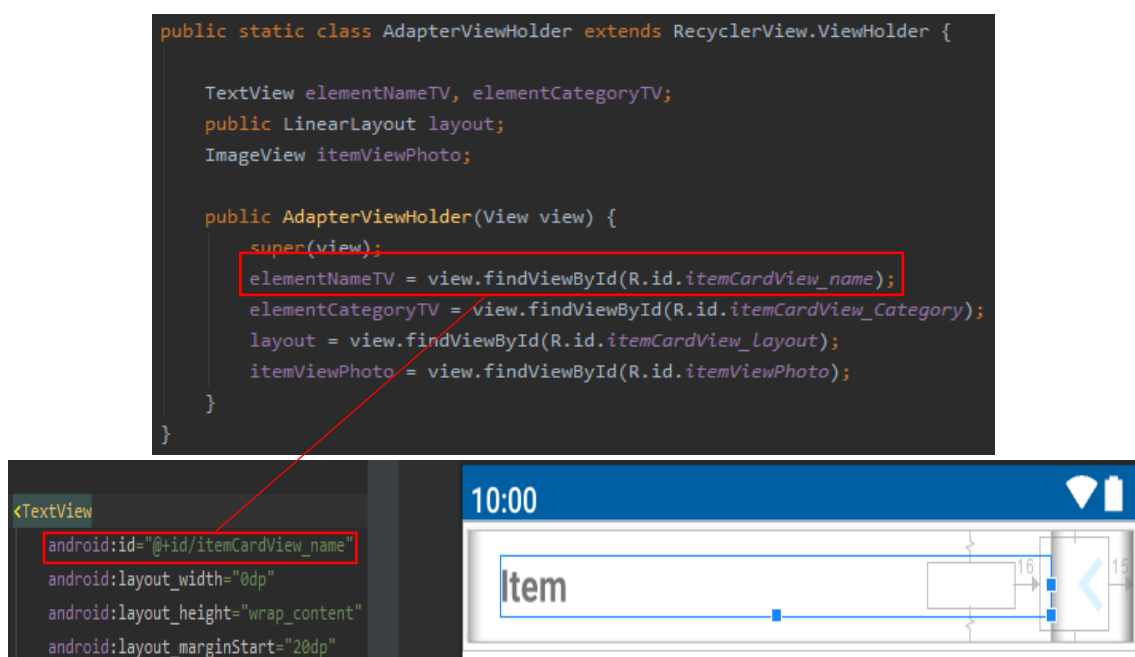


FIGURA 28: CAPTURA DE PANTALLA DE (ARRIBA) LA CLASE *ADAPTERVIEWHOLDER* PARA *ADAPTER_ITEM* Y (ABAJO) DEL RECURSO XML *CARD_VIEW_ITEM.XML*

Por otro lado, mientras que en la imagen anterior muestra la vista que se va a reciclar, en la siguiente imagen (Figura 29), obtenida del recurso xml *recyclerview_addbutton*, se visualiza el diseño del *RecyclerView*. Cuando se vaya a cargar cualquier la lista, ésta será la interfaz que verá el usuario.

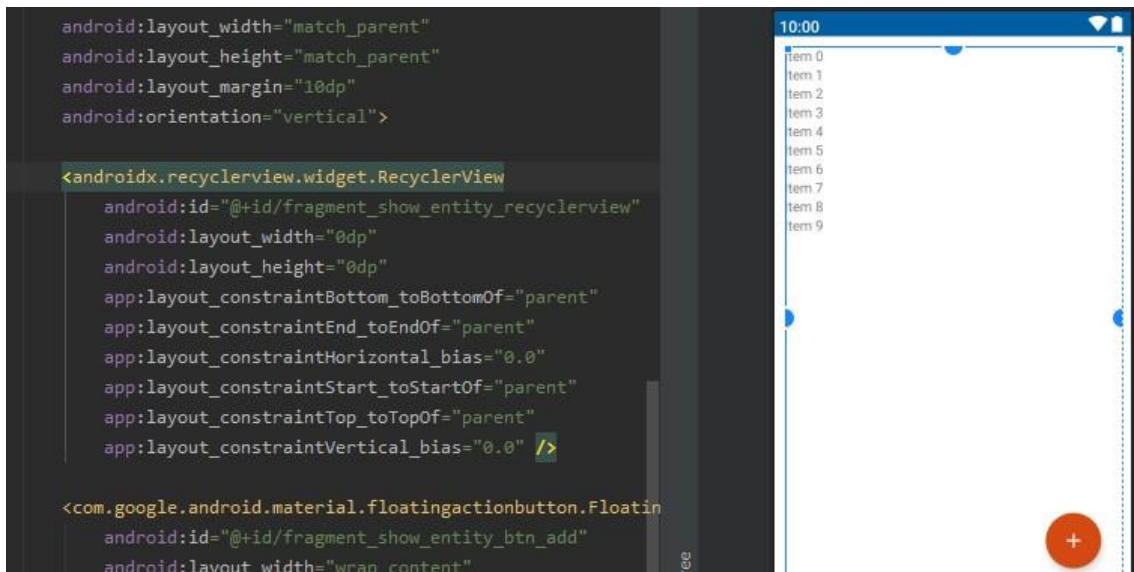


FIGURA 29: CAPTURA DE PANTALLA DEL RECURSO *RECYCLERVIEW_ADDBUTTON.XML*

Por último, en el siguiente fragmento de código (Figura 30), obtenido del proyecto, se muestra la implementación a nivel de código del *RecyclerView* de la clase *Fragment_ShowItems*. Cuando el usuario viaja a la pestaña en la que se muestra la lista de ítems, internamente, lo que ocurre es: se crea su adaptador (①), después se le asigna al *RecyclerView* (②) y, por último, se le agrega el evento *OnLongClickListener* para permitir que, al pulsar en algún elemento de la lista, se desencadene un evento (③).

```
recyclerView = view.findViewById(R.id.fragment_show_entity_recyclerview);
ItemTouchHelper.SimpleCallback simpleCallback =
    new Item_RecyclerViewItemTouchHelper( dragDirs: 0, ItemTouchHelper.LEFT, listener: this);
new ItemTouchHelper(simpleCallback).attachToRecyclerView(recyclerView);

recyclerView.setLayoutManager(new LinearLayoutManager(getContext()));
adapter = new Adapter_Item(dataset); ①
adapter.setActivity(getActivity());
recyclerView.setAdapter(adapter); ②

adapter.setListener((View v) -> { ③
    Item item = dataset.get(recyclerView.getChildAdapterPosition(v));
    editItemDialog(item);
    return true;
});
```

FIGURA 30: EXTRACTO DEL CÓDIGO DE LA CLASE *FRAGMENT_SHOWITEMS*

De esta manera ya se ha cumplido con el objetivo de mostrar la lista de ítems. Además, gracias a la manera en la que se ha diseñado el *RecyclerView* es la posibilidad de utilizarlo para cualquier lista que se quiere mostrar.

En las siguientes imágenes (Figura 31), capturadas en la aplicación en ejecución, se presentan tres pantallas donde se muestran tres listas diferentes utilizando *RecyclerView*. La primera captura se corresponde a la lista de ítems, de la cual se ha descrito su construcción anteriormente, mientras que la segunda se trata de la lista de maletas y la tercera la lista de categorías. En todas las imágenes el *RecyclerView* es el mismo, de manera que solamente hay que crear otra clase *RecyclerView.Adapter* junto con su *ViewHolder* asociado.

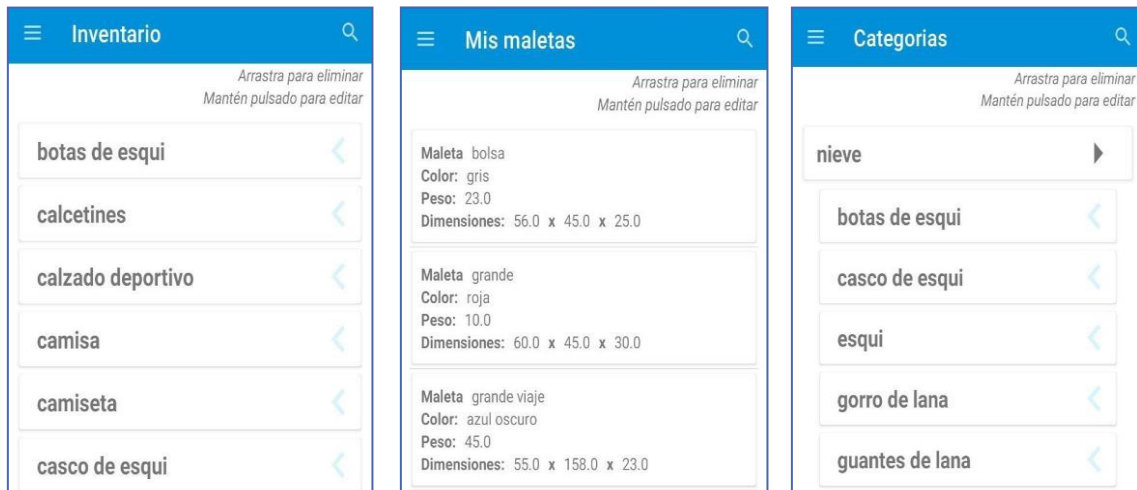


FIGURA 31: CAPTURAS DE PANTALLA DE LA APLICACIÓN EN EJECUCIÓN

Por otro lado, a la hora de trabajar con una lista de elementos es importante que ésta pueda ordenarse para mayor comodidad del usuario. Por defecto, las listas están ordenadas alfabéticamente, exceptuando los viajes, que se ordenan por fecha de salida. Además, como se vio anteriormente, es posible agregar el componente *Filterable* al adaptador del *RecyclerView*, permitiendo desarrollar un filtro para la lista y que ésta cambie en tiempo de ejecución automáticamente por medio de la herramienta de búsqueda *SearchView* [40].

En la siguiente figura (Figura 32), obtenida de la aplicación en ejecución, se exponen dos capturas de pantalla donde, en la imagen de la izquierda, está la lista de ítems y, en la de la derecha, la lista filtrada, en este caso, muestra aquellos elementos que empiecen por 'c'.

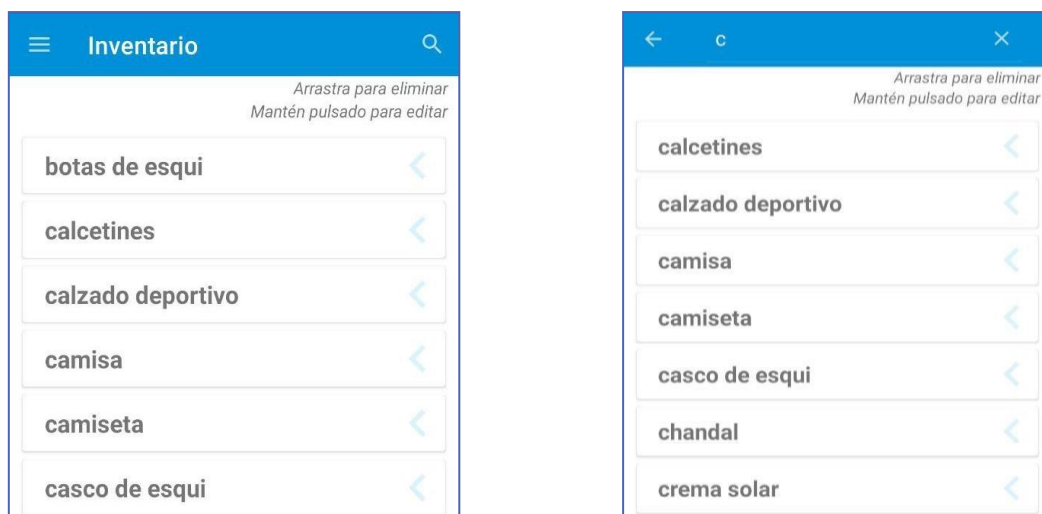


FIGURA 32: CAPTURA DE PANTALLA DE (A LA IZQUIERDA) LA LISTA DE ÍTEMS Y DE (A LA DERECHA) LA LISTA DE ÍTEMS FILTRADA DE LA APLICACIÓN EN EJECUCIÓN

3.1 Arquitectura

Para el desarrollo de la aplicación inicialmente, se planteó utilizar el patrón arquitectónico Modelo-Vista-Controlador (en siglas MVC); sin embargo, dada la naturaleza de las aplicaciones Android, al final se optó del Modelo-Vista-Presentador (en inglés Model-View-Presenter o sus siglas MVP).

MVP es un patrón arquitectónico derivado del MVC, orientado para la creación de interfaces de usuario y diseñado para facilitar pruebas de unidad y mejorar la separación de la lógica en la capa de presentación. Para ello, MVP divide la aplicación en tres partes:

- **Modelo:** son los datos, el estado y la lógica de negocio.
- **Vista:** interfaz gráfica con la que interacciona el usuario.
- **Presentador:** entidad intermedia entre modelo y vista.

De esta manera, se crea una clase *presentador* que aúna todos los métodos necesarios para interactuar con la capa de datos por lo que, cada vez que se quiera realizar una petición a la base de datos, ésta pasa por el presentador, quien se encarga de devolver la respuesta a la capa de presentación. En la siguiente imagen (Figura 33), se muestra una representación de ambos patrones junto con una tabla comparativa. La diferencia fundamental entre MVC y MVP radica en la clase intermediaria y su interacción con el resto de los componentes.

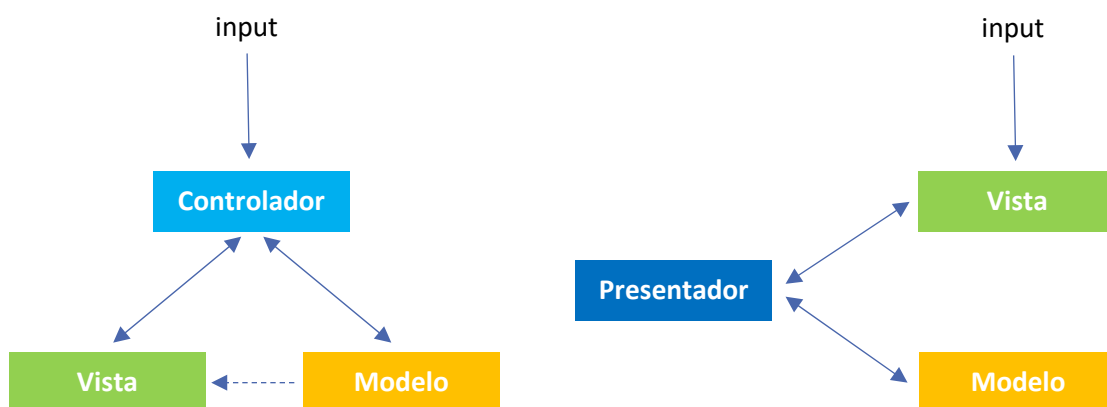


FIGURA 33: REPRESENTACIÓN DE LOS PATRONES (A LA IZQUIERDA) MVC Y (A LA DERECHA) MVP

| <i>Modelo Vista Controlador</i> | <i>Modelo Vista Presentador</i> |
|--|--|
| Gran acoplamiento entre componentes | Los elementos están separados, el Presentador es el intermediario entre la Vista y el Modelo |
| El Controlador es el responsable de determinar que Vista mostrar ante cualquier acción | El Presentador recoge peticiones, interactúa con el Modelo y devuelve la información, la Vista cambia por sí misma |
| Cada acción en la Vista genera una llamada al Controlador, el cual la procesa y devuelve una nueva Vista | La Vista recoge las acciones y las redirige al Presentador |
| La Vista no tiene ninguna lógica de manera que solo se encarga de renderizar | La Vista posee un mínimo de lógica para que, de manera independiente, pueda variar en función de las acciones |

Como se ha explicado anteriormente, existen componentes como son las Actividades y los Fragmentos que tienen tanto una parte lógica como una parte visual, ambas vinculadas. La parte lógica contiene, como mínimo, el código necesario para cargar la interfaz. Es por ello por lo que MVP funciona mucho mejor que MVC para proyectos Android. El patrón MVP se puede implementar de dos maneras distintas:

- **Vista pasiva:** el objetivo es hacer que la Vista tenga la menor lógica posible. La Vista sigue recibiendo las peticiones, enviándosela al Presentador y sin interactuar en ningún momento con el Modelo. En este caso, el presentador se encarga de indicar qué debe actualizar la Vista en función de la información proporcionada por el Modelo; es decir, aunque la Vista siga siendo quien se encarga de cargar los elementos, su contenido es configurado por el Presentador.
- **Vista activa:** la Vista se une al Modelo directamente a través de un enlace de datos proporcionado por el Presentador. El inconveniente al hacer esto es que la Vista interactúa con el Modelo directamente por lo que se pierde esa separación de inquietudes que se pretende conseguir en un principio.

