UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Diseño e implementación de un simulador de rutas para dispositivos Android. Aplicación RouteFlyerDroid

Trabajo Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

AUTOR/A: Muñoz Fuentes, Carlos

Tutor/a: Rodas Jordá, Ángel

CURSO ACADÉMICO: 2021/2022

Diseño e implementación de un simulador de rutas aéreas para dispositivos Android. Aplicación RouteFlyerDroid

RouteFlyerDroid

Autor
Carlos Muñoz Fuentes

Tutor/es
Ángel Rodas Jordá
ETSINF - Dpto. de Informática de Sistemas y Computadores

Máster en Ingeniería Aeronáutica

Valencia, 4 de Septiembre de 2022

Agradecimientos

Desde que decidí el tema del proyecto supe que iba a ser una experiencia muy enriquecedora, pero que iba a precisar de un alto tiempo de aprendizaje. Sin embargo, en todo momento Ángel ha sabido guiarme en un ámbito que para mí era desconocido, entregándose al proyecto con una dedicación extraordinaria. Tanto por el tiempo invertido como por su entusiasmo a la hora de ayudarme a progresar, quiero expresar mi más sincero agradecimiento a mi tutor Ángel Rodas Jordá.

Quiero agradecer a mi familia, a mis padres y a mi hermano, por escucharme y apoyarme incondicionalmente durante el desarrollo de este proyecto, así como durante toda mi etapa universitaria. Agradecer también a mis amigos, por haberme motivado a seguir desarrollando la aplicación, así como a desconectar cuando lo he necesitado.

Este proyecto pone fin a seis años dedicados al estudio de la Ingeniería Aeronáutica, los cuales me han ayudado a crecer tanto académica como personalmente. Gracias a todas las personas que han formado parte de esta etapa.

Life is trying things to see if they work

Ray Bradbury

Resumen

El presente Trabajo Final de Máster tiene como objetivo el desarrollo de una aplicación Android (RouteFlyerDroid) que permite simular rutas aéreas entre más de 14000 aeródromos a nivel mundial, haciendo uso del lenguaje de programación Java. Esta aplicación genera rutas destinadas al vuelo instrumental y permite simular aeronaves que las recorren, monitorizando sus principales parámetros tanto de manera gráfica como numérica. La interfaz de usuario se basa en un mapa del mundo, extraído de librerías correspondientes al proyecto colaborativo de mapas libres y editables OpenStreetMap, sobre el cual se muestran los aeródromos, rutas y las propias aeronaves simuladas. Asimismo, mediante diversos menús de navegación se permite al usuario interactuar con todos los elementos implicados en las rutas. Esto engloba tanto la posibilidad de configurar con precisión las simulaciones previamente a su inicio, como la consulta en tiempo real de gran cantidad de parámetros de las aeronaves, los aeródromos y las propias rutas. Si bien la finalidad del proyecto no está enfocada a un uso aeronáutico profesional, se tiene como objetivo que los elementos simulados en la aplicación se ajusten en lo posible a la realidad.

Palabras clave: Android, aplicación, java, simulador de vuelos; aeronavegación, OpenStreetMap.

Abstract

The objective of this Master's Final Project is the development of an Android application, called RouteFlyerDroid, that allows to simulate air routes between more than 14000 airfields worldwide, using the Java programming language. This application generates routes for instrument flight and simulates aircraft that fly them, monitoring their main parameters both graphically and numerically. The user interface is based on a map of the world, extracted from libraries corresponding to the OpenStreetMap collaborative project of free and editable maps, on which the airports, routes and the simulated aircraft themselves are shown. Likewise, navigation menus allow the user to interact with all the elements involved in the routes. This includes both the possibility of accurately configuring the simulations prior to their start and the real-time consultation of numerous aircraft, airport and route parameters. Although the aim of the project is not focused on professional aeronautical use, the objective is to ensure that the elements simulated in the application are as close to reality as possible.

Keywords: Android, app, Java, flight simulator, air navigation, OpenStreetMap.

