Escuela Politécnica Superior de Jaén

Grado en Ingeniería Informática

UNIVERSIDAD DE JAÉN

Nombre del Centro

Trabajo Fin de Grado

PROTOTIPO DE APLICACIÓN MÓVIL PARA LA TRAZABILIDAD DE RESTRICCIONES CONTRA LA COVID-19 EN VIAJES INTERNACIONALES

Alumno: Jesús Manuel Palomo Vega

Tutor: Prof. D. Juan Manuel Jurado Rodríguez
Dpto: Departamento de Informática

Tutor: Prof. Dña. Lidia Mª Ortega Alvarado
Dpto: Departamento de Informática

Septiembre, 2021

Universidad de Jaén
Escuela Politécnica Superior de Jaén
Departamento de Informática

Don NOMBRE_TUTOR , tutor del Proyecto Fin de Carrera titulado: NOMBRE_PROYECTO, que presenta NOMBRE_ALUMNO, autoriza su presentación para defensa y evaluación en la Escuela Politécnica Superior de Jaén.

Jaén, Septiembre de 2021

El alumno: Los tutores:

PALOMO VEGA JESUS MANUEL - 77379337T

Firmado digitalmente por PALOMO VEGA JESUS MANUEL - 77379337T
Fecha: 2021.09.10 14:44:21 +02'00'

Jesús Manuel Palomo Vega

D. Juan Manuel Jurado Rodríguez
Dña. Lidia Mª Ortega Alvarado

Resumen del trabajo

En este documento se halla documentado el proceso para la realización del presente Trabajo de Fin de Grado. Dicho trabajo fue asignado durante el segundo cuatrimestre del curso 2020/2021. Este trabajo consiste en un proyecto individual de ingeniería para el Grado en Ingeniería Informática en idioma Castellano.

La idea para el desarrollo del mismo surge el 16 de Abril de 2021 durante la realización de un concurso Hackathon organizada por la universidad de Pécs (Hungría). Dicho concurso estaba enfocado en el prototipado de aplicaciones móviles relacionadas con el ámbito de la salud. El prototipo de la aplicación desarrollada durante este trabajo obtuvo el primer premio en dicho concurso, lo que supuso una motivación para presentar dicho prototipo como idea para el Trabajo de Fin de Grado.

La aplicación estará enfocada a los viajes internacionales, ya sea por motivos turísticos o por cualquier otro. Esta permitirá planear un viaje en avión seleccionando un país de origen y un país de destino. Tras esto, la aplicación ofrecerá las mejores opciones de vuelo para realizar dicho viaje. Estas contarán con información sobre los aeropuertos de origen y destino, distancia entre ambos y precio estimado del billete. Una vez seleccionado un vuelo por el usuario, se mostrará una información acorde al mismo. Esta información constará de países de origen y destino, distancia de viaje, fronteras cruzadas y, por último, restricciones y nivel de alarma relativos a la pandemia originada por el virus COVID-19.

La aplicación será desarrollada para dispositivos móviles con el sistema operativo Android.

Brief description

This document shows the process of the development of the Final Thesis of the Bachelor's degree. This project was assigned during the second semester of the 2020/2021 course. This Final Thesis consist of an individual engineering project for the Computer Science Bachelor's Degree. It has been developed and documented in Spanish.

The beginning of the idea for this project took place on the 16th of April of 2021 during the celebration of a Hackathon quiz organized by the University of Pécs (Hungary). This quiz was focused on the development of a prototype of a mobile application focused on health purposes. This application's prototype won the first price of this Hackathon so it supposed a big motivation to keep the development of the application as a Final Thesis.

The application is focused on international trip, which can be motivated for touristic purposes or any other purposes. It will allow the user to plan and manage a plane trip by selecting an origin country and a destination country. Having this, the application will show the user several options for the plane trip. This option will consist on the origin and destination countries, the distance of the trip, borders crossed and, finally, information about the restrictions and level of alarm due to the pandemic situation caused by the COVID-19 disease.

This application has been developed for Smartphones with the Android operative system.

Agradecimientos

Pese a que este proyecto ha sido bastante complejo, este capítulo ha sido el más difícil de redactar. Y es que son tantas las personas que me han acompañado y ayudado durante mi vida, que mencionarlas a todas haría este capítulo más extenso que el propio documento.

En primer lugar me gustaría darle las gracias a mis padres, Jesús y Conchi. Gracias por luchar tanto por mí incluso antes de venir a este mundo. Sabéis que aunque tengamos nuestros roces, sois para mí un ejemplo de valentía, esfuerzo, constancia y superación. Gracias a mi tía Luisa y a mi tío Pepe, que junto a mis padres me han cuidado desde pequeño y me han hecho ser la persona que hoy soy. Gracias a mi primo José por ser el motivo por el que me enamoré de la informática y a mi primo Carlos por haberme enseñado tantas nuevas experiencias en la vida. Gracias a mi prima Leticia, por tantos consejos sobre la universidad y tantas otras cosas, por tu cercanía y, sobre todo, por tu espíritu luchador e incansable, fuiste y serás un ejemplo para todos nosotros. Por último no me quiero olvidar de mi abuelo Alfonso, junto a mis padres la persona que más me ha querido en este mundo y de mis hermanas Cristina e Inmaculada. Sé que aunque nunca os llegué a conocer siempre habéis estado junto a mí.

También quiero dar las gracias a Alfonso Ureña por insistir en que probase la experiencia del Erasmus en Eslovenia y al país de Eslovenia en general por acogerme en el mejor año de mi vida. Gracias a todos mis compañeros de la carrera, en especial, a mis amigos de Send Futbol. Gracias también a todos los que estuvieron junto a mí en Eslovenia y con los que compartí momentos inolvidables. Gracias a Yago y a Sergio por tantas experiencias juntos y por tener la misma pasión que yo por lo que hacemos.

Finalmente me gustaría dar las gracias a mis amigos, en especial a Alberto, que siempre ha estado a mi lado en las buenas y especialmente en las malas. Y por último me gustaría darle las gracias a mi pareja, Alba, por enseñarme tantas lecciones de vida y enseñarme que quien te quiere, te espera.

Palabras Clave

En la siguiente lista se encuentran palabras descriptivas que pueden catalogar el presente proyecto:

- Android
- Aplicación
- Móvil
- Pandemia
- Viajes
- Vuelos
- Clínicas
- Test
- Restricciones
- COVID-19

Glosario

- Smartphone: Teléfono con capacidad para conectarse a internet o a una red de datos móviles.
- IDE: Entorno de desarrollo integrado(Integrated Development Enviroment).
- Obsolescencia programada: Programación del fin de la vida útil de un dispositivo.
- SDK: (Software Development Kit) Conjunto de herramientas para desarrollo software
- API: Interfaz de programación de aplicaciones(Application Programming Interface)
- Mockup: Modelo o prototipo del diseño de una aplicación.
- Actividad/Activity: Componente de la app que proporciona al usuario una vista y una clase programable
- JSON: (JavaScript Object Notation) formato de texto para el intercambio de datos
- Label: Estructura textual usada en una interfaz de usuario
- Marcador: Icono para señalar un punto geográfico dentro de un mapa
- Polilínea: Línea que une dos coordenadas
- UTF8: Formato estándar de codificación de caracteres

Índice

Índice de Ilustraciones

Índice de Tablas

1. Especificación del trabajo

En este capítulo se presenta la especificación del presente Trabajo de Fin de Grado, con una estructura y contenidos inspirados en los criterios y recomendaciones que establece la norma UNE 157801:2007 – "Criterios Generales para la elaboración de proyectos de Sistemas de Información". A lo largo del documento se utilizarán términos y acrónimos cuya descripción aparecen en la sección *Glosario.*

1.1. Introducción

Hoy en día no se imagina una vida sin teléfonos móviles, más en concreto los Smartphones, estos dispositivos son usados diario y prácticamente a todas horas. El uso de los mismos permite estar permanentemente actualizados y tener acceso de manera constante todo tipo de información, ya sea a través de otras personas o a través de un servicio web, que permitirá a cientos de miles de personas acceder a dicha información de manera concurrente.

Es por esto que la industria del Smartphone no deja de crecer, incluso durante estos últimos meses dónde la economía se ha resentido, el mercado del Smartphone ha experimentado un aumento del 35% en sus ingresos durante el primer trimestre del año 2021 frente al mismo periodo de 2020 **[1].**

Cada vez hay más marcas que buscan incorporarse a este mercado, de entre estas, la mayoría se decanta por el uso del sistema operativo Android en sus terminales. Se estima que en 2021 el 72.5% del tráfico web en dispositivos móviles provenía de terminales con dicho sistema operativo frente al 26.9% que suponía su competencia directa iOS **[2]**.

El motivo de esta cifras no es otro que el hecho de ser un sistema operativo de código abierto(open source), lo que significa que todo fabricante puede integrarlo en sus terminales sin necesidad de pagar una licencia. Este hecho será determinante a la hora de desarrollar un terminal telefónico, ya que, muchas empresas cuentan con un presupuesto limitado frente al resto de competidores, lo que hace que Android sea una propuesta muy interesante a la hora de implementar sus productos.

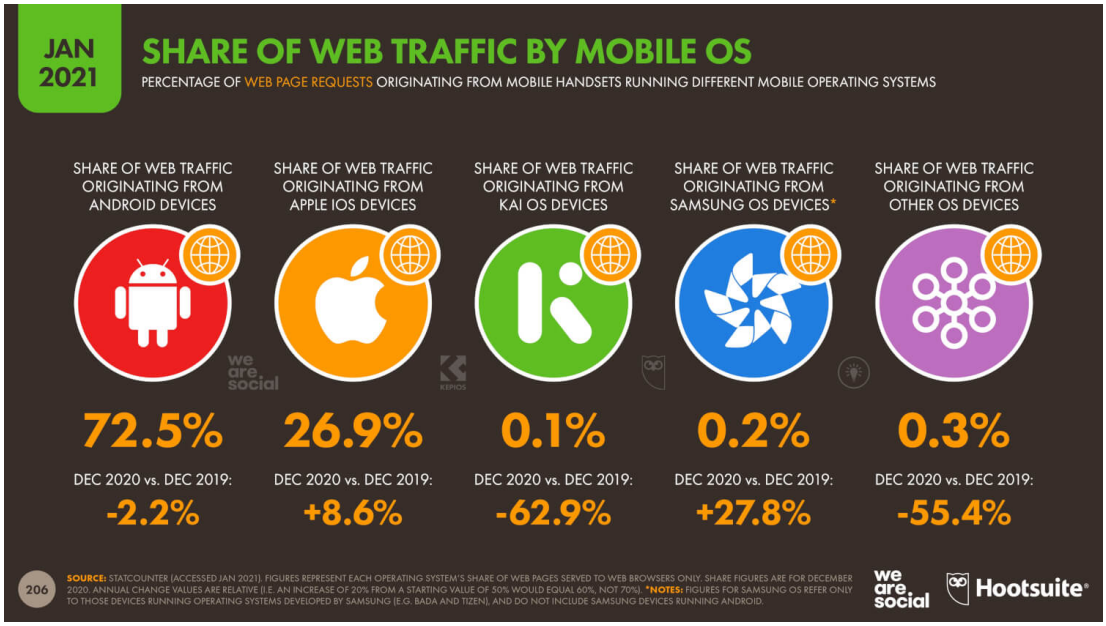

Ilustración 1. Gráfico del tráfico web por tipo de sistema operativo en SmartPhones[2]

El mismo sistema operativo Android afirma en su web: “Todo el mundo, incluidos los competidores de Android, puede optar por descargar, instalar, modificar y distribuir su ‘código fuente’ de manera gratuita para desarrollar aplicaciones, dispositivos móviles e incluso otros sistemas operativos” **[3]**.

Por otro lado, durante este duro año 2020 y esta primera mitad de 2021, como hemos podido observar, el mercado del Smartphone se ha visto muy beneficiado.

Esto se deberá, entre otros factores, a la etapa en la que la mayoría de la población fue confinada en sus casa y que, aún a día de hoy, el contacto con otras personas suponga un riesgo. Estos hechos han conllevado que el

Smartphone se haya vuelto una solución para que la gente pueda seguir en contacto con otras personas de su entorno a través de servicios de mensajería, redes sociales, aplicaciones de videollamada, etc… El Smartphone además, se ha posicionado como una herramienta muy útil para llevar a cabo tareas deldía a día de manera rápida y sencilla y, en la mayoría de los casos, sin necesidad de desplazarse fuera de casa.

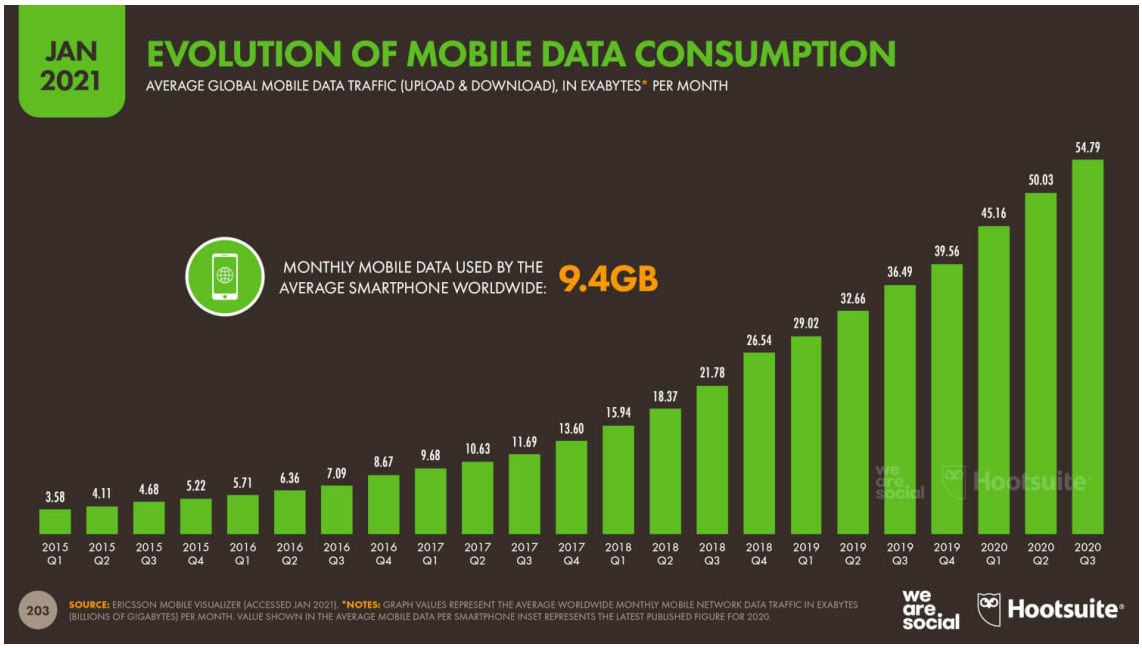

Ilustración 2. Evolución del consumo de datos móviles por cuatrimestre[2]

Estas tareas ya se podían realizar antes al igual que ahora, pero mucha gente no había reparado en ellas hasta verse en esta situación. Es aquí donde entran empresas tales como Amazon, Microsoft(gracias a su aplicación Teams), Apple, Paypal, Alphabet, Shopify, Zoom, etc… como las grandes beneficiadas. Si analizamos a las 15 empresas que más se han beneficiado económicamente por la pandemia, podemos apreciar que 14 de ellas están directamente relacionadas con el ámbito del Smartphone **[4]**.

Nombre de la empresa	Puesto en el ranking
Amazon	1
Microsoft	2
Apple	3
Tesla	4
Tencent	5
Facebook	6
Nvidia	7
Alphabet	8
PayPal	9
T-Mobile	10
Pinduoduo	11
Netflix	12
Meituan Dianping	13
Shopify	14
Zoom	15

Tabla 1. Empresas con mayor crecimiento durante la pandemia

Por contraparte, las empresas que mayores pérdidas han registrado han sido las empresas de viajes y de turismo. Y es que viajar hoy en día se ha convertido en una tarea engorrosa en la que muchas veces, más que el placer y el descanso que buscamos con ella, encontramos una gran frustración debido a las medidas y restricciones que se aplican en cada país **[4]**.

Desde mi experiencia personal, durante el curso 2020-2021, he realizado una movilidad Erasmus+, lo que me ha permitido vivir durante 10 meses en Eslovenia. Allí, como cualquier estudiante Erasmus, he deseado viajar desde el primer día y por el camino he encontrado decenas de trabas. Y es que, con una situación tan cambiante dónde las restricciones de los cruces de fronteras cambiaban semanalmente, resultaba muy frustrante tratar de encontrar dichas restricciones. Esto se debía a la necesidad de buscar información en las páginas oficiales del gobierno de cada país, las cuales se encontraban en el idioma de dicho país y suponían un conflicto lingüistico.

Es por eso que me aventuré a tratar de diseñar esta aplicación para tratar de ayudar a esas personas que, como yo, queremos seguir viajando de forma segura y seguir conociendo la cultura de otros países de una forma casi tan fácil como antes de la pandemia. Dicha aplicación nos permitirá conocer las restricciones de cada país al que queramos viajar dentro de Europa. Así como conocer los posibles vuelos desde un país a otro y saber de las principales clínicas del país en el que nos encontremos. En estas, podremos someternos a los test pertinentes en función de las restricciones del país donde deseemos viajar.

A lo largo de este documento, se presenta la aplicación realizada junto a una descripción de las características y requisitos principales. También se analizarán todas las tecnologías utilizadas en su desarrollo, así como un análisis de las fases del proyecto y un manual de usuario para comprender el funcionamiento de la misma.

1.2. Objetivos

El objetivo principal del proyecto es el de desarrollar una aplicación que permita planificar un viaje en avión entre un país de origen y un país de destino europeos. Para esto, la aplicación mostrará las restricciones del país de destino, los posibles vuelos mediante los cuáles se puede realizar dicho viaje. A su vez, permitirá ubicar en un mapa tanto los aeropuertos de origen y destino como las clínicas del país seleccionado para poder someterse las pertinentes pruebas médicas previas al viaje.

Otro objetivo es el de hacer que la aplicación sea atractiva, el flujo de navegación dentro de la misma sea lógico e intuitivo y el uso sea sencillo y eficiente.

Por último, a nivel personal, otro de los objetivos es el de familiarizarme con un IDE como Android Studio y desarrollar un proyecto de este tamaño por mi mismo desde cero, documentándome y estudiando el entorno por mi cuenta y utilizando los conocimientos sobre Java adquiridos en la carrera.

1.3. Aplicaciones existentes

En esta sección se mostrarán algunas de las aplicaciones del ámbito de la planificación de viajes que más impacto han tenido en el mercado. Estas aplicaciones comparten con la aplicación desarrollada en el presente proyecto una herramienta para la elección de vuelos. Aunque, no existe como tal una aplicación que combine la búsqueda de vuelos con la información sobre restricciones así como con información sobre clínicas médicas. Esto hará que la aplicación desarrollada suponga una oferta interesante e innovadora para el usuario, como se comenta más adelante en el apartado *Conclusiones*.

1.3.1. SkyScanner:

Skyscanner es un motor de búsqueda global que compara vuelos, hoteles y alquiler de automóviles. El servicio es libre para usuarios que se dirigen a la aerolínea, hotel o proveedor de servicios automovilísticos o agencia de viajes para completar el proceso de reserva.

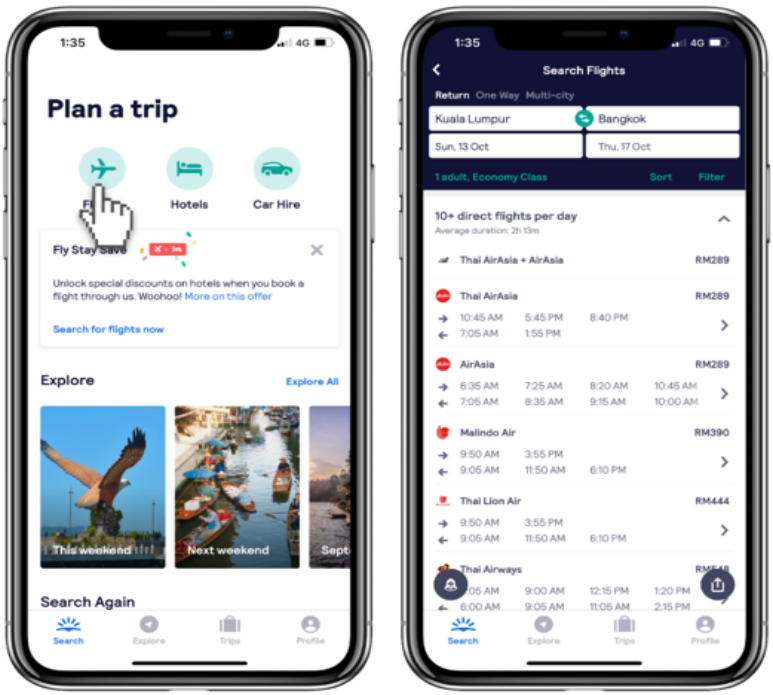

Ilustración 3. Interfaz de la aplicación SkyScanner[31]

1.3.2. Kayak:

Kayak es un agregador de tarifas y metabuscador de viajes en línea. La aplicación de Kayak permite a sus usuarios comparar y reservar vuelos, hoteles, coches de alquiler y paquetes de vacaciones de cientos de proveedores a la vez

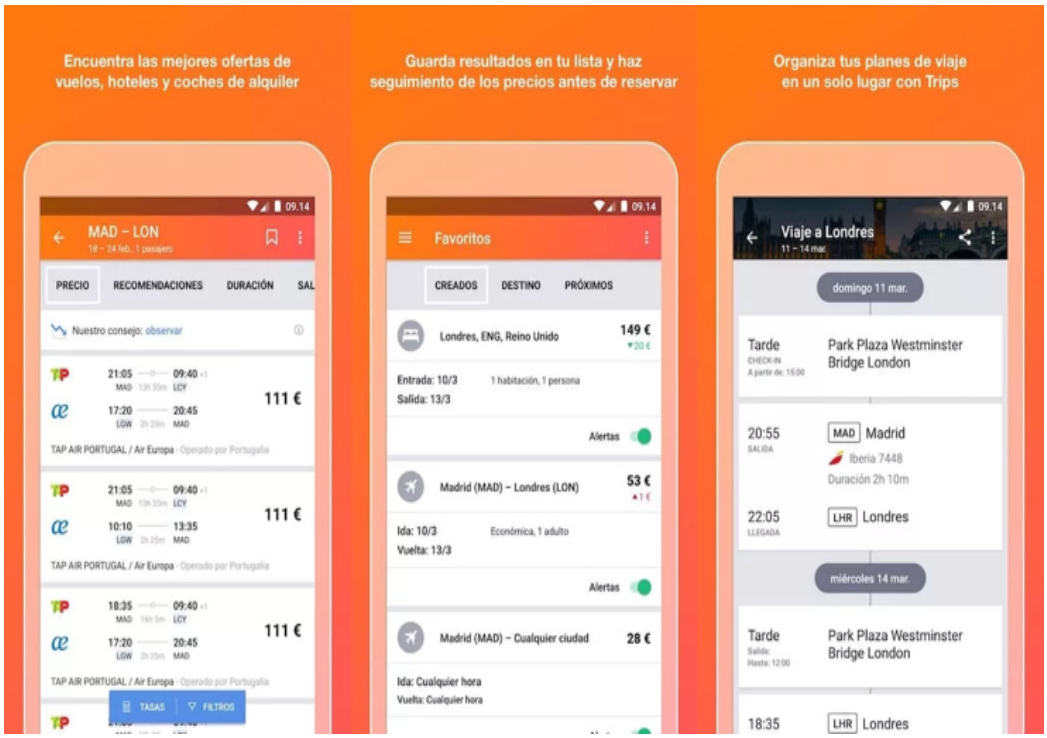

Ilustración 4. Interfaz de la aplicación Kayak[32]

1.4. Metodología utilizada

En este apartado, se mostrará la metodología seguida para el desarrollo de este proyecto. Al principio del mismo, se contemplaron diferentes metodologías a seguir, dichas metodologías se expondrán brevemente a continuación y finalmente se procederá a comentar de manera más detallada la metodología elegida y la razón por la que se ha decidido seguir dicha metodología.

En primer lugar, las metodologías se clasifican en función de su enfoque y características esenciales, esto dividirá las metodologías en dos tipos, metodologías tradicionales y metodologías ágiles.

Metodologías tradicionales

Ventajas	Inconvenientes
Claridad de objetivos	Los resultados se obtienen de forma más lenta
Mejores para proyectos de menor dimensión	No se admiten cambios
No necesitan de equipos especializados o recursos	Reiniciar el proyecto eleva mucho el coste del mismo

Tabla 2. Ventajas e inconvenientes de las metodologías tradicionales

Metodologías ágiles

Ventajas	Inconvenientes
Los resultados se obtienen de manera muy rápida	Necesidad de un buen grado de comunicacón
Tiempos de respuesta y resultados mucho más rápidos y reutilizables	Menos indicados para proyectos de menor dimensión
Fiabilidad	Depende mucho del cliente y puede no funcionar si no tiene claro lo que quiere
Satisfacción del cliente	Coste variable

Tabla 3. Ventajas e inconvenientes de las metodologías ágiles

Tras los motivos expuestos previamente, se decide optar por una metodología ágil, esta nos permitirá realizar cambios de manera más sencilla y obtener los resultados de manera más rápida.