Para este proyecto se ha optado utilizar un punto intermedio, donde la Vista no interactúa en ningún momento con el Modelo, pero si tiene más lógica aparte de cargar elementos de la interfaz. El presentador se encarga de recoger las peticiones del usuario que requieran del uso de la capa de datos, resolverlas y devolver una respuesta que la Vista será capaz de interpretar y de actualizarse por sí misma.

La capa de datos de una aplicación Android está compuesta por dos tipos de almacenamientos:

- **Compartido:** espacio accesible para el resto de las aplicaciones. Puede ser interno (memoria del propio dispositivo) o externo (volúmenes extraíbles, como las tarjetas SD). Para acceder al almacenamiento compartido hace falta declarar los permisos necesarios de lectura y escritura en el archivo manifiesto.
- **Específico:** se trata de archivos internos propios de la aplicación. Se utiliza para guardar información sensible y que no sea accesible por otros programas.

En el caso de este trabajo, se requieren dos espacios de almacenamientos. Uno compartido para las imágenes de los ítems de manera que, tanto las imágenes capturadas desde la aplicación como las que no, puedan ser utilizadas por cualquier programa. El otro espacio será para los datos internos, por lo que se utilizará un fichero específico dentro de la aplicación. Existen varias formas de almacenar información, siendo la principal el uso de archivos dentro de los directorios (ya sean compartido o específicos).

Otra manera es usando pares clave-valor, los cuales permiten guardar datos primitivos de forma sencilla. Una forma de manejarlo es a través de un objeto *SharedPreferences* [41] el cual facilita la creación de ficheros que contengan pares clave-valor además de facilitar métodos para leer y escribir en ellos. Este objeto es útil para guardar las preferencias del usuario; es decir, aquellos valores que ayudan a personalizar la experiencia del usuario como, por ejemplo, opciones de presentación, idioma, opciones de configuración, etc.

Por ejemplo, utilizando un *SharedPreferences*, se puede guardar localmente el modo en el que el usuario quiera que se le muestren los ítems pues, dentro de la aplicación, se pueden crear *Categorías* que son agrupaciones de ítems que facilitan el manejo de la lista de ítems. De esta manera se puede cambiar la presentación para que se muestre en diferentes bloques en función a la categoría a la cual estén asociados. En la siguiente imagen (Figura 34), obtenidas de la aplicación en ejecución, se puede observar, a la izquierda, la lista de ítems y, a la derecha, la misma lista, pero separada por categorías.

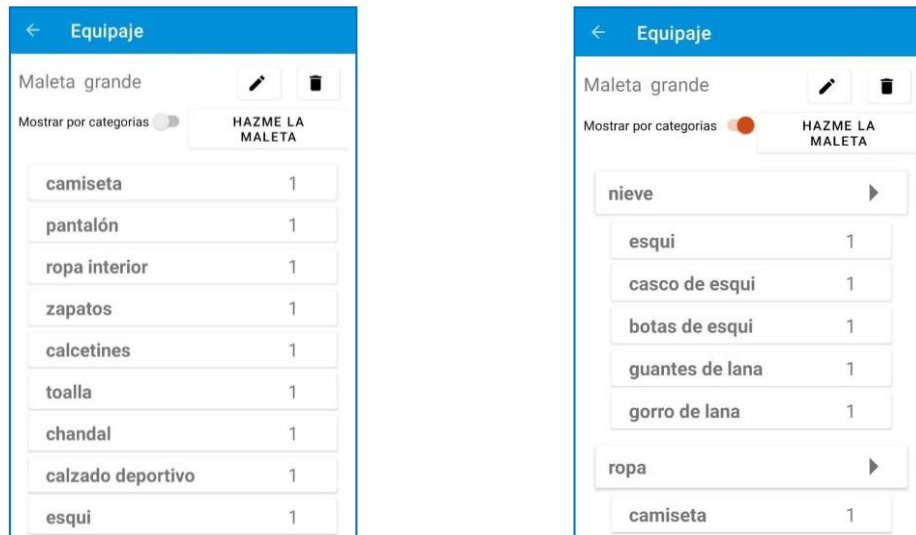


FIGURA 34: CAPTURA DE PANTALLA DE (A LA IZQUIERDA) LA LISTA DE ÍTEMS Y (A LA DERECHA) LA LISTA DE ÍTEMS SEPARADA POR CATEGORÍAS

Para cambiar de una vista a otra basta con pulsar el *Switch* que está encima de la lista. Este botón tiene dos estados: activado o desactivado, de manera que se puede aprovechar para trabajar con un valor booleano el cual se almacena en un *SharedPreferences*.

En el siguiente fragmento de código (Figura 35), extraído del código del proyecto, se muestra la implementación de esta funcionalidad: primero se obtiene el valor actual almacenado en el *SharedPreferences* (①) y después se agrega un evento *setOnCheckedChangeListener()* para que, en caso de que se pulse el botón, se altere la vista (②).

```

① switchShowCategories = view.findViewById(R.id.switch_ShowCategory);
   sp = getContext().getSharedPreferences( name: "settings", Context.MODE_PRIVATE);
   SharedPreferences.Editor editor = sp.edit();
   boolean showCategories = sp.getBoolean( key: "showCategories", defValue: false);

② switchShowCategories.setOnCheckedChangeListener((buttonView, isChecked) -> {
   editor.putBoolean("showCategories", true);
   editor.apply();
   getNav().navigate(R.id.action_fragment_ShowBaggageByItem_to_fragment_ShowBaggageByCategory, bundle);
});

```

FIGURA 35: EXTRACTO DEL CÓDIGO DE LA CLASE *FRAGMENT_SHOWBAGGAGEBYITEM*

Aparte del SharedPreferences, es posible agregar una base de datos a un proyecto Android. Para ello, Android utiliza el lenguaje SQLite para su creación y manejo. Se trata de un gestor de bases de datos relacional que se enlaza a la aplicación formando parte de ésta. Destaca por ser ligero, de código abierto y autónomo, además de permitir almacenar información persistente de forma sencilla sin necesidad de un servidor. Todos estos atributos más el hecho de que cumple con las características *ACID* (atomicidad, consistencia, aislamiento y durabilidad) hacen que se opte SQLite por encima de otros gestores de bases de datos, pues los dispositivos móviles tienen recursos más limitados que otros computadores.

En el siguiente esquema (Figura 36) se muestra la estructura de la base de datos de la aplicación.

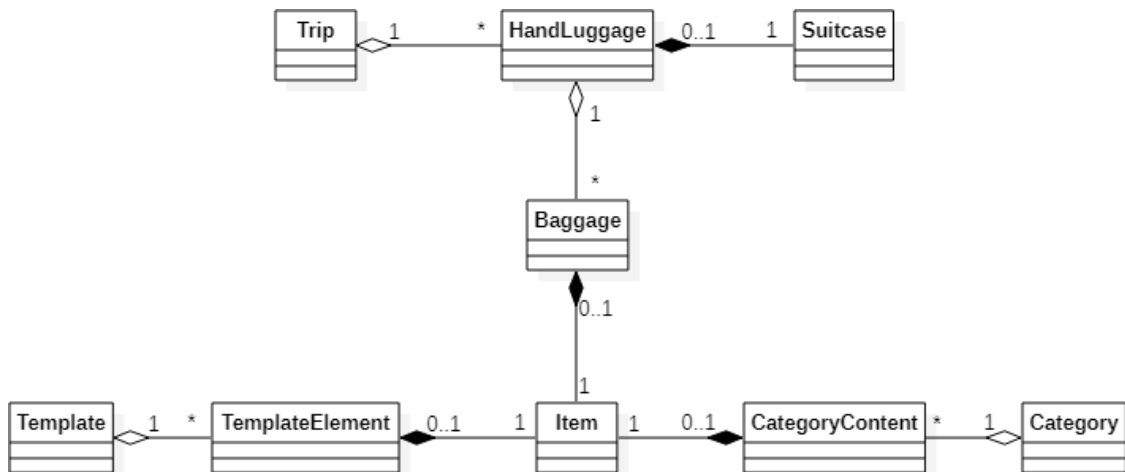


FIGURA 36: ESQUEMA DE LA BASE DE DATOS DE LA APLICACIÓN

- **Item:** representa los diferentes elementos que se pueden agregar a una maleta.
- **Category:** son agrupaciones de ítems. Un ítem no puede estar en más de una categoría simultáneamente.
- **Suitcase:** se trata de las maletas que se utilizará para un viaje.
- **Trip:** son los viajes que va a realizar, esté realizando y haya realizado el usuario.
- **CategoryContent:** indica que ítems están dentro de una categoría.
- **HandLuggage:** equipaje (ítems dentro de una maleta) que se lleva en un viaje.
- **Baggage:** entidad que asocia que ítems van dentro de una maleta
- **Template:** plantillas de maletas
- **TemplateElement:** elementos que forman una determinada plantilla

Se puede distinguir un total de nueve entidades, las cuales se pueden clasificar en: principales o secundarias, siendo las principales aquellas que son independientes una de otras. Estas son *Item*, *Category*, *Suitcase*, *Template* y *Trip*. Es a través de las entidades secundarias, *CategoryContent*, *HandLuggage*, *TemplateElement* y *Baggage*, que se establece las relaciones entre entidades principales.

Se ha decidido este diseño debido a su flexibilidad y versatilidad, por ejemplo, si se crea una categoría a la cual se le van a agregando ítems, si más adelante se elimina, los ítems se mantienen y, por otro lado, si se eliminan ítems, simplemente dejan de aparecer en la categoría. Esto es gracia a la entidad intermedia *CategoryContent*.

Una característica fundamental de la base de datos es el concepto de *maleta* y *equipaje*. Para evitar confusiones y solapamiento de funcionalidades se definen de la siguiente manera:

- **Maleta:** es el contenedor que se usa para transportar el equipaje.
- **Equipaje:** ítems que se lleva en un viaje determinado. En un viaje se puede llevar tantas maletas como desee el usuario y cada maleta, a través de la entidad *HandLuggage*, tendrá asociado un equipaje.

Por otro lado, aunque no exista una relación directa entre el equipaje y las categorías, es bastante sencillo establecer un vínculo directo a través de la entidad *Item*.

Para la implementación de la base de datos se ha hecho uso de la librería de base de datos *Room* [42], desarrollada por el equipo de Android, que proporciona una capa de abstracción sobre SQLite, facilitando la creación y gestión de la base de datos. En la siguiente imagen (Figura 37), extraída de la página para desarrolladores de Android [42] se muestran los tres componentes que componen *Room*:

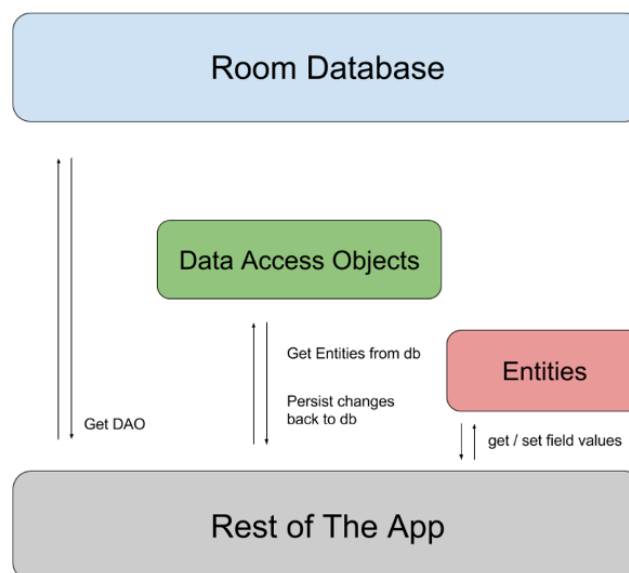


FIGURA 37: DIAGRAMA DE LA ARQUITECTURA DE ROOM

- **Room Database:** definida por la etiqueta `@Database`, se trata de una única clase abstracta que incluye todas las entidades declaradas y que posee un método con el cual se puede instanciar, de manera que cumple el patrón *singleton*.
- **Entidad:** identificada con `@Entity`, representa una tabla dentro de la base de datos.
- **Data Access Objects:** más conocida por sus siglas DAO, es una clase que contiene los métodos utilizados para acceder a la base de datos. Se le agrega la etiqueta `@Dao` y es donde se declaran las sentencias SQL.

Por cada tabla hay una clase Entidad y una clase DAO, de manera que se puede trabajar individualmente con cada elemento de la base de datos y, como se ha explicado anteriormente, las peticiones hacia la base de datos se hacen a través de la clase *presentador*.

En el siguiente fragmento de código (Figura 38), extraído del proyecto, se observa la estructura de la clase *Item* que se corresponde a la entidad ítem de la base de datos. Se define con la etiqueta *@Entity* indicando que el nombre de la tabla será *items* y, además, podemos ver que tiene dos campos: el identificador del ítem designado como clave primaria *itemID* y el nombre del ítem en formato String. Además, la clave primaria se generará automáticamente para cada nuevo elemento y su manejo lo gestiona la librería *Room* de manera interna.

```
@Entity(tableName = "items")
public class Item implements Serializable {
    @PrimaryKey(autoGenerate = true)
    public int itemID;

    @ColumnInfo(name = "itemName")
    public String itemName;
```

FIGURA 38: DECLARACIÓN DE LA CLASE ITEM

Por otro lado, en la siguiente captura (Figura 39), obtenida del proyecto, se muestra la implementación de la clase *itemDAO* que, señalizada con la etiqueta *@Dao*, contiene los métodos de acceso a la tabla *items*. A cada método se le asocia una petición SQL a la base de datos la cual se recoge utilizando etiquetas. Existen etiquetas que llevan implícita una operación como son *@Insert*, *@Update* y *@Delete* que agregan, actualizan y eliminan un registro de la tabla asociada respectivamente mientras que, para construir una sentencia SQL personalizada se utiliza la etiqueta *@Query*. De esta manera es muy sencillo definir métodos de acceso a la base de datos ya que *Room* se encargará, de manera interna, de la conexión, la validación y la persistencia de la misma.

```
@Dao
public interface ItemDAO {
    @Query("SELECT * FROM items ORDER BY itemName")
    List<Item> selectAll();

    @Update
    void update(Item item);

    @Insert
    void insert(Item item);

    @Delete
    void delete(Item item);
```

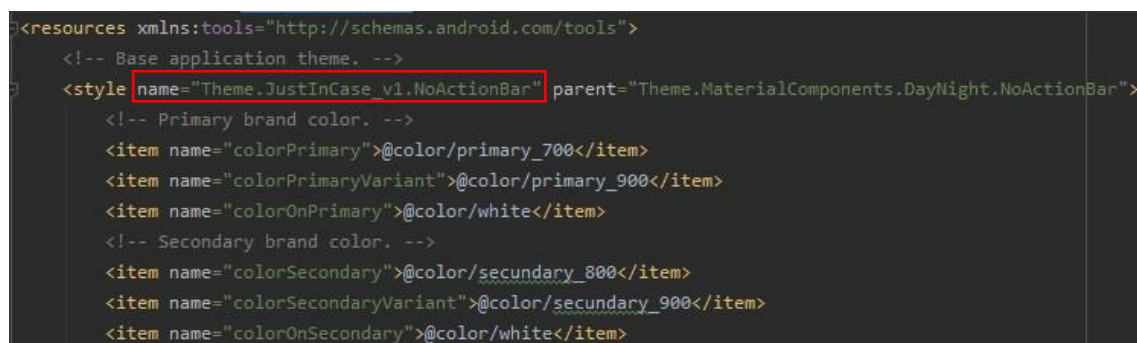
FIGURA 39: CAPTURA DE PANTALLA DE LA CLASE ITEMDAO

3.2 Diseño

A medida que se iba desarrollando tanto la parte lógica como la interfaz de usuario también se fue trabajando en lo que se conoce como el “*estilo de la aplicación*”. En Android existen dos conceptos de diseño: los estilos y los temas. Mientras que un *estilo* (en inglés, *style*) es un conjunto de características que definen la apariencia de un solo elemento de la interfaz, como puede ser un botón, un texto o una imagen; un *tema* (en inglés, *theme*) se aplica a toda la aplicación. Existe una jerarquía de diseño donde primero se aplica el *tema* y después los *estilos*, siendo una forma de trabajo similar a la utilizada en aplicaciones web mediante las hojas de estilo y el lenguaje de diseño gráfico CSS (por sus siglas en inglés, *Cascading Style Sheets*).