Índice general

Índice de figuras

Índice de tablas

1. Introducción

El propósito de este Capítulo es describir de una manera general el proyecto realizado, así como exponer los motivos que llevaron a su realización y los objetivos que se plantearon durante su desarrollo.

1.1. Descripción del proyecto

Este proyecto consiste en el desarrollo de una aplicación aeronáutica para Android cuyo principal objetivo es la simulación de vuelos instrumentales entre aeródromos a nivel mundial. Todo ello se lleva a cabo mediante una interfaz fundamentada en un mapa, sobre el que se generan representaciones 2D de todos los elementos implicados, tales como aviones, rutas, aeródromos y puntos de ruta.

Se trata de una aplicación con diversas funcionalidades que permiten al usuario la representación y configuración de innumerables rutas áreas reales extraídas desde bases de datos de navegación, así como la simulación y parametrización de aviones que las recorren. Tanto los parámetros de vuelo de las aeronaves como las características de navegación de la ruta son configurables por el usuario previamente al inicio de las simulaciones. Asimismo, la aplicación incorpora y procesa una amplia base de datos de aeronavegación, de la que se extrae información de más de 14000 aeródromos a nivel mundial. Esta estructura de datos de aeródromos permite al usuario la consulta de sus principales características, así como su selección como puntos de origen y destino de las simulaciones.

Como aspecto fundamental de la aplicación se destaca la creación de mapas de tráfico, puesto que esta permite la simulación y monitorización simultánea de diversas aeronaves. Estas aeronaves pueden ser monitorizadas durante todo su tiempo de simulación, tanto de manera gráfica en el mapa como de manera numérica mediante la consulta de sus principales parámetros a través de pestañas desplegables. Esta monitorización de los parámetros de vuelo de las aeronaves se acompaña también de la posibilidad de interaccionar con las mismas, tanto pausando y reanudando sus simulaciones como pudiendo eliminar las aeronaves del mapa. Asimismo, se incorpora la posibilidad de que el usuario interrelacione la simulación de dos aeronaves mediante la consulta de la distancia entre ambas durante su vuelo.

En síntesis, las diversas funcionalidades incorporadas en esta aplicación dotan al usuario de gran cantidad de posibilidades para la simulación de aeronaves y monitorización de tráfico aéreo simulado, el cual busca asemejar su comportamiento al que podrían tener aeronaves reales. Además, se ofrece la posibilidad de consultar información de más de 14.000 aeródromos a nivel mundial. La aplicación incluye además otras funcionalidades extra, enfocadas a ofrecer al usuario cierta personalización del mapa y la representación de los elementos implicados en las simulaciones. Todas estas funcionalidades se expondrán con más detalle en la Sección 4.1.

Antes de continuar con la descripción de otros aspectos del proyecto, cabe destacar que durante la presente memoria se va a utilizar el término aeródromo para referirse a todo elemento extraído de la base de datos de navegación. Estos incluyen desde emplazamientos destinados a vuelos recreativos hasta aeropuertos comerciales a nivel mundial. Asimismo, los términos aeronave y avión se van a emplear indistintamente para referirse a los aviones simulados en la aplicación.

1.2. Alcance del proyecto

A la hora de definir el propósito del proyecto se ha tenido en cuenta en primera instancia la necesidad de aprendizaje por parte del alumno acerca del desarrollo de aplicaciones móviles, así como la necesidad de profundizar en los conocimientos de Java adquiridos durante los estudios de Máster en Ingeniería Aeronáutica. Si bien es cierto que al inicio del proyecto se partía de una base de conocimiento del lenguaje Java adquirida durante el desarrollo de la asignatura *Sistemas de Gestión de Vuelo por Computador [33584]*, se carecía de conocimientos sobre el desarrollo de aplicaciones para dispositivos Android, puesto que esta materia no formaba parte del plan de estudios del Máster. Es por ello que se previó un tiempo considerable para el aprendizaje de todo lo relativo a la programación enfocada a Android y más concretamente para la familiarización con el software de desarrollo Android Studio. Esto supuso que no se fijase como objetivo el desarrollo de una aplicación de carácter profesional, sino más bien de carácter académico, sin que esto impidiese que pudiese estar enfocada a todo tipo de público interesado en el ámbito aeronáutico y más concretamente en el campo de la aeronavegación.