Las metodologías ágiles se basan en la evolución tanto de la necesidad como de la solución a la misma y en el trabajo coordinado en equipo. Para esto, destacan como cualidades el desarrollo evolutivo y flexible del producto, la autonomía de los equipos encargados del proyecto así como la comunicación entre los mismos y la planificación de objetivos. Así, toda metodología ágil ha de cumplir con el manifiesto ágil, compuesto por 12 principios y agrupado en 4 valores fundamentales **[5]**. Estos valores serán:

- Individuos e interacciones sobre procesos y herramientas
- Software funcionando sobre documentación extensiva
- Colaboración con el cliente sobre negociación contractual
- Respuesta ante el cambio sobre seguir un plan

Dentro de las metodologías ágiles existen varias opciones, de entre las cuales destacan SCRUM, Progamación Extrema(XP) y Kanban.

- **SCRUM**

  Esta metodología es un marco de trabajo de procesos ágiles que trabaja con el ciclo de vida iterativo e incremental. El producto se libera por pares de manera periódica, aplicando las buenas prácticas de trabajo colaborativo en equipo. A su vez, facilita el hallazgo de soluciones óptimas con los problemas que pueden surgir durante el desarrollo del proyecto.

  Con SCRUM se realizan entregas regulares y parciales llamadas *Sprints* del producto final. Todas ellas con una prioridad previamente establecida que nace según el beneficio que aporten al cliente minimizando además los riesgos por un desarrollo excesivamente largo. Es por esto que SCRUM se indica para proyectos en entornos complejos dónde se buscan resultados de manera inmediata. SCRUM destaca por su innovación, productividad, flexibilidad y competitividad **[5]**.

- **Programación Extrema (XP)**

  Esta metodología está basada en un conjunto de reglas y buenas prácticas para el desarrollo de software en ambientes muy cambiantes con requisitos imprecisos. Por tanto, está enfocada a la retroalimentación contínua entre el equipo de desarrollo y el cliente.

  Esto conlleva que, iniciado el proyecto, se deban definir todos los requisitos, para luego invertir el esfuerzo en manejar los cambios que se presenten y así minimizar las posibilidades de error. La programación extrema resulta simple y cuenta con un alto grado de satisfacción del cliente **[5]**.

- **Kanban**

  Esta metodología requiere una comunicación en tiempo real sobre la capacidad del equipo. Se utiliza para controlar el avance de trabajo en una línea de producción, en la cual se clasifican las tareas en sub estatus. El motivo no es otro que determinar los niveles de productividad en cada fase del proyecto.

  Para el desarrollo del software, se simplifica la planificación y la asignación de responsabilidades. Los procesos de flujo de trabajo se representan en un tablero de un mínimo de 3 columnas (Pendiente, En Progreso y Terminado). La cantidad de tarjetas en la columna *Pendiente* forma parte de lo solicitado por el cliente, mientras que la cantidad en la columna *En Progreso* dependerán de la capacidad del equipo desarrollador **[5]**.

Para este proyecto, se elige la metodología SCRUM por ser la más usada en este tipo de proyectos y la que mejores resultados ofrece en los mismos, así como ser la más popular dentro de las metodologías ágiles. Dicha metodología se ha seguido en la medida de lo posible, teniendo en cuenta las limitaciones derivadas de la ejecución del Trabajo de Fin de Grado en sí mismo.

La metodología SCRUM contará con una serie de roles específicos imprescindibles para el desarrollo del proyecto. En el caso de este proyecto, todos los roles del equipo SCRUM serán interpretados por el autor del Trabajo de Fin de Grado excepto el rol de cliente, que será asumido por los tutores del mismo. El equipo SCRUM está formado por 4 roles principales.

- **StakeHolder(Cliente)**

  Su responsabilidad radica en definir los requerimentos (Backlog), recibir el producto al final de cada *Sprint* y proporcionar feedback.

- **Product Owner**

  Es el intermediario entre el cliente y el equipo de desarrollo. Será el encargado de priorizar los requerimentos en función de las necesidades de la solicitud.

- **SCRUM Master**

  Actúa como facilitador ante todo el equipo de desarrollo. Eliminará todos los impedimentos que se identifiquen durante el proceso y se encarga de revisar que el equipo siga los valores y principios ágiles, las reglas y los procesos SCRUM.

- **SCRUM Team**

  Se encarga de desarrollar los casos de uso definidos en los requerimentos. Se trata de un equipo auto gestionado, por lo que no existe un jefe de equipo como tal, por lo que todos los integrantes se encargarán de realizar las estimaciones. En base a la velocidad de los *Sprints* se irá construyendo el *Sprint Backlog.*

El otro pilar fundamental de SCRUM son las reuniones. Los tipos de reuniones existentes son:

- **Reunión de planificación**

  Se realiza al inicio de cada *Sprint*, el objetivo es planificar la cantidad de trabajo a la que el equipo se someterá durante dicho *Sprint*.

- **Reunión diaria o *daily***

  Se realizan a diario con una duración máxima de 15 minutos, en ellas se comentan el trabajo del día previo y el objetivo del presente y los problemas surgidos. Se establecen planes de 24 horas.

- **Reunión de revisión**

  Se lleva a cabo al final de cada Sprint y se revisan qué puntos han sido completados y cuáles no.

- **Reunión de retrospectiva**

  Se lleva a cabo también al final del Sprint con el fin de analizar posibles mejoras. A esta reunión deberá asistir todo el equipo y es de las más importantes

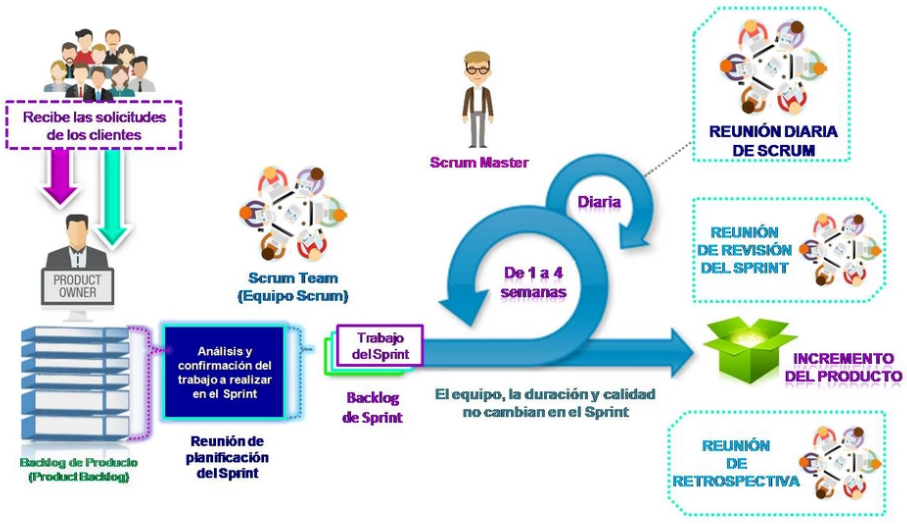

Ilustración 5. Representación gráfica de la metodología SCRUM[5]

1.5. Tecnologías Utilizadas

En este capítulo, se mencionarán las herramientas utilizadas para la elaboración del presente proyecto. Los criterios de elección han sido la versatilidad de las mismas, sencillez de uso y familiaridad del autor con las mismas.

1.5.1. Java como lenguaje de programación

Ilustración 6. Logo de Java[33]

Este paso fue sencillo, ya que las dos únicas opciones contemplables eran Java y Kotlin. Si bien a nivel de documentación en lo relativo a Android son prácticamente idénticos, ya que Android ofrece la misma documentación para sendos lenguajes, se decide utilizar Java.

Esto se debe a que, además de ser uno de los lenguajes más utilizados y más interesantes de masterizar de cara a un futuro laboral. Es un lenguaje con el que he trabajado bastante durante la carrera y esa ausencia de necesidad de familiarizarme con el lenguaje de programación me brindaría más tiempo para poder hacerlo con el IDE.

1.5.2. Android Studio como IDE

Ilustración 7. Logo de Android Studio[6]

Cuando se habla de desarrollo de aplicaciones para el sistema operativo Android, la primera idea suele ser el IDE de Android Studio **[6]**. Existen otros IDEs como Eclipse **[7]**, el cual fue reemplazado por Android Studio en 2013 como IDE oficial para el desarrollo de aplicaciones Android **[8]** incluso SDKs como Flutter **[9]**. Se decide el uso de Android Studio para el desarrollo de este proyecto por su sencillez y por la calidad y amplitud de su documentación **[10]**, así como también por la extensión de su comunidad. Esto hace que resolver dudas sobre la implementación de la aplicación o aprender sobre las herramientas que nos brinda Android Studio sea una tarea sencilla.

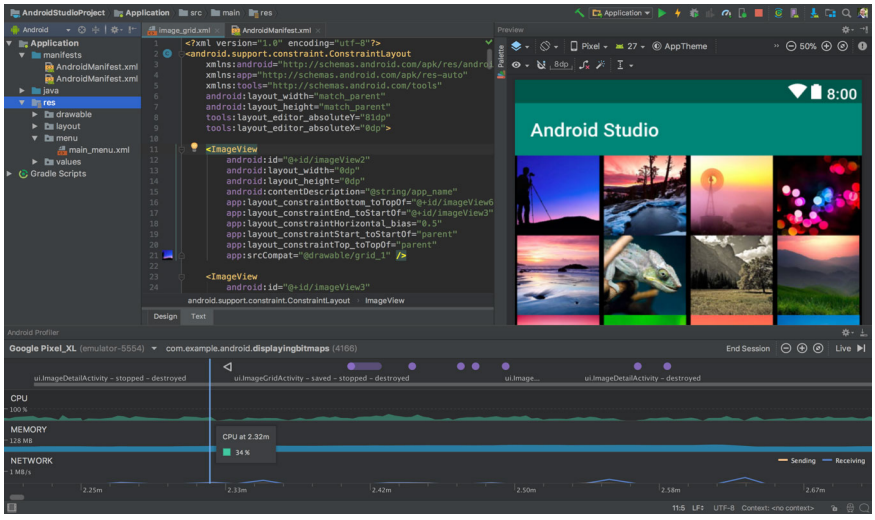

Ilustración 8. Interfaz de Android Studio[6]

1.5.3. XAMPP y MySQL

Ilustración 9. Logo de XAMPP[12]

Ilustración 10. Logo de MySQL[11]

Como se ve previamente, Android implementa la biblioteca de SQLite para la gestión de bases de datos relacionales. Pese a eso, en este caso se decide utilizar MySQL **[11]** a través de XAMPP **[12]** para poder gestionar las bases de datos que aportarán información a la aplicación de manera remota.

Si bien conectarla con la aplicación es un proceso más complejo que utilizar la biblioteca SQLite, es mucho más cómoda a la hora de gestionar las bases de datos.

1.5.4. Notepad++

Ilustración 11. Logo de Notepad++[13]

Para la conexión a las bases de datos, ha sido necesario programar una serie de scripts en lenguaje PHP. Para ello, se ha elegido el editor de código Notepad++ **[13]** ya que, si bien es muy simple, es el editor de código usado durante gran parte de la carrera por lo que es una tecnología conocida.

1.5.5. Mockflow

Ilustración 12. Logo de MockFlow[14]

Mockflow **[14]** es una herramienta para elaborar mockups online. Esta herramienta es la que se usó en el hackathon dónde surgió la idea de esta aplicación así que ya que estaba familiarizado con ella así que se decide seguir usándola. Por otro lado, permite añadir vistas con mapas de google maps, lo cual me resulta muy interesante y fue definitivo para decantarse por esta herramienta.

1.5.6. Visual Paradigm

Ilustración 13. Logo de Visual Paradigm[15]

Visual Paradigm **[15]** es una herramienta de diseño de diagramas para la organización de proyectos software. Se elige por ser una herramienta que utilizada previamente en la carrera y que permite diseñar diagramas UML con sencillez.

1.5.7. Tom's Planner

Ilustración 14. Logo de Tom's Planner[16]

Tom's planner es un proveedor de servicios de aplicaciones y herramientas basado en la web para la planificación, gestión y colaboración de proyectos **[16]**.

1.6. Introducción a Android

Android **[3]** es un sistema operativo móvil basado en núcleo Linux y otros software de código abierto. Fue diseñado para dispositivos móviles con pantalla táctil, como Smartphones, Tablets, Smartwatches automóviles a través de Android Auto, televisiones…

Android es presentado en 2007 por la compañía Android Inc., propiedad de Google desde 2005, abogando por los estándares abiertos para dispositivos móviles. Desde entonces, son numerosas las actualizaciones y nuevas versiones que han ido apareciendo hasta hoy. Muchas de ellas se instalan con el fin de arreglar fallos pero también para restringir según qué tipo de hardware, por lo que se considera a Android uno de los mayores promotores de la famosa obsolescencia programada

Ilustración 15. Logo de Android[3]

1.6.1. Arquitectura de Android

El sistema operativo Android se estructura en 5 capas como se puede apreciar en la siguiente imagen, las cuales se analizan desde el núcleo de Linux. Este supone la capa más interna del sistema operativo, a la capa más superficial dónde se encuentran las aplicaciones.

Ilustración 16. Representación gráfica de las capas de la arquitectura de Android [30]

1.6.1.1. Capa del Núcleo(o kernel) de Linux:

Cómo se menciona anteriormente, el núcleo de Android está basado en el núcleo del sistema operativo Linux, éste actuará como capa de abstracción entre el hardware y el resto de la pila software.

1.6.1.2. Capa intermedia

1.6.1.2.1. Runtime de Android:

El Android Runtime o entorno de ejecución de Android incluye un set de bibliotecas que proporcionan casi todas las funciones principales de las bibliotecas del lenguaje Java.

Aparte, Google desarrolla Dalvik, una máquina virtual basada en registros y que corre clases compiladas por el compilador Java. Esta máquina destacaba por su optimización de recursos ya que dichas clases son almacenadas en formato .dex y desde la versión 5.0 compila totalmente al momento de instalación de la aplicación.

1.6.1.2.2. Bibliotecas

Al mismo nivel encontramos el conjunto de bibliotecas de C/C++ que son utilizadas por varios componentes del sistema, entre estas bibliotecas encontramos:

- System C Library: Implementación de biblioteca C estándar.
- WebKit: Para el navegador web, usada también en Google Chrome o Safari.
- OpenGL: Biblioteca de gráficos 3D.
- SQLite: Motor de bases de datos relacionales.
- *Secure Socket Layer* (SSL): Protocolos criptográficos para comunicaciones seguras.

1.6.1.3. Marco de trabajo de aplicaciones

Diseñado para simplificar la reutilización de componentes, cualquier aplicación podrá publicar sus capacidades y cualquier otra aplicación podrá hacer uso de ellas dentro de unas reglas. Los componentes pueden ser reemplazados por el usuario. Dentro de los servicios de este marco de trabajo podemos encontrar el sistema de vistas, el sistema de localización, el sistema que maneja el flujo de vida de las actividades o el sistema de notificaciones entre otros.

1.6.1.4. Aplicaciones

Por último, se encuentra la capa de aplicaciones, en ella está ubicado todo el conjunto de aplicaciones del dispositivo, ya sean nativas del sistema operativo o de un desarrollador externo e instaladas por el usuario. Éstas deberán funcionar en la máquina virtual *Dalvik* para garantizar seguridad y la mayoría están escritas en Java.

1.6.2. Versiones de Android

El sistema operativo se inicia con el lanzamiento de la versión beta en 2007, más tarde, en septiembre de 2008, se lanza la primera versión comercial, Android 1.0. En febrero de 2009, Android lanza la primera actualización de su sistema operativo. Desde entonces, hasta la versión 9, cada nueva versión ha recibido el nombre de un postre siguiendo un orden alfabético, a continuación se aprecian todas las versiones de Android lanzadas hasta la fecha:

Nombre	Versión	Lanzamiento	Nivel de API	Distribución[4]
Apple Pie	1.0	Septiembre 2009	1	
Banana Bread	1.1	Febrero 2009	2	
Cupcake	1.5	Abril 2009	3	
Donut	1.6	Septiembre 2009	4	
Éclair	2.0 - 2.1	Octubre 2009	5 - 7	
Froyo (Frozen Yogurth)	2.2 - 2.2.3	Mayo 2010	8	
Gingerbread	2.3 – 2.3.7	Diciembre 2010	9 – 10	
Honeycomb	3.0 – 3.2.6	Febrero 2011	11 - 13	
Ice Cream Sandwich	4.0 – 4.0.5	Octubre 2011	14 – 15	
Jelly Bean	4.1 – 4.3.1	Julio 2012	16 – 18	
KitKat	4.4 – 4.4.4	Octubre 2013	19 – 20	
Lollipop	5.0 – 5.1.1	Noviembre 2014	21 – 22	
Marshmallow	6.0 – 6.0.1	Octubre 2015	23	4.39%
Nougat	7.0 – 7.1.2	Junio 2016	24 – 25	4.1%
Oreo	8.0 – 8.1	Agosto 2017	26 – 27	8.79%
Pie	9.0	Agosto 2018	28	16.48%
10	10.0	Septiembre 2019	29	20.94%
11	11.0	Septiembre 2020	30	34.17%
12	12.0	Agosto 2021	31	

Tabla 4. Versiones de Android

1.7. Estructura de la Aplicación

En este apartado se analizan los elementos más importantes de la estructura de una aplicación Android **[22]** dentro del IDE de Android Studio.

1.7.1. Actividades

1.7.1.1. Ciclo de vida de una actividad

En este esquema se puede observar el ciclo de vida **[23]** de una aplicación, desde que se inicia por primera vez hasta que se cierra y se terminan sus procesos.

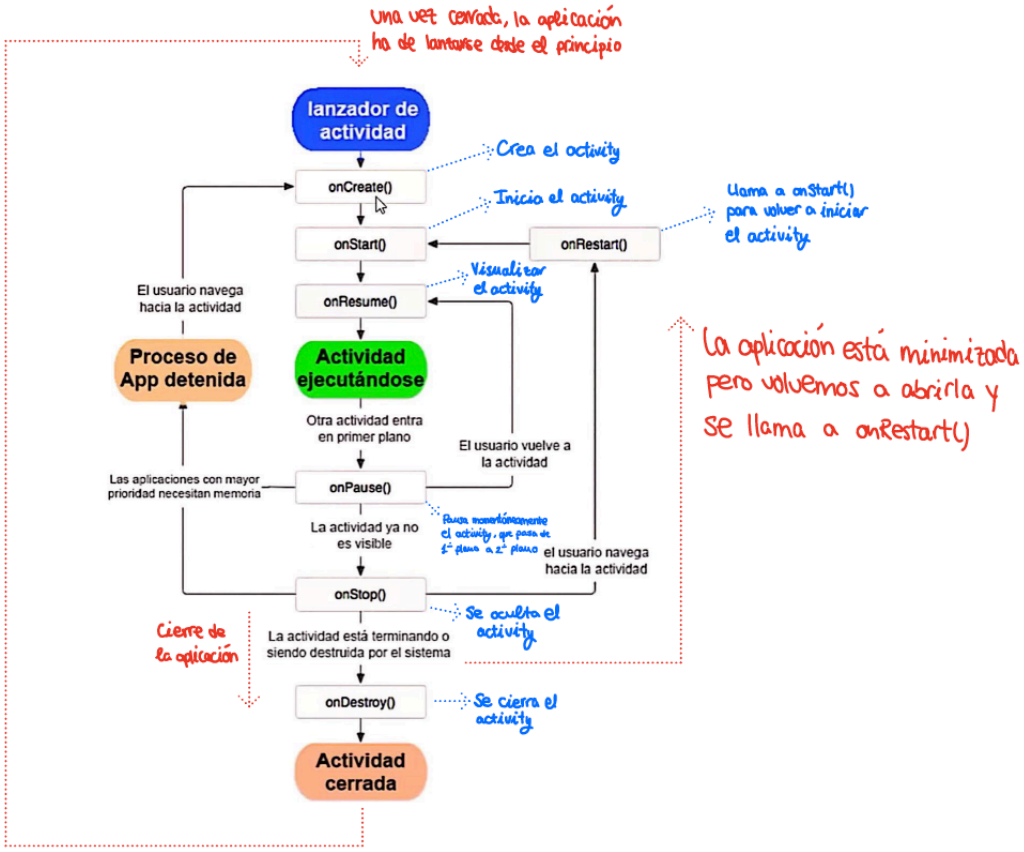

Ilustración 17. Representación gráfica del ciclo de vida de una actividad

1.7.1.1.1. onCreate():

En este método se crea el activity y se ejecuta la lógica de arranque básica de la misma. Dentro de este método será donde se asocian los valores por defecto que se desea que contengan las variables previamente inicializadas al inicio de la actividad, así como a otro tipo de estructuras de datos.

Por otro lado también será el lugar donde tendremos que llamar a los métodos que queremos que se ejecuten al inicio del ciclo de vida de la app.

1.7.1.1.2. onStart():

En este método inicia un activity que ya ha sido creado previamente mediante el método onCreate().

1.7.1.1.3. onResume():

En este método, se pasa a visualizar el activity en primer plano, este método perdura hasta que otra actividad pase a primer plano o la aplicación se minimice o cierre.

1.7.1.1.4. onPause():

En este método se pausará el activity, haciendo que este pase de primer a segundo plano. Esto no significa que el activity desaparezca, éste sigue existiendo y puede volver a visualizarse en primer plano conservando los mismos valores. Para ello, se ha de llamar otra vez al método onResume(). En caso de que no, se llamará al método onStop().

1.7.1.1.5. onStop():

Se puede considerar a éste método el método contrario a onResume(). Mientras que onResume() permite visualizar el activity en primer plano, onStop() hace que dejemos de hacerlo, ya sea porque la aplicación se ha minimizado o se ha cerrado.

En caso de querer volver a ver la actividad en primer plano tras haber minimizado la aplicación, será necesario llamar al método onRestart().

En caso de haberse cerrado la aplicación, será necesario volver a crear el activity desde el principio empezando por el método onCreate().

1.7.1.1.6. onRestart():

Éste método permitirá llamar directamente al método onStart() de un activity después de haber llamado al método onStop() de dicho activity pero sin haber sido este destruido. En este caso, se puede volver a visualizar el activity con los mismos valores que tenía al detenerse.

1.7.1.1.7. onDestroy():

Éste método cierra o destruye el activity cuando la aplicación se cierra.

1.7.1.2. Clase java asociada

En esta clase será donde se programe todo el comportamiento de la actividad y donde se llame a los métodos previamente comentados.

Aparte, será aquí donde se programen el resto de métodos que determinarán el funcionamiento de la aplicación y permitirán realizar una serie de funciones dentro de la misma.

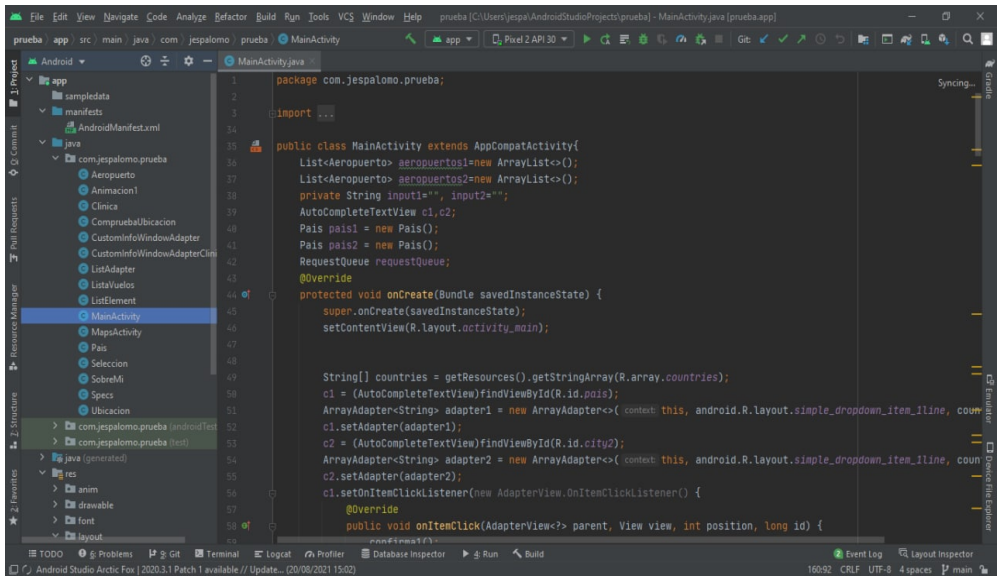

Ilustración 18. Ejemplo de una clase Java

1.7.1.3. Layout

En esta clase se encuentran una serie de archivos en formato XML. Estos archivos permitirán programar la interfaz grafica del activity ya sea programando directamente el código XML o mediante una interfaz que nos brinda el IDE de Android Studio.