A la hora de diseñar la interfaz de usuario, es fundamental definir el *theme* principal con el objetivo de mantener una coherencia cromática entre las diferentes pantallas, así como usar siempre los mismos elementos para acciones similares. Para ello, Android Studio separa en una carpeta los distintos recursos *xml* que contengan valores de estilo de manera que son identificables y de fácil acceso. Dentro de estos recursos *xml* se puede definir el *theme* y, mediante el archivo manifiesto, vincularlo a la interfaz.

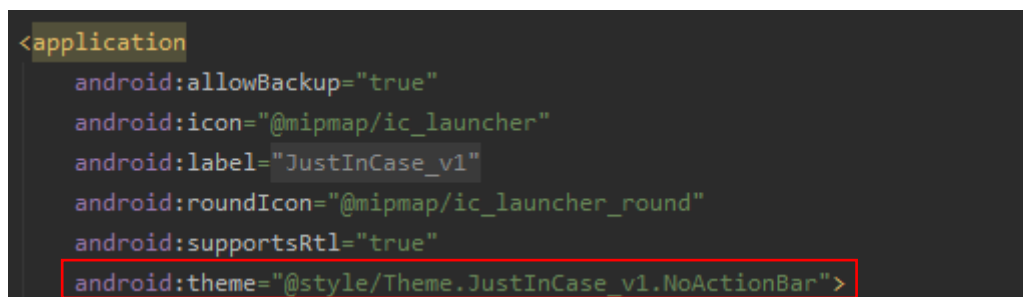
En la siguiente imagen (Figura 40), extraída del proyecto, se muestra el archivo *themes.xml*, donde podemos ver que el *theme* de la aplicación está formado por varios elementos designados con las etiquetas *<item>*, a las cuales se les asocia un identificador, siendo el nombre del atributo de estilo, y un valor, siendo el color asignado. Estos pares clave-valor se agrupan bajo la etiqueta *<style>* que, a través del nombre, identifica el estilo de manera exclusiva.



```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.JustInCase_v1.NoActionBar" parent="Theme.MaterialComponents.DayNight.NoActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/primary_700</item>
        <item name="colorPrimaryVariant">@color/primary_900</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/secondary_800</item>
        <item name="colorSecondaryVariant">@color/secondary_900</item>
        <item name="colorOnSecondary">@color/white</item>
    </style>
</resources>
```

FIGURA 40: CAPTURA DE PANTALLA DEL ARCHIVO *THEMES.XML*

En adición a la imagen preliminar, se presenta la siguiente captura de pantalla (Figura 41), extraída del archivo manifiesto *AndroidManifest.xml*, en la que se ha señalado la sentencia donde se vincula el recurso mostrado en la figura anterior con la aplicación. De esta manera, los valores descritos en el fichero *theme.xml* afectarán a todos los elementos de la interfaz.



```
<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="JustInCase_v1"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.JustInCase_v1.NoActionBar">
```

FIGURA 41: CAPTURA DE PANTALLA DEL ARCHIVO *ANDROIDMANIFEST.XML*

Un detalle importante que podemos identificar del fichero *theme.xml* es que los colores no están designados directamente, sino que el valor que se ha utilizado es una referencia a otro documento. Este archivo es *colors.xml* y, como su nombre indica, su función es la de guardar los colores que se van a utilizar en la aplicación. Esto se ha hecho de esta manera para facilitar el acceso a los mismos recursos, tanto por parte del *theme* general, como por parte del resto de estilos específicos que se apliquen a la interfaz y así todos los elementos de la interfaz usen la misma paleta de colores.

En la siguiente imagen se muestra el fichero *colors.xml* (Figura 42), extraído del código del proyecto. En él se pueden distinguir dos partes diferenciadas: la primera denominada “*primaria*”, compuesta por varios tonos azules, y la segunda, como “*secundaria*” la cual utiliza tonos naranjas. Estos bloques de colores son los que se van a utilizar principalmente en la aplicación, aunque también se puede apreciar que, debajo, hay más colores. Éstos son usados para propósitos específicos como, por ejemplo, el `<color name="hover">` que se corresponde con el color del botón de la cámara cuando éste es pulsado, o el `<color name="item_selected">` que es la tonalidad de azul que se utiliza para resaltar aquellos elementos de un *RecyclerView* que hayan sido seleccionados, como ocurre cuando se quiere agregar un nuevo ítem a categoría o cuando se agrega una maleta a un viaje.

```

1      <?xml version="1.0" encoding="utf-8"?>
2      <resources>
3
4          <!-- primary -->
5          <color name="primary_300">#4dc9fc</color>
6          <color name="primary_700">#0090d9</color>
7          <color name="primary_900">#005ea3</color>
8
9          <!-- secondary color -->
10         <color name="secondary_900">#bc3d0a</color>
11         <color name="secondary_800">#d54a12</color>
12         <color name="secondary_400">#fc7542</color>
13
14         <color name="black">#000000</color>
15
16         <color name="item_selected">#b3e8fd</color>
17
18         <color name="white">#ffffff</color>
19         <color name="hover">#b3e8fd</color>
20
21     </resources>

```

FIGURA 42: CAPTURA DE PANTALLA DEL ARCHIVO *COLORS.XML*

La razón que hay detrás de esta estructura en bloque de colores, radica en la teoría del color [43] y cómo, mediante tonos, se puede construir una interfaz más amigable, clara y concisa. De esta manera, los fondos, las barras de herramientas y todos los elementos principales de la aplicación utilizan los tonos azules del bloque primario, mientras que botones y otros elementos secundarios utilizarán los tonos naranjas del bloque secundario.

La elección de estos colores no ha sido totalmente arbitraria puesto que, primero, se partió de querer usar un color similar al que utiliza la universidad de Las Palmas de Gran Canaria como seña identidad, tanto en su logo como en sus productos (páginas webs, *merchandising*, presentaciones, etc.) y después se buscó cual sería el complementario más adecuado. Para ayudarnos con las tonalidades, se hizo uso de la página *Material Design* [44] cuyo sistema de diseño de código abierto está administrado y gestionado por Google, por lo que además posee facilidades para su uso en aplicaciones Android. En esta página se puede encontrar y utilizar una herramienta para la creación de una paleta de colores [45], así como documentación acerca de teorías del color y de diseño interfaces.

En la siguiente imagen (Figura 43), se puede observar la herramienta mencionada anteriormente para la creación de una paleta de colores. En ella, se ha puesto el valor del color principal de la aplicación generando el código de colores correspondiente.

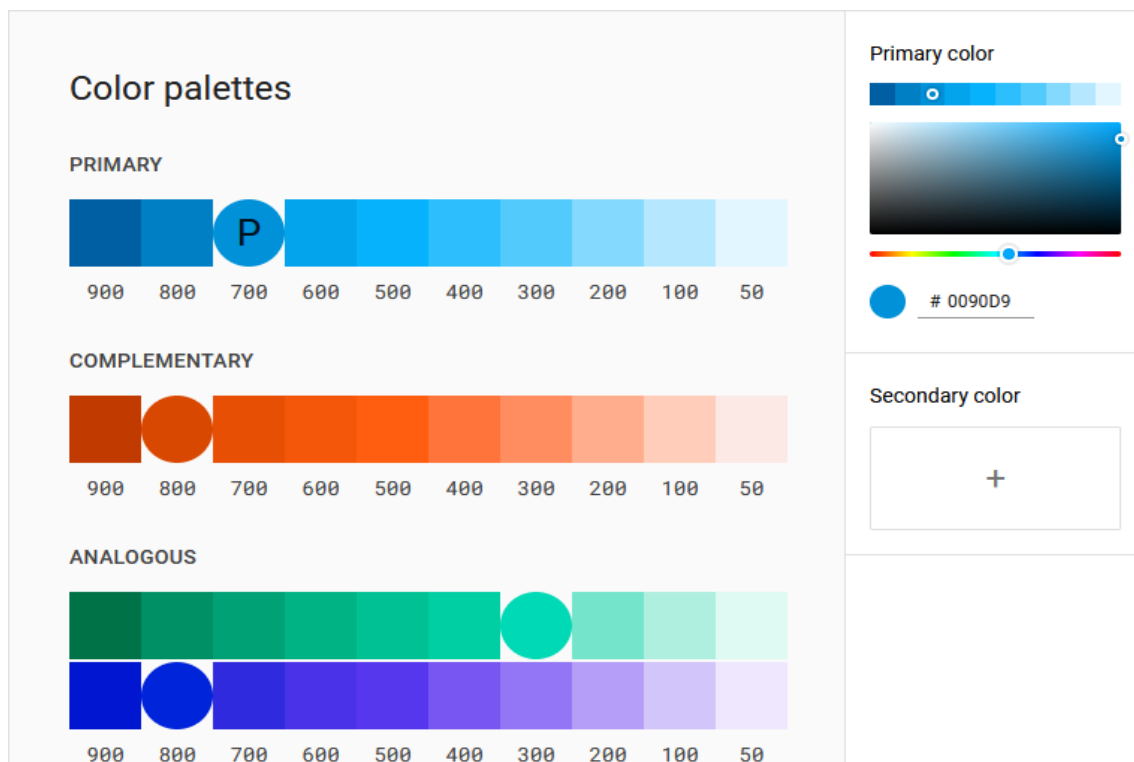


FIGURA 43: CAPTURA DE PANTALLA DE LA HERRAMIENTA PARA LA CREACIÓN DE PALETAS DE COLORES DE LA PÁGINA WEB *DESIGN MATERIAL*

En función de la cantidad de elementos y la profundidad de detalles que queramos darle a la interfaz, utilizaremos más, o menos, tonalidades e, incluso, otros colores. En el caso de esta aplicación, como se buscaba un diseño más limpio, sencillo y claro, se optó por utilizar una paleta de colores bastante simplificada; sin embargo, y como se puede ver, añadir más colores es una tarea sencilla pues bastaría con agregarlo a la paleta actual a través del archivo *colors.xml*.

También es posible definir varios *themes* dentro de la aplicación permitiendo que el usuario pueda escoger entre una amplia gama de colores que desee. Esta *preferencia* se puede guardar en un *SharePreference* que indicará a la aplicación la paleta a utilizar a la hora de cargar los elementos. Además, el uso de otros *themes* es útil también de cara a diseñar una *versión oscura*, más conocida como *modo noche* o *modo nocturno*, de la aplicación para que ésta no moleste a la vista cuando se use en entornos con poca luz.

Aparte de dar una apariencia única a la aplicación, el uso del color puede servir para facilitar la navegación mediante técnicas de diseño como *calls to action* [46] y *espacio en blanco* [47]:

- **Call to action:** se trata de diseñar elementos de la interfaz que destaquen y actúen como reclamo para el usuario. Un ejemplo muy utilizado en las aplicaciones webs, son los botones de “únete” o “regístrate”, botones que contrastan con el resto de los elementos y que suelen resaltar de algún modo gracias al tamaño, su posición y/o su contraste.
- **Espacio en blanco o *whitespace*:** es todo lo que rodea a un elemento de la interfaz pero que no aporta nada como, por ejemplo, un fondo en blanco. De esta manera, los elementos ganan peso en la pantalla incentivando a que los usuarios los utilicen.

Estas técnicas, sumada a otras más, ayudan a diseñar una interfaz tanto de apariencia más agradable al usuario, como funcional. En la siguiente figura (Figura 44) se presentan dos imágenes capturadas de la aplicación en ejecución. En la primera se muestra la lista de ítems cuando no se ha creado ninguno, mientras que, en la otra, se muestra como se ve una vez se han agregado varios ítems. En esta interfaz podemos observar varias características:

1. La barra de herramientas (①) situada en la parte superior de la pantalla utiliza el color principal de la aplicación, el azul y los textos e iconos se resaltan utilizando el blanco.
2. El botón flotante (②) para crear un nuevo ítem usa el naranja, contrastando con el *theme* principal y está ubicado en la esquina inferior derecha para mayor comodidad del usuario, que, al ser un teléfono móvil lo tiene más fácil de alcanzar y presionar.
3. Entre las dos pantallas no hay cambios de interfaz: los ítems se van agregando uno debajo de otro (③) en encuadres concretos, respetando márgenes y espacios.
4. Con una flecha del color principal (④) pero en una tonalidad muy clara, indica la acción que debe realizar el usuario para eliminar un ítem de la lista.

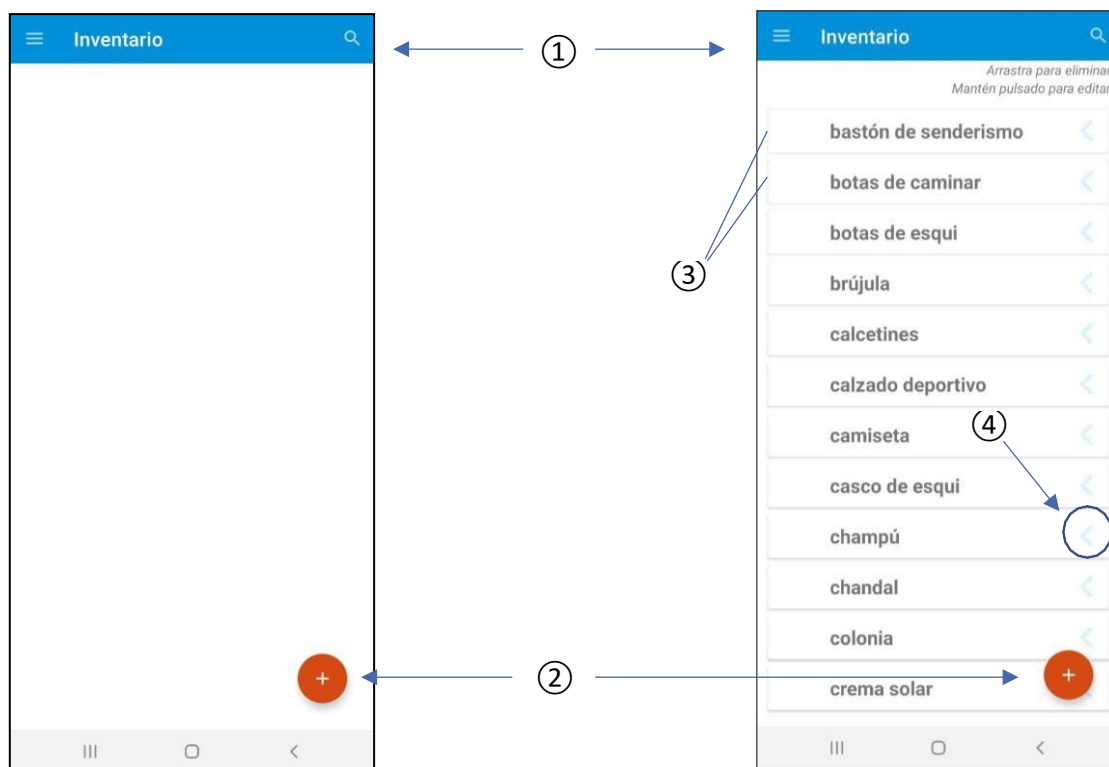


FIGURA 44: ANÁLISIS DE CAPTURAS DE PANTALLA DE LA APLICACIÓN EN EJECUCIÓN, A LA IZQUIERDA LA PANTALLA “INVENTARIO” SIN ÍTEMS Y, A LA DERECHA, CON ÍTEMS

Otro aspecto importante para tener en cuenta a la hora de realizar el diseño de la interfaz, es el menú de la aplicación pues, cómo se vio anteriormente en el apartado de desarrollo (Sección 3), se utilizó un menú desplegable o *drawer menú*. Para su implementación se utilizó el espacio situado en la parte superior izquierda de la pantalla y, en función de a qué profundidad se encuentre el usuario, el icono alternará entre las tres barras horizontales, a una flecha horizontal señalando hacia la izquierda. Esto ayuda al usuario a poder acceder con facilidad a los diferentes puntos de la aplicación y permite rehacer los pasos que haya llevado a cabo sin entorpecer la navegación.