Un punto importante a destacar, que ha influido en gran medida en el alcance del proyecto, es que el alumno partía de un código base desarrollado por el tutor, el cual ofrecía gran cantidad de posibilidades de desarrollo y daba lugar a la creatividad del alumno a la hora de implementarlo en una aplicación Android. Este código, que se ha ido modificando durante el desarrollo de la app, contenía todo lo relativo al ámbito

aeronáutico. Es decir, se partía de un paquete en lenguaje Java que contenía diversas clases y métodos básicos para la simulación de aviones, rutas, aeródromos y mapas de tráfico, entre otros elementos. Por su parte, el alumno tuvo desde el primer momento la libertad de manipular el código e implementar las modificaciones y mejoras pertinentes, así como adaptarlo a una representación gráfica y numérica dentro de un entorno de desarrollo de aplicaciones móviles.

A pesar del tiempo dedicado al aprendizaje del desarrollo de apps mediante Android Studio y el propio del progreso del alumno en los conocimientos del lenguaje Java, el alcance del proyecto fue incrementando su ambición a medida que se iban logrando avances en el desarrollo de la aplicación. Concretamente, como propuesta base del proyecto se sugirió la implementación del código aeronáutico en un mapa que permitiese simular tanto rutas directas entre aeródromos como rutas preestablecidas extraídas desde bases de datos de navegación, siendo tanto las rutas como los aeródromos elementos almacenados en la aplicación mediante formato .txt. Sin embargo, a medida que la aplicación se fue desarrollando, el alcance del proyecto continúo incrementándose. Este nuevo alcance incluyó funcionalidades tales como la simulación de cualquier ruta que el usuario precise, la incorporación de bases de datos de aeródromos y rutas, la posibilidad de almacenar las rutas generadas y la consulta de gran cantidad de información de los aeródromos, entre otras mejoras incorporadas.

Resulta fundamental destacar en este apartado que desde el primer momento en el que se plantearon los requisitos de desarrollo de la app se descartó la idea de llevar a cabo un procesamiento interno de las rutas desde bases de datos de navegación. Este procesamiento requiere un gran trabajo de programación, que excede los objetivos impuestos para el desarrollo de la app. Es por ello que se prefirió centrar el foco en la simulación y navegación de las rutas, por encima del procesamiento o generación de las mismas. Este procesamiento se delegó desde un principio en software de generación de rutas aéreas de terceros, tal y como se expondrá en el siguiente capítulo.

Tras haber fijado los principales objetivos a conseguir durante el desarrollo de la app, se terminó concluyendo que el propósito de esta pudiese sobrepasar el puro ámbito académico. A pesar de que sigue tratándose de una aplicación para el uso recreativo enfocada a todo aquel que tenga interés en la aeronavegación, puede servir de utilidad para llevar a cabo aproximaciones de vuelos reales y monitorizar los distintos parámetros de las aeronaves a lo largo de estos, así como consultar información aeronáutica que pueda ser de interés para el usuario. Cabe destacar que no se contempla su uso comercial por dos motivos principales. Por un lado, el software de terceros empleado no es adecuado para uso comercial. Por otro lado, el código aeronáutico desarrollado tiene un enfoque académico y no es apto para uso profesional. Por último, durante el desarrollo de la app se han asumido ciertas hipótesis que deberían ser sustituidas por aproximaciones más cercanas a la aeronavegación real.