Ésta interfaz es muy útil ya que permite ver en tiempo real la apariencia del activity, además de ofrecer un modo de visualización *blueprint* que será muy útil a la hora de la alineación y situación de los elementos del activity.

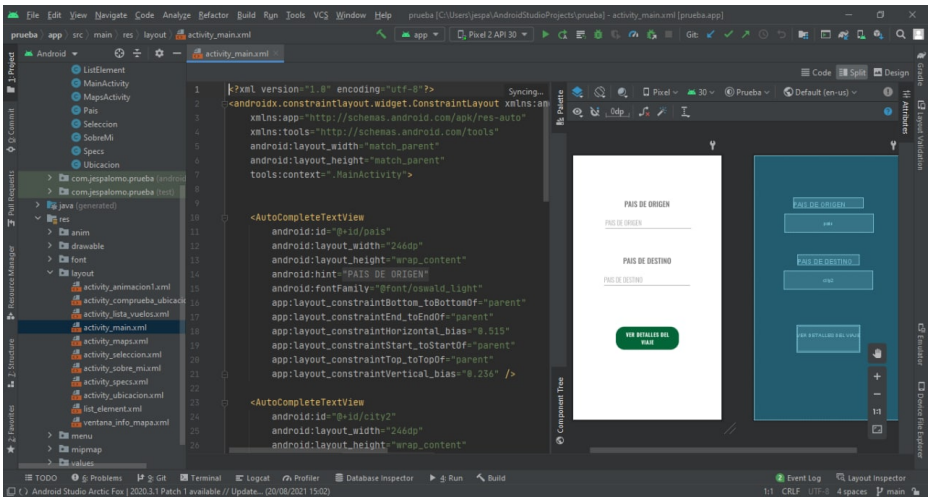

Ilustración 19. Ejemplo de un fichero XML y diseño de interfaz en Android Studio

1.7.2. Archivo Strings.xml

Este archivo contendrá las cadenas de caracteres de tipo String que se podrán visualizar de forma estática en los elementos de la interfaz de la aplicación. Cuando se programa la interfaz de una actividad y se añade un elemento que incluye una cadena de texto, se debe asociar el texto de dicho elemento a una cadena perteneciente al archivo Strings.xml

1.7.3. Directorio Drawable

En este directorio se almacenan todas las imágenes que se utilizan dentro de la aplicación

1.7.4. Archivo Android Manifest

Éste archivo en formato XML es un denominado archivo de manifiesto **[22]**. Éste archivo es necesario en cualquier proyecto Android y será el que describe la información de la aplicación para las herramientas de creación de Android, el sistema operativo y la tienda de Google Play. Dentro de este archivo será donde se determina la jerarquía de las actividades para establecer una de ellas como actividad principal, que será la actividad que se creará cuando se inicie la aplicación.

En éste archivo también será dónde se otorguen los permisos para que la aplicación pueda acceder al GPS del dispositivo o a internet entre otros.

1.7.5. El paquete Gradle

El paquete Gradle **[24]** es el sistema de compilación oficial de Android. Este permite, entre otras cosas, determinar las versiones mínima y máxima con las que será compatible la aplicación. También permite implementar librerías externas como Volley o MaterialDesign en este caso.

1.7.6. Elementos de la interfaz

1.7.6.1. TextView

Esta clase pública permite mostrar un texto en la interfaz, a continuación se muestra un ejemplo de un *TextView* de la aplicación:

```
<TextView
    android:id="@+id/textView"
    android:layout_width="194dp"
    android:layout_height="29dp"
    android:fontFamily="@font/oswald_medium"
    android:gravity="center"
    android:text="PAIS DE ORIGEN"
    android:textSize="20sp"
    app:layout_constraintBottom_toTopOf="@+id/pais"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.497"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.882" />
```

Ilustración 20. Ejemplo del código XML de un elemento TextView

```
<string name="city1">PAIS DE ORIGEN</string>
```

Ilustración 21. Ejemplo de una variable del archivo Strings.xml

Ilustración 22. Ejemplo de un elemento TextView en la interfaz

1.7.6.2. Button

Esta clase pública permitirá ejecutar un evento programado dentro de la clase java correspondiente a la interfaz cuando se hace click sobre dicho botón. A continuación se muestra un ejemplo de un *Button* de la aplicación:

```
<Button
    android:id="@+id/avanzar"
    android:layout_width="175dp"
    android:layout_height="75dp"
    android:backgroundTint="@color/accent"
    app:cornerRadius="24dp"
    android:fontFamily="@font/oswald_medium"
    android:onClick="botonSpecs"
    android:text="VER VUELOS"
    android:textSize="14sp"
    android:textStyle="bold"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/city2"
    app:layout_constraintVertical_bias="0.347" />
```

Ilustración 23. Ejemplo del código XML de un elemento Button

Ilustración 24. Ejemplo de un elemento Button en la interfaz

1.7.6.3. AutoCompleteTextView

Esta clase pública se permitirá escribir una cadena de texto y mostrará una serie de sugerencias en función del texto ya escrito por el usuario.

Estas sugerencias formarán parte de un vector dentro del archivo *strings.xml*. A continuación se muestra un ejemplo de un *AutoCompleteTextView* de la aplicación:

```
<AutoCompleteTextView
    android:id="@+id/pais"
    android:layout_width="246dp"
    android:layout_height="wrap_content"
    android:hint="PAIS DE ORIGEN"
    android:fontFamily="@font/oswald_light"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.515"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.236" />
```

Ilustración 25. Ejemplo del código XML de un elemento AutoCompleteTextView

```
<string name="city1">PAIS DE ORIGEN</string>
```

Ilustración 26. Ejemplo de una variable del archivo Strings.xml

```
<string-array name="countries">
    <item>Albania</item>
    <item>Alemania</item>
    <item>Andorra</item>
    <item>Austria</item>
    <item>Belgica</item>
    <item>Bielorrusia</item>
    <item>Bosnia y Herzegovina</item>
    <item>Bulgaria</item>
    <item>Ciudad del Vaticano</item>
```

Ilustración 27. Ejemplo de un vector en el archivo Strings.xml

Ilustración 28. Ejemplo de una variable AutoCompleteTextView en la interfaz

1.7.6.4. Switch

Esta clase pública funcionará de manera similar a *button*, sólo que contará con un estado “activo” y un estado “no activo”. Estos estados se mantendrán hasta que se clique en el botón, lo cual hará que cambie de un estado a otro. Por otro lado, la estética es distinta. A continuación se muestra un ejemplo de un *Switch* de la aplicación

```
<Switch
    android:id="@+id/switch1"
    android:layout_width="152dp"
    android:layout_height="38dp"
    android:fontFamily="@font/oswald_light"
    android:onClick="bot"
    android:text="Incluir destinos cercanos"
    android:textSize="12sp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="@+id/lista"
    app:layout_constraintHorizontal_bias="0.941"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.023" />
```

Ilustración 29. Ejemplo del código XML de un elemento Switch

Ilustración 30. Ejemplo de un elemento Switch en la interfaz

1.7.6.5. ImageView

Esta clase pública permitirá visualizar una imagen en la interfaz. A continuación se muestra un ejemplo de un *ImageView* de la aplicación:

```
<ImageView
    android:id="@+id/imagenLogo"
    android:layout_width="372dp"
    android:layout_height="385dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.515"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.583"
    app:srcCompat="@drawable/logo_app" />
```

Ilustración 31. Ejemplo del código XML de un elemento ImageView

Ilustración 32. Ejemplo de un elemento ImageView en la interfaz

1.7.6.6. RecyclerView

Esta clase pública permitirá visualizar una lista de elementos en la interfaz e interactuar con ellos. A continuación se muestra un ejemplo de un *RecyclerView* de la aplicación:

```
<androidx.recyclerview.widget.RecyclerView
    android:id="@+id/lista"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginStart="1dp"
    android:layout_marginTop="8dp"
    android:layout_marginEnd="1dp"
    android:layout_marginBottom="1dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/vuelos" />
```

Ilustración 33. Ejemplo del código XML de un elemento RecyclerView

Ilustración 34. Ejemplo de un elemento RecyclerView en la interfaz

1.7.6.7. CardView

Esta clase pública, perteneciente a la biblioteca de Material Design, permitirá diseñar individualmente la apariencia de los elementos de una lista.

```
<androidx.cardview.widget.CardView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:id="@+id/cv"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:animateLayoutChanges="true"
    app:cardCornerRadius="20dp"
    app:cardElevation="4dp"
    app:cardUseCompatPadding="true">
```

Ilustración 35. Ejemplo del código XML de un elemento Cardview

Ilustración 36. Ejemplo de un elemento CardView en la interfaz

1.7.7. Atributos importantes en XML

- android:id: Id del elemento.
- android:layout_width: Anchura del element.
- android:layout_height: Altura del element.
- android:fontFamily: Fuente del texto mostrado.
- android:gravity: Situación del elemento dentro del contenedor.
- android:text: Texto mostrado.
- android:textSize: Tamaño del texto.
- android:textStyle: Estilo del texto(negrita, cursiva..).
- android:backgroundTint: Color del fondo.
- android:cornerRadius: Radio de las esquinas del botón.
- android:onClick: Evento de la clase asociada que se activará al clicar el botón.
- app:srcCompat: Ruta de la imagen mostrada.
- app:layoutConstraint…: Mostrarán el ajuste del elemento dentro del activity.

1.8. Planificación temporal

En este apartado, se observa la planificación temporal del proyecto mediante un Diagrama de Gantt, el cual permitirá apreciar las diferentes tareas realizadas y en qué período de tiempo fueron realizadas.

Si bien existen 3 procesos principales, los cuales serán “Análisis y Diseño”, “Implementación” y “Documentación”, estos procesos tendrán, a su vez, otros procesos internos los cuales se verán también reflejados dentro del diagrama.

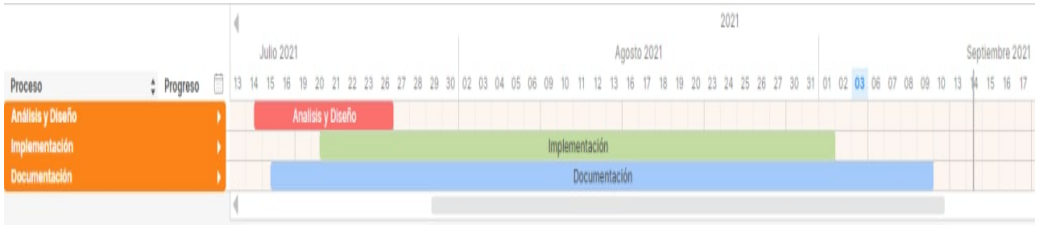

Ilustración 37. Diagrama de Gantt del proyecto

En la siguiente imagen, aparece de manera más detallada los subprocesos de cada uno de los procesos principales. En caso de este proyecto, se decidió seguir una metodología de tipo SCRUM, por tanto, la implementación aparece dividida en una serie de *Sprints*.

Estos Sprints debieron tener una duración aproximada de una semana. Pese a esto, debido a que el proyecto ha sido mayoritariamente desarrollado durante el mes de agosto, el último sprint ha tenido una mayor duración, ya que durante este mes no se han producido reuniones.

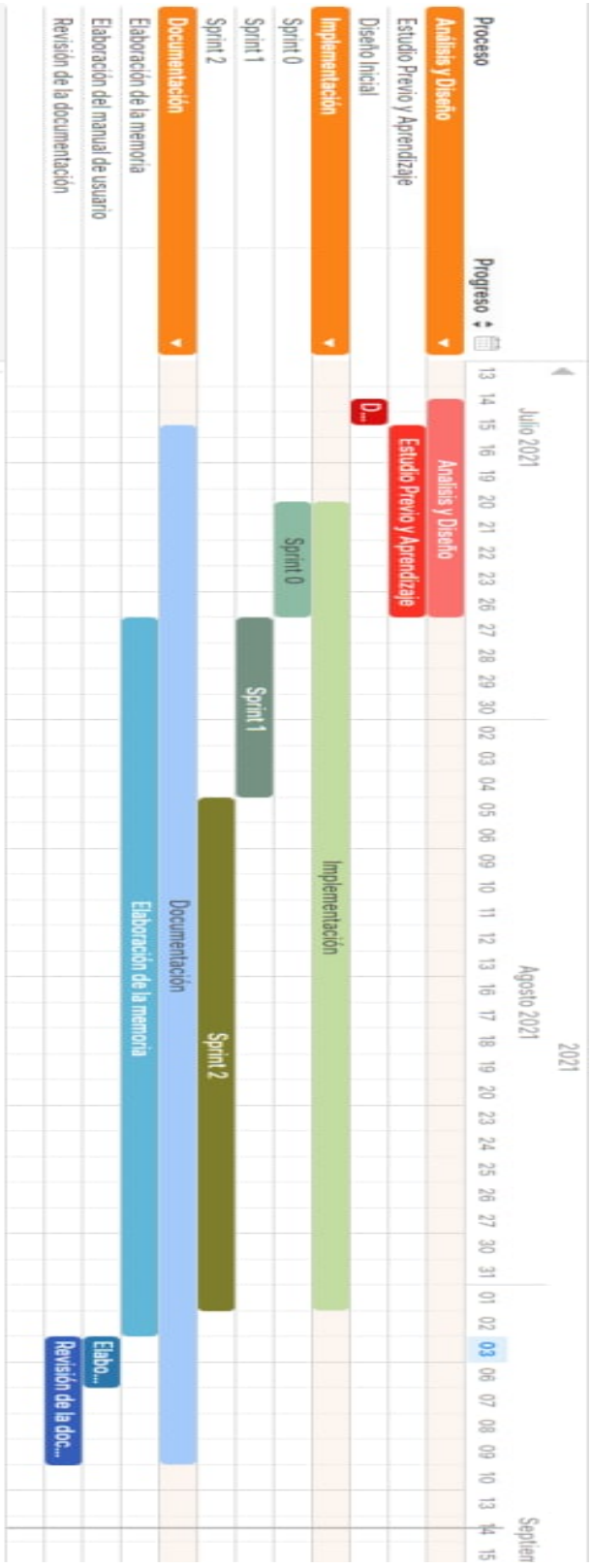

Ilustración 38. Diagrama de Gantt ampliado del proyecto

1.9. Presupuesto

En este apartado se determina y justifica el desembolso económico que asumirá el cliente para el desarrollo del proyecto. Para calcular dicho coste, se han tenido en cuenta tanto el tiempo de desarrollo como los costes a nivel de hardware, software y personal para hacer una estimación lo más fiel posible a la realidad.

1.9.1. Costes Hardware

Para la realización de este proyecto, se ha precisado un equipamiento informático, si bien se contempló la adquisición de un teléfono móvil, se descartó por el buen funcionamiento del emulador que incorpora Android Studio, por lo que sólo ha sido precisa la adquisición de un ordenador portátil.

El ordenador aquirido es un equipo portátil de la marca Lenovo, modelo IdeaPad 515-SK con un valor de 789€. Para este equipo se estima una vida útil de 5 años, por lo que, considerando ese valor y un período de 5 años, se estima un coste de 13.15€ por mes.

El coste total del Hardware es de 789€.

1.9.2. Costes Software

Para el desarrollo de esta app se han utilizado una serie de herramientas software tales como Android Studio, MockFlow, Tom's Planner o Visual Paradigm.

Para el uso de dichas aplicaciones, sólo Tom's Planner y Visual Paradigm han conllevado gastos, ya que tanto Android Studio como MockFlow son herramientas gratuitas. Sin embargo, Tom's Planner ha requerido una suscripción de 3 meses, ya que es la más corta que se puede adquirir, por un costo de 39$ (32,84€). Por último, se han precisado dos meses de suscripción a Visual Paradigm por valor de 70€ (35€ cada mes). Por lo que el coste total del Software asciende a 102,84€.

1.9.3. Costes de Personal

Para estos costes, se han tomado como referencia valores salariales de diferentes portales como PayScale **[27]**. En este caso, ante la diversidad de salarios que ofrecen estos portales, se ha hecho la estimación en función de un valor medio salarial, el cual ronda los 11/12€ por hora.

Sabiendo que el proyecto se ha realizado durante un periodo de 45 días, que descontando fines de semana suponen 33 días laborables con 8 horas de trabajo por día suponen un total de 264 horas.

Por tanto, el precio estimado del personal teniendo en cuenta el salario medio de un programador y las horas trabajadas es de 3168€.

1.9.4. Costes Adicionales

Para esos costes, se consideran principalmente el consumo eléctrico y de ADSL. Debido a la situación actual, no ha sido posible realizar reuniones presenciales por lo que no han ido necesarios desplazamientos, por lo que se omite ese gasto. Por otro lado, se ha trabajado en modo remoto desde casa, por lo que no se ha precisado el alquiler de un estudio u oficina para el desarrollo.

Para la estimación del costo eléctrico, se han utilizado una serie de herramientas web que han arrojado unas cifras de 1.40 kW de consumo y 0.24 kW de iluminación. Contando con que el desarrollo del proyecto ha tenido lugar durante el periodo estival, que cuenta con un número mayor de horas de luz, se ha considerado reducir la estimación de iluminación a la mitad.

Teniendo en cuenta las 264 horas estimadas de trabajo y el precio medio actual del kW/h que se sitúa en 0.22566€ se calcula el gasto de la siguiente manera

*Coste total = Tiempo en horas * kW consumidos*

*kW consumidos = Consumo en kW * Horas de trabajo*

1.4 + 0.12 = 1.52

*264 horas * 1.52 * 0.22566€ = 90.55€*

Este gasto, sumado a los 30€ de costo fijo por la contratación de la tarifa de potencia suponen un total de 120.55€, que añadiendo el impuesto eléctrico del 5,1127% que en este caso supone 4.63€ supone un total de 95.18€

Por último, teniendo en cuenta el coste de una tarifa ADSL por duración de 2 meses, se estiman unos 60€ adicionales en valor de tarifa ADSL.

En total los costes adicionales suponen 155.18€

1.9.5. Cálculo total de costes

Tipo de Coste	Coste Total
Coste Hardware	789€
Coste Software	102.84€
Coste de Personal	3168€
Costes Adicionales	155.18€
Total	4215.20€

Tabla 5. Desglose del presupuesto para el proyecto

2. Análisis

Esta es la primera etapa del proyecto, una vez elegidas las herramientas que se van a usar para el desarrollo del mismo, se procede a fijar una serie de objetivos. También se realizará un estudio sobre como cumplimentar debidamente los mismos con dichas herramientas.

2.1. Introducción

En primer lugar, se busca la manera de desarrollar una app que sea compatible con las últimas versiones de Android, aprovechando los recursos que estas ofrecen, esto permitirá alcanzar el mayor volumen de usuarios posible

Tras esto, se definen los requisitos que se busca que sean satisfechos por la aplicación, para ello, hay que ponerse en el lugar del usuario y tratar de pensar qué necesita el usuario cuando planea un viaje.

Por un lado, se busca ofrecer la mayor cantidad de información sobre el mismo de la manera más breve posible para no resultar muy pesado. Por otro lado, se busca que el usuario pueda acceder a ella de manera fácil e intuitiva, por lo que será también importante tener en cuenta una serie de objetivos a la hora de diseñar la interfaz de la aplicación.

Por último, se buscará que el acceso a la información sea fluido y sencillo, tratando de minimizar las interacciones del usuario con la interfaz y gestionando la información procedente de bases de datos o APIs de manera fluida.

2.2. Requisitos

Todo proyecto software ha debe contar con una serie de requisitos previamente definidos para el correcto funcionamiento y comportamiento del mismo. Para ésta aplicación se han estipulado una serie de requisitos concretos basados en otras aplicaciones similares a la aplicación objetivo y en la documentación de Android Studio.

2.2.1. Requisitos Funcionales

Según la ingeniería de requisitos, un requisito funcional es aquel que describe el comportamiento de un sistema o software cuando se cumplen una serie de condiciones **[17]**. En caso de esta aplicación se encuentran una serie de requisitos funcionales.

2.2.1.1. Pantalla de inicio:

En la pantalla de inicio se encuentran tres botones:

- Planear viaje: Permitirá acceder a la actividad de planear viaje
- Buscar clínicas: Permitirá acceder a la actividad de buscar clínicas

2.2.1.2. Formularios planear viaje y buscar clínicas:

En estas actividades se encuentran unos huecos donde se debe introducir el nombre de un país y un botón que permitirá acceder a la siguiente actividad. En caso de no estar debidamente rellenos dichos huecos, la aplicación no permitirá al usuario avanzar a la siguiente actividad.

2.2.1.3. Lista vuelos:

En esta actividad se pueden encontrar una lista con elementos seleccionables, cada uno con diferente información sobre un vuelo. Al pulsar cualquiera de ellos, se accede a la siguiente actividad. También cuenta con un botón deslizable Switch que permitirá alterar dicha lista en función de las preferencias del usuario.

2.2.1.4. Detalles del viaje:

En esta actividad se puede pulsar un botón que permite avanzar a la siguiente actividad

2.2.1.5. Mapas de ubicación de clínicas o viaje:

En estas actividades se puede navegar por el mapa y en el caso del mapa de ubicación de clínicas se podrá interactuar con los marcadores. Cuando estos se pulsan, se invoca una ventana con información sobre dicho marcador.

2.2.1.6. Menú:

Situado en la parte superior, permite el desplazamiento a una actividad u otra en función del icono que se pulse.

- Icono *pantalla de inicio*
- Icono *buscar clínicas*
- Icono *planear viaje*
- Icono *sobre mí*

2.2.1.7. Notificaciones:

Se mostrarán unas notificaciones temporales en la parte inferior de la pantalla cuando se trate de avanzar a la siguiente actividad sin haber rellenado previamente la información pertinente en los campos de texto.

2.2.2. Requisitos No Funcionales

Se tratan de las características generales y restricciones de la aplicación desarrollada **[18]**.

2.2.2.1. Seguridad:

Para usar la aplicación no es necesario incluir ningún tipo de dato personal al usarse en modo consulta, por lo que no habrá información sensible en riesgo.

2.2.2.2. Usabilidad:

El tiempo de aprendizaje de la aplicación será mínimo, la aplicación será responsiva y visualizable en multitud de dispositivos diferentes. Además cuenta con un correcto sistema de mensajes de error si estos se produjesen.

2.3. Bases de datos

Las bases de datos utilizadas han sido diseñadas en MySQl mediante la herramienta PHPMyAdmin de XAMPP. Para poder obtener la información de dichas bases de datos en nuestra aplicación, se ha utilizado la biblioteca Volley **[19]** que permite interactuar con éstas.

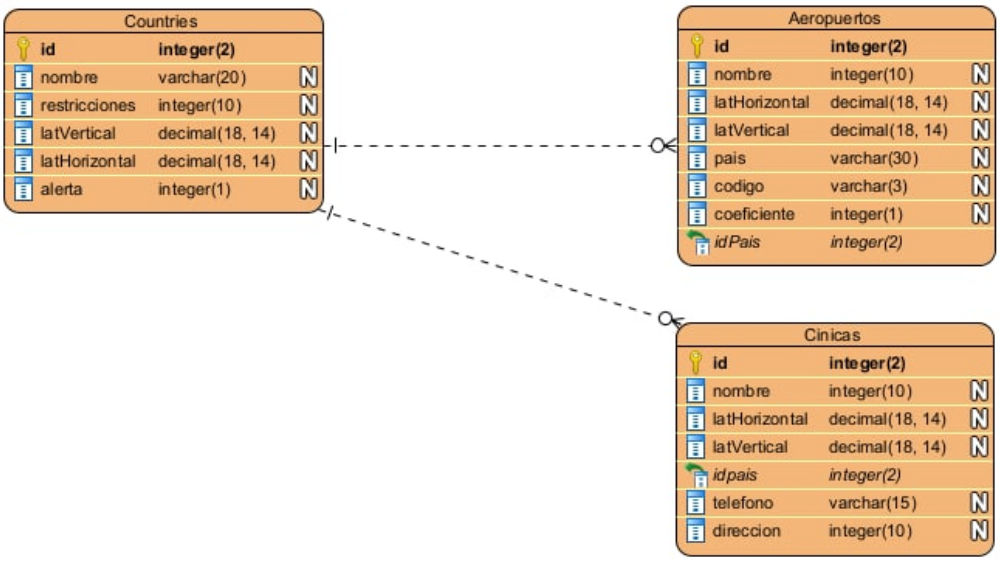

Ilustración 39. Estructura de la base de datos

2.4. Diseño Material

El Diseño Material **[20]** o Material Design es una biblioteca diseñada para Android Studio, ésta biblioteca permite utilizar elementos más interactuables y visuales como la lista de vuelos que ofrece la aplicación o el menú superior, está disponible para las últimas versiones de Android.

2.5. APIs

En este proyecto, se ha incluído la API de Google Maps, que permitirá visualizar un mapa dentro de una actividad e interactuar con el mismo pudiendo añadir marcadores y polilíneas a nivel de desarrollador o moverse por el mapa a nivel de usuario.