Existen más conceptos de diseño, como es el caso de las micro-interacciones, las cuales consisten en agregar pequeñas acciones en la interacción del usuario con la aplicación a la hora de realizar una funcionalidad, con el objetivo de notificarle de manera visual sobre algún cambio de estado. Un ejemplo de esto es el “doble check” de las aplicaciones de comunicación como Whatsapp o Telegram, informando que el mensaje ha sido enviado y recibido por el otro usuario.

Por otro lado, otro recurso similar son los cuadros de diálogos, los cuales se pueden utilizar para guiar la acción. Un ejemplo es cuando el usuario navega hasta la pantalla de la lista de ítems, pero está vacía. Entonces, el sistema lo detecta automáticamente y, mediante una ventana emergente pregunta al usuario por la acción a realizar.

Es a partir del empleo de estos conocimientos que se diseña y desarrolla una plantilla, la cual es utilizada para la creación de la interfaz. En el siguiente conjunto de imágenes (Figura 45) se muestra varias capturas de diferentes pantallas de la aplicación.

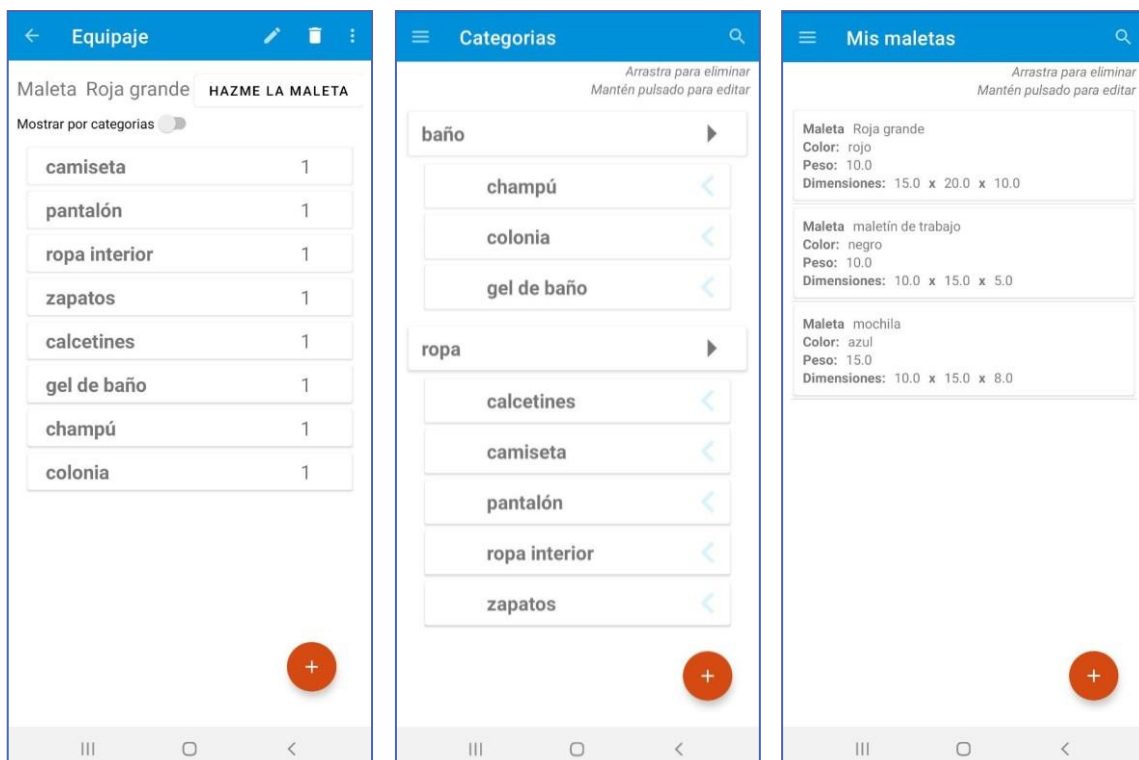


FIGURA 45: CAPTURAS DE PANTALLA DE (A LA IZQUIERDA) “EQUIPAJE” DE UN VIAJE, (AL CENTRO) “CATEGORÍAS” Y (A LA DERECHA) “MIS MALETAS”

Cómo se puede apreciar son prácticamente iguales, siguiendo la misma estructura y variando en el contenido que se presenta. La ubicación de los diferentes recursos, el color y su posición se mantienen, dando una coherencia funcional a la aplicación, haciendo que su uso sea intuitivo y sencillo. También se observa la aplicación de los conceptos mencionados, como el caso del *whitespace* puesto que existe un gran espacio entre los principales elementos y son claramente diferenciables unos de otros, o como el caso del *call to action* que genera el botón flotante estando ubicado en una zona exclusiva de la interfaz, apartado del resto de interacciones y contrastando mediante el uso del color.

Para aquellos casos en los que la interfaz se complica, es importante tratar de mantener la relación con el resto de las pantallas. En la siguiente figura (Figura 46), extraída de la aplicación en ejecución, se presenta un ejemplo de adaptación para tres casos: cuando se muestra la lista de viajes, a la hora de agregar ítems a una maleta y en la funcionalidad de generar el equipaje. En los tres casos la interacción con la aplicación es diferente con respecto al resto de funcionalidades por lo que, usando los mismos elementos, es necesario hacer ciertas modificaciones como, por ejemplo, el botón de acción o cómo se distribuye el contenido.

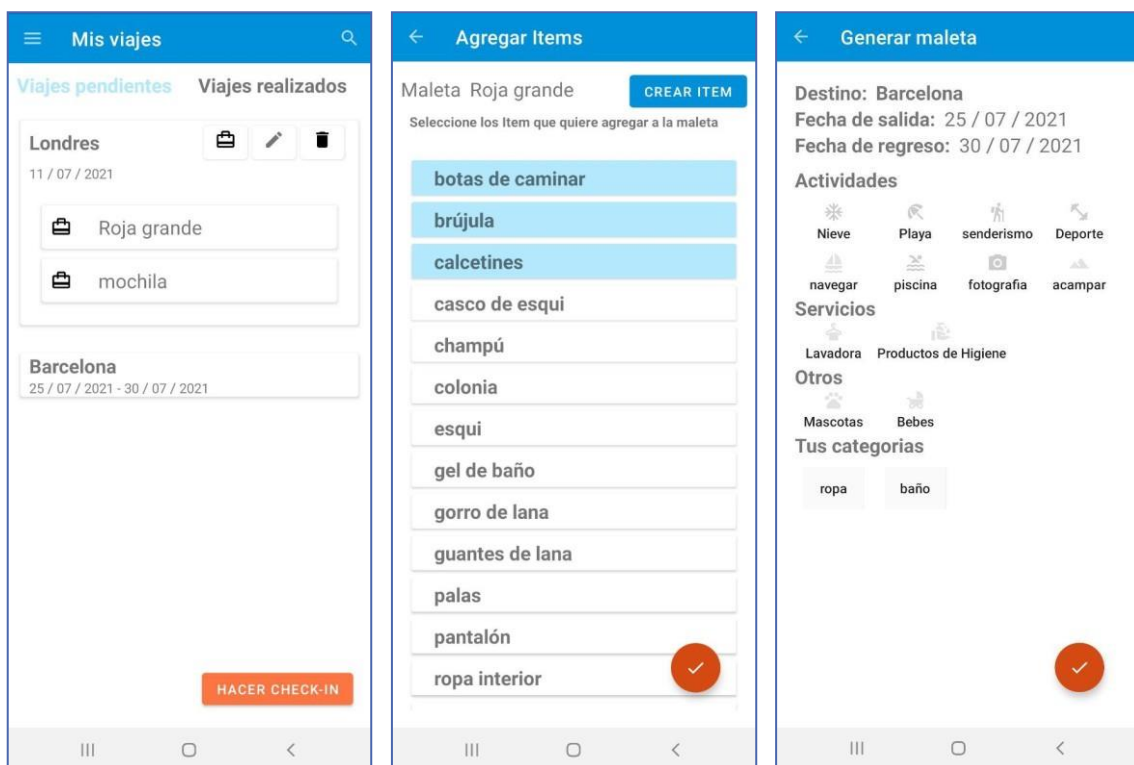


FIGURA 46 CAPTURA DE PANTALLA DE (A LA IZQUIERDA) “MIS VIAJES”, (AL CENTRO) “AGREGAR ITEMS” A UNA MALETA Y (A LA DERECHA) “GENERAR MALETA”

A la hora de trabajar con el *estilo* de los elementos gráficos, hay que tener en cuenta de que se pueden especificar una gran variedad de atributos, desde el color y las dimensiones hasta el tipo y el tamaño de la fuente, así como profundidad, contraste, animaciones, efectos, etc. Por tanto, la importancia de establecer un *theme* radica en darle una apariencia propia, funcional y coherente a la interfaz de la aplicación.

3.3 Inconvenientes en el desarrollo

Durante la implementación del código han surgido varios inconvenientes que han afectado de manera negativa al desarrollo del proyecto. Estas complicaciones se deben, en la mayoría de los casos, al desconocimiento de la tecnología en la que estamos trabajando puesto que, aunque tengamos acceso a toda la información y que, antes de implementar cualquier método, se hace una fase previa de estudio y análisis, en varias ocasiones surgen detalles y complicaciones que no fueron contempladas en dicha fase. Es por ello por lo que ciertas funcionalidades se han ido replanteando conforme se avanzaba.

El caso más importante fue con las funcionalidades de *compartir maletas y viajes en grupo*. Según un el documento TFT01 “*Se podrán crear grupos de listas para los viajes en grupo de manera que se pueda visualizar el estado del equipaje individual y compartido de todos sus miembros*”. Con esto, la idea que se quería llevar a cabo era permitir la capacidad de, por un lado, crear grupos de viajes donde un usuario podría ver las maletas de los demás miembros y, por otro, poder compartir las maletas con otros usuarios de la aplicación. Un ejemplo de uso de esta funcionalidad sería el de una familia donde la madre o el padre pueda consultar las maletas de los hijos.

Al no ser una característica que forme parte del núcleo de la aplicación, ésta se fue posponiendo para implementar más adelante; sin embargo, a la hora de plantearla y ver cuáles son las opciones para llevarla a cabo, nos encontramos con que se requiere de una infraestructura bastante grande y compleja. Esto es debido a que necesitaría un método para la identificación de usuarios, poder realizar conexiones entre ellos y un sistema de comunicación. A raíz de esta problemática se optó por buscar una alternativa menos costosa y, en un futuro, mejorar dicha funcionalidad.

Otro problema con el que nos enfrentamos, fue el manejo de las listas utilizando *RecyclerView*. Como se ha dicho anteriormente, *RecyclerView* utiliza dos listas simultáneas, una es la que se muestra al usuario por pantalla y la otra es la que va guardando los datos que se han ido cargado pero que se han dejado de visualizar. Un problema que surgió a causa del manejo de estas dos listas es el solapamiento de datos, mostrando datos duplicados o colocándose en lugares incorrectos además de que también afectaba a los eventos *OnClickListener()* de manera que, al pulsar sobre un elemento de la lista, se seleccionaba otro o se hacía una selección múltiple generando errores de ejecución que provocaban el fallo completo de la aplicación. La solución para evitar que se produjera este error fue encapsular por completo los datos que carga el *RecyclerView*, tratando de minimizar el número de variables locales con las que trabaja.

Otra complicación que surgió en el desarrollo fue con la funcionalidad de la cámara. En primer lugar, se presentó un problema con la variabilidad entre versiones. En la API de Android, se pueden encontrar 3 librerías diferentes: *Camera*, *Camera2* y *CameraX* siendo esta última la versión más moderna pero que, para el momento en el que se iba agregar la nueva funcionalidad, estaba en fase de desarrollo, concretamente, la página oficial para desarrolladores Android señalaba que *CamaraX* se encontraba en fase *Beta*; es decir, una versión temprana aún sin terminar, sujeta a cambios y, posiblemente, inestable.

Por estas razones, al principio se hizo uso de *Camera2*; sin embargo, en mitad del desarrollo salió una nueva actualización de *CameraX* donde, entre otras cosas, indicaba que era ésta la librería que se tenía que usar, pues tanto *Camera* como *Camera2* pasaban a quedar obsoletas y a la mayoría de los métodos se les agregó la etiqueta *@Deprecated*. A causa de esto, se tardó el

doble de tiempo en implementar la funcionalidad de cámara, teniendo en cuenta, además, que *CameraX* no poseía tanta documentación y ejemplos como *Camera2*.

En segundo lugar, tras la implementación de la funcionalidad de cámara, ocurre que, debido a que las fotos obtenidas eran de muy alta calidad, provocan que la aplicación se ralentice notablemente. Para solventar esto se tuvo que hacer un estudio exhaustivo de formatos, escalado de imágenes y manejo de memorias puesto que los formatos a utilizar, JPG y PNG, tienen procesos de compresión distintos y generan diferentes resultados en función del tratamiento que se le quiera dar a la imagen [48] [49].

En tercer lugar, *Room* no permite guardar imágenes ni objetos complejos como *Bitmap* o *Drawable* en la base de datos, por lo que para vincular un ítem a una imagen se tiene que hacer mediante una string la cual recoge la URI donde ésta se ubica. En consecuencia, cada vez que se vaya a mostrar un ítem al que se le haya asignado una imagen se debe efectuar un proceso de búsqueda y carga en la memoria del dispositivo, lo que ralentiza mucho la aplicación. Para resolver este inconveniente se ha hecho uso de un método interno encargado de cargar y almacenar las URIs manera local y temporal cuando se quiera visualizar las diferentes imágenes, por lo que el procedimiento solo se tiene que ejecutar una sola vez.

Por último, existe otro problema de versión relacionado con el objeto a utilizar para el manejo de imágenes a nivel de código. Actualmente, Android recomienda usar un *ImageDecode* [50] para trabajar con cualquier elemento gráfico, desde dibujos hasta fotografías, pero, para utilizarlo, se requiere la versión 28 del SDK; es decir, de la API de Android la cual se corresponde a la versión 9 de Android. Eso provocaría que, a fecha de realización de este proyecto, aproximadamente el 39'5% de los dispositivos podrían instalar y utilizar la aplicación. Este caso se abordó implementando de ambas maneras: por un lado, la versión actualizada “tal como indica los parámetros de Android” y por otro, la versión antigua.

En la siguiente imagen (Figura 47), extraída de la clase *Adapter_Item* del código del proyecto, se muestra una parte del código desarrollado para cargar la imagen asociada a un ítem. Con la sentencia “if (*Build.VERSION.SDK_INT* < 28)” se comprueba si el dispositivo soporta, o no, la versión actualizada y ejecuta el método correspondiente. Para cuando más dispositivos hayan sido actualizados solo será necesario eliminar la rama obsoleta y dejar la actualizada.

```
if (Build.VERSION.SDK_INT < 28) { //pre-Android 9
    Bitmap bitmap = MediaStore.Images.Media.getBitmap(activity.getContentResolver(), Uri.parse(item.getItemPhotoURI()));
    Bitmap resizedBitmap = Bitmap.createScaledBitmap(bitmap, dstWidth: 25, dstHeight: 25, filter: true);
    photoDataset.add(resizedBitmap);
} else { // post-Android 9
    ImageDecoder.Source source = ImageDecoder.createSource(activity.getContentResolver(), Uri.parse(item.getItemPhotoURI()));
    //This method may take several seconds to complete, so it should only be called from a worker thread - API Android
    Bitmap bitmap;
    bitmap = ImageDecoder.decodeBitmap(source, (decoder, info, src) -> {
        decoder.setTargetSampleSize(4);
    });
    photoDataset.add(bitmap);
}
```

FIGURA 47: EXTRACTO DE CÓDIGO DEL PROYECTO DE LA CLASE *ADAPTER_ITEM*

Con todo esto, aunque al final se consiguió efectuar la tarea de asociar una foto a un ítem y que ésta se muestre en pantalla, el resultado ha provocado varios inconvenientes, destacando la poca eficiencia que existe en el manejo de imágenes.