1.3. Motivación y objetivos

La principal motivación por parte del alumno para llevar a cabo este proyecto ha sido indudablemente la posibilidad de progresar en el aprendizaje del lenguaje Java, a la vez que iniciar un proceso de formación en el desarrollo de aplicaciones para dispositivos Android. Sin duda, el reto de diseñar una aplicación Android sin tener conocimientos previos de la materia resultó realmente motivador, puesto que se consideró una vía de aprendizaje sumamente útil en el contexto tecnológico actual. Asimismo, el contenido del proyecto se consideró atractivo desde un primer momento, tanto por tratar cuestiones de aeronavegación relacionadas con los estudios cursados como por ofrecer amplias posibilidades de creatividad en el desarrollo de la aplicación.

Durante las primeras semanas de desarrollo de proyecto, se propuso como objetivo primero el aprendizaje del entorno de desarrollo Android Studio, el cual trajo consigo la necesidad de adquirir conocimientos sobre aspectos informáticos propios de su uso. Entre otros muchos elementos, fue necesario aprender acerca del lenguaje XML, el software Gradle, el software GIT y otros aspectos de carácter gráfico, como por ejemplo el uso de vectores en formato SVG. Asimismo, resultó necesario el progreso en los conocimientos iniciales del lenguaje Java, puesto que a lo largo de los estudios de grado y máster únicamente se había utilizado durante las sesiones prácticas de la asignatura Sistemas de Gestión de Vuelo por Computador.

Tras la familiarización con el entorno de Android Studio se llevaron a cabo diversas pruebas de inyección de dependencias y uso de librerías, destacando el aprendizaje acerca de la librería OSMDroid (la librería para Android de OpenStreetMap), la cual se expone en el Capítulo 3. Una vez se realizaron las pruebas necesarias para la implementación del código base en el entorno Android y la interacción del mismo con el mapa ofrecido por la librería de OpenStreetMap, se procedió a fijar los objetivos relativos al desarrollo de la aplicación.

Como se ha anticipado, la idea que dio lugar al proyecto es el desarrollo de una aplicación Android en lenguaje Java que formase parte del ámbito de la aeronáutica, y más concretamente de la aeronavegación. Sin embargo, desde que se planteó esta idea hasta el fin del desarrollo de la aplicación, se ha ido concretando el tipo de aplicación a desarrollar. Para definir los objetivos de desarrollo se llevó a cabo un proceso de búsqueda de aplicaciones Android relacionadas con la aeronavegación. Además, se consultaron también páginas webs que ofrecieran servicios similares. Se llegó a valorar la posibilidad de monitorizar tráfico aéreo real, pero tras haber consultado otras apps y webs del mismo ámbito, se centró el foco del trabajo en la monitorización de aerona-

ves simuladas, puesto que se trata de un segmento sobre el cual no se ha encontrado software en los proveedores oficiales de Android. Si bien es cierto que existen diversas webs y aplicaciones de búsqueda y representación de rutas aéreas, la simulación de aeronaves y su monitorización no está apenas extendida en el mercado de aplicaciones Android. Es por ello que finalmente se planteó como objetivo principal el desarrollo de una aplicación de simulación de rutas reales que ofreciese gran cantidad de posibilidades al usuario, tratando de innovar con respecto al software actual del ámbito de la simulación de rutas aéreas.

2. Estado del arte

A la hora de fijar los objetivos del proyecto y determinar las funcionalidades que podía presentar la aplicación, ha resultado fundamental la búsqueda y análisis de software relativo a la aeronavegación. Tras investigar sobre los distintos servicios que se ofrecían en páginas web y aplicaciones Android gratuitas, se llegó a diversas conclusiones sobre el mercado de las aplicaciones de aeronavegación:

- Existe una gran cantidad de software enfocado a la monitorización de tráfico aéreo real. Tanto en formato web como en aplicaciones Android, se puede encontrar software profesional muy completo con amplias posibilidades para el usuario. Cabe destacar por encima del resto el software FlightRadar24 [1], tanto en su formato web como en el desarrollado para aplicaciones móviles.