2.6. Iconos, fuentes y logo

Los íconos y las fuentes utilizadas son las ofrecidas por Google y Android Studio, por otra parte, el logo utilizado es un gráfico vectorial en formato .svg **[21].**

3. Ejecución del proyecto

En este apartado se procede a mostrar en detalle todos los *Sprints* que han supuesto el desarrollo del presente proyecto. Dichos *Sprints* han estado delimitados por una serie de reuniones realizadas entre el cliente, labor interpretada por los tutores y el desarrollador del proyecto, que ha ejercido del resto de roles.

Debido al período vacacional durante el mes de Agosto, los *Sprints* no tienen duraciones similares. La duración de los mismos se puede ver en los diagramas de Gantt del apartado *Planificación.*

3.1. Introducción.

El primer *Sprint* comienza con la reunión inicial, en ella se acuerdan las bases del proyecto. Durante esta etapa, se desarrollan las primeras actividades de la app siendo lo más fiel posible al mockup inicial.

La app entonces contaba con la actividad planear viaje como actividad principal, la actividad detalles del viaje y las dos actividades con mapas. Para esto, se había incluido ya la API de google maps para los mapas.

Tras la segunda reunión tiene lugar el segundo *Sprint*. En él se incluyen marcadores y polilíneas en los mapas y se crea la base de datos y sus tablas. También se hace la primera conexión a la base de datos desde la aplicación usando la biblioteca Volley y se crea un menú superior con un botón que permite el desplazamiento del usuario a la actividad principal.

Tras la tercera reunión, comienza el tercer y último *Sprint*. En él, se mejora definitivamente la obtención de información desde las bases de datos permitiendo hacer múltiples consultas a la vez. A su vez, se crea una vista principal en la que se incluyen animaciones, cambio los colores de la interfaz y la fuente de la aplicación.

También se incluye interactividad con los marcadores añadiendo ventanas de información nuevas. Se crean dos nuevas tablas en las bases de datos y una nueva actividad con una nueva lista con elementos interactuables. En concreto, la lista de los vuelos que podemos elegir en función de los países elegidos.

Se crean por último nuevas clases para hacer más sencillo el manejo de objetos y arreglo errores menores de parseo de datos etc…

3.2. Sprint 0

3.2.1. Mockup

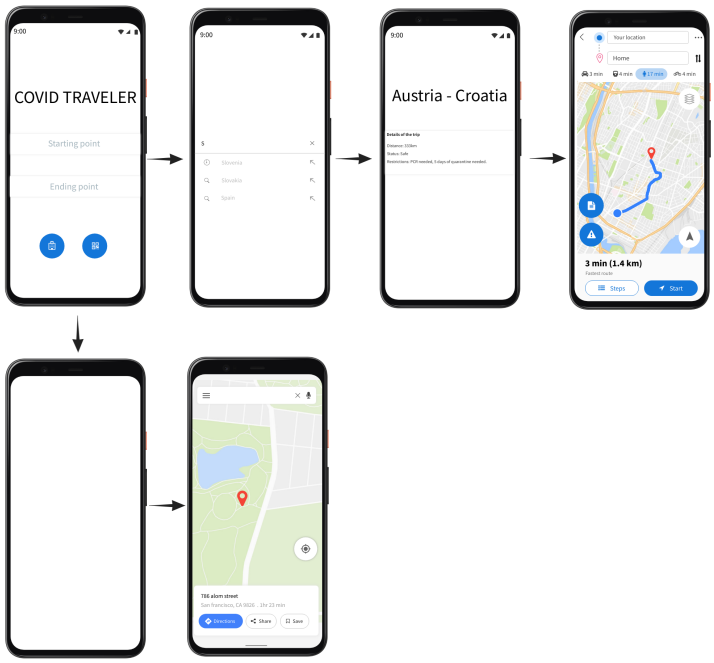

Ilustración 40. Mockup del Sprint 0

3.2.2. Diagrama de clases

En el siguiente diagrama se pueden apreciar las 4 clases que componen el proyecto en este Sprint

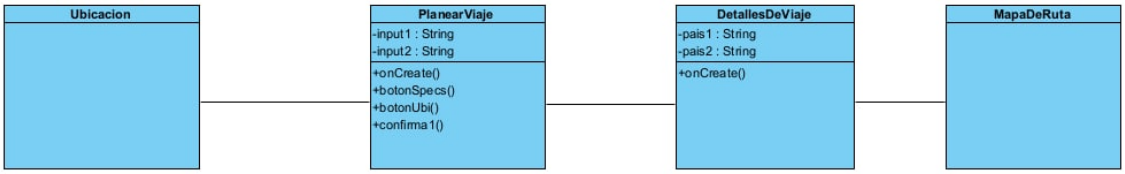

Ilustración 41. Diagrama de clases del Sprint 0

3.2.3. Implementación

3.2.3.1. Implementación cliente

En esta etapa, la implementación por parte del cliente comprenderá la creación de las primeras actividades, las cuales son *MainActivity(PlanearViaje)*, *Specs(Detalles del viaje)*, *MapsActivity(Mapa de ruta), Ubicación(Mapa de clínicas)*.

Al no haberse creado aún las bases de datos y, por tanto, no tener acceso a los datos de las mismas, la aplicación es sometida a un primer testeo con datos estáticos a modo de prueba, simulando datos reales. Por último, se cambia la fuente original de la aplicación a modo de prueba de cara a la versión final de la aplicación.

3.2.3.1.1. Planear Viaje

Ilustración 42. Interfaz de la vista Planear Viaje

En esta vista se añaden dos elementos de tipo *AutoCompleteTextView* llamados c1 y c2. Estos elementos permiten introducir una cadena de texto, proporcionándonos sugerencias en función del texto introducido.

```
String[] countries = getResources().getStringArray(R.array.countries);
c1 = (AutoCompleteTextView)findViewById(R.id.pais);
ArrayAdapter<String> adapter1 = new ArrayAdapter<>( context: this, android.R.layout.simple_dropdown_item_1line, countries);
c1.setAdapter(adapter1);
c2 = (AutoCompleteTextView)findViewById(R.id.city2);
ArrayAdapter<String> adapter2 = new ArrayAdapter<>( context: this, android.R.layout.simple_dropdown_item_1line, countries);
c2.setAdapter(adapter2);
```

Ilustración 43. Código Java para el elemento AutoCompleteTextView

Estas sugerencias surgen por cadenas de caracteres almacenadas en un vector dentro del archivo *strings.xml*.

```
<string-array name="countries">
    <item>Albania</item>
    <item>Alemania</item>
    <item>Andorra</item>
    <item>Austria</item>
    <item>Belgica</item>
    <item>Bielorrusia</item>
    <item>Bosnia y Herzegovina</item>
    <item>Bulgaria</item>
    <item>Ciudad del Vaticano</item>
```

Ilustración 44. Vector en el archivo Strings.xml

En este punto del desarrollo, era necesario pulsar cada uno de los botones situados a la derecha de c1 y c2. Estos botones tendrían asignadas las funciones confirma1 y confirma2 que almacenarían el resultado de c1 y c2 en dos variables de tipo *String* input1 e input2 respectivamente con el nombre del país elegido.

```
public void confirma1(View view){
    input1 = c1.getText().toString();
}
```

Ilustración 45. Código del método Confirma1 en el Sprint 0

Por último, existían 2 botones dentro de la vista que permitían pasar a otra actividad, ya fuese la actividad Detalles del viaje o Clínicas en mi zona. Cada uno de ellos llamaba a un método específico en el que se inicializaba un objeto de tipo *Intent* pasándole como parámetro el nombre de la actividad a la que se desea ir.

Este tipo de objeto permite añadir variables que pasarán de una actividad a otra, por último, se llamará al método *StartActivity()* pasándole por cabecera el objeto de tipo *Intent* y esto hará que nos movamos a la siguiente actividad.

Esta acción sólo se llevará a cabo en caso de haberse rellenado ambos campos de texto, en caso de que no, la aplicación mostrará una notificación de tipo *Toast* indicando que es necesario rellenar un campo y se especificará cual.

```
public void botonSpecs(View view){
    if(!input1.isEmpty() && !input2.isEmpty() ){
        Intent next = new Intent( packageContext: this, Specs.class);
        next.putExtra( name: "Pais1", input1);
        next.putExtra( name: "Pais2", input2);
        startActivity(next);
    }else if(input1.isEmpty()){
        Toast.makeText(getApplicationContext(), text: "Confirma pais de origen",Toast.LENGTH_SHORT).show();
    }else{
        Toast.makeText(getApplicationContext(), text: "Confirma pais de destino",Toast.LENGTH_SHORT).show();
    }
}
```

Ilustración 46. Código del método botonSpecs()

3.2.3.1.2. Detalles del viaje

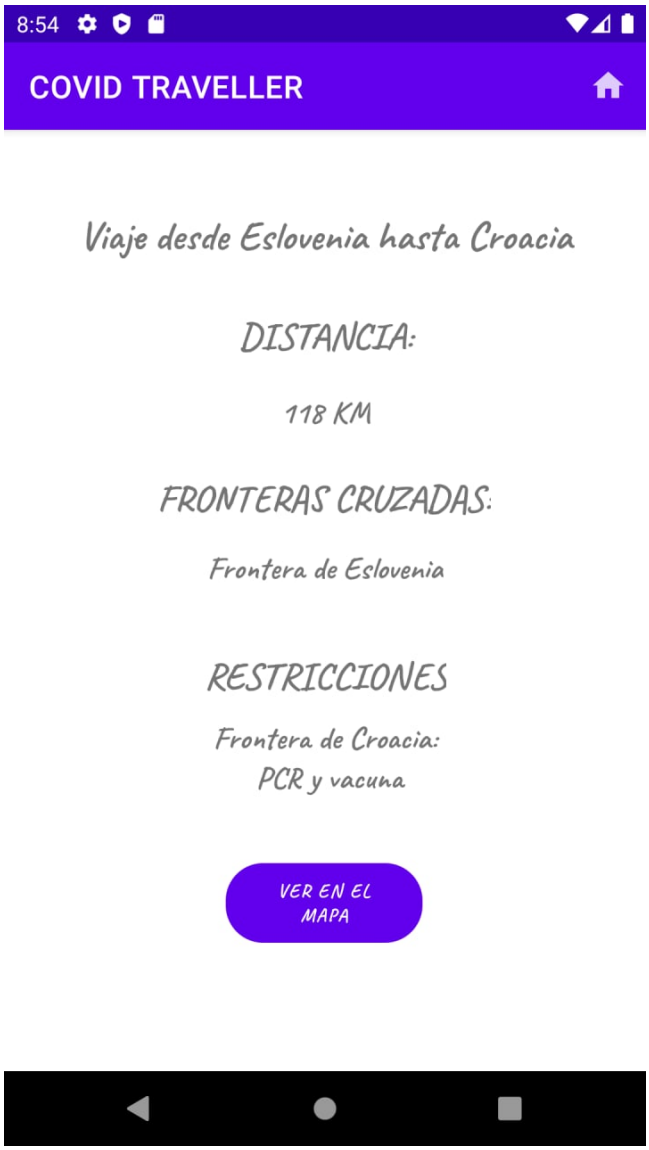

Ilustración 47. Interfaz de la actividad Detalles del Viaje

Esta actividad cuenta con una serie de cadenas de texto, estas cadenas se rellenarán de forma estática en algunos casos mientras que en otros variarán en función de los datos de los países seleccionados. En esta etapa del proyecto, sólo los nombres de los países variaban dinámicamente.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_specs);
    ListElement element = (ListElement) getIntent().getSerializableExtra( name: "ListElement");
    pais1 = (String) getIntent().getSerializableExtra( name: "Pais1");
    pais2 = (String) getIntent().getSerializableExtra( name: "Pais2");

    trip = findViewById(R.id.pregunta);
    borders = findViewById(R.id.borders);
    trip.setText("Viaje desde " + pais1 + " hasta " + pais2);
}
```

Ilustración 48. Código del método onCreate de la actividad Detalles del Viaje

3.2.3.1.3. Mapa de ruta y Ubicación

En esta etapa del proyecto estas dos actividades sólo mostrarán un mapa por el que nos podremos desplazar libremente pero que no muestra ninguna información adicional.

3.2.3.2. Implementación servidor

La única implementación por parte del servidor que se realiza en este apartado, es la solicitud y creación de una clave de API de Google Maps para acceder a los datos de dicho servidor con objetivo de crear diferentes elementos dentro del mapa, tales como marcadores, polilíneas… etc, los cuales serán implementados más adelante.

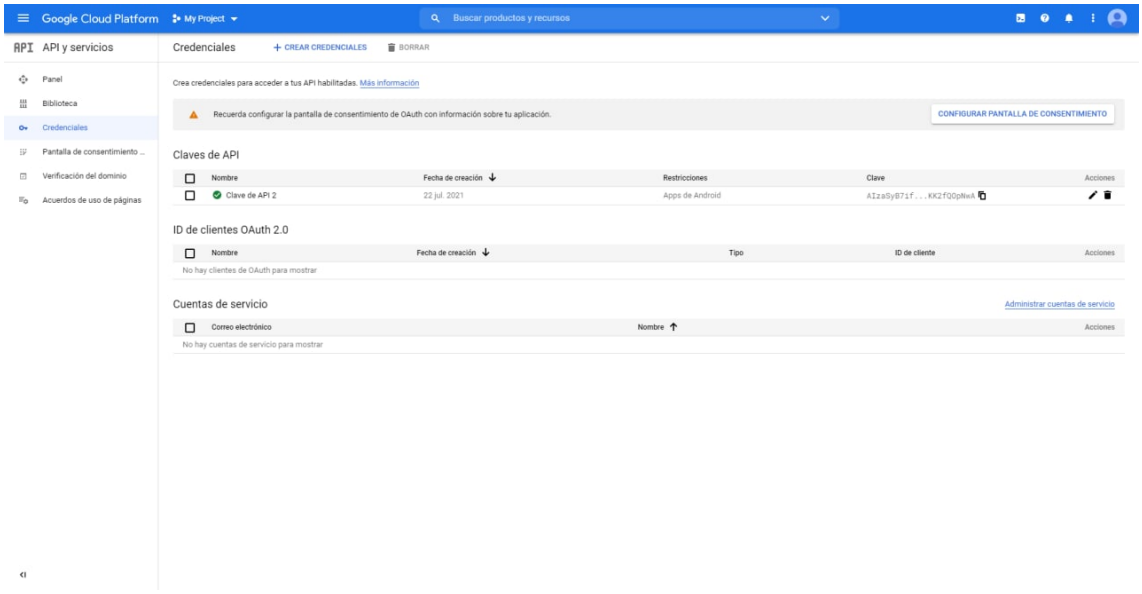

Ilustración 49. Repositorio de APIs de Google

3.3. Sprint 1

3.3.1. Mockup

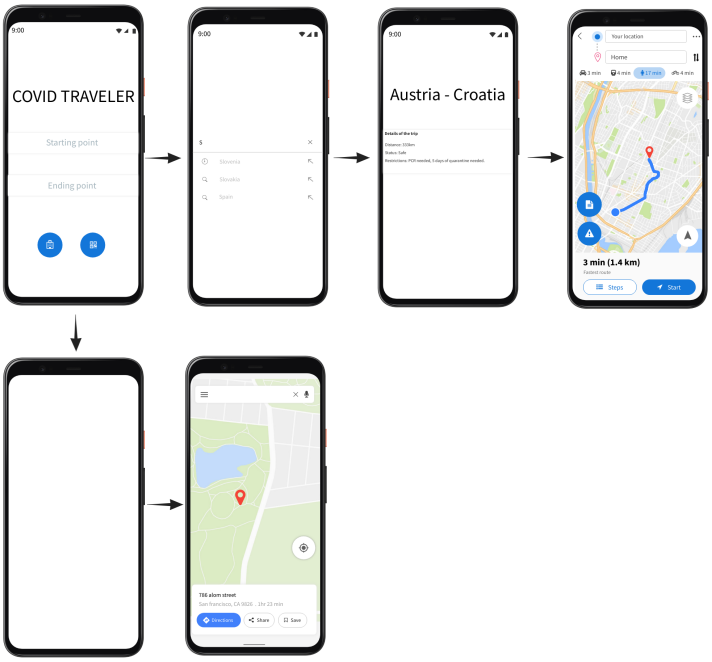

Ilustración 50. Mockup del Sprint 1

3.3.2. Diagrama de clases

En este diagrama se puede apreciar la adición de nuevos métodos y de la clase país. En este *Sprint* se añade una conexión de la clase PlanearViaje a la base de datos

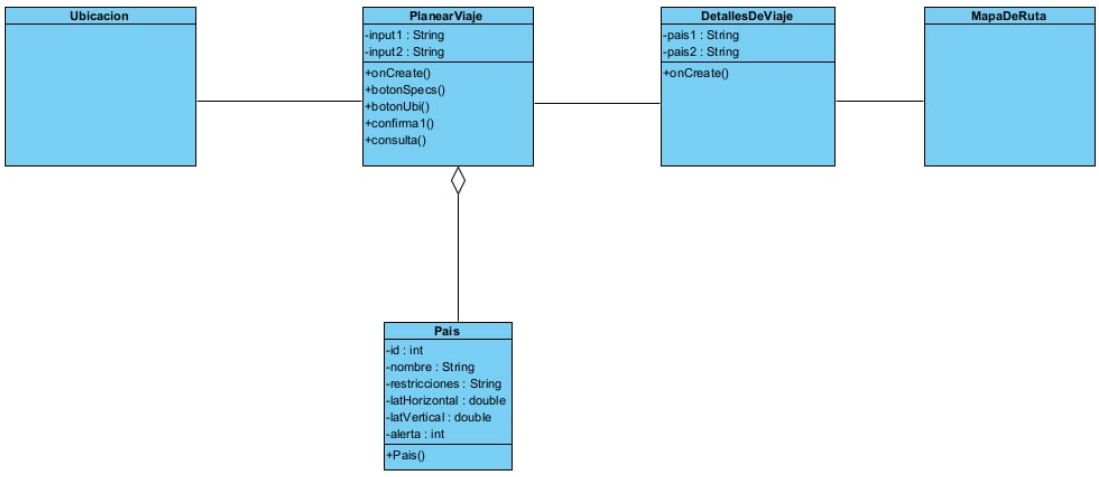

Ilustración 51. Diagrama de clases del Sprint 1

3.3.3. Implementación

3.3.3.1. Implementación cliente

En esta etapa, el desarrollo se centra en la parte del servidor, por lo que la parte del cliente no cambia mucho.

Se implementan las polilíneas y marcadores añadidos a las actividades de los mapas y un menú en la parte superior con un ícono que al ser pulsado redirige al usuario hasta la actividad principal.

3.3.3.1.1. Implementación del menú

Por un lado se implementa un nuevo archivo menu.xml donde se incluye un elemento de tipo menú con un solo ítem en este caso, al cual se le asigna el icono de una casa.

Este elemento permitirá añadir tantos ítems como se desee y estos aparecerán en el mismo orden en el que se añaden al archivo menú.xml. En este caso, al solo haber un elemento, no es apreciable pero, en la siguiente etapa se podrá ver como se han añadido nuevos elementos que serán situados a la derecha del ya existente.

Por último, los elementos del menú no interfieren con el nombre de la vista que está situado en la parte izquierda de la barra superior.

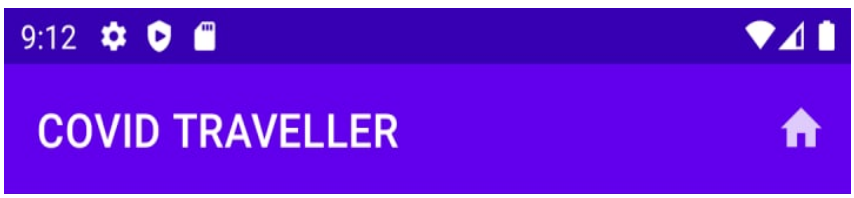

Ilustración 52. Menú en la parte superior en el Sprint 1

También se le asigna un color con *android:iconTint* y se utiliza *app:showAsAction* con el valor "*always*" para indicar que el icono aparecerá permanentemente en el menú. La etiqueta *app:showAsAction* permite también hacer un menú desplegable en función del valor que se le asigne.

```
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/home"
            android:icon="@drawable/home_icono"
            android:title="IR A INICIO"
            android:iconTint="@color/white"
            app:showAsAction="always"/>

    </group>
</menu>
```

Ilustración 53. Código de un item del menú

Por otro lado, se han creado dos métodos en las clases donde se deseaba incluir dicho menú. El primer método consiste en la creación de un objeto de tipo *MenuInflater* y mediante el método *inflate()* se vincula el archivo menu.xml en este caso mediante su identificador.

```
public boolean onCreateOptionsMenu(Menu menu){
    getMenuInflater().inflate(R.menu.menu, menu);
    return true;
}
```

Ilustración 54. Código del método onCreateOptionsMenu

En un segundo método, se identifica el ítem del menú mediante el identificador y se realiza un procedimiento similar a los botones vistos previamente. Se crea un objeto de tipo I*ntent* con el nombre de la clase deseada y se cambia a dicha actividad.

```
public boolean onOptionsItemSelected(MenuItem item){
    int id= item.getItemId();

    if(id == R.id.home){
        Intent next = new Intent( packageContext: this, Seleccion.class);
        startActivity(next);
        return true;
    }return super.onOptionsItemSelected(item);
}
```

Ilustración 55. Código del método onOptionsItemSelected

3.3.3.1.2. Implementación de marcadores y polilíneas

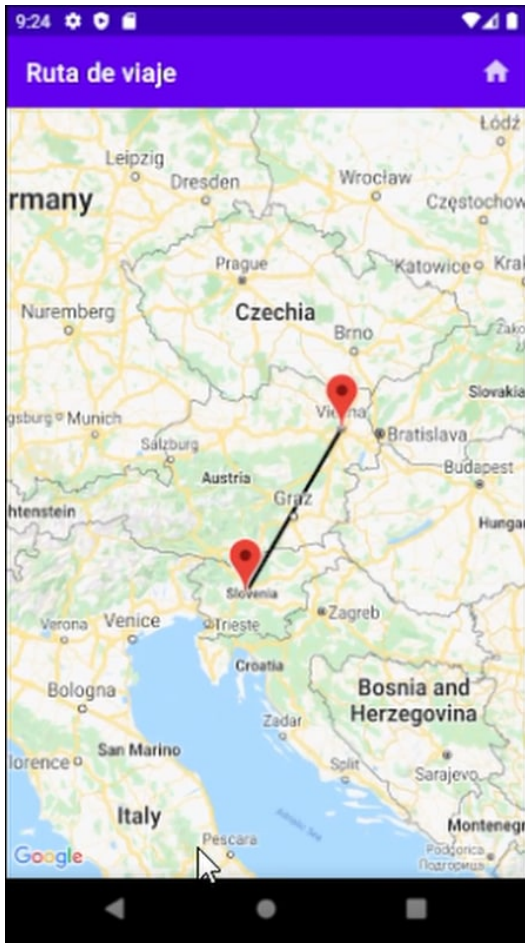

Ilustración 56. Interfaz de la actividad ruta de viaje con marcadores y polilíneas en el Sprint 1

En primer lugar, se han usado las coordenadas de los puntos de origen y destino para establecer los puntos de latitud y longitud de cada uno de los marcadores. Se han creado dos objetos de la clase *LatLng* (uno por coordenada) de tipo final, ya que no se desea que sean modificados en ningún momento.

```
final LatLng aeropuerto1 = new LatLng(pais1.getLatVertical(), pais2.getLatHorizontal());
final LatLng aeropuerto2 = new LatLng(pais1.getLatVertical(), pais2.getLatHorizontal());
```

Ilustración 57. Declaración de las variables aeropuerto1 y aeropuerto2

Tras esto, se añaden los marcadores indicando como coordenadas el objeto de tipo *LatLng* correspondiente y como *tag* o etiqueta el nombre del país correspondiente.

```
mMap.addMarker(new MarkerOptions().position(aeropuerto1)
        .title(pais1.getNombre()));

mMap.addMarker(new MarkerOptions().position(aeropuerto1)
        .title(pais2.getNombre()));
```

Ilustración 58. Código para la creación de marcadores

Tras esto, se crea un objeto de la clase *Polyline* que añade una *polyline* o polilínea entre ambas coordenadas. Para ello, se establece el inicio de la polilínea en el punto de coordenadas del país de origen y el final de la misma en el punto de coordenadas del país de destino. Se añade también una etiqueta que indicará el nombre de la polilínea.

```
Polyline polyline1 = googleMap.addPolyline(new PolylineOptions()
        .clickable(true)
        .add(
                aeropuerto1,
                aeropuerto2)
        );
polyline1.setTag("Ruta mas corta");
```

Ilustración 59. Código para la creación de una polilínea

Por último, se situará posición inicial de la cámara desde la que el usuario podrá visualizar el mapa. Para ello, se crea un objeto de la clase *CameraPosition* al que se le establecen las coordenadas del país de origen para que estas salgan en el centro de la cámara y un nivel de zoom.