3.4 Testeo

Como se mencionó anteriormente en el apartado de “*Tecnologías utilizadas*” (Sección 2.1), a lo largo de este proyecto se ha utilizado un dispositivo físico para compilar y ejecutar el código, probando las nuevas funcionalidades desarrolladas en cada iteración. Al mismo tiempo, gracias a que Android Studio puede generar un archivo *.APK* se ha podido también hacer pruebas en otros dispositivos, pudiendo comprobar la funcionalidad de la aplicación en diferentes dimensiones, preferencias y configuraciones.

Sin embargo, aunque estas pruebas tengan un valor importante, no es suficiente para validar que la aplicación no esté exenta de fallos, pues puede darse el caso de que no se hayan contemplado ciertas situaciones y excepciones que en una experiencia normal no se llegan a generar. Para ello es imprescindible realizar una fase de pruebas donde se somete el código a diferentes test con el objetivo de detectar posibles fallos en su ejecución. Estas pruebas pueden clasificarse en tres tipos en función de su objetivo y nivel de actuación.

1. En primer lugar, están los *test unitarios* o pruebas unitarias, las cuales están diseñadas para comprobar una funcionalidad específica de una clase y asegurar que cada unidad de código opere correcta y eficientemente por separado. En este nivel se verifica que los métodos “*hacen lo que tienen que hacer*” y que no tengan fallos de estructura.
2. En segundo lugar, encontramos las pruebas de integración cuyo objetivo es asegurar que el sistema en conjunto funciona como se espera. Para ello se hacen pruebas sobre las relaciones existentes entre las diferentes clases y servicios. Con este tipo de pruebas se busca encontrar los problemas que surgen al mezclar diferentes capas de la aplicación.
3. Por último, existen las pruebas funcionales. A diferencia de las de integración, las pruebas funcionales abarcan la aplicación por completo y lo que se busca es verificar el comportamiento del sistema al completo y todas sus funcionalidades.

A medida que se ha ido desarrollando el código se iban realizando pruebas funcionales con el dispositivo móvil en cada iteración. De esta manera se iba verificando que cada funcionalidad cumplía su cometido y se integraba correctamente con el sistema. Sin embargo, y como se ha dicho anteriormente, estas pruebas no abarcan todos los casos posibles por lo que se hace necesario utilizar pruebas unitarias y de integración para comprobar que todo está correctamente. Además, gracias al patrón MVP y las técnicas de *clean code*, la construcción del código ha sido lo más modular posible permitiendo que diseñar estos test sea bastante sencillo.

Dado que la lógica del proyecto está hecha en Java se utilizó JUnit, en concreto la librería JUnit4 [51], para la fase de pruebas, además de que el IDE de Android Studio posee facilidades para su implementación. JUnit es un framework orientado a la realización de test de aplicaciones Java que permite ejecutar, de manera controlada, métodos y clases con el objetivo de evaluar su comportamiento.

En la siguiente captura de pantalla (Figura 48), extraída del fichero *build.gradle*, se muestra cómo se han agregados las diferentes dependencias para poder implementar JUnit4 en la aplicación.

```
dependencies {

    // Required for local unit tests (JUnit 4 framework)
    testImplementation 'junit:junit:4.13.2'

    // Required for instrumented tests
    androidTestImplementation 'com.android.support:support-annotations:28.0.0'
    androidTestImplementation 'com.android.support.test:runner:1.0.2'

    androidTestImplementation 'androidx.test.ext:junit:1.1.2'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.3.0'
```

FIGURA 48: CAPTURA DE PANTALLA DEL FICHERO *BUILD.GRADLE*

Sin embargo, hay que tener en cuenta que muchas funcionalidades están ligadas, o bien a la base de datos, o bien a la interfaz de usuario por lo que, para comprobarlas correctamente, es imprescindible ejecutarlas en un entorno Android.

Para poder llevar a cabo esta tarea, Android Studio permite desarrollar las denominadas pruebas instrumentadas [52], las cuales se realizan en un dispositivo Android, ya sea físico o emulándolo, y que permite acceder a los métodos de la clase *Instrumentation*. Esta clase es capaz de proveer la información necesaria como, por ejemplo, el *contexto* o *context*, para testar métodos que actúan en la interfaz de la aplicación sin necesidad de la interacción de un usuario y de manera automática. El *contexto* es imprescindible para el funcionamiento de la interfaz de usuario pues es quien proporciona el estado en el que se encuentra la aplicación (en ejecución, en pausa, parado...), la actividad o fragmento donde se tiene el foco y el acceso al resto de recursos.

De esta manera, en el archivo *build.gradle* se configura agregando las dependencias correspondientes para la realización de las pruebas y, por otro lado, se construye el fichero *androidTest* para las pruebas instrumentadas. En el caso de que no se requiera de un entorno Android, es posible crear un fichero *test* que recoja diferentes pruebas que se ejecutarán con la máquina virtual de Java, siendo este método más eficiente.

Una vez se realicen las pruebas, los resultados se pueden ver a través de la consola interna del IDE de Android Studio que notificará si ha surgido un error o si se han pasado todos los test. También es posible indicar el ámbito de las pruebas para que se ejecuten todas las diseñadas o pruebas específicas, así como la ubicación del fichero donde se almacenarán.

4 Trabajo futuro

De cara al futuro es importante tener en cuenta que este proyecto solo ha abarcado desde la fase de planificación y análisis hasta la fase de implementación y pruebas, por lo que aun faltaría una última fase de publicación y despliegue además de definir una serie de características fundamentales, como son su mantenimiento, su rentabilidad y las futuras actualizaciones.

En el caso del mantenimiento sería imprescindible primero hacer una fase de despliegue de prueba, donde se pone a prueba la aplicación a un número amplio de usuarios con el objetivo de encontrar fallos que no hayan aparecido durante todo el desarrollo. Una vez la aplicación quede a disposición del público, habría que habilitar una opción para reportar errores e ir recogiendo las futuras incidencias para poder ir corrigiéndolas en cada nueva versión.

Con respecto a la rentabilidad, en un producto móvil se puede obtener ingresos, principalmente, de cuatro métodos diferentes:

1. **Aplicación de pago:** es necesario efectuar un pago para poder instalar y utilizar la aplicación al completo. Este pago puede ser una compra directa o una suscripción por un tiempo determinado que se debe ir renovando.
2. **Anuncios:** a lo largo de la experiencia de usuario se van agregando diferentes anuncios los cuales pueden ubicarse como parte de la interfaz, por ejemplo, en una banda horizontal encima de la barra de herramientas o debajo del menú de opciones. También existe la opción de que, tras determinadas acciones, se interrumpa la ejecución de la aplicación mostrando el anuncio.
3. **Funcionalidades de pago:** se trata limitar la experiencia de usuario de manera que pueda utilizar la aplicación de forma gratuita, pero deba pagar para tener acceso a funcionales o características específicas que la mejorarían. Un ejemplo aplicando a este proyecto de esta técnica sería limitar el número de maletas que el usuario puede diseñar y que, pagando, pueda ampliar dicha cantidad.
4. **Comisión por acciones:** se agrega la posibilidad de realizar transacciones que impliquen un costo económico como, por ejemplo, para este producto, comprar un billete o reservar una estancia de un hotel. Por realizar dicha acción a través de la aplicación se puede cobrar un porcentaje. A más usuario realicen estas transacciones mayores ingresos se obtienen y, además, permite diseñar estrategias competitivas para fomentar que los clientes inviertan en la aplicación como, por ejemplo, descuentos, ofertas y promociones.

Sobre las futuras actualizaciones, hay que separar por un lado aquellas que amplían y/o perfilan funcionalidades ya existentes y por otro la creación de nuevas funcionalidades. Esta distinción es importante porque no es lo mismo agregar una nueva característica al sistema que cambiarlo y, en ambos casos, requiere de una planificación, un análisis y, una vez implementado, probar que la aplicación sigue siendo funcional.

Existen una gran variedad de métodos que se pueden añadir, muchos de los cuales han surgido a lo largo del desarrollo del proyecto. De hecho, hay ciertas características del proyecto que se han diseñado de cierta manera de cara a esas futuras actualizaciones como, por ejemplo, el cambio de idioma y la resolución de pantalla:

- **Idioma:** Actualmente la aplicación está completamente en español; sin embargo, todas las frases y palabras están guardadas en el archivo *strings.xml*, por lo que bastaría con duplicar ese archivo *strings.xml* pero configurado para que se lea si el dispositivo tiene puesto otro idioma. Esta utilidad es una de las características de aplicaciones Android que facilitan mucho poder agregar más idiomas en una misma aplicación y que se cambien automáticamente en función de las preferencias del usuario o a través de un botón de *ajustes* dentro de la propia aplicación.
- **Resolución de pantalla:** a pesar de que se ha construido la interfaz utilizando píxeles dependientes de la densidad y con elementos que se reescalan automáticamente, puede darse el caso de que en ciertas resoluciones no sea adapte correctamente. También puede suceder esto si cambiamos de dispositivo, como puede ser una *tablet*, un televisor o un ordenador, pues la aplicación está diseñada para una interfaz de un teléfono móvil. En adición a esto, hay que tener en cuenta la disposición horizontal o *landscape*, puesto que, si se gira un teléfono, la interfaz debería también cambiar y adaptarse a esa nueva perspectiva.

Aparte de estas características pendiente por implementar, también existen funcionalidades que, aunque no estén contempladas, podrían agregarse en un futuro como podría ser la implementación de notificaciones, la posibilidad de compartir maletas en tiempo real, calcular el peso o permitir más filtros. También es posible obtener ideas nuevas si se plantean una diversificación del producto como, por ejemplo, que desde la misma aplicación se permita reservar viajes y alojamiento. Todas estas ideas son apuntadas y, en caso de seguir con el desarrollo, entrarían dentro del ciclo de trabajo.

Para publicar el proyecto y que cualquier usuario pueda descargarlo en su móvil, al ser un producto Android, se debe realizar a través de la plataforma de Google denominada *Android Market* o *Play Store de Google*, aunque su nombre oficial es *Google Play* [53]. Esta plataforma de aplicaciones móviles permite a los usuarios navegar y descargar aplicaciones Android, las cuales están disponibles, generalmente, de forma gratuita, aunque existen algunas que son de pago. En cualquier caso, Google no cobra ni por la publicación de un producto ni por su descarga, de hecho, existe una aplicación de móvil denominada *Google Play Store* que facilita este procedimiento de forma cómoda y automática, aunque no es necesaria para poder instalar aplicaciones en un dispositivo móvil.

Por otro lado, hay que tener en cuenta que toda la aplicación, así como sus funcionalidades actuales y futuras están diseñadas y desarrolladas para dispositivos Android. Sin embargo, existen más dispositivos que no usan este sistema operativo como es el caso de los productos de la compañía multinacional *Apple Inc.* que utilizan su propio sistema operativo para dispositivos móviles llamado *IOS* [54]. Por tanto, para que la aplicación pueda instalarse en dispositivos *IOS* es necesario desarrollar un proyecto en otro IDE específico, puesto que *Android Studio* no es válido para ello.

5 Conclusiones

El desarrollo de este trabajo ha sido una experiencia completa y compleja, que ha abarcado diversas fases y donde se ha podido poner en prácticas varios de los conocimientos adquiridos a lo largo de la carrera. También ha supuesto un gran reto tanto a nivel personal, como académico y una gran oportunidad para aprender a desarrollar en un entorno como es el desarrollo móvil, muy diferente de las aplicaciones de escritorio y aplicaciones webs estudiadas durante el grado. Ha sido una gran oportunidad para participar en un proyecto de gran envergadura y cuyo objetivo ha sido el obtener un producto real y útil, el cual podría publicarse y por tanto competir contra otras aplicaciones ya existentes en el mercado.

Tras cumplir con las especificaciones detalladas en el apartado “Descripción del proyecto” del documento TFT01 y, a pesar de las dificultades e inconvenientes que han surgido a lo largo de su desarrollo, se ha obtenido un producto funcional. Con la realización de este proyecto se han cubierto las competencias generales propias del Trabajo de Fin de Grado especificadas en el proyecto docente, así como la descripción, objetivos y metodología descritas en el documento TFG01 y las competencias específicas IS01 e IS04 correspondiente a la mención de Ingeniería del Software las cuales se describen a continuación:

“Ejercicio original a realizar individualmente y presentar y defender ante un tribunal universitario, consistente en un proyecto en el ámbito de las tecnologías específicas de la Ingeniería en Informática de naturaleza profesional en el que se sinteticen e integren las competencias adquiridas en las enseñanzas – TFG01” [55].

“Capacidad para desarrollar, mantener y evaluar servicios y sistemas software que satisfagan todos los requisitos del usuario y se comporten de forma fiable y eficiente, sean asequibles de desarrollar y mantener y cumplan normas de calidad, aplicando las teorías, principios, métodos y prácticas de la ingeniería del software – IS01” [55].

“Capacidad de identificar y analizar problemas y diseñar, desarrollar, implementar, verificar y documentar soluciones software sobre la base de un conocimiento adecuado de las teorías, modelos y técnicas actuales. – IS04” [55].

Bibliografía

- [1 Real Academia de la lengua Española, «<https://www.rae.es/>,» [En línea]. Available: <https://dle.rae.es/viajar>. [Último acceso: 11 2020].
- [2 Organización Mundial del Turismo, «<https://www.unwto.org/es/glosario-terminos-turisticos>,» [En línea]. Available: <https://www.unwto.org/es/glosario-terminos-turisticos>. [Último acceso: Noviembre 2020].
- [3 Organización Mundial del Turismo, «<https://www.unwto.org/es/turismo>,» [En línea]. Available: <https://www.unwto.org/es/turismo>. [Último acceso: Noviembre 2020].
- [4 Organización Mundial del Turismo, «Panorama del turismo internacional,» 2019. [En línea]. Available: <https://www.e-unwto.org/doi/pdf/10.18111/9789284421237>. [Último acceso: Noviembre 2020].
- [5 Organización Mundial de la Salud, «coronavirus 2019,» [En línea]. Available: <https://www.who.int/es/emergencies/diseases/novel-coronavirus-2019/advice-for-public/q-a-coronaviruses#:~:text=sintomas>. [Último acceso: Noviembre 2020].
- [6 Organización Mundial del Turismo, «<https://www.unwto.org/international-tourism-and-covid-19>,» 27 Octubre 2020. [En línea]. Available: <https://www.unwto.org/international-tourism-and-covid-19>. [Último acceso: Noviembre 2020].
- [7 Epdata, «llegada de turistas internacionales epdata,» Europa Press, [En línea]. Available: <https://www.epdata.es/llegada-turistas-internacionales-espana/2aa0879a-0255-4122-8378-32ef30ba6259>. [Último acceso: Noviembre 2020].
- [8 Instituto Nacional de Estadística, «Encuesta de Gasto Turístico 2019 INE,» 3 Febrero 2020. [En línea]. Available: <https://www.ine.es/daco/daco42/egatur/egatur1219.pdf>. [Último acceso: Noviembre 2020].
- [9 Wikipedia, «turismo en españa,» [En línea]. Available: https://es.wikipedia.org/wiki/Turismo_en_Espa%C3%B1a. [Último acceso: Noviembre 2020].
- [1 «Wikipedia - minimalismo,» 09 03 2021. [En línea]. Available: <https://es.wikipedia.org/wiki/Minimalismo>. [Último acceso: 2021 24 14].
- [1 Google, «Google play - Aplicaciones sobre maletas,» [En línea]. Available: https://play.google.com/store/search?q=maletas&c=apps&hl=es_419&gl=US. [Último acceso: 04 2021].
- [1 Wikipedia, «Wikipedia - Historias de usuario,» Octubre 2020. [En línea]. Available: https://es.wikipedia.org/wiki/Historias_de_usuario. [Último acceso: Junio 2021].
- [1 scrum manager, «Scru manager - historias de usuario,» [En línea]. Available: https://scrummanager.net/files/scrum_manager_historias_usuario.pdf. [Último acceso: Junio 2021].