```
CameraPosition cameraPosition = new CameraPosition.Builder()
        .target(aeropuerto1)
        .zoom(5)
        .build();
mMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));
```

Ilustración 60. Código de creación y parametrización de la cámara para el mapa

Aun así, el usuario podrá desplazarse por todo el mapa haciendo el gesto de arrastrar y podrá hacer zoom tanto positivo como negativo haciendo el gesto de pinzas, aunque el mismo mapa implementa teclas con los símbolos “+” y “-” para hacer zoom.

3.3.3.2. Implementación servidor

Como se mencionaba anteriormente, para el desarrollo de la aplicación se ha usado una base de datos remota. Ya que era mucho más fácil de gestionar a la hora de hacer inserciones que una base de datos local en el dispositivo con *SQLite*. Por tanto, la arquitectura de la aplicación será una arquitectura cliente servidor.

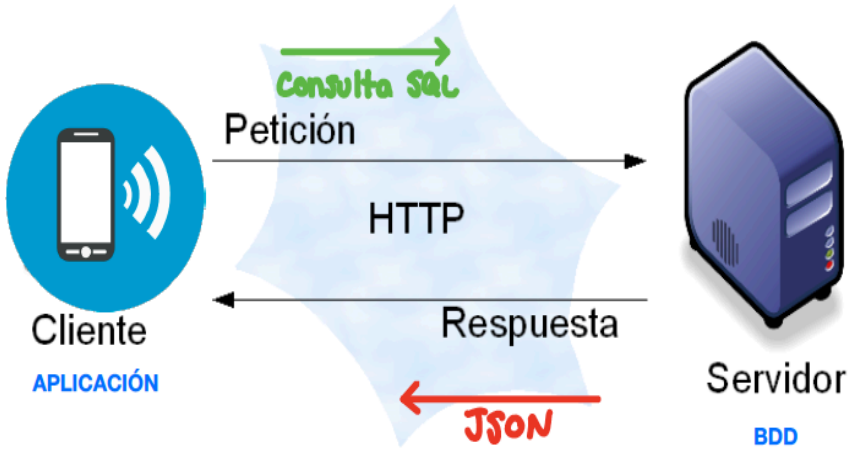

Ilustración 61. Esquema de la arquitectura Cliente – Servidor

En esta aplicación, se realizarán una serie de peticiones http por parte del cliente. Estas ocurrirán generalmente cuando se introduzca una información o un dato en la aplicación. Entonces, utilizando ese dato, se realizarán las peticiones en forma de consulta a la base de datos gracias a la biblioteca *Volley*.

Generalmente, para cada consulta se desea obtener información de una o varias filas de una tabla concreta de la base de datos. Una vez obtenidos esos datos, estos se alojan en un objeto de un tipo correspondiente a la tabla consultada. En el siguiente ejemplo, se puede ver el constructor de la clase País junto a la estructura de la tabla *countries,* correspondiente a los diferentes países de la app.

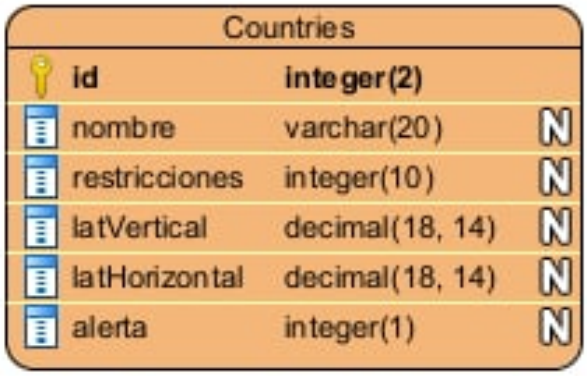

Ilustración 62. Tabla countries

```
public Pais(int id, double latVertical, double latHorizontal,
            String restricciones, String nombre, int alerta) {
    this.id = id;
    this.latVertical = latVertical;
    this.latHorizontal = latHorizontal;
    this.restricciones = restricciones;
    this.nombre = nombre;
    this.alerta = alerta;
}
```

Ilustración 63. Código del constructor de la clase Pais

Para las consultas, se crea una función que recibirá por cabecera una URL. Esta URL estará formada por el protocolo http, la dirección IP del dispositivo donde se alojan los archivos PHP que albergarán el código SQL de las consultas y la ruta del mismo dentro del dispositivo.

A estos elementos les siguen los carácteres “?” y “=” antes y después del atributo que se desea filtrar en la base de datos. Por último, el valor que se desea buscar irá concatenado al final de la cadena. Por último, se pasará también por la cabecera el objeto mencionado previamente correspondiente al tipo de consulta.

En el siguiente ejemplo, vemos como el link está formado por la dirección IP de mi equipo, se especifica el archivo *consultabien.php* alojado en el directorio */dev*.

Se especifica también que se quiere filtrar el atributo nombre de la tabla y se busca el valor correspondiente al valor asignado a la variable input1 en dicha columna. Finalmente, se pasa un objeto de tipo Pais creado previamente con valores por defecto porque se está haciendo una consulta a la tabla *countries*.

```
consulta( URL: "http://192.168.0.43/dev/consultabien.php?nombre="+input1+"",pais1);
```

Ilustración 64. Código de llamada al método consulta()

En este ejemplo podemos apreciar que las funciones confirma1 y confirma2 pasan a no solo almacenar un valor dentro de las variables input1 e input2 sino también a llamar a funciones de consulta.

```
public void confirma1(){
    input1 = c1.getText().toString();
    if(!input1.isEmpty()){
        consulta( URL: "http://192.168.0.43/dev/consultabien.php?nombre="+input1+"",pais1);
    }else {
        Toast.makeText(getApplicationContext(), text: "Introduce pais de origen",Toast.LENGTH_SHORT).show();
    }
}
```

Ilustración 65. Código del método confirma1() en el Sprint 1

El proceso es similar a una consulta a la base de datos dentro del navegador, aunque en caso de la aplicación, se ha de cambiar la dirección l*ocalhost* por la IP del dispositivo donde están alojados los archivos PHP.

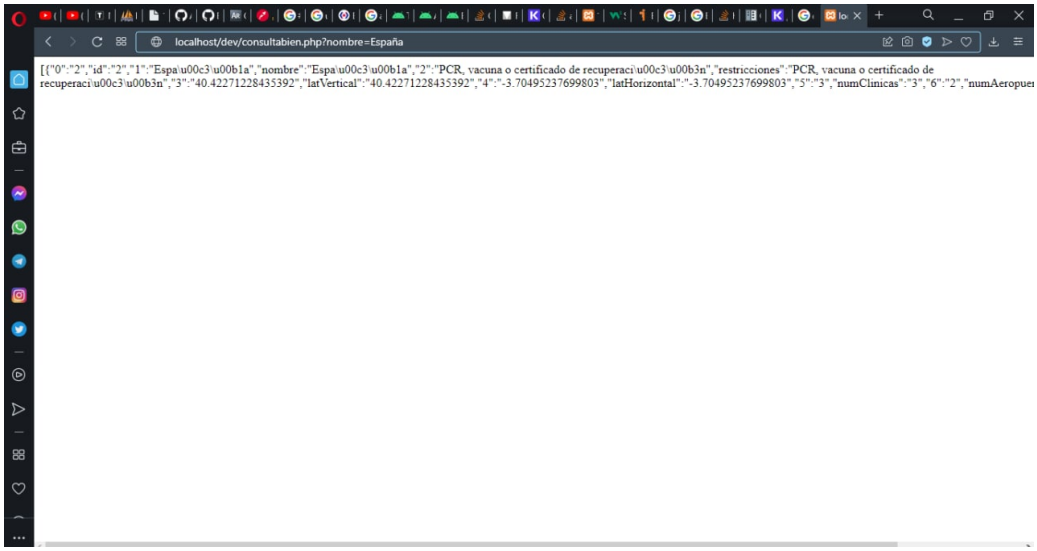

Ilustración 66. Consulta a la base de datos desde el navegador

En la siguiente imagen, se puede apreciar el código del archivo PHP del ejemplo anterior. En él se especifican las credenciales de la base de datos y se escribe el código de la consulta que se desea realizar. La columna filtrada en esta columna ha de coincidir con la que aparece en al URL de la función de consulta.

```php
<?php
$hostname='localhost';
$username='root';
$password='';
$database='tfgl';
$conexion= new mysqli($hostname,$username,$password,$database);
$nombre=$_GET['nombre'];
$consulta = "select * from countries where nombre= '$nombre'";
$resultado = $conexion -> query($consulta);

while($fila=$resultado -> fetch_array()){
    $country[] = array_map('utf8_encode', $fila);
}
echo json_encode($country);
$resultado -> close();

?>
```

Ilustración 67. Código del archivo consultabien.php

Por último, dentro de la función de consulta, podemos apreciar que se hace una petición al servidor con la URL que hemos pasado por cabecera. Tras esto se captará una respuesta o response del servidor, esta será un array de objetos de tipo JSON que contendrán la información de todas las filas coincidentes con los criterios de la consulta.

La función determinará la longitud del vector y asignará uno a uno los valores de la fila a cada uno de los atributos del objeto que se pasa por cabecera mediante métodos *setters*. En caso de ser varios resultados y sólo tener un objeto, se asignarán los valores del último resultado, aunque, al tratarse en este caso de países y filtrarse por nombre, no se da dicha situación porque cada país tiene un nombre distinto.

```
private void consulta(String URL, Pais p){
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(URL, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            JSONObject jsonObject = null;
            for (int i = 0; i < response.length(); i++) {
                try {
                    jsonObject = response.getJSONObject(i);
                    p.setId(jsonObject.getInt( name: "id"));
                    p.setNombre(codificar(jsonObject.getString( name: "nombre")));
                    p.setRestricciones(codificar(jsonObject.getString( name: "restricciones")));
                    p.setLatVertical(jsonObject.getDouble( name: "latVertical"));
                    p.setLatHorizontal(jsonObject.getDouble( name: "latHorizontal"));
                    p.setAlerta(jsonObject.getInt( name: "alerta"));
                    Toast.makeText(getApplicationContext(), text: "Pais: "+ p.getNombre(),Toast.LENGTH_SHORT).show();
                } catch (JSONException e) {
                    Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHORT).show();
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast.makeText(getApplicationContext(), error.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
    );
    requestQueue = Volley.newRequestQueue( context: this);
    requestQueue.add(jsonArrayRequest);
}
```

Ilustración 68. Código del método consulta()

3.4. Sprint 2

3.4.1. Mockup

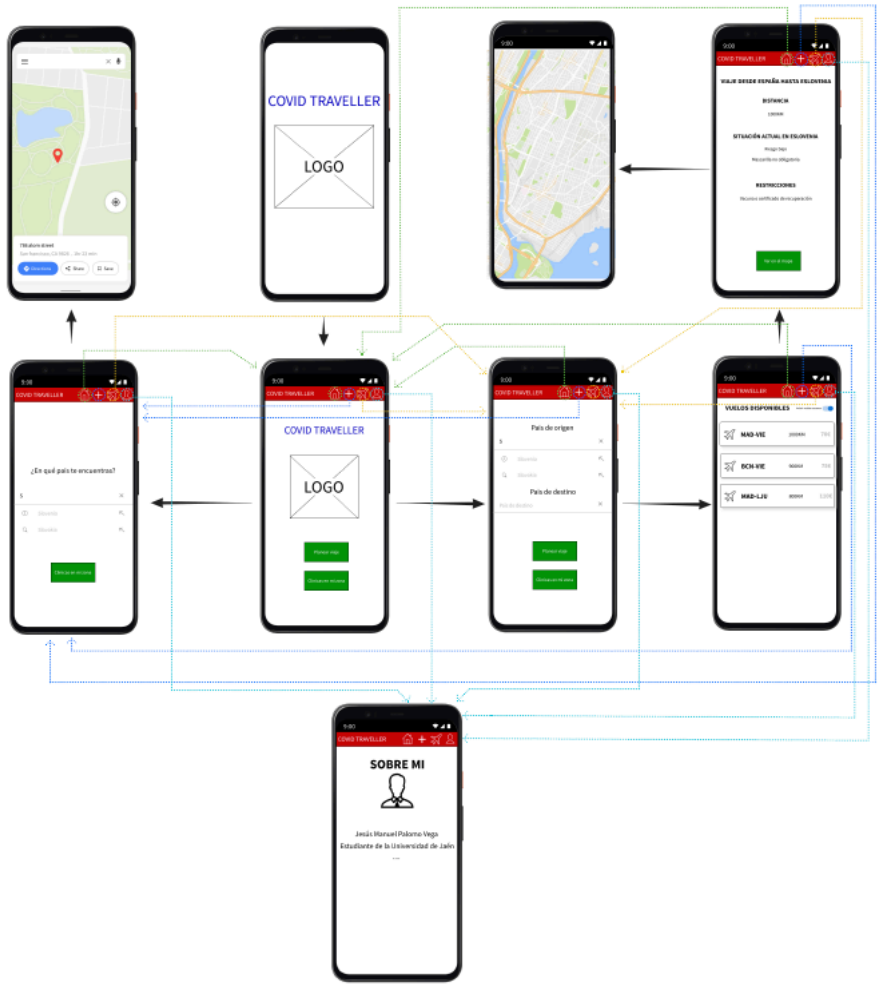

Ilustración 69. Mockup del Sprint 2

3.4.2. Diagrama de clases

Este diagrama representará la estructura de la aplicación en el último Sprint. Las clases ClinicasEnMiZona, PlanearViaje y Listavuelos estarán conectadas a la base de datos.

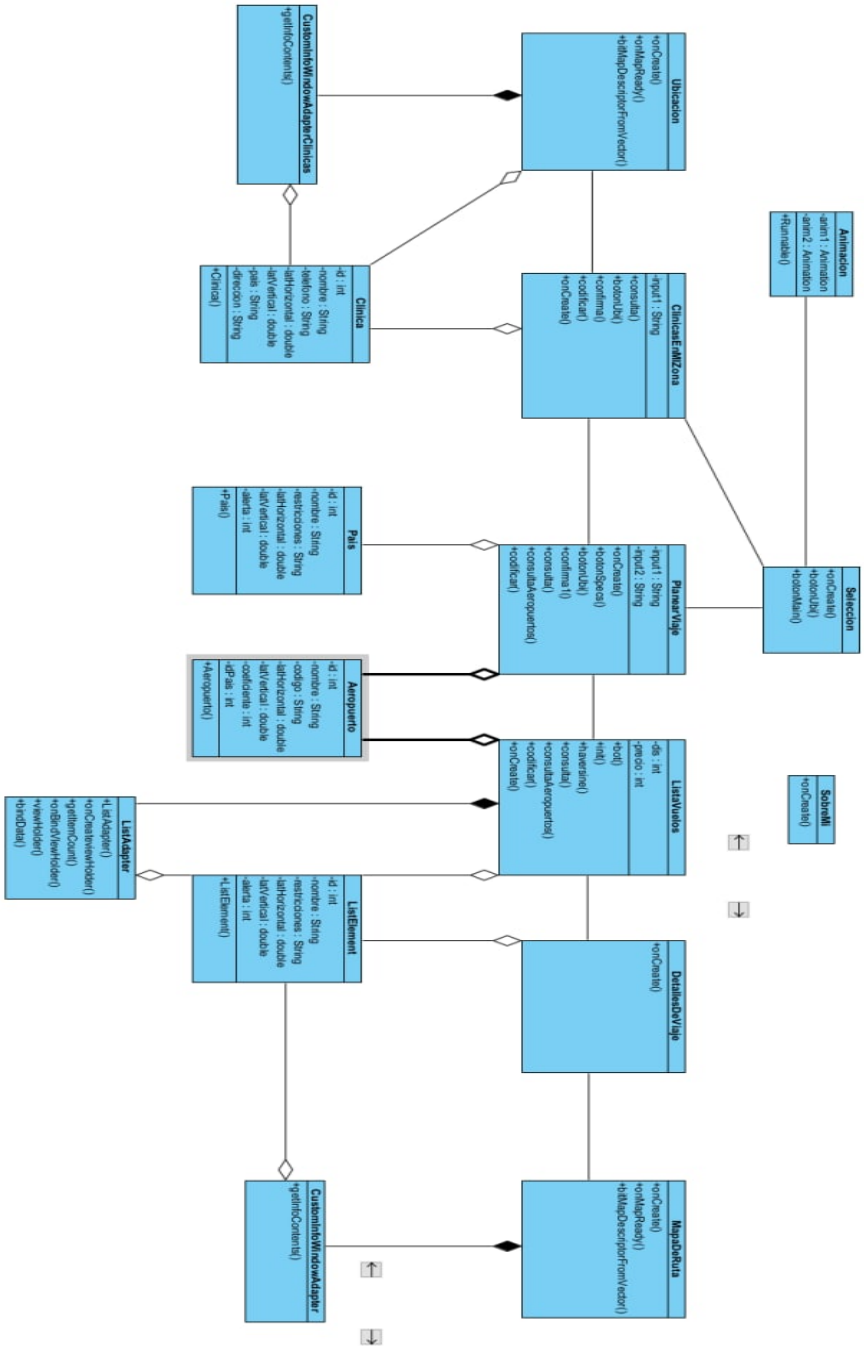

Ilustración 70. Diagrama de clases del Sprint 2

3.4.3. Implementación

3.4.3.1. Implementación cliente

En esta etapa, se implementan nuevas actividades para hacer a la aplicación más atractiva a nivel visual, a la vez que se cambian valores como los colores de la misma, y se añaden nuevas imágenes.

Por otro lado, se añaden nuevas clases correspondientes a las nuevas tablas creadas en las bases de datos con el fin de facilitar el tratamiento de los datos volcados de las mismas.

Por último, se añaden nuevos métodos a las actividades para facilitar el tratamiento o la muestra de datos.

3.4.3.1.1. Cambio de colores de la interfaz

En primer lugar, se ubica el archivo colors.xml dentro de la carpeta values la cual contiene archivos como strings.xml entre otros. Tras esto, se procede a añadir dentro de un vector los colores deseados. Cabe destacar que, sólo en este caso, Android admitirá códigos de colores RGB.

```
<resources>
    <color name="texto">#092BE4</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="primary_color">#C12A04</color>
    <color name="primary_dark">#992204</color>
    <color name="primary_light">#FFCDD2</color>
    <color name="accent">#046538</color>
    <color name="primary_text">#212121</color>
    <color name="secondary_text">#757575</color>
    <color name="icons">#FFFFFF</color>
    <color name="divider">#BDBDBD</color>
</resources>
```

Ilustración 71. Colores declarados en el archivo colors.xml

Tras esto, se accede al archivo *themes.xml* el cual se encuentra dentro de este mismo directorio, pero en un directorio particular aparte. En este directorio, se asigna por nombre los colores previamente declarados en el archivo *colors.xml*.

```
<resources xmlns:tools="http://schemas.android.com/tools">
    <!-- Base application theme. -->
    <style name="Theme.Prueba" parent="Theme.MaterialComponents.DayNight.DarkActionBar">
        <!-- Primary brand color. -->
        <item name="colorPrimary">@color/primary_color</item>
        <item name="colorPrimaryVariant">@color/accent</item>
        <item name="colorOnPrimary">@color/white</item>
        <!-- Secondary brand color. -->
        <item name="colorSecondary">@color/teal_200</item>
        <item name="colorSecondaryVariant">@color/teal_700</item>
        <item name="colorOnSecondary">@color/black</item>
        <!-- Status bar color. -->
        <item name="android:statusBarColor" tools:targetApi="l">?attr/colorPrimary</item>
        <!-- Customize your theme here. -->
        <item name="colorButtonNormal">@color/accent</item>
    </style>
</resources>
```

Ilustración 72. Código del archivo themes.xml

3.4.3.1.2. Animación de inicio

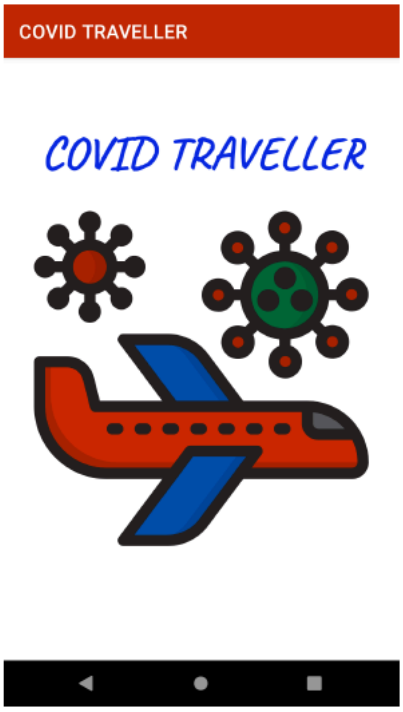

Ilustración 73. Interfaz de la actividad Animacion1

En primer lugar, se añaden los elementos al archivo de interfaz, tales como las imágenes y los botones. Como esta actividad pasará a ser la actividad que se creará y mostrará al iniciar la actividad, se ha de modificar el archivo *AndroidManifest.xml* especificando a esta actividad como la nueva actividad principal.

```
<activity android:name=".Animacion1">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Ilustración 74. Código del método confirma1() en el Sprint 2

Tras esto, gracias a la librería *AnimationUtils*, se pueden programar dos animaciones en este caso, que moverán con fluidez elementos de la interfaz de manera descendente o ascendente. Estas animaciones estarán programadas en los archivos *desplazamiento_arriba.xml* y *desplazamiento_abajo.xml*.

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:fromXDelta="0%"
        android:fromYDelta="-30%"
        android:duration="2000"/>
    <alpha
        android:fromAlpha="0.1"
        android:toAlpha="1.0"
        android:duration="2000"
        />
</set>
```

Ilustración 75. Código del archivo desplazamiento_arriba.xml

```
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <translate
        android:fromXDelta="0%"
        android:fromYDelta="100%"
        android:duration="2000"/>
    <alpha
        android:fromAlpha="0.1"
        android:toAlpha="1.0"
        android:duration="2000"
        />
</set>
```

Ilustración 76. Código del archivo desplazamiento_abajo.xml

Dichas animaciones se asignarán a las variables de tipo *Animation anim1* y *anim2* de la clase java, las cuales, a su vez, serán asignadas a los elementos que se desea animar, en este caso el logo y el título de la app.

```
Animation anim1 = AnimationUtils.loadAnimation( context: this, R.anim.desplazamiento_arriba);
Animation anim2 = AnimationUtils.loadAnimation( context: this, R.anim.desplazamiento_abajo);
```

Ilustración 77. Código de declaración de las animaciones anim1 y anim2

```
TextView titulo = (TextView)findViewById(R.id.tituloApp);
ImageView logo = (ImageView) findViewById(R.id.imagenLogo);

titulo.setAnimation(anim2);
logo.setAnimation(anim1);
```

Ilustración 78. Asignación de las animaciones anim1 y anim2

Tras esto, se utilizará el método de la siguiente imagen, el cual ejecutará las animaciones. Para esto, se utilizará un vector de *Pair,* el cual almacenará cada uno de los elementos que se desea animar junto a un *tag*.

Tras esto, se comprueba que la versión del sistema operativo es superior a *Lollipop* [ver tabla de versiones] y en caso afirmativo se llevará a cabo la animación. Al terminar dicha animación se abrirá la siguiente actividad designada mediante un objeto de tipo *Intent*, método que ya se vio anteriormente. Por último, se establecerá el tiempo de duración en milisegundos, en este caso 4000 milisegundos lo que supondría una animación de 4 segundos.

```
new Handler().postDelayed(new Runnable() {
    @Override
    public void run() {
        Intent next = new Intent( packageContext: Animacion1.this, Seleccion.class);

        Pair[] pairs = new Pair[2];
        pairs[0] = new Pair<View, String>(logo, "logoTrans");
        pairs[1] = new Pair<View, String>(titulo, "textTrans");

        if(Build.VERSION.SDK_INT>=Build.VERSION_CODES.LOLLIPOP){
            ActivityOptions options = ActivityOptions.makeSceneTransitionAnimation( activity: Animacion1.this, pairs);
            startActivity(next, options.toBundle());
        }else{
            startActivity(next);
            finish();
        }
    }
}, delayMillis: 4000);
```

Ilustración 79. Código de programación de las animaciones

3.4.3.1.3. Elementos añadidos al menú

Ilustración 80. Menú de la parte superior en el Sprint 2

Se modifican tanto el código del archivo *menú.xml*, en el que se añaden 3 nuevos *ítems.*

```
<item
    android:id="@+id/clinicas"
    android:icon="@drawable/clinica_icono"
    android:title="CLÍNICAS EN MI ZONA"
    android:iconTint="@color/white"
    app:showAsAction="always"/>
<item
    android:id="@+id/planearviaje"
    android:icon="@drawable/ic_travel_explore_white_24dp"
    android:title="PLANEAR UN VIAJE"
    android:iconTint="@color/white"
    app:showAsAction="always"/>
<item
    android:id="@+id/about"
    android:icon="@drawable/persona"
    android:title="SOBRE EL AUTOR"
    android:iconTint="@color/white"
    app:showAsAction="always"/>
```

Ilustración 81. Items añadidos al menu

A su vez, se modifica el método implementado en las actividades para la visualización del mismo

```
public boolean onOptionsItemSelected(MenuItem item){
    int id= item.getItemId();

    if(id == R.id.home){
        Intent next = new Intent( packageContext: this, Seleccion.class);
        startActivity(next);
        return true;
    }else if(id == R.id.clinicas){
        Intent next = new Intent( packageContext: this, CompruebaUbicacion.class);
        startActivity(next);
        return true;
    }else if(id == R.id.planearviaje){
        Intent next = new Intent( packageContext: this, MainActivity.class);
        startActivity(next);
        return true;
    }else if(id == R.id.about){
        Intent next = new Intent( packageContext: this, SobreMi.class);
        startActivity(next);
        return true;
    }return super.onOptionsItemSelected(item);
}
```

Ilustración 82. Código del método para la creación del menu en el Sprint 2

3.4.3.1.4. Actividad de selección

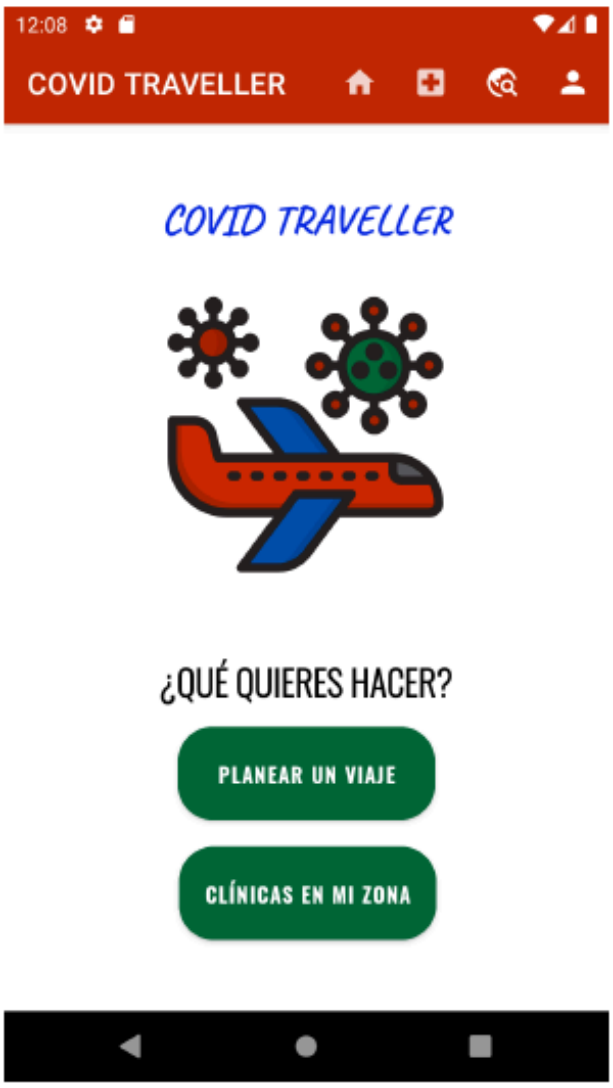

Ilustración 83. Interfaz de la actividad Selección

Esta actividad muestra el logo de la aplicación así como el título de la misma. Además, implementa dos botones que permiten desplazarse a una actividad u otra.

3.4.3.1.5. Clínicas cercanas

Ilustración 84. Interfaz de la actividad Clínicas Cercanas

Esta actividad presenta un elemento *AutoCompleteTextView* que tendrá como opciones las cadenas de caracteres que forman parte del vector *countries* del archivo *strings.xml*.

En esta etapa, se implementa en cada elemento *AutoCompleteTextView* un *ItemClickListener*, que permitirá ejecutar instrucciones cuando se seleccione una de las opciones sugeridas.

```
c1.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        input1 = c1.getText().toString();
        confirma1();
    }
});
```

Ilustración 85. Código del listener para el elemento AutoCompleteTextView

Por último, en esta actividad y en todas las actividades en las que se haga una consulta al servidor, se añadirá un método de codificación.

Este método es necesario ya que, uno de los problemas de la biblioteca *Volley* hace que, en ciertas ocasiones, las cadenas de texto obtenidas no estén en formato *UTF-8*. Por tanto, para asegurar la correcta visualización de los datos, todas las cadenas de texto obtenidas desde la base de datos son codificadas mediante este método.

Dicho método obtendrá la cadena a codificar por cabecera, creará una nueva cadena y le asignará los datos codificados. Tras esto, devolverá la cadena creada.

```
private String codificar(String cadena){
    String str = "";
    try {
        str = new String(cadena.getBytes( charsetName: "ISO-8859-1"), charsetName: "UTF-8");
    } catch (UnsupportedEncodingException e) {

        e.printStackTrace();
    }

    return Html.fromHtml(str).toString();
}
```

Ilustración 86. Código del método codificar()

3.4.3.1.6. Ubicación y clínicas añadidas al mapa

Ilustración 87. Mapa de clínicas y ubicación del dispositivo

Para establecer un marcador en la ubicación del dispositivo dentro del mapa, se ha procedido primero a calcular la misma con el siguiente método:

```
public void onMapReady(GoogleMap googleMap) {
    mMap = googleMap;
    if (ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_FINE_LOCATION)
            != PackageManager.PERMISSION_GRANTED &&
            ActivityCompat.checkSelfPermission( context: this, Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        return;
    }
    mMap.setMyLocationEnabled(true);
    mMap.getUiSettings().setMyLocationButtonEnabled(false);
    LocationManager locationManager = (LocationManager) Ubicacion.this.getSystemService(Context.LOCATION_SERVICE);
    LocationListener locationListener = new LocationListener() {
        @Override
        public void onLocationChanged(Location location) {
            LatLng miUbicacion = new LatLng(location.getLatitude(), location.getLongitude());
            mMap.addMarker(new MarkerOptions().position(miUbicacion).title("Ubicacion actual"))
            .setIcon(bitmapDescriptorFromVector(getApplicationContext(), R.drawable.ic_baseline_my_location_24));;

          mMap.moveCamera(CameraUpdateFactory.newLatLng(miUbicacion));
            CameraPosition cameraPosition = new CameraPosition.Builder()
                    .target(miUbicacion)
                    .zoom(14)
                    .bearing(90)
                    .tilt(45)
                    .build();
            mMap.animateCamera(CameraUpdateFactory.newCameraPosition(cameraPosition));

        }
        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
        }
        @Override
        public void onProviderEnabled(String provider) {
        }
        @Override
        public void onProviderDisabled(String provider) {
        }
    };
    locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,  minTimeMs: 0,  minDistanceM: 0, locationListener);
```

Ilustración 88. Código del método para la obtener la ubicación del dispositivo

Tras esto, se creará un bucle que creará un marcador por cada uno de los objetos de tipo Clínica dentro de la lista obtenida en la actividad anterior. Cada marcador se colocará en función de los valores de latitud y longitud almacenados en cada uno de los objetos.

```
for(int i=0; i<clinicas.size(); i++){
    clinica = new LatLng(clinicas.get(i).getLatVertical(), clinicas.get(i).getLatHorizontal());
    Clinica c = new Clinica(clinicas.get(i).getId(),clinicas.get(i).getPais(),
            clinicas.get(i).getLatVertical(),clinicas.get(i).getLatHorizontal(),
            clinicas.get(i).getNombre(),clinicas.get(i).getTelefono(), clinicas.get(i).getDireccion());
    MarkerOptions markerOptions = new MarkerOptions();
    markerOptions.position(clinica)
            .title(c.getNombre());
    Marker mar = mMap.addMarker(markerOptions);
    mar.setTag(c);
    m.add(mar);
    Toast.makeText(getApplicationContext(), clinicas.get(i).getNombre(), Toast.LENGTH_SHORT).show();
}
```

Ilustración 89. Código para establecer los marcadores para cada clínica

Por último, en cada una de ellas se creará una ventana de información con el resto de datos almacenados en el objeto, para ello se utilizará el siguiente método

```
CustomInfoWindowAdapterClinicas customInfoWindow = new CustomInfoWindowAdapterClinicas( context: this);
mMap.setInfoWindowAdapter(customInfoWindow);
```

Ilustración 90. Creación del objeto de la clase CustomInfoWindowAdapterClinicas

Este método creará un objeto de tipo *CustomInfoWindowAdapterClinicas*. Esta clase será la que genere la ventana en el marcador. Se creará una clase específica para la clase ubicación y otro para la clase Mapa de Rutas, los parámetros son diferentes pero el funcionamiento es el mismo.

```
private static final int coloresmasc []= {0xff3be155,0xff3be155,0xfff50505};
private static final int colores []= {0xff3be155,0xfff5a905,0xfff50505};
private String riesgo[]= {"Pais con riesgo bajo",
"Pais con riesgo intermedio",
"Pais con alto riesgo"};
private String mascarillas[]= {"Mascarilla no obligatoria al aire libre",
        "Mascarilla no obligatoria al aire libre",
        "Mascarilla obligatoria al aire libre"};
@Override
public View getInfoContents(final Marker m) {
    //Carga layout personalizado.
    View v = ((Activity)context).getLayoutInflater()
            .inflate(R.layout.ventana_info_mapa,  root: null);
    if(m!=null) {
        vuelo = (ListElement) m.getTag();
        ((TextView) v.findViewById(R.id.info_window_tercera)).setText(mascarillas[vuelo.getP().getAlerta()]);
        ((TextView) v.findViewById(R.id.info_window_segunda)).setText(riesgo[vuelo.getP().getAlerta()]);
        ((TextView) v.findViewById(R.id.info_window_segunda)).setTextColor(colores[vuelo.getP().getAlerta()]);
        ((TextView) v.findViewById(R.id.info_window_primera)).setText(vuelo.getP().getNombre());
        ((ImageView) v.findViewById(R.id.info_window_imagen)).setColorFilter(coloresmasc[vuelo.getP().getAlerta()]);
    }
    return v;
}
```

Ilustración 91. Código de la clase CustomInfoWindowAdapter

Esta clase programará una serie de cadenas de texto, una polilínea y un icono. Se aplicarán colores de tipo ARGB(Alpha RGB), ya que Android no soporta RGB en estos casos. Dichos colores serán almacenados en un vector y dependerán de los valores almacenados en la variable correspondiente al marcador en cuestión.

3.4.3.1.7. Creación del adaptador para el RecyclerView

En esta clase se programa el adaptador que permitirá mostrar cada elemento de una lista de tipo *ListElement* como un *CardView*[8.1.7]. Para ello, se emplean dos clases, en primer lugar, una primera clase que nos permitirá crear el *CardView,* para ello crearemos un objeto *ViewHolder.*

```
@Override
public ListAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType){
    View view = mInflater.inflate(R.layout.list_element,  root: null);
    return new ListAdapter.ViewHolder(view);
}

@Override
public void onBindViewHolder(final ListAdapter.ViewHolder holder, final int position){
    holder.bindData(mData.get(position));
}
```

Ilustración 92. Código del método onCreateViewHolder

Tras esto, en el mismo fichero, se crea la clase *ViewHolder* que permitirá añadir los datos a cada uno de estos *CardView* y establecer un *OnClickListener* para cada uno de ellos. Esto permitirá que, cuando se haga *click* en ellos, lleve al usuario a la siguiente actividad portando la información contenida en dicho *CardView*

```
public class ViewHolder extends RecyclerView.ViewHolder{
    ImageView iconImage;
    TextView ruta, duracion, tipo;

    ViewHolder(View itemView){
        super(itemView);
        iconImage=itemView.findViewById(R.id.iconImageView);
        ruta = itemView.findViewById(R.id.ruta);
        duracion = itemView.findViewById(R.id.duracion);
        tipo = itemView.findViewById(R.id.tipo);
    }

    void bindData(final ListElement item){
        ruta.setText(item.getRuta());
        duracion.setText(item.getDuracion());
        tipo.setText(item.getTipo());
        itemView.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) { listener.onItemClick(item); }
        });
    }
}
```

Ilustración 93. Código de la clase ViewHolder

3.4.3.1.8. Opciones de vuelo

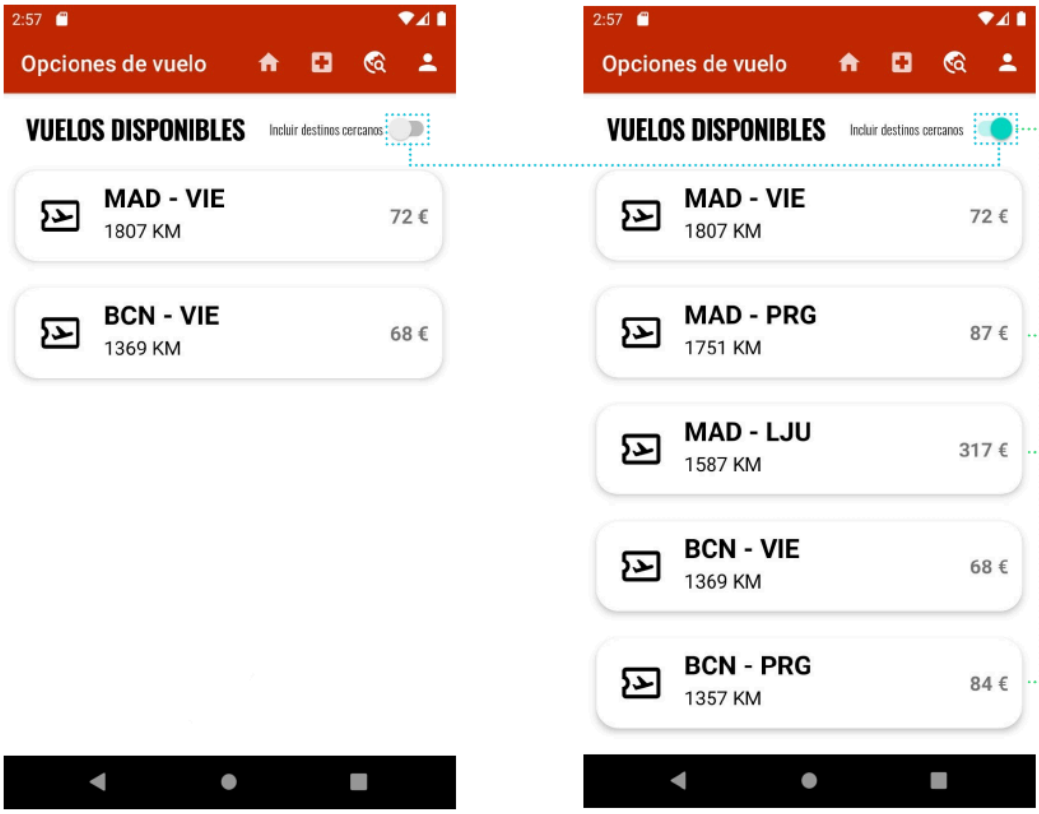

Ilustración 94. Interfaz de la actividad Opciones de Vuelo

En esta clase se programará el *RecyclerView* para que cada elemento de la misma tenga la estética y estructura previamente designadas en la clase *ListAdapter*, a las cuales se añade la información pertinente en cada caso.

Para esto, se utiliza una lista de elementos de tipo *ListElement* la cual se rellenará dentro del método *init()* con datos que provienen de dos listas de objetos de tipo *Aeropuerto* (*aeropuertos1 y aeropuertos2*).

Estos objetos contienen los datos de los aeropuertos de los países de origen y destino respectivamente.

```
public void init(){
    elements = new ArrayList<>();
    for(int i=0; i<aeropuertos1.size(); i++){
        for(int j=0; j<aeropuertos2.size(); j++){
            dis = haversine(aeropuertos1.get(i).getLatHorizontal(), aeropuertos1.get(i).getLatVertical(),
                    aeropuertos2.get(j).getLatHorizontal(), aeropuertos2.get(j).getLatVertical());
            if(dis>1000){
                precio= (int) Math.round(dis/(aeropuertos1.get(i).getCoeficiente()*aeropuertos2.get(j).getCoeficiente()));
            }else{
                precio= (int) Math.round(dis/(aeropuertos1.get(i).getCoeficiente()*aeropuertos2.get(j).getCoeficiente())+50);
            }
            elements.add(new ListElement( ruta: aeropuertos1.get(i).getCodigo()+" - "+aeropuertos2.get(j).getCodigo(),
                    duracion: Integer.toString(dis)+" KM", tipo: precio+ " €",aeropuertos1.get(i).getNombre(),aeropuertos2.get(j).getNombre(),
                    aeropuertos1.get(i).getLatVertical(), aeropuertos2.get(j).getLatVertical(),
                    aeropuertos1.get(i).getLatHorizontal(),aeropuertos2.get(j).getLatHorizontal(), aeropuertos2.get(j).getP()));
        }
    }
```

Ilustración 95. Código del método init()

Este objeto de tipo *ListElement* por tanto, contará con toda la información relevante sobre el viaje. Tras esto, en el mismo método crearemos un objeto de tipo *ListAdapter* que nos permitirá mostrar cada elemento de esta lista como un elemento del *RecyclerView*. Cada objeto de tipo ListAdapter llevará un *Click Listener* como ya se vió antes con los elementos *AutoCompleteTextView*

```
ListAdapter listAdapter = new ListAdapter(elements,  context: this, new ListAdapter.OnItemClickListener() {
    @Override
    public void onItemClick(ListElement item) { moveToDescription(item); }
});
```

Ilustración 96. Declaración de un objeto de tipo ListAdapter

Por último, al final del método, se asignará dicha lista al elemento *RecyclerView* creado previamente y se le asignará como adaptador de la lista el objeto de tipo *ListAdapter* creado anteriormente.

```
RecyclerView recyclerView = findViewById(R.id.lista);
recyclerView.setHasFixedSize(true);
recyclerView.setLayoutManager(new LinearLayoutManager( context: this));
recyclerView.setAdapter(listAdapter);
```

Ilustración 97. Código de configuración del elemento RecyclerView

Esta función *init()* debe ser llamada cada vez que se desee crear o modificar el *RecyclerView,* es por esto que se llama en el método de creación de la actividad (onCreate) [ciclo de vida de act].

También deberá ser llamada cada vez que se quiera modificar esta lista en función de si el botón *Switch* está activo o desactivado. Se ha de llamar de llamar a la función *init()* para volver a crear dicha lista con los elementos que formen parte de las listas aeropuertos1 y aeropuertos2 que serán los que se modificarán en todo caso.

Para esto, dentro del método *bot()* que será el método asignado al *Switch*, primero se comprueba el estado del botón.

```
public void bot(View v){
    if(v.getId()==R.id.switch1){
```

Ilustración 98. Comprobación del estado del elemento Switch

Tras esto se comprueban que los identificadores de los aeropuertos de los países adyacentes al país de destino están dentro de los límites acotados y por lo tanto son correctos. Estos identificadores han sido añadidos a los objetos de tipo Aeropuerto a1 y a2 respectivamente.

Tras esto, procederemos a añadir dichos objetos a las listas en caso de estar el botón activo o, en caso de estar inactivo, borraremos el último elemento añadido a dicha lista eliminando el elemento en última posición. Sabemos que es este porque sabemos que el método *.add()* de *List* hace la inserción del elemento al final de la lista.

```
if(switch_.isChecked()){
    if(a1!=0 && a1!=pais1.getId()){
        aeropuertos2.add(aeropuerto1);
    }
    if(a2!=34 && a2!=pais1.getId()){
        aeropuertos2.add(aeropuerto2);
    }
    init();
}else{
    if(a1!=0 && a1!=pais1.getId()){
        aeropuertos2.remove( index: aeropuertos2.size()-1);
    }
    if(a2!=42 && a2!=pais1.getId()){
        aeropuertos2.remove( index: aeropuertos2.size()-1);
    }
    init();
}
```

Ilustración 99. Código del método bot()

Por último, se añade una interpretación en código de la fórmula del semiverseno (haversine en inglés) **[25]**. Esta función permite calcular, en función de las coordenadas proporcionadas y el radio de la tierra, la distancia entre ambos puntos en kilómetros.

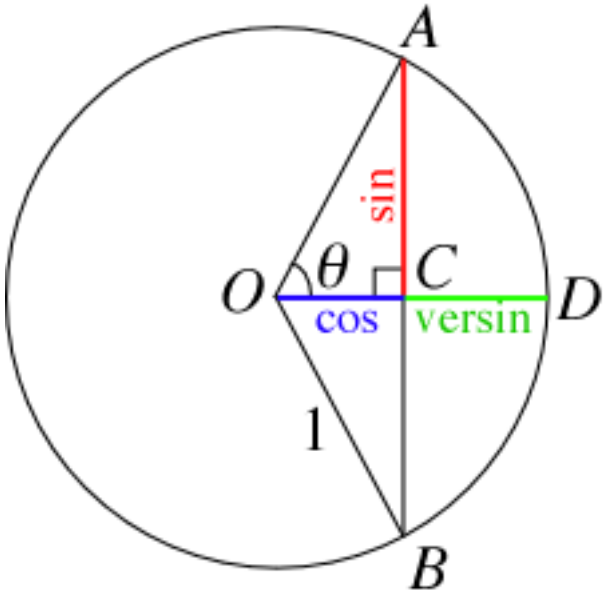

Ilustración 100. Representación gráfica de la fórmula del semiverseno[25]

```
private static int haversine(double lon1, double lat1, double lon2, double lat2) {

    double radioTierra = 6371;

    lat1 = Math.toRadians(lat1);
    lon1 = Math.toRadians(lon1);
    lat2 = Math.toRadians(lat2);
    lon2 = Math.toRadians(lon2);

    double dlon = (lon2 - lon1);
    double dlat = (lat2 - lat1);

    double sinlat = Math.sin(dlat / 2);
    double sinlon = Math.sin(dlon / 2);

    double a = (sinlat * sinlat) + Math.cos(lat1)*Math.cos(lat2)*(sinlon*sinlon);
    double c = 2 * Math.asin (Math.min(1.0, Math.sqrt(a)));

    double distancia = radioTierra * c;

    return (int)distancia;

}
```

Ilustración 101. Código de la función para implementar la fórmula del semiverseno

Este dato, además de para permitir conocer la distancia entre ambos, permitirá hacer una estimación del precio. Al no tener acceso a ninguna API sobre vuelos, se ha decidido hacer una estimación del precio del vuelo en función de la distancia y el coeficiente de cada aeropuerto.

Dicho coeficiente es un valor que se ha asignado en función del tamaño, país y tráfico de cada aeropuerto. Se ha tenido también en cuenta que si la distancia es de menos de 1000km se añadirá un plus de 50 euros con el fin de obtener precios más realistas. Si bien no calcula precios reales, calcula precios aceptablemente parecidos.

```
dis = haversine(aeropuertos1.get(i).getLatHorizontal(), aeropuertos1.get(i).getLatVertical(),
        aeropuertos2.get(j).getLatHorizontal(), aeropuertos2.get(j).getLatVertical());
if(dis>1000){
    precio= (int) Math.round(dis/(aeropuertos1.get(i).getCoeficiente()*aeropuertos2.get(j).getCoeficiente()));
}else{
    precio= (int) Math.round(dis/(aeropuertos1.get(i).getCoeficiente()*aeropuertos2.get(j).getCoeficiente())+50);
```

Ilustración 102. Código del cálculo estimado del precio del billete

3.4.3.1.9. Detalles del viaje

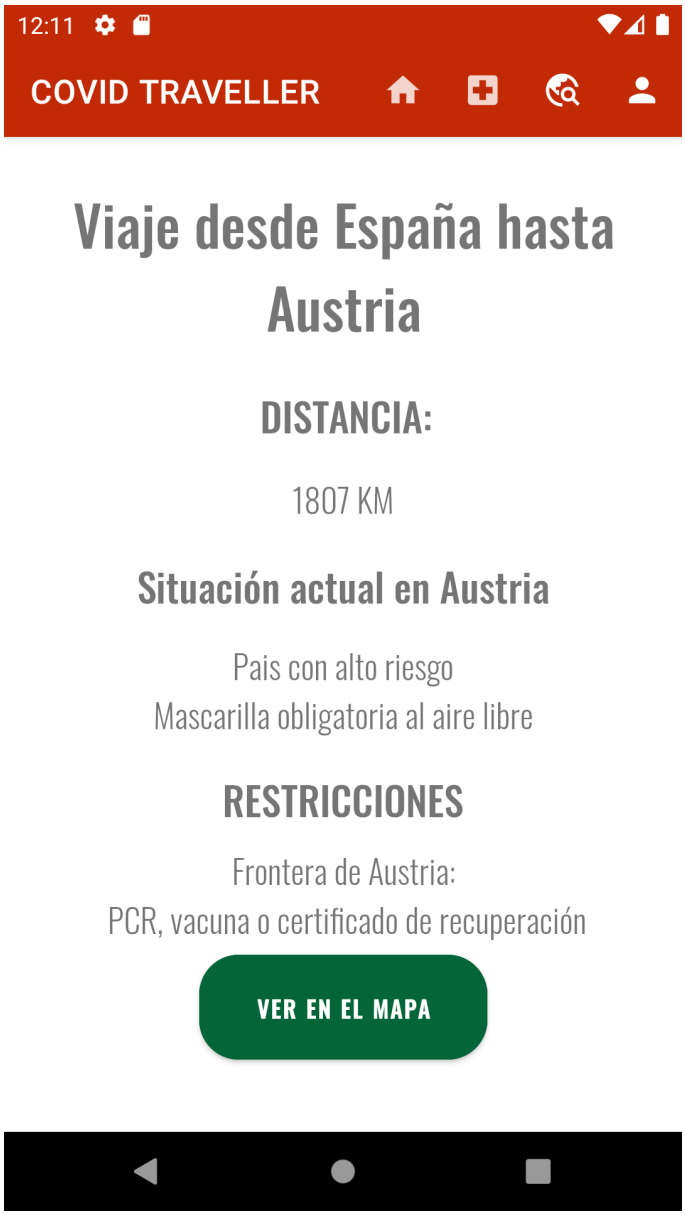

Ilustración 103. Interfaz de la actividad Detalles del Viaje en el Sprint 2

En esta vista se reciben los datos del vuelo seleccionado previamente, los cuales se mostrarán en ella. A diferencia de la etapa anterior dónde la mayoría del texto era estático en esta etapa se modifica para mostrar la información en función del vuelo escogido. Se muestra mayoritariamente información sobre el país del destino.