- [1 Scrum manager, «Scrum manager - INVEST,» Abril 2021. [En línea]. Available:
4] <https://www.scrummanager.net/bok/index.php?title=INVEST>. [Último acceso: Junio 2021].
- [1 H. México, «HostGator,» 14 Abril 2020. [En línea]. Available:
5] <https://www.hostgator.mx/blog/clean-code-codigo-limpio/>. [Último acceso: Mayo 2021].
- [1 Microsoft , «Microsoft - Microsoft Word,» 2021. [En línea]. Available:
6] <https://www.microsoft.com/es-es/microsoft-365/word>. [Último acceso: Junio 2021].
- [1 Microsoft, «Microsoft - Outlook,» 2021. [En línea]. Available:
7] <https://www.microsoft.com/es-es/microsoft-365/outlook/email-and-calendar-software-microsoft-outlook>. [Último acceso: Junio 2021].
- [1 Balsamiq Studios, «Balsamiq,» 2021. [En línea]. Available: <https://balsamiq.com/>. [Último
8] acceso: Junio 2021].
- [1 MKLabs Co.,Ltd, «Start UML,» 2021. [En línea]. Available: <https://staruml.io/>. [Último
9] acceso: Junio 2021].
- [2 Wikipedia, «Wikipedia - UML,» 2021. [En línea]. Available:
0] https://es.wikipedia.org/wiki/Lenguaje_unificado_de_modelado. [Último acceso: Junio
2021].
- [2 Google, «Android Studio - Descargar,» 2021. [En línea]. Available:
1] https://developer.android.com/studio?hl=es&gclid=Cj0KCQjwraqHBhDsARIsAKuGZeEagnDYzAly5SIqphLG4dPpCseMyx5kkQjHPJndB9I7WL6Kk2tvZpwaAnaeEALw_wcB&gclidsrc=aw.ds.
[Último acceso: Junio 2021].
- [2 Gradle Inc., «Gradle,» 2021. [En línea]. Available: <https://gradle.org/>. [Último acceso: Mayo
2] 2021].
- [2 Google, «Android para desarrolladores - Píxeles independientes de la densidad,» 01 Junio
3] 2020. [En línea]. Available:
<https://developer.android.com/training/multiscreen/screendensities?hl=es-419#TaskUseDP>. [Último acceso: Mayo 2021].
- [2 «Prototypr - resolución de pantallas y densidad de pantalla en android,» Julio 2018. [En
4] línea]. Available: <https://blog.prototypr.io/designing-for-multiple-screen-densities-on-android-5fba8afe7ead>. [Último acceso: Mayo 2021].
- [2 «Altova,» Altova GmbH, 2020. [En línea]. Available:
5] https://www.altova.com/manual/es/MobileTogether/mobiletogetherdesigner/mtdobjsfatures_sizes.html. [Último acceso: Mayo 2021].
- [2 Google, «Android para desarrolladores - Guía,» Enero 2021. [En línea]. Available:
6] <https://developer.android.com/guide>. [Último acceso: Mayo 2021].
- [2 «Source Tree,» Atlasian, 2021. [En línea]. Available: <https://www.sourcetreeapp.com/>.
7] [Último acceso: 2021].

- [2 Google, «Android para desarrolladores - Actividades,» [En línea]. Available:
8] <https://developer.android.com/guide/components/activities/intro-activities>. [Último acceso: Mayo 2021].
- [2 Digital Learning SL, «Academia Android - actividades y fragmentos,» 08 2014. [En línea].
9] Available: <https://academiaandroid.com/activity-y-fragmentos/>. [Último acceso: 05 2021].
- [3 Google, «Android para desarrolladores - Fragmentos,» 12 2019. [En línea]. Available:
0] <https://developer.android.com/guide/components/fragments?hl=es>. [Último acceso: 05 2021].
- [3 Google, «Youtube - Single Activity,» 9 Noviembre 2018. [En línea]. Available:
1] <https://www.youtube.com/watch?v=2k8x8V77CrU>. [Último acceso: Mayo 2021].
- [3 Google, «Android para desarrolladores - Componente Navigation,» 2021. [En línea].
2] Available: <https://developer.android.com/guide/navigation?hl=es-419>. [Último acceso: Mayo 2021].
- [3 Google, «Android para desarrolladores - Pila de Actividades,» [En línea]. Available:
3] <https://developer.android.com/guide/components/activities/tasks-and-back-stack?hl=es>. [Último acceso: Mayo 2021].
- [3 Google, «Android para desarrolladores - documentación, onNavDestinationSelected,»
4] Febrero 2021. [En línea]. Available:
[https://developer.android.com/reference/androidx/navigation/ui/NavigationUI#onNavDestinationSelected\(android.view.MenuItem,%20androidx.navigation.NavController\)](https://developer.android.com/reference/androidx/navigation/ui/NavigationUI#onNavDestinationSelected(android.view.MenuItem,%20androidx.navigation.NavController)). [Último acceso: Mayo 2021].
- [3 Google, «Android para desarrolladores - Permisos,» Mayo 2021. [En línea]. Available:
5] <https://developer.android.com/training/permissions/requesting?hl=es-419>. [Último acceso: Mayo 2021].
- [3 Google, «Android para desarrolladores - CamaraX,» Marzo 2021. [En línea]. Available:
6] <https://developer.android.com/training/camerax#consistency>. [Último acceso: Mayo 2021].
- [3 Google, «Android para desarrolladores - ListView,» 18 Mayo 2021. [En línea]. Available:
7] <https://developer.android.com/reference/android/widget/ListView>. [Último acceso: Junio 2021].
- [3 Google, «Android para desarrolladores - RecyclerView,» 17 Mayo 2021. [En línea].
8] Available: <https://developer.android.com/jetpack/androidx/releases/recyclerview>. [Último acceso: Junio 2021].
- [3 Google, «Android para desarrolladores - listas dinámicas con RecyclerView,» 22 Enero
9] 2021. [En línea]. Available:
<https://developer.android.com/guide/topics/ui/layout/recyclerview>. [Último acceso: Junio 2021].

- [4 Google, «Android para desarrolladores - SearchView,» 08 Julio 2020. [En línea]. Available: 0] <https://developer.android.com/training/search/setup?hl=es>. [Último acceso: Junio 2021].
- [4 Google, «Android para desarrolladores - Shared Preferences,» Junio 2020. [En línea]. 1] Available: <https://developer.android.com/training/data-storage/shared-preferences>. [Último acceso: Mayo 2021].
- [4 Google, «Android para desarrolladores - Room,» 04 2021. [En línea]. Available: 2] <https://developer.android.com/training/data-storage/room?hl=es-419>. [Último acceso: 05 2021].
- [4 Wikipedia, «Wikipedia - Teoría del color,» 2 Junio 2021. [En línea]. Available: 3] https://es.wikipedia.org/wiki/Teor%C3%ADa_del_color. [Último acceso: Junio 2021].
- [4 Google, «Material Design,» 2021. [En línea]. Available: <https://material.io/>. [Último acceso: 4] Junio 2021].
- [4 Google, «Material Design - Herramienta para escoger colores,» 2021. [En línea]. Available: 5] <https://material.io/design/color/the-color-system.html#tools-for-picking-colors>. [Último acceso: Junio 2021].
- [4 Wikipedia, «Wikipedia (inglés) - call to action,» 30 Abril 2021. [En línea]. Available: 6] [https://en.wikipedia.org/wiki/Call_to_action_\(marketing\)](https://en.wikipedia.org/wiki/Call_to_action_(marketing)). [Último acceso: Junio 2021].
- [4 Canvas, «Canvas - white space design,» 2021. [En línea]. Available: 7] <https://www.canva.com/learn/white-space-design/>. [Último acceso: Junio 2021].
- [4 Google, «Android para desarrolladores - Cómo reducir los tamaños de descarga de 8] imágenes,» Mayo 2020. [En línea]. Available: <https://developer.android.com/topic/performance/network-xfer?hl=es>. [Último acceso: Junio 2021].
- [4 Google, «Android para desarrolladores - Cómo administrar mapas de bits,» Enero 2021. [En 9] línea]. Available: <https://developer.android.com/topic/performance/graphics?hl=es>. [Último acceso: Junio 2021].
- [5 Google, «Android para desarrolladores - ImageDecoder,» Febrero 2021. [En línea]. 0] Available: [https://developer.android.com/reference/android/graphics/ImageDecoder#createSource\(android.content.ContentResolver,%20android.net.Uri\)](https://developer.android.com/reference/android/graphics/ImageDecoder#createSource(android.content.ContentResolver,%20android.net.Uri)). [Último acceso: Junio 2021].
- [5 JUnit, «Página oficial de JUnit,» Febrero 2021. [En línea]. Available: 1] <https://junit.org/junit4/>. [Último acceso: Junio 2021].
- [5 Google, «Android para desarrolladores - Pruebas Instrumentadas,» Junio 2020. [En línea]. 2] Available: <https://developer.android.com/training/testing/unit-testing/instrumented-unit-tests?hl=es-419>. [Último acceso: Junio 2021].
- [5 Google, «Google Play,» 2021. [En línea]. Available: <https://play.google.com/store>. [Último 3] acceso: Junio 2021].

- [5 Apple, 2021. [En línea]. Available: <https://www.apple.com/es/>. [Último acceso: Junio 4] 2021].
- [5 ULPGC, «Objetivos y Competencias del GII de la ULPGC,» [En línea]. Available: 5] https://www2.ulpgc.es/archivos/plan_estudios/4008_40/ObjetivosyCompetenciasdelGII.pdf. [Último acceso: Junio 2021].
- [5 Ministerio de Transportes, Movilidad y Agenda Urbana, «<https://www.mitma.gob.es>,» [En 6] línea]. Available: <https://www.mitma.gob.es/aviacion-civil/centenario-transporte-aereo-espania/historia/historia-de-los-100-anos-del-transporte-aereo>. [Último acceso: 11 2020].
- [5 Instituto Nacional de Estadística, «<https://www.ine.es>,» [En línea]. Available: 7] https://www.ine.es/infografias/infografia_turismo_viajeros.pdf. [Último acceso: 11 2020].
- [5 Google, «API de Android,» [En línea]. Available: 8] <https://developer.android.com/reference/packages>. [Último acceso: Mayo 2021].
- [5 R. C. Martin, Clean Code: A Handbook of Agile Software Craftsmanship, Upper Saddle River, 9] Nueva Jersey, Estados Unidos: Prentice Hall, 2009.
- [6 Google, «Guía para desarrolladores Android,» 08 Enero 2021. [En línea]. Available: 0] <https://developer.android.com/guide>. [Último acceso: Mayo 2021].
- [6 Google, «Android para desarrolladores - Navigation y la pila de actividades,» [En línea]. 1] Available: <https://developer.android.com/guide/navigation/navigation-navigate?hl=es#back-stack>. [Último acceso: Mayo 2021].
- [6 A. Butt, "bestcodeway," blogspot, Septiembre 2014. [Online]. Available: 2] <http://bestcodeway.blogspot.com/2014/09/mvc-vs-mvvm-vs-mvp.html>. [Accessed Mayo 2021].
- [6 Google, «Google play store - PackTeo,» 2021. [En línea]. Available: 3] <https://play.google.com/store/apps/details?id=com.createo.packteo>. [Último acceso: Mayo 2021].
- [6 TNX Apps, «Google Play Store - TNX Apps,» Google, 2021. [En línea]. Available: 4] <https://play.google.com/store/apps/details?id=com.tnx.packed>. [Último acceso: Mayo 2021].
- [6 appscapes, «Google play store - appscapes,» Google, 2021. [En línea]. Available: 5] <https://play.google.com/store/apps/details?id=com.appscapes.packinglist>. [Último acceso: Mayo 2021].
- [6 Google, «Android para desarrolladores - Toast,» Abril 2021. [En línea]. Available: 6] <https://developer.android.com/guide/topics/ui/notifiers/toasts?hl=es-419>. [Último acceso: Junio 2021].

ANEXO I: MANUAL DE USUARIO

El siguiente anexo recoge los pasos a seguir para instalar y ejecutar la aplicación “JustInCase”.

1. Instalación de la aplicación:

1. Descargar el ejecutable .apk. Lo puede encontrar en:
 - a. <https://github.com/SrMiki/TFG/blob/main/app-debug.apk>
 - b. <https://drive.google.com/drive/folders/1KBUnTypfxFjzRT4toz1VS7zyeB27TKKg?usp=sharing>
2. Ejecutarlo en el teléfono móvil. **Imprescindible que sea un dispositivo Android y que tenga una versión igual o posterior a la versión 5 de Android.**
 - a. En caso de que salga algún cuadro de diálogo confirmar
3. Iniciamos la aplicación.

Si todo se ha hecho correctamente debería mostrarse la página principal de la aplicación ya sería posible navegar por ella y acceder a sus diferentes funcionalidades.



FIGURA 49: CAPTURA DE PANTALLA DE LA VENTANA PRINCIPAL DE LA APLICACIÓN

A continuación, se muestran cómo llevar cabo las funcionales principales de la aplicación mediante capturas de pantallas obtenidas de la aplicación en ejecución.

2. Crear un viaje:

1. Abrimos el menú pulsando en el botón situado en la esquina superior izquierda.
2. Seleccionamos la opción “Mis viajes”

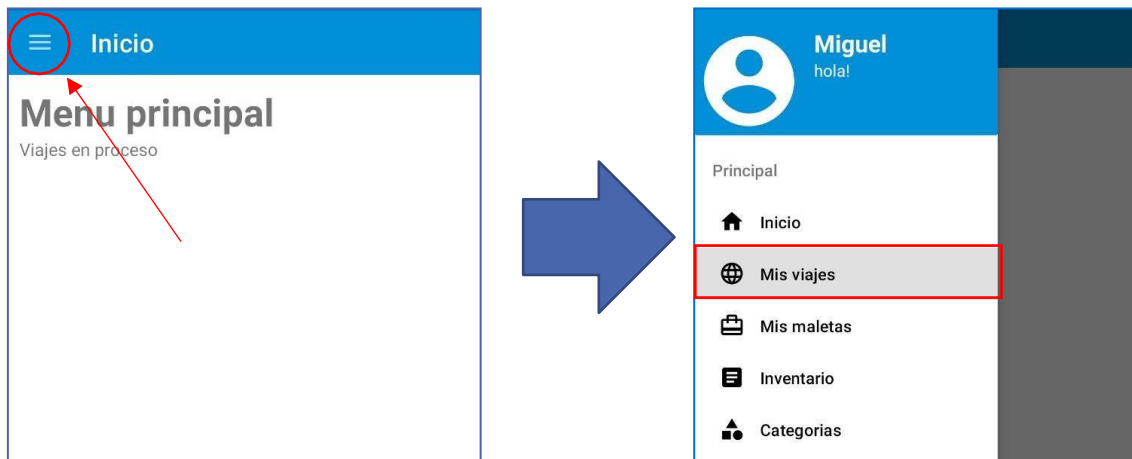


FIGURA 50: PRESENTACIÓN DEL PASO 1 Y 2 PARA CREAR UN NUEVO VIAJE

3. Presionamos el botón ubicado en la parte inferior derecha que pone “Nuevo Viaje”
4. Rellenamos los campos del formulario. Obligatoria “Destino” y “Fecha de salida”. Podemos indicar si el viaje es solo de ida presionando el switch o dejando en blanco.