```
private String riesgo[]= {"Pais con riesgo bajo",
        "Pais con riesgo intermedio",
        "Pais con alto riesgo"};
private String mascarillas[]= {"Mascarilla no obligatoria al aire libre",
        "Mascarilla no obligatoria al aire libre",
        "Mascarilla obligatoria al aire libre"};
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_specs);
    ListElement element = (ListElement) getIntent().getSerializableExtra( name: "ListElement");
    pais1 = (Pais) getIntent().getSerializableExtra( name: "Pais1");
    pais2 = (Pais) getIntent().getSerializableExtra( name: "Pais2");
    vuelo = (ListElement) getIntent().getSerializableExtra( name: "Vuelo");

    trip = findViewById(R.id.pregunta);
    borders = findViewById(R.id.borders);
    borders_ = findViewById(R.id.borders_);
    distancia= findViewById(R.id.distancia_);
    restricciones_ = findViewById(R.id.restrictions_);
    trip.setText("Viaje desde " + pais1.getNombre() + " hasta " + vuelo.getP().getNombre());
    distancia.setText(vuelo.getDuracion());
    borders.setText("Situación actual en "+ vuelo.getP().getNombre());
    borders_.setText(riesgo[vuelo.getP().getAlerta()]+"\n"+mascarillas[vuelo.getP().getAlerta()]);
    restricciones_.setText("Frontera de "+vuelo.getP().getNombre()+":\n "+vuelo.getP().getRestricciones());
}
```

Ilustración 104. Código de la actividad Detalles del Viaje

3.4.3.1.10. Sobre mi

Esta actividad sólo mostrará un texto estático con información sobre el autor

Ilustración 105. Interfaz de la actividad SobreMi

3.4.3.2. Implementación servidor

En esta etapa, en primer lugar, se añadirán dos nuevas tablas a la base de datos, la tabla *clínicas* y la tabla *aeropuertos*.

Se crean también una serie de clases que permitirán crear objetos con atributos similares a las tablas de la base de datos. Al igual que en la etapa anterior se creó la clase *Pais,* en esta etapa se crearán 3 clases más con el mismo fin. Dichas clases extenderán de la clase abstracta *Serializable*, lo que permitirá que los objetos de dichas clases puedan transferirse de una actividad a otra.

Por otro lado, se crearán distintos métodos para consultar dichas tablas con sus correspondientes ficheros PHP para cada tipo de consulta.

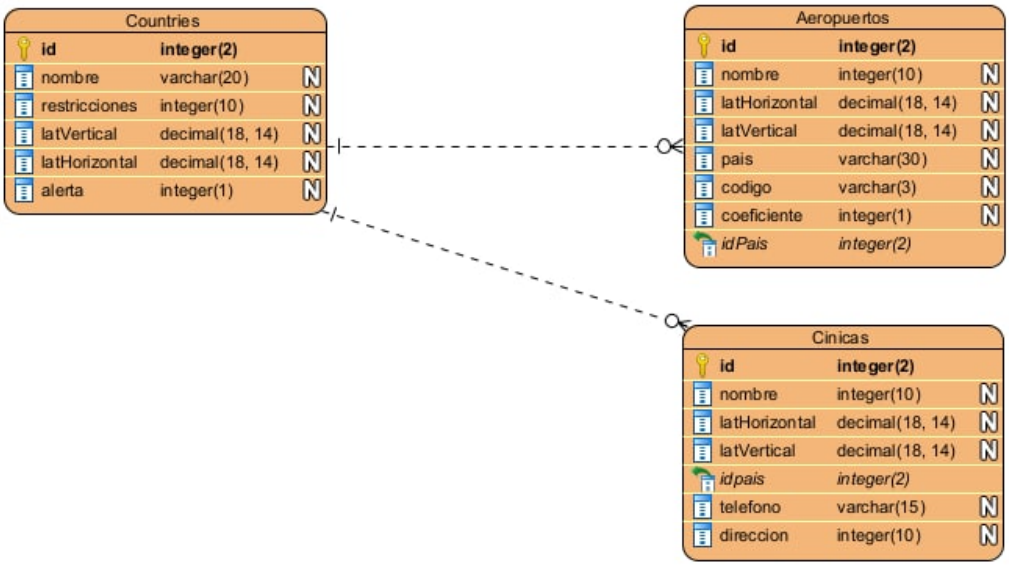

Ilustración 106. Esquema de la base de datos en el Sprint 2

3.4.3.2.1. Tabla clinicas

Esta tabla contará con toda la información necesaria sobre las clínicas donde el usuario se podrá someter a test médicos. Para el manejo de los datos de dicha tabla se crea la clase *Clinica* con atributos similares a los de la tabla.

Ilustración 107. Tabla Clinicas

```
public Clinica(int id, String pais, double latVertical, double latHorizontal, String nombre, String telefono, String direccion) {
    this.id = id;
    this.pais = pais;
    this.latVertical = latVertical;
    this.latHorizontal = latHorizontal;
    this.nombre = nombre;
    this.telefono = telefono;
    this.direccion = direccion;
}
```

Ilustración 108. Código del constructor de la clase clínicas

Tras esto, se implementa el método para realizar la consulta a dicha tabla. La cabecera de dicho método tendrá una estructura similar a la función de consulta de países.

Cambiarán en este caso, el archivo PHP asignado, el atributo que se filtra y el valor buscado. En este caso se elimina el objeto que se enviaba por cabecera ya que dicha consulta sólo se realiza una vez .

```
consulta( URL: "http://192.168.0.43/dev/consultaubi.php?pais="+input1+"");
```

Ilustración 109. Código para llamar a la función consulta()

A su vez, el código de la consulta será similar al de la función de la consulta de países, sólo que, en este caso, en vez de almacenar un resultado, se almacenarán varios. Es por esto que no se utiliza un objeto sino una lista de objetos de tipo *Clinica*.

```
private void consulta(String URL){
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(URL, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            JSONObject jsonObject = null;
            for (int i = 0; i < response.length(); i++) {
                try {
                    jsonObject = response.getJSONObject(i);
                    clinicas.add(new Clinica(jsonObject.getInt( name: "id"), codificar(jsonObject.getString( name: "pais")),
                            jsonObject.getDouble( name: "latVertical"),jsonObject.getDouble( name: "latHorizontal"),
                            jsonObject.getString( name: "nombre"),codificar(jsonObject.getString( name: "telefono")),
                            codificar(jsonObject.getString( name: "direccion"))));
                    Toast.makeText(getApplicationContext(), clinicas.get(i).getNombre(), Toast.LENGTH_SHORT).show();
                } catch (JSONException e) {
                    Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHORT).show();
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast.makeText(getApplicationContext(), error.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
    );
    RequestQueue requestQueue = Volley.newRequestQueue( context: this);
    requestQueue.add(jsonArrayRequest);
}
```

Ilustración 110. Código del método consulta()

Por último, se muestra en la siguiente imagen el código del archivo PHP dónde se encuentra la consulta.

```php
<?php
$hostname='localhost';
$username='root';
$password='';
$database='tfgl';
$conexion= new mysqli($hostname,$username,$password,$database);
$pais=$_GET['pais'];
$consulta = "select * from clinicas where pais= '$pais'";
$resultado = $conexion -> query($consulta);

while($fila=$resultado -> fetch_array()){
    $clinica[] = array_map('utf8_encode', $fila);
}
echo json_encode($clinica);
$resultado -> close();

?>
```

Ilustración 111. Código del archivo consultaubi.php

3.4.3.2.2. Tabla aeropuertos

Esta tabla contará con toda la información necesaria sobre los aeropuertos de cada país. Para el manejo de los datos de dicha tabla se crean las clases *Aeropuerto* y *ListElement* con atributos similares a los de la tabla.

Mientras que la clase Aeropuerto estará orientada a almacenar información sobre un único aeropuerto, la clase *ListElement* guardará la información de dos. Esto sucede porque la finalidad de la clase *ListElement* es la de almacenar información sobre un viaje. Para ello, guarda información sobre el aeropuerto del país de origen y el aeropuerto del país de destino.

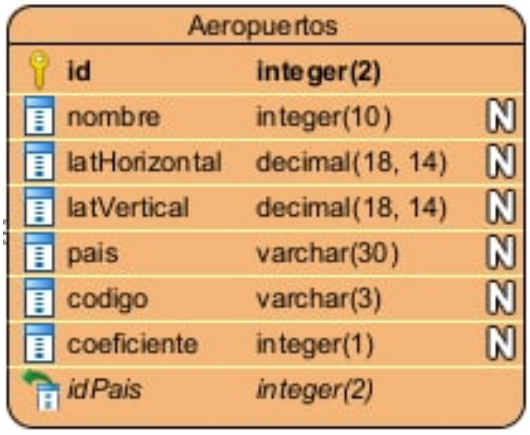

Ilustración 112. Tabla Aeropuertos

```
public Aeropuerto(int id, String nombre, double latVertical, double latHorizontal, String codigo, int coeficiente, int idPais, Pais p) {
    this.id = id;
    this.nombre = nombre;
    this.latVertical = latVertical;
    this.latHorizontal = latHorizontal;
    this.codigo = codigo;
    this.coeficiente = coeficiente;
    this.idPais = idPais;
    this.p = p;
}
```

Ilustración 113. Código del constructor de la clase Aeropuerto

```
public ListElement(String ruta, String duracion, String tipo, String nombre1, String nombre2,
                   Double latVertical1, Double latVertical2, Double latHorizontal1, Double latHorizontal2, Pais p) {
    this.ruta = ruta;
    this.duracion = duracion;
    this.tipo = tipo;
    this.nombre1 = nombre1;
    this.nombre2 = nombre2;
    this.latVertical1 = latVertical1;
    this.latVertical2 = latVertical2;
    this.latHorizontal1 = latHorizontal1;
    this.latHorizontal2 = latHorizontal2;
    this.p = p;
}
```

Ilustración 114. Código del constructor de la clase ListElement

Tras esto, se implementa el método para realizar la consulta a dicha tabla. La cabecera de dicho método tendrá una estructura similar a los métodos de consulta anteriormente mencionados.

Cambiarán en este caso, el archivo PHP asignado, el atributo que se filtra, el valor buscado que en este caso será un objeto de tipo *Pais* y se añadirá una lista de objetos de tipo aeropuerto. Esto se debe a que, al igual que la anterior consulta, se almacenarán varios resultados

```
consultaAeropuertos( URL: "http://192.168.0.43/dev/consultaaeropuertos.php?pais=" +input1+ "", aeropuertos, pais1);
```

Ilustración 115. Código de llamada al método consultaAeropuertos()

En este caso se crea un objeto de tipo Aeropuerto al que se le asignan los datos obtenidos y se almacena en la lista.

```
private void consultaAeropuertos(String URL, List<Aeropuerto> aeropuertos, Pais p){
    JsonArrayRequest jsonArrayRequest = new JsonArrayRequest(URL, new Response.Listener<JSONArray>() {
        @Override
        public void onResponse(JSONArray response) {
            JSONObject jsonObject = null;
            for (int i = 0; i < response.length(); i++) {
                try {
                   jsonObject = response.getJSONObject(i);
                    aeropuertos.add(new Aeropuerto(jsonObject.getInt( name: "id"),codificar(jsonObject.getString( name: "nombre")),
                            jsonObject.getDouble( name: "latVertical"), jsonObject.getDouble( name: "latHorizontal"),
                            jsonObject.getString( name: "codigo"), jsonObject.getInt( name: "coeficiente"),
                            jsonObject.getInt( name: "idPais"), p));
                    Toast.makeText(getApplicationContext(),aeropuertos.get(i).getCodigo(),Toast.LENGTH_SHORT).show();

                } catch (JSONException e) {
                    Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_SHORT).show();
                }
            }
        }
    }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            Toast.makeText(getApplicationContext(), error.getMessage(), Toast.LENGTH_SHORT).show();
        }
    }
    );
    requestQueue = Volley.newRequestQueue( context: this);
    requestQueue.add(jsonArrayRequest);
}
```

Ilustración 116. Código del método consultaAeropuertos()

A continuación, se muestra el código del archivo PHP dónde se encuentra la consulta.

```php
<?php
$hostname='localhost';
$username='root';
$password='';
$database='tfgl';
$conexion= new mysqli($hostname,$username,$password,$database);
$pais=$_GET['pais'];
$consulta = "select * from aeropuertos where pais= '$pais'";
$resultado = $conexion -> query($consulta);

while($fila=$resultado -> fetch_array()){
    $country[] = array_map('utf8_encode', $fila);
}
echo json_encode($country);
$resultado -> close();

?>
```

Ilustración 117. Código del archivo consultaaeropuertos.php

3.4.4. Pruebas de usabilidad final

Al ser una aplicación orientada a todos los públicos, se ha tratado de realizar estas pruebas de usabilidad en una población lo más dispersa posible. Para ello, se han seleccionado una muestra de usuarios tanto de género masculino como femenino.

Además, se han establecido 3 rangos principales de edad teniendo en cuenta los rangos con mayor número de usuarios de Smartphone. Dichos rangos son de 15 a 30 años, de 30 a 50 y 50 años o más. Cada combinación de rango y género contará con 2 representantes para enriquecer la muestra. El usuario se someterá a 6 preguntas evaluables en un rango del 1 al 5:

- 1. Totalmente en desacuerdo
- 2. Algo en desacuerdo
- 3. Ni de acuerdo ni en desacuerdo
- 4. Algo de acuerdo
- 5. Totalmente de acuerdo

Los usuarios sometidos al cuestionario serán:

- Usuario masculino de 17 años
- Usuario masculino de 23 años
- Usuario femenino de 18 años
- Usuario femenino de 26 años
- Usuario masculino de 35 años
- Usuario masculino de 46 años
- Usuario femenino de 33 años
- Usuario femenino de 46 años
- Usuario masculino de 58 años
- Usuario masculino de 61 años
- Usuario femenino de 54 años
- Usuario femenino de 59 años

A continuación, se podrán ver las preguntas a las que se sometieron los usuarios:

1. Las instrucciones para el uso de la aplicación eran claras y me ayudaron a comprender el funcionamiento de la aplicación.
2. No encontré problemas ni errores durante el uso de la aplicación.
3. El diseño de la aplicación es intuitivo y facilita el uso de la misma.
4. La aplicación es fácil de usar y no precisa de un alto nivel de conocimiento de ningún ámbito particular.
5. La información mostrada es útil así como las notificaciones.
6. La aplicación es útil y en caso de ser lanzada al mercado la descargaría.

Los resultados de las mismas se muestran en la siguiente tabla:

Usuario	P1	P2	P3	P4	P5	P6
M17	5	5	5	4	3	4
M23	5	5	4	5	4	5
F18	5	5	4	5	3	5
F26	4	5	3	4	3	5
M35	4	5	5	5	4	5
M46	3	5	4	4	3	5
F33	4	5	4	4	4	5
F46	4	5	4	4	3	5
M58	4	5	5	5	3	4
M61	3	5	3	3	4	5
F54	4	5	3	4	3	4
F59	3	5	3	4	4	5
MEDIA	4	5	3,92	4,25	3,42	4,75

Tabla 6. Tabla de puntuaciones del cuestionario

4. Modelo de Negocio

En este proyecto, se incluye un estudio sobre el futuro de la aplicación, así como su posición e impacto en el mercado. Se presenta así un modelo de negocio para esta aplicación destinada a facilitar a sus usuarios la planificación de sus viajes.

4.1. Objetivos

El objetivo principal del proyecto será ofrecer al usuario una herramienta para viajar que pueda competir o complementar a las ya existentes, aportando nuevas funcionalidades como la muestra de información sobre las restricciones.

4.2. Análisis de entorno

Para analizar el entorno hacia el que se dirige el negocio, se ha realizado un embudo de mercado. En un principio, la aplicación sólo está en Español, aunque se desea su desarrollo multilenguaje en una futura versión. Esto limita el negocio momentáneamente a los 47 millones de habitantes de España, de los cuales se estima que alrededor de 32 millones poseen un Smartphone **[28]**.

Al ser una aplicación para Android, se estima que el 72,5% **[2]** de los usuarios de Smartphone puedan acceder a ella, lo que supone 23,2 millones de usuarios. Sabiendo que, en el primer trimestre de 2021, el porcentaje de viajes de españoles al extranjero fue de un 3% **[29],** se estiman unos 696000 usuarios por trimestre, lo que supondrían unos 2.09 millones de usuarios al año. Aunque estas cifras son bastante pesimistas ya que, con la mayoría de la población vacunada, el tráfico de turistas españoles aumentará.

4.3. Plan de mercadotecnia

El plan de mercado consistiría en tratar de obtener apoyo por parte del gobierno español, que apoyaría el proyecto y lo integraría como ya ha hecho previamente con otras aplicaciones relacionadas con el mismo tema. Este apoyo sería importante a la hora de la implementación de APIs para la mejora del funcionamiento. En caso de no contar con una subvención por parte del gobierno, se estudiaría el tema de la inclusión de anuncios o la sinergia con aerolíneas con el fin de ofrecer sus vuelos dentro de la aplicación.

Una vez la aplicación fuese multi-lenguaje, las posibilidades previamente mencionadas se multiplicarían, apareciendo los gobiernos de otros países así como nuevas aerolíneas y patrocinadores.

5. Conclusiones

5.1. Conclusiones

Este proyecto permitirá al usuario planear futuros viajes contemplando las restricciones y nuevas formas de viajar debido a la situación de pandemia acontecida durante los últimos meses.

Si bien existen portales web con información sobre restricciones y aplicaciones para planear un viaje, ninguna de ellas combina ambos ámbitos. Esto convierte a este proyecto en una idea interesante a corto, medio y largo plazo.

Esta aplicación permite además obtener información de contacto de las clínicas del país donde se encuentra el usuario. Esto supone una ventaja, ya que combina dos búsquedas que hoy en día están relacionadas, para ofrecer ambas en una única aplicación.

5.2. Trabajos futuros

De cara a las siguientes versiones de la aplicación, uno de los principales factores a implementar sería el soporte multi-lenguaje. Este, como se ha mencionado previamente, abriría las puertas a infinidad de nuevos usuarios y oportunidades de negocio. Esto supondría un alto crecimiento en cuanto a la popularidad y reconocimiento de la aplicación respecta.

Uno de los puntos positivos de la presente aplicación es que es una herramienta reutilizable, ya que, al mostrar contenidos específicos por países, puede tener otros usos aparte de la muestra de información sobre restricciones.

Es así que, aparte del enfoque inicial orientado a la situación pandémica, se puede enfocar hacia otros ámbitos como la información cultural sobre cada país o cualquier otro ámbito informativo.

Este crecimiento facilitaría mucho la oportunidad de obtener oportunidades de negocio ya fuese a través de subvenciones públicas por parte de gobiernos o de colaboraciones con empresas privadas.

Dichas colaboraciones implicarían una modificación de ciertos aspectos de la aplicación, por ejemplo, en caso de colaborar con aerolíneas, se permitiría al usuario filtrar los vuelos por aerolíneas. A su vez, se mostrarían de manera fija en la parte superior de la vista, una serie de vuelos "destacados" por los que se cobraría un extra a la aerolínea correspondiente en concepto de publicidad.

Por otro lado, se podrían implementar anuncios de otras compañías a través de empresas de marketing, además de un sistema de códigos descuento para ciertos vuelos. Cada vez que estos códigos fuesen usados por los usuarios, supondrían un cobro extra a la empresa ofertante. De igual manera, se contempla las posibles sinergias con clínicas privadas en las que se cobra un extra por cada vez que se consulta información sobre la misma a través de la aplicación.

6. Apéndices

6.1. Manual de Usuario

En este apartado se hace un breve tutorial sobre el uso de cada una de las actividades que forman la aplicación.

6.1.1. Icono de la aplicación

Para iniciar la aplicación, el usuario deberá pulsar el icono de la misma dentro del menú de aplicaciones.

COVID TR...

Ilustración 118. Icono de la aplicación

6.1.2. Pantalla de inicio:

Esta será la única pantalla que estará formada por dos actividades, la primera de ellas nos muestra a pantalla completa el logo y el nombre de la aplicación con una animación. Tras 4 segundos, se realiza una transición automática hacia la segunda actividad con otra animación.

Esta segunda actividad será la que permanecerá en pantalla hasta que el usuario pulse uno de los dos botones que lo llevará a la siguiente vista, la cual podrá ser la actividad para buscar clínicas o la actividad para planificar un viaje.

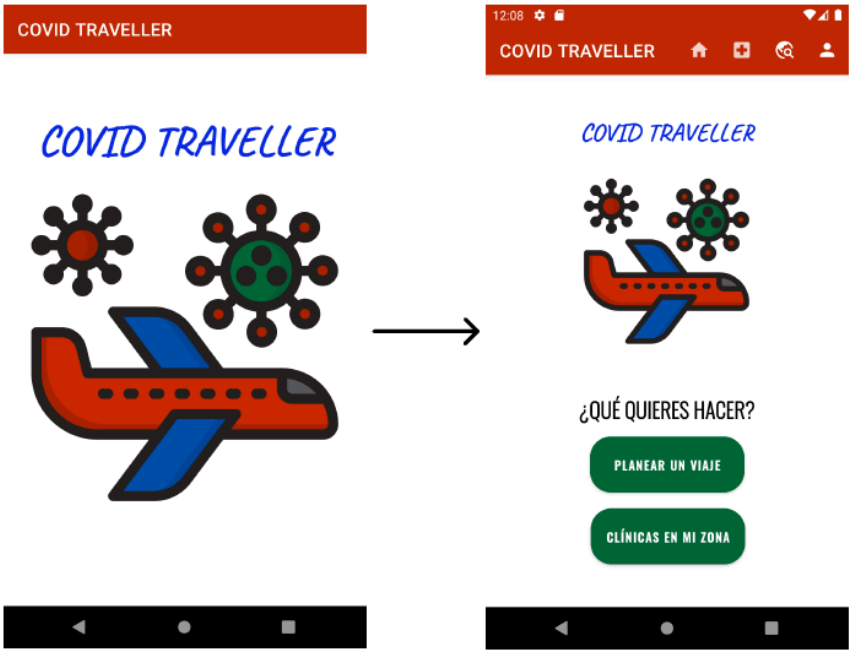

Ilustración 119. Vista de la pantalla de inicio

6.1.3. Menú superior

En este menú, presente en prácticamente todas las vistas mencionadas anteriormente, se aprecia el nombre de cada vista y una serie de iconos. Estos, al ser pulsados, llevarán al usuario a diferentes vistas en función de cuál sea el icono pulsado. A continuación, se mencionan cada uno de ellos con la correspondiente vista hacia la que se desplaza en caso de ser pulsado.

- Botón con icono de casa -> Vista principal
- Botón con icono de cruz -> Vista buscar clínicas
- Botón con bola del mundo y lupa -> Vista planear viaje
- Botón con icono de persona -> Vista sobre mí

Ilustración 120. Menú de la aplicación

6.1.4. Búsqueda de clínicas en mi país:

En esta vista, el usuario debe introducir el nombre de un país en la caja de texto que aparece en el centro de la misma al empezar a escribir el nombre. La aplicación lanzará una serie de sugerencias para autocompletar el nombre del país que se desea introducir. Tras esto, el usuario pulsará el botón que lo llevará a la siguiente vista.

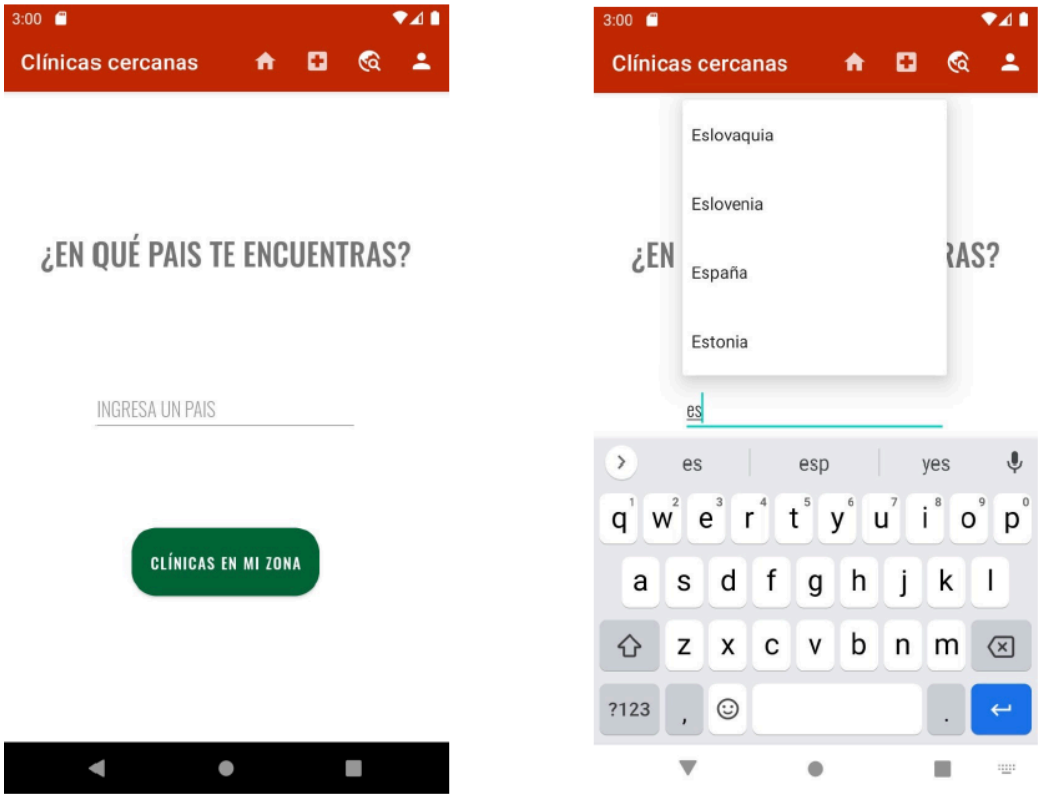

Ilustración 121. Vista de la pantalla de búsqueda de clínicas

En caso de no introducir ningún país y pulsar el botón, aparte de no avanzar a la siguiente vista, la aplicación mostrará una notificación indicando que es necesario introducir un país en la caja de texto.

Ilustración 122. Vista de notificación lanzada debido a error del usuario

6.1.5. Mapa para mostrar la clínicas del país y ubicación del usuario:

En esta vista, se pueden observar en el mapa una serie de marcadores en color rojo que mostrarán la posición de las diferentes clínicas donde el usuario podrá someterse a un test.

Al pulsar cualquiera de estos marcadores, emergerá una ventana en la que visualizar la información más importante de dicha clínica. Por último, el usuario podrá visualizar un marcador en color azul, el cual mostrará la ubicación del dispositivo.

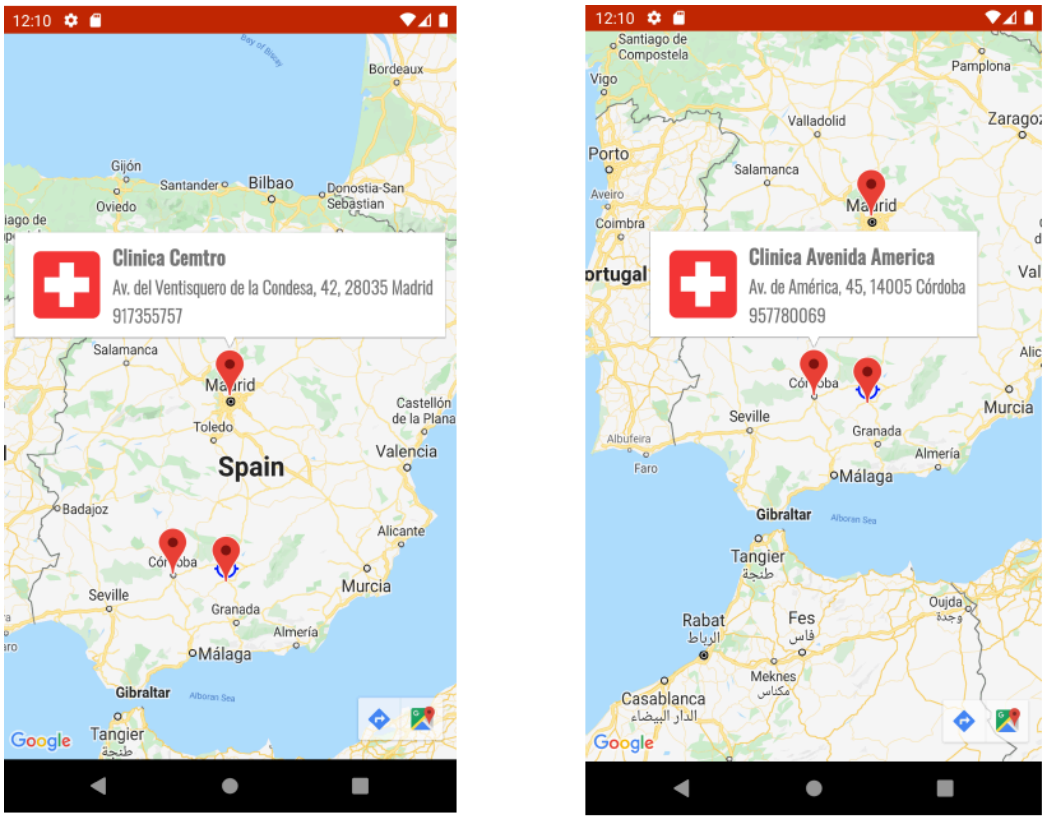

Ilustración 123. Vista del mapa de clínicas y ubicación

6.1.6. Planificar viaje

En esta vista, existen dos cajas de texto. De igual manera que en la vista de búsqueda de clínicas, el usuario debe de introducir el nombre de un país en cada una de ellas. Al empezar a escribir, la aplicación mostrará una serie de sugerencias en función de lo que haya escrito. Una vez rellenas las cajas de texto, el usuario podrá pulsar el botón de la parte inferior que lo llevará a la siguiente lista.

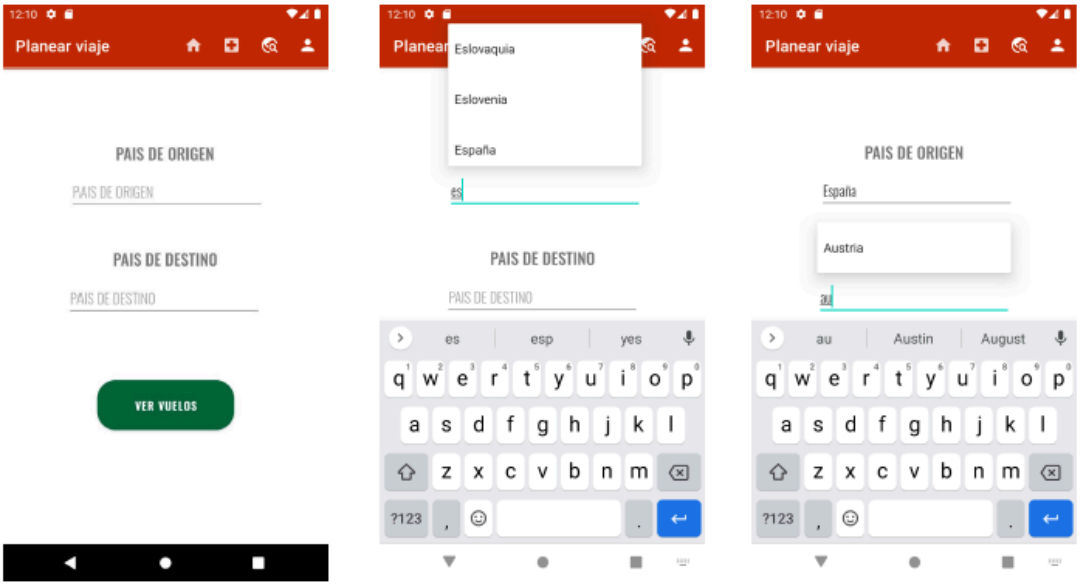

Ilustración 124. Vista de la pantalla de Planear viaje

En esta vista, en caso de no haber rellenado alguna de las dos cajas de texto, al pulsar el botón, el usuario no avanzará hacia la siguiente vista. En este caso, la aplicación mostrará una alerta indicando que se rellene la caja de texto que falta por rellenar.

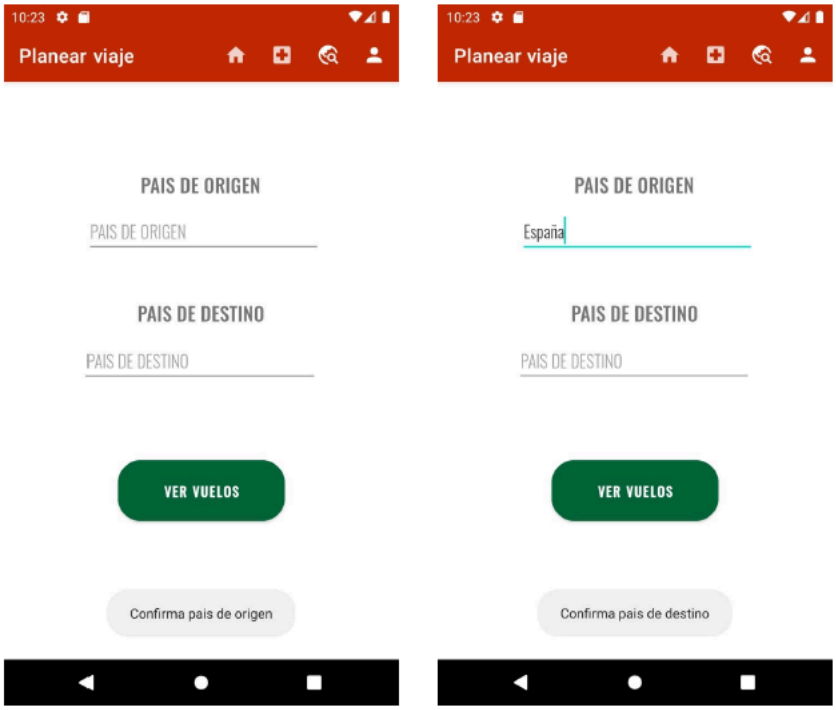

Ilustración 125. Vista de notificaciones por error de usuario

6.1.7. Lista de elección de vuelos

En esta vista, se puede observar una lista con todos los vuelos entre los principales aeropuertos del país de origen y los principales aeropuertos del país de destino. Dentro de cada una de las opciones, se muestra la distancia entre ambos aeropuertos, los códigos IATA de cada uno de ellos indicando la ruta que seguirá el avión y una estimación del precio del billete para dicho vuelo.

También se incluye un botón desplazable en la parte superior derecha. Cuando éste se encuentre activo, el usuario podrá visualizar, aparte de los ya vistos, una serie de vuelos entre los aeropuertos del país de origen y los aeropuertos de los países colindantes al país de destino. En caso de desactivar dicho botón, el usuario dejará de visualizar dichos vuelos visualizando sólo los que veía en un primer momento.

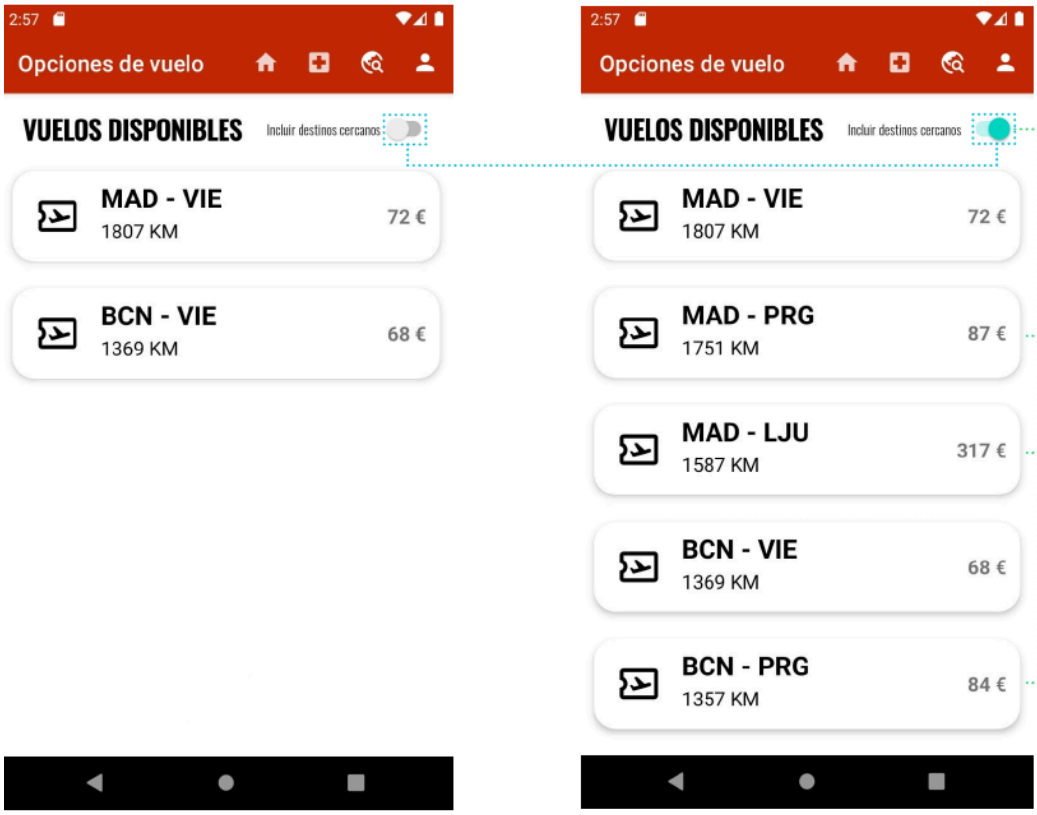

Ilustración 126. Vista de la pantalla de opciones de vuelos

6.1.8. Detalles del viaje

En esta vista, se puede observar la información más relevante del viaje. En la parte superior se muestra un título con los países de origen y destino. Justo debajo, se muestra la distancia entre los aeropuertos de salida y llegada. Tras esto, se muestra la situación actual en el país con respecto a la pandemia, además de informar sobre la obligatoriedad del uso de mascarilla en el mismo. Por último se indicarán las restricciones sanitarias de entrada al país. Una vez el usuario pulse el botón, avanzará a la siguiente vista en la que podrá ver la ruta del viaje en el mapa.

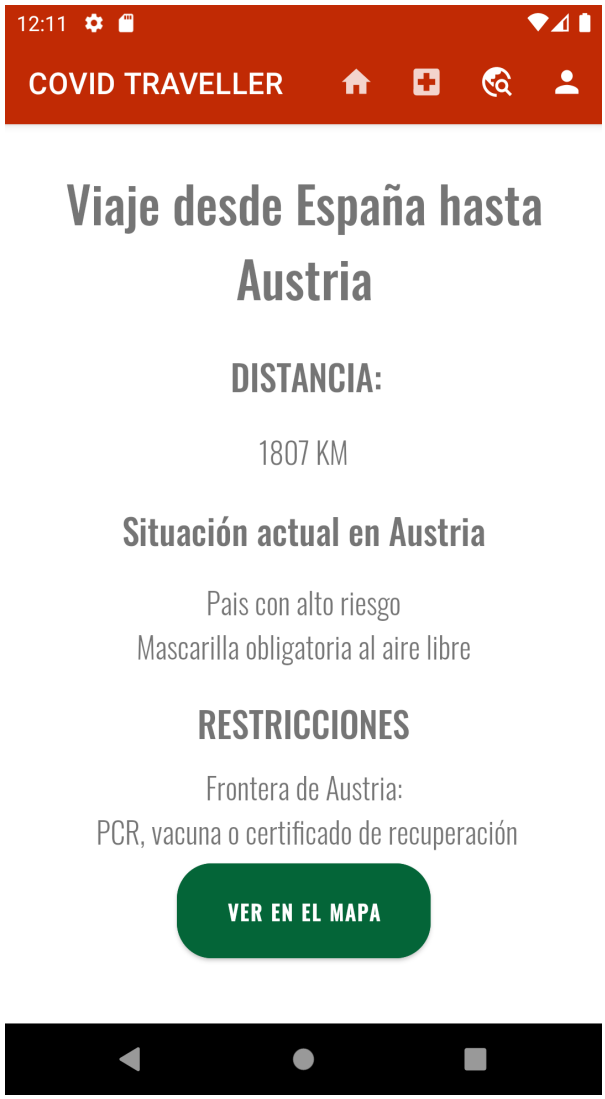

Ilustración 127. Vista de la pantalla Detalles de Viaje

6.1.9. Mapa para mostrar la ruta del viaje seleccionado

En esta vista se muestra la ruta que seguirá el avión durante el viaje. Se puede apreciar un icono de un avión despegando indicando la ubicación en el mapa del aeropuerto de origen. Por otro lado, también habrá un icono de un avión aterrizando indicando la ubicación del aeropuerto de destino. Este último contará con una ventana de información en la que se indicará el nombre del país, la situación actual con respecto a la pandemia y la obligatoriedad del uso de mascarilla dentro del mismo.

El usuario también podrá saber esta información en función del color del icono de mascarilla que habrá dentro de la ventana de información. El color rojo de la misma indicará la necesidad de usar la mascarilla al aire libre y verde que no es obligatorio el uso de la misma en espacios abiertos.

Por último, la aplicación mostrará una polilínea entre ambos marcadores. El color de la misma indicará también la situación del país con respecto a la pandemia siendo el color verde indicativo de riesgo bajo, naranja indicativo de riesgo intermedio y rojo indicativo de alto riesgo.

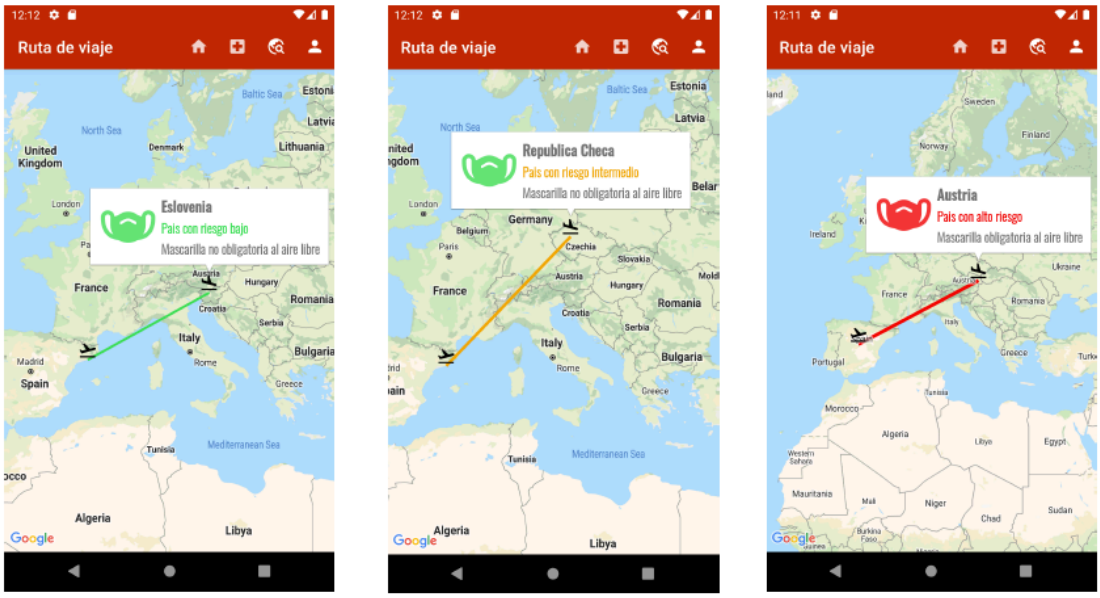

Ilustración 128. Vista del mapa de Ruta de Viaje

6.1.10. Sobre mí

En esta vista podremos observar información sobre el autor de la aplicación

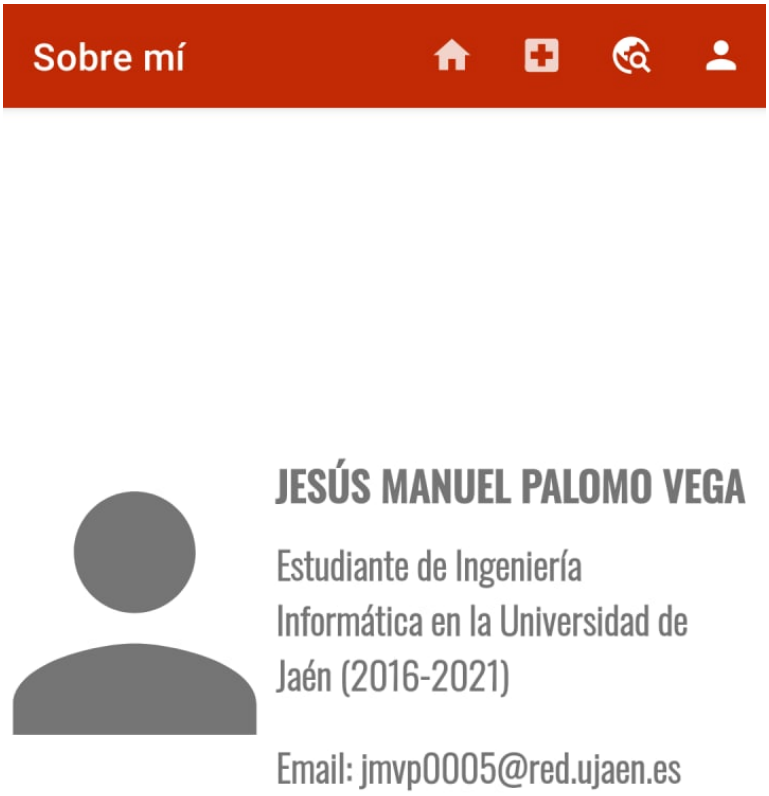

Ilustración 129. Vista de la pantalla Sobre Mí

7. Bibliografía

[1] https://cincodias.elpais.com/cincodias/2021/04/29/companias/1619716587_988266.html [Último acceso el 4 de Septiembre de 2021]
[2] https://yiminshum.com/mobile-movil-mundo-2021/ [Último acceso el 5 de Septiembre de 2021]
[3] https://www.android.com/intl/es_es/ [Último acceso el 8 de Septiembre de 2021]
[4] https://www.businessinsider.es/20-empresas-dinero-han-ganado-pandemia-covid-19-663015 [Último acceso el 8 de Septiembre de 2021]
[5] https://openwebinars.net/blog/conoce-las-3-metodologias-agiles-mas-usadas/ [Último acceso el 8 de Septiembre de 2021]
[6] https://developer.android.com/studio [Último acceso el 1 de Septiembre de 2021]
[7] https://www.eclipse.org [Último acceso el 1 de Septiembre de 2021]
[8] https://www.silicon.es/android-studio-el-ide-de-google-que-sustituye-a-eclipse-36919 [Último acceso el 1 de Septiembre de 2021]
[9] https://flutter.dev [Último acceso el 1 de Septiembre de 2021]
[10] https://developer.android.com/docs [Último acceso el 9 de Septiembre de 2021]
[11] https://www.mysql.com [Último acceso el 1 de Septiembre de 2021]
[12] https://www.apachefriends.org/es/index.html [Último acceso el 1 de Septiembre de 2021]
[13] https://notepad-plus-plus.org [Último acceso el 1 de Septiembre de 2021]
[14] https://www.mockflow.com [Último acceso el 1 de Septiembre de 2021]
[15] https://www.visual-paradigm.com [Último acceso el 1 de Septiembre de 2021]
[16] https://www.tomsplanner.es [Último acceso el 1 de Septiembre de 2021]
[17] http://www.pmoinformatica.com/2017/02/requerimientos-funcionales-ejemplos.html [Último acceso el 2 de Septiembre de 2021]
[18] http://www.pmoinformatica.com/2015/05/requerimientos-no-funcionales-ejemplos.html [Último acceso el 2 de Septiembre de 2021]
[19] https://developer.android.com/training/volley?hl=es [Último acceso el 7 de Septiembre de 2021]
[20] https://material.io/develop/android [Último acceso el 7 de Septiembre de 2021]
[21] https://icon-icons.com/es/icono/virus-covid19-viajes-coronavirus-avión/141745 [Último acceso el 1 de Septiembre de 2021]
[22] https://www.develou.com/android-studio-proyecto-en/ [Último acceso el 1 de Septiembre de 2021]
[23] https://developer.android.com/guide/components/activities/activity-lifecycle?hl=es [Último acceso el 1 de Septiembre de 2021]
[24] https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419 [Último acceso el 2 de Septiembre de 2021]
[25] https://openwebinars.net/blog/que-es-gradle/ [Último acceso el 2 de Septiembre de 2021]
[26] https://es.wikipedia.org/wiki/Fórmula_del_semiverseno [Último acceso el 4 de Septiembre de 2021]
[27] https://www.payscale.com [Último acceso el 4 de Septiembre de 2021]

[28] https://es.statista.com/estadisticas/493856/pronostico-de-usuarios-de-smartphone-en-espana/ [Último acceso el 6 de Septiembre de 2021]
[29] https://www.epdata.es/datos/turismo-residente-datos-graficos/249 [Último acceso el 6 de Septiembre de 2021]
[30] https://es.wikipedia.org/wiki/Android
[31] https://www.skyscanner.es
[32] https://www.kayak.es
[33] https://www.java.com/es/