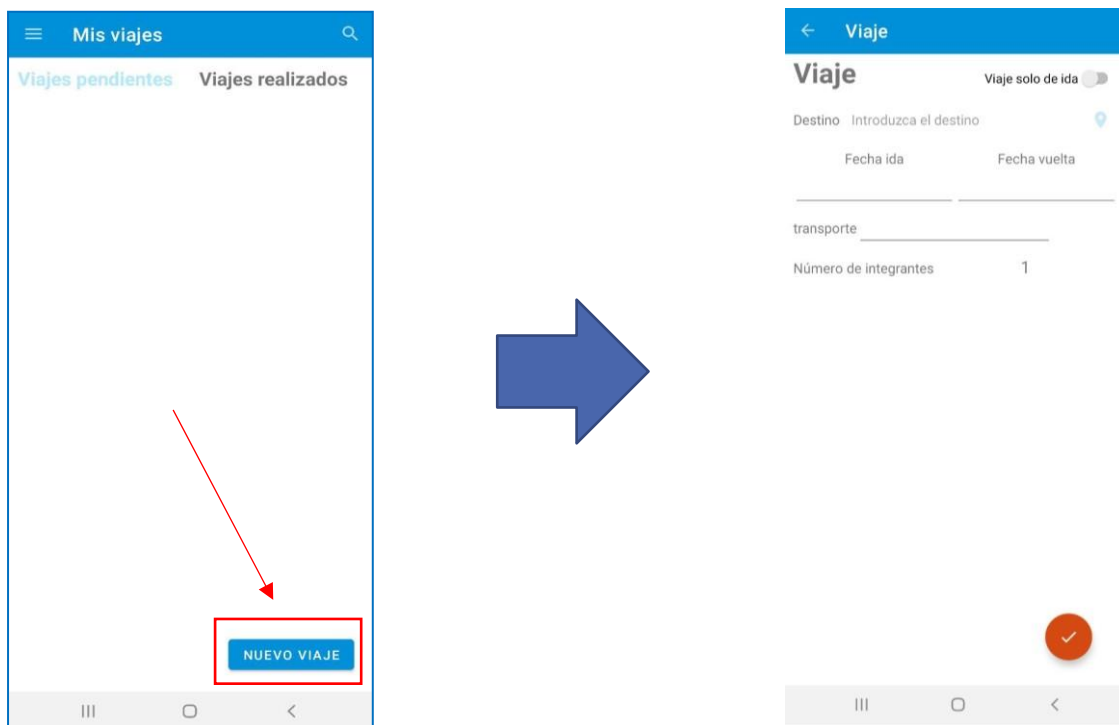


FIGURA 51: PRESENTACIÓN DEL PASO 3 Y 4 PARA CREAR UN NUEVO VIAJE

5. Una vez hayamos completado el formulario, pulsamos el botón para finalizar y se abrirá un cuadro de diálogo preguntando si queremos, o no, asignar las maletas al viaje:
 - a. Si presionamos que **no**, la operación se dará por **terminada** y el viaje habrá sido creado, por lo que nos redirigirá a la pestaña donde se muestran los viajes planificados.

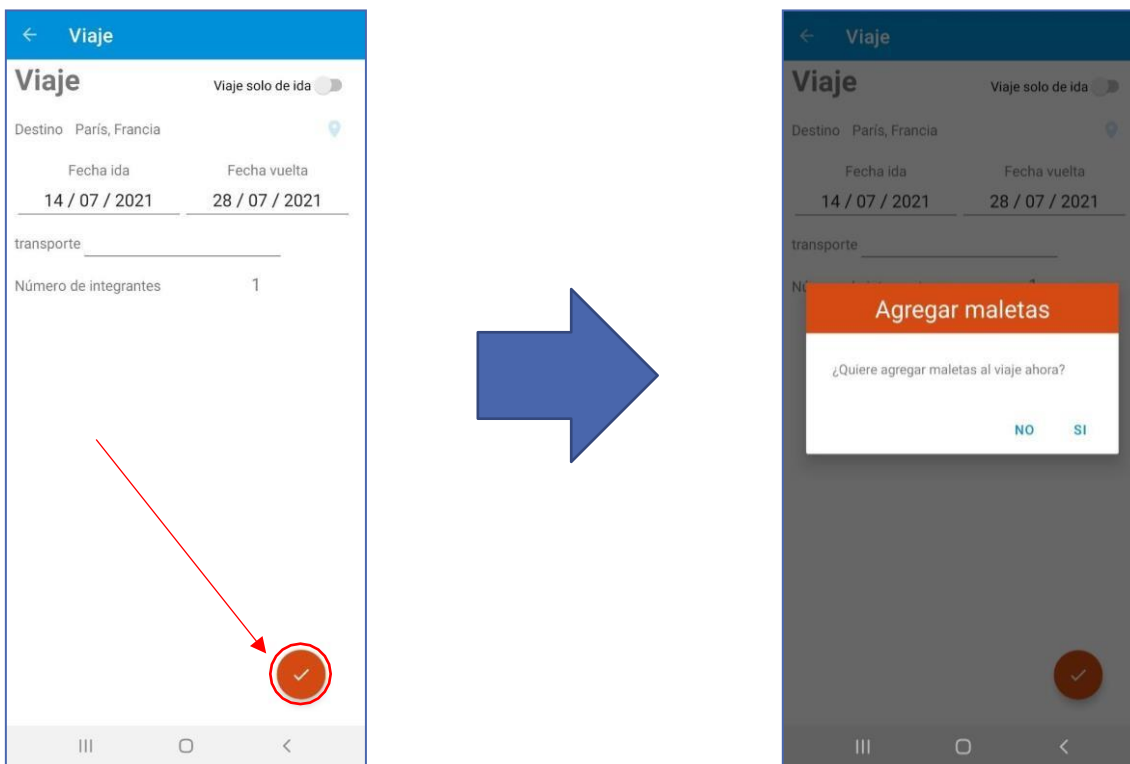


FIGURA 52: PRESENTACIÓN DEL PASO 5 PARA CREAR UN NUEVO VIAJE

- b. Si presionamos que **sí**, nos redirigirá a la ventana donde se muestra la lista de maletas que hemos creado en la aplicación. En caso de no tener ninguna disponible, automáticamente nos aparecerá un cuadro de diálogo que nos permitirá crear una nueva maleta.

6. Seleccionamos las distintas maletas que queremos llevar en nuestro viaje y presionamos en el botón de aceptar ubicado en la parte inferior derecha.

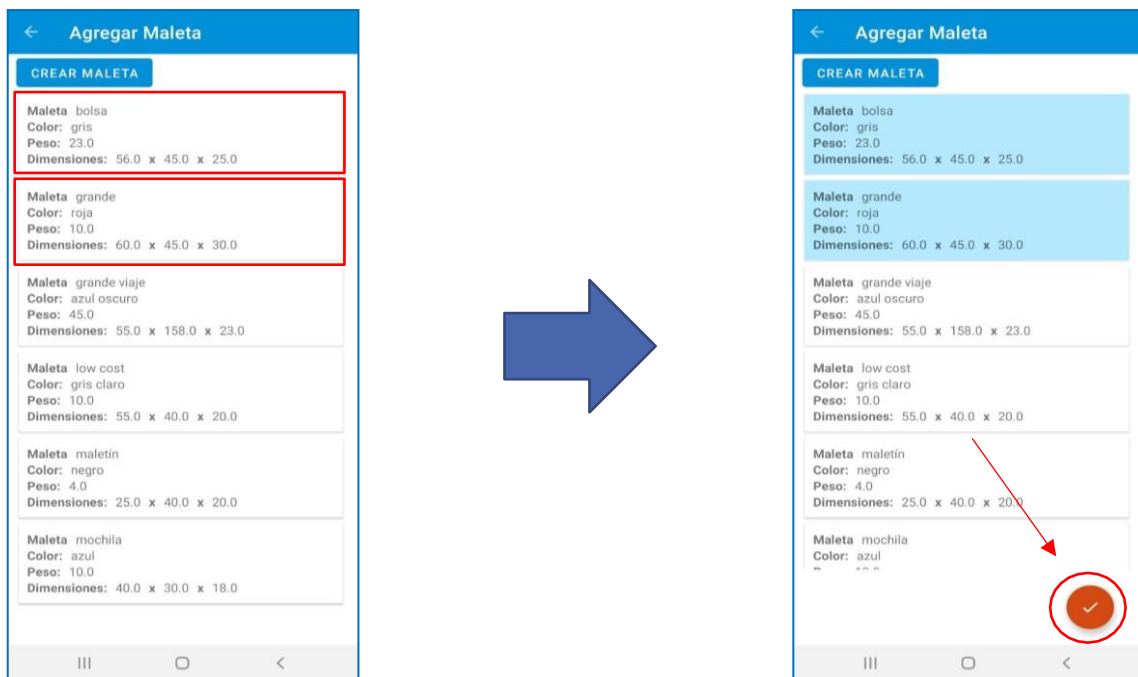


FIGURA 53: PRESENTACIÓN DEL PASO 5.B Y 6 PARA CREAR UN NUEVO VIAJE

Podemos acceder al viaje que acabamos de crear en cualquier momento para ver más detalles, editarlo o eliminándolo.

Para iniciar un viaje, primero se selecciona el viaje que queremos iniciar y después pulsamos el botón “Hacer check-in” ubicado en la parte inferior izquierda de la pantalla, donde nos llevará a la pantalla de “Hacer la maleta”. En ella se visualizarán los diferentes ítems que contiene la maleta a los cuales podemos ir haciendo *check*.

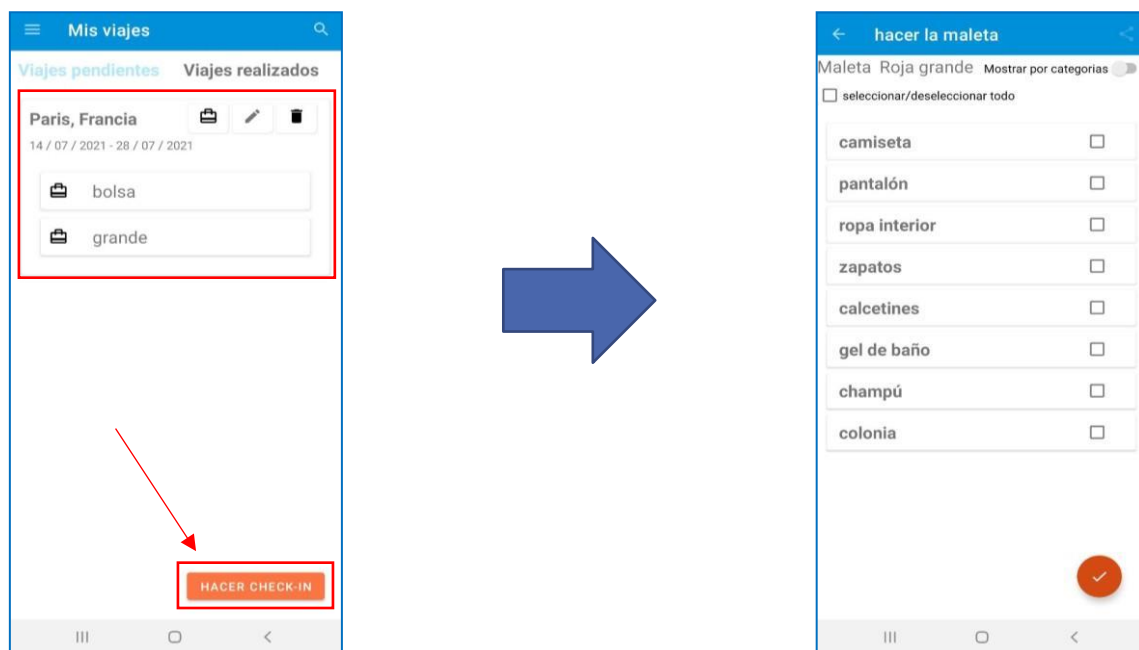


FIGURA 54: REPRESENTACIÓN DE “HACER CHECK-IN” DE UNA MALETA

3. Crear elementos básicos

A continuación, se muestra el procedimiento a seguir para gestionar los diferentes elementos de la aplicación. Los siguientes pasos sirven para crear ítems, categorías, maletas y plantillas.

1. Abrimos el menú pulsando en el botón situado en la esquina superior izquierda.
2. Seleccionamos la opción con la que queramos trabajar, para este ejemplo, *Inventario*

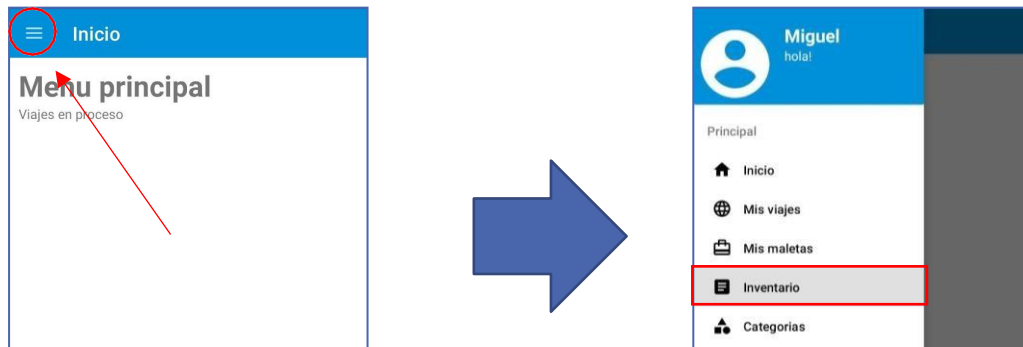


FIGURA 55: PRESENTACIÓN DEL PASO 1 Y 2 PARA CREAR UN NUEVO ÍTEM

3. Presionamos el botón ubicado en la parte inferior derecha
4. Aparecerá un cuadro de diálogo donde habrá que rellenarlo con datos. En este caso con el nombre del ítem.

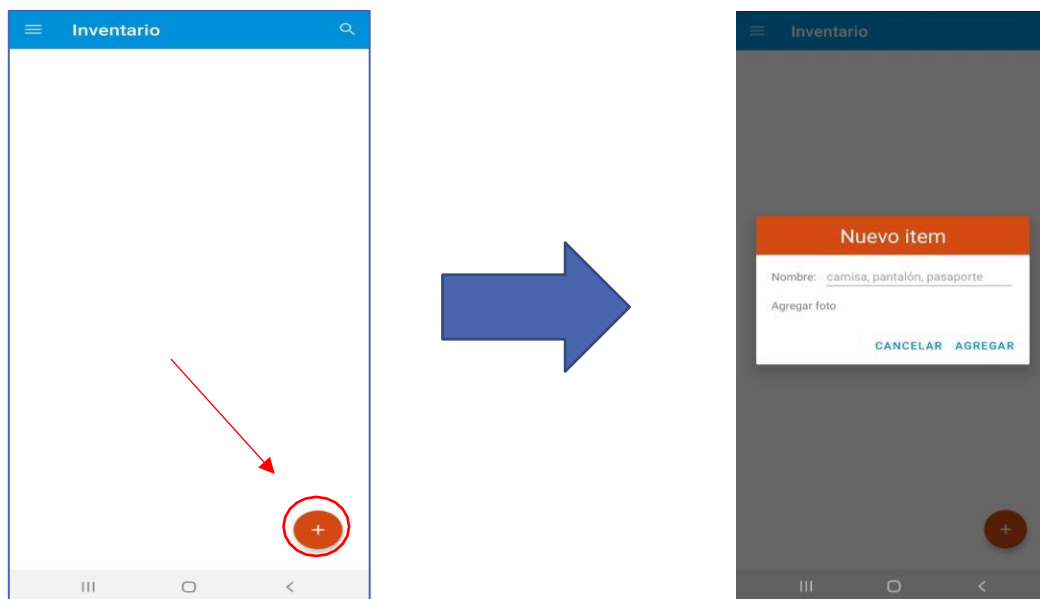


FIGURA 56: PRESENTACIÓN DEL PASO 3 Y 4 PARA CREAR UN NUEVO ÍTEM

Para el resto de los casos se presentan las siguientes imágenes: la primera es para categorías, la segunda para plantillas y la tercera para maletas.



FIGURA 57: CUADROS DE DIÁLOGO PARA AGREGAR UNA NUEVA CATEGORÍA, PLANTILLA Y MALETA

5. Una vez lo hayamos completado, pulsamos el botón para confirmarla acción, en este caso, “Agregar”.

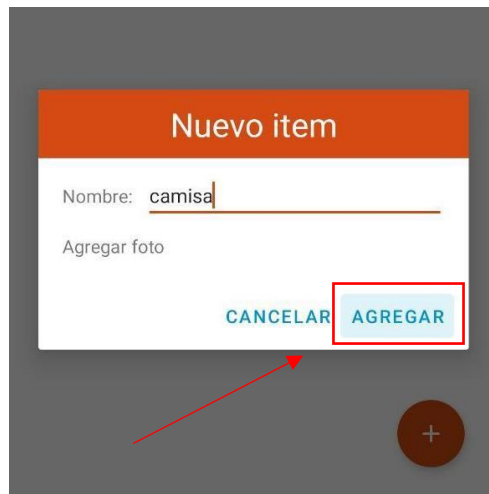


FIGURA 58: PRESENTACIÓN DEL PASO 5 CREAR UN NUEVO ÍTEM

Si la operación se ha realizado correctamente, nos devolverá a al listado donde aparecerá el nuevo elemento que hayamos añadido como se puede observar en la primera imagen. Si realizamos varias veces la misma acción, introduciendo nuevos elementos, éstos se irán colocando ordenadamente uno debajo de otro, como se puede apreciar en la segunda imagen.

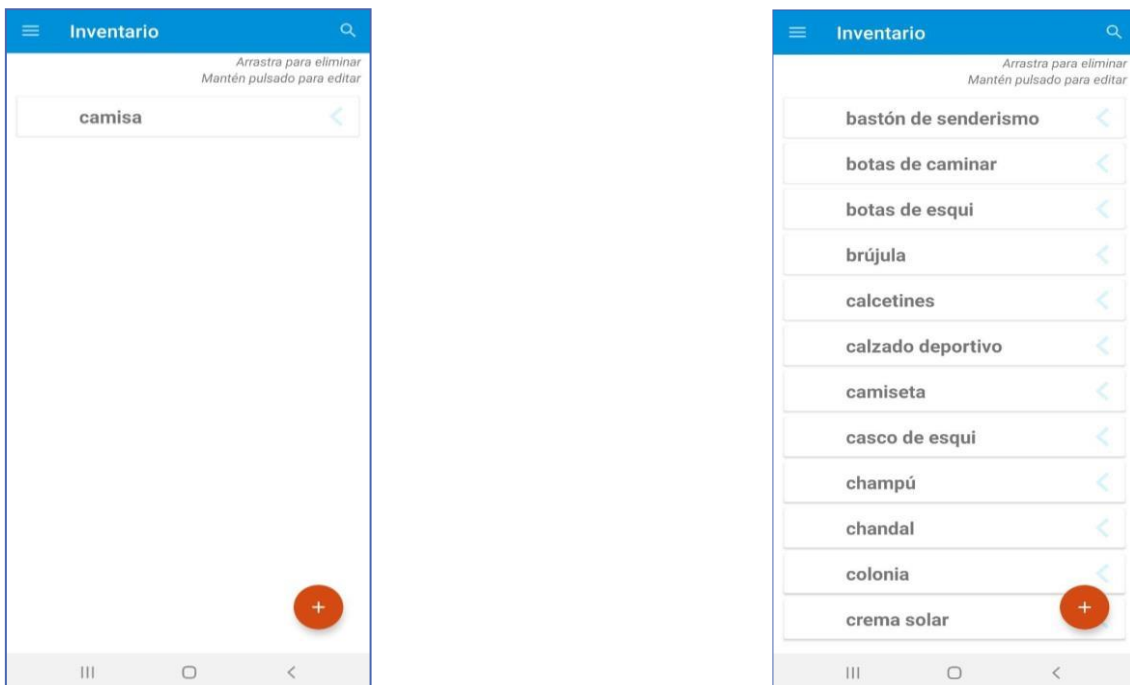


FIGURA 59: CAPTURA DE PANTALLA DE LA LISTA DE ÍTEMS, A LA IZQUIERDA, CON SOLO UN ÍTEM CREADO Y, A LA DERECHA, CON VARIOS ÍTEMS CREADOS.

3.1 Editar y Eliminar elementos básicos

Si mantenemos pulsado cualquier elemento de la lista, saldrá un cuadro de diálogo para editarla. En el caso de *Categorías* y *Plantillas* se debe presionar por más de un segundo para que aparezca.

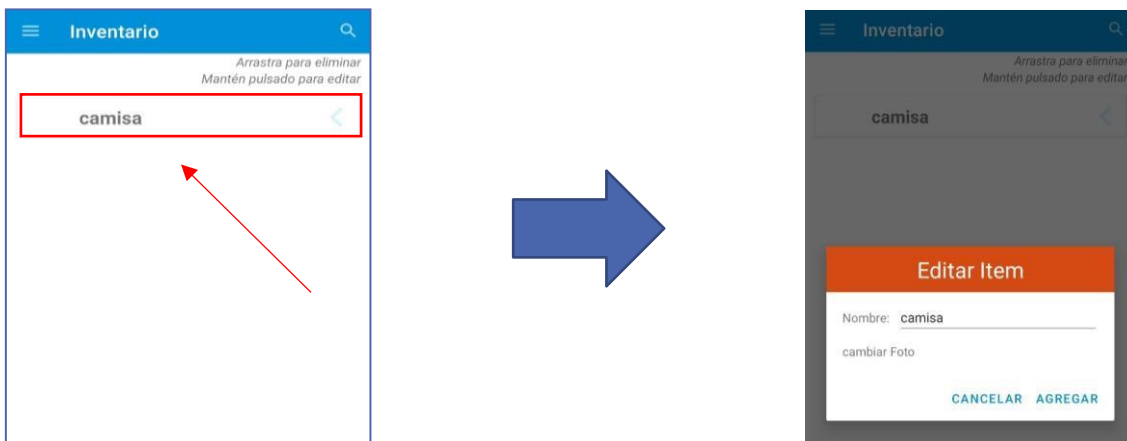


FIGURA 60: REPRESENTACIÓN DEL PROCEDIMIENTO PARA EDITAR UN ÍTEM

Para eliminar cualquier elemento de una lista solo hay que arrastrarlo de izquierda a derecha, fuera de la pantalla.

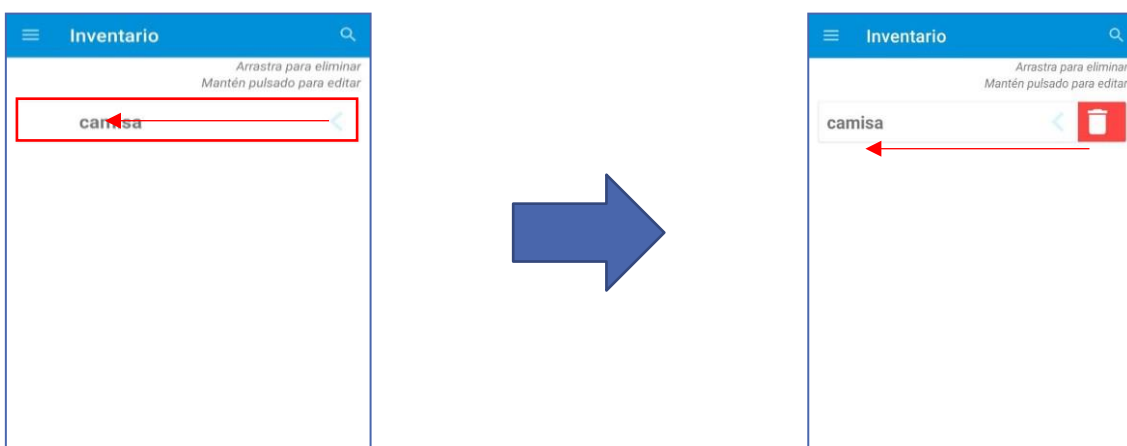


FIGURA 61: REPRESENTACIÓN DEL PROCEDIMIENTO PARA ELIMINAR UN ÍTEM

4. Planificar el equipaje

Para agregar ítems a la maleta de un viaje, se sigue el siguiente procedimiento:

1. debemos seleccionar la maleta que queremos editar.

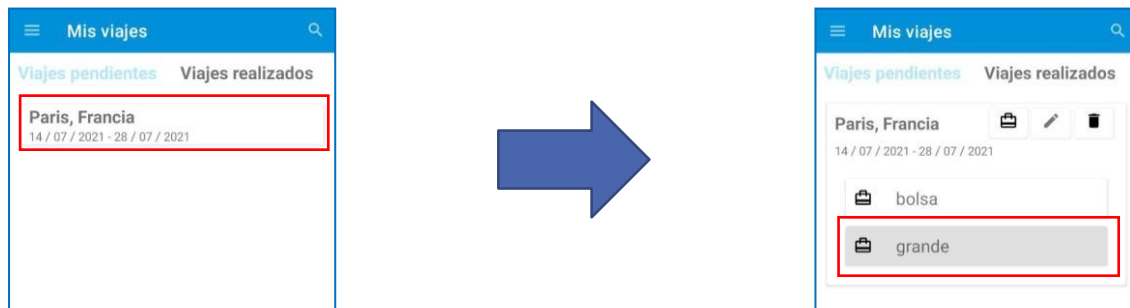


FIGURA 62: REPRESENTACIÓN DEL PASO 1 PARA AGREGAR UN EQUIPAJE A UNA MALETA

2. Una vez dentro, podremos agregar los ítems de tres formas distintas.



FIGURA 63: PANTALLA DE “EQUIPAJE” DE UNA MALETA DE UN VIAJE

- a. usando una plantilla. Para ello seleccionamos la opción de la barra de herramientas “Plantilla”

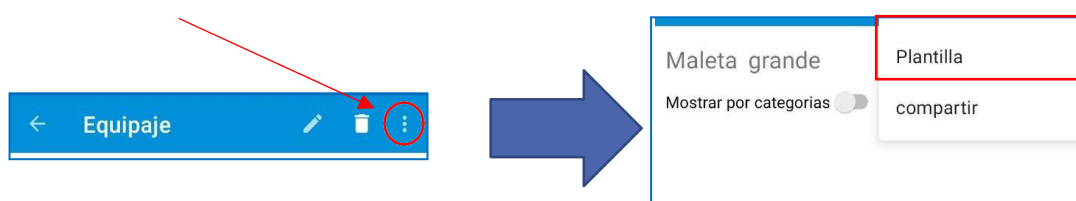


FIGURA 64: REPRESENTACIÓN DEL PASO 2.A PARA AGREGAR UN EQUIPAJE A UNA MALETA

- b. Usando la herramienta de generación automática. Para ello, pulsamos el botón “Hazme la maleta” ubicado en la parte superior derecha de la pantalla y seleccionamos las opciones que se adecuen a nuestro viaje.

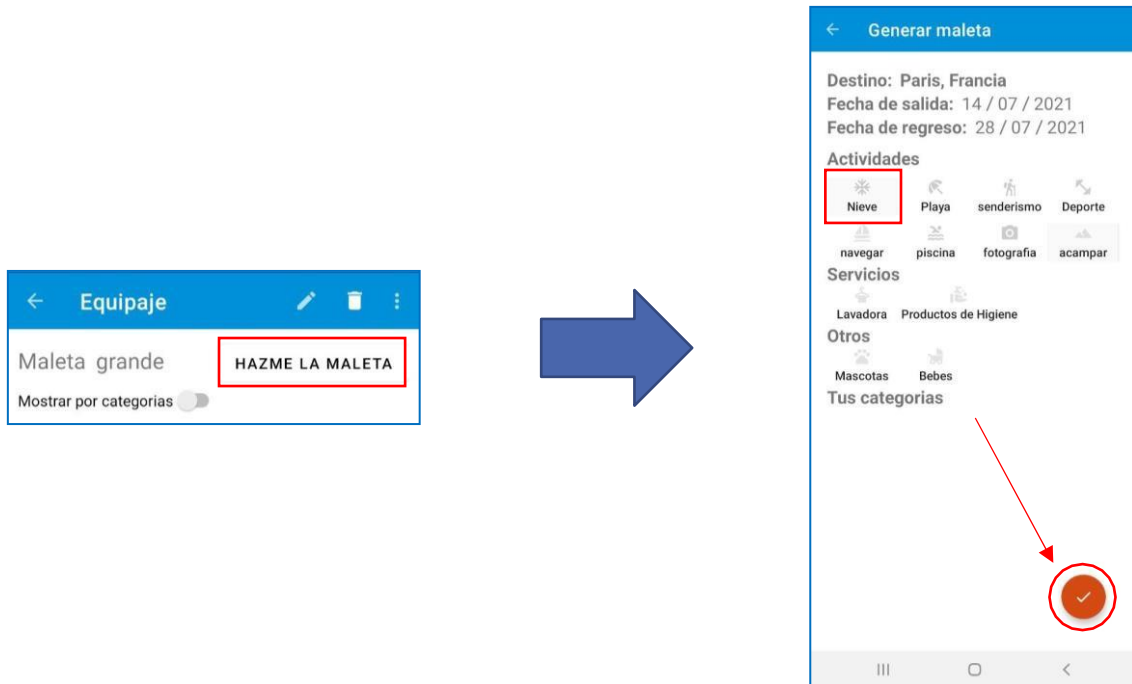


FIGURA 65: REPRESENTACIÓN DEL PASO 2.A PARA AGREGAR UN EQUIPAJE A UNA MALETA

- c. Manualmente. Primero pulsamos el botón de acción ubicado en la parte inferior derecha de la pantalla, seleccionamos los ítems para añadir y pulsamos el botón de confirmación situado también en la parte inferior derecha.

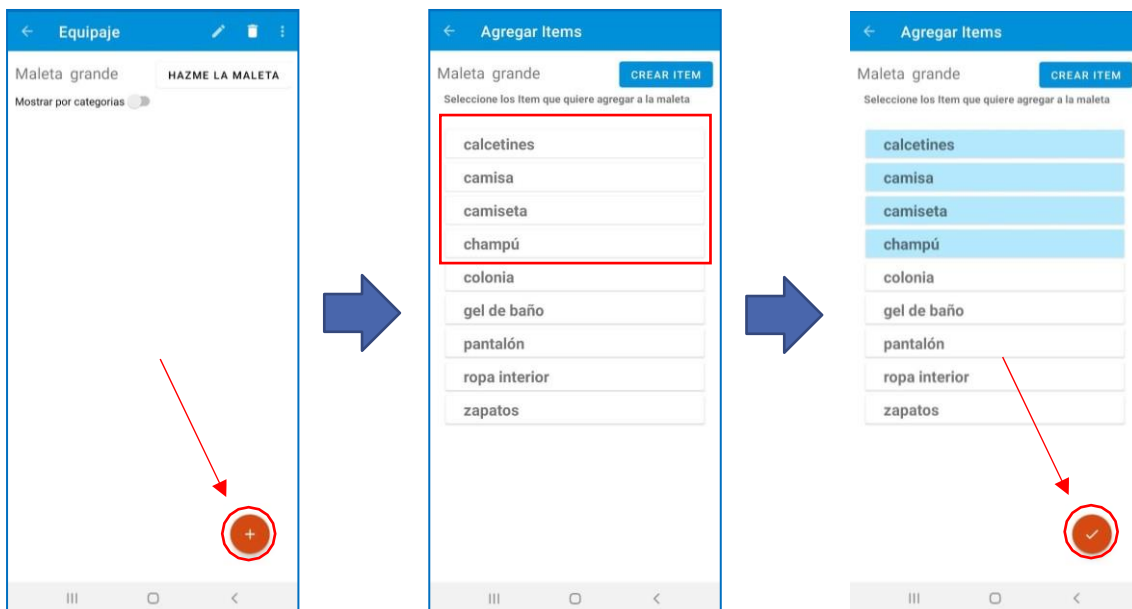


FIGURA 66: REPRESENTACIÓN DEL PASO 2.A PARA AGREGAR UN EQUIPAJE A UNA MALETA