- Otro de los campos ampliamente desarrollado en el sector de la aeronavegación es el de la simulación de vuelos 3D, enfocada a ofrecer una experiencia en primera persona como piloto de una aeronave tanto a público profesional como aficionado. Entre el amplio mercado de simuladores para Android se podría destacar una de las aplicaciones que cuenta con mayor número de descargas, y que destaca por su realismo: la app Airline Commander [2], desarrollada por Rortos. Sin embargo, la simulación 3D escapaba del alcance que se le dio inicialmente al proyecto.

- Existe otro sector de aplicaciones enfocado a ofrecer a pilotos no comerciales la posibilidad de personalizar al detalle sus propias rutas mediante bases de datos de navegación reales y cartas de navegación actualizadas, ofreciéndoles una herramienta de utilidad para generar sus propios planes de vuelo. Además, este tipo de aplicaciones suelen incluir seguimiento GPS para navegar en tiempo real. Algunos ejemplos de este tipo pueden ser Garmin Pilot [3], desarrollada por la conocida compañía Garmin, o SkyDemon [4], la cual se posiciona como la app para vuelos VFR (del inglés *Visual Flight Rules*) con mejor valoración media. Si bien es cierto que este sector no está tan desarrollado y podría tratarse de un segmento a explotar en el ámbito de las aplicaciones móviles, se prefirió enfocar los esfuerzos en el desarrollo del código base hacia otro segmento que no presente apenas software. Sin embargo, este tipo de funcionalidades se recogen como posible desarrollo futuro en el Capítulo 7.

- El segmento de generación y representación de rutas aéreas entre aeródromos obtenidas mediante el procesamiento de bases de datos de aeronavegación presenta

diversos servicios tanto en formato web como para dispositivos Android. Como se expondrá a continuación, en este segmento destaca la herramienta de generación de rutas ofrecida por SimBrief [5]. Sin embargo, estas aplicaciones están orientadas en su mayor parte al procesamiento y manipulación de bases de datos de navegación, generando como resultado rutas aéreas realistas que ocasionalmente se encuentran representadas sobre un mapa. En cambio, la idea de simular en dos dimensiones aeronaves que recorran estas rutas no se ha encontrado implementada en ninguna aplicación disponible en la plataforma de distribución oficial de Android [6]. Este hecho es uno de los motivos principales por los que se ha puesto el foco de la aplicación en la simulación 2D de rutas procesadas desde bases de datos de navegación aeronáutica.

Como se ha adelantado en el apartado sobre alcance del proyecto (Sección 1.2), la generación de rutas aéreas mediante el procesamiento de bases de datos de aeronavegación escapa de los objetivos del proyecto. Es por ello que se precisa de software de terceros que se encargue de esta tarea, con el fin de extraer un código de ruta que mediante un procesamiento interno se adapte posteriormente a la aplicación. En consecuencia, fue necesario llevar a cabo una búsqueda de los posibles servicios web a utilizar para obtener estas rutas. Algunos de estos servicios se exponen a continuación.

2.1. Páginas web con servicio de planificación de rutas aéreas

2.1.1. RocketRoute

RocketRoute [7], propiedad de la compañía AFV Partners LLC, es un servicio profesional de planificación de vuelos y navegación para la industria de la aviación. Según los datos que figuran en su sitio web, este servicio ha procesado más de un millón de rutas aéreas y tiene en torno a 150.000 usuarios registrados en todo el mundo.

Concebido inicialmente para pilotos privados, RocketRoute es un servicio de pago utilizado a día de hoy por operadores de aeronaves comerciales y compañías aéreas a nivel mundial. Además de su herramienta de planificación de vuelos, RocketRoute ofrece servicios de asesoramiento y seguimiento durante el vuelo. Para ello cuenta con una amplia plantilla que ofrece soporte técnico ininterrumpido a los pilotos. Se trata de una herramienta profesional basada en la nube que funciona tanto en su formato web como en dispositivos móviles y tabletas.

A pesar de tratarse de un servicio de pago, ofrece una prueba gratuita de 14 días en la que se puede utilizar su depurada herramienta de planificación de rutas. La Figura

2.1 muestra un ejemplo de generación de ruta desde su herramienta web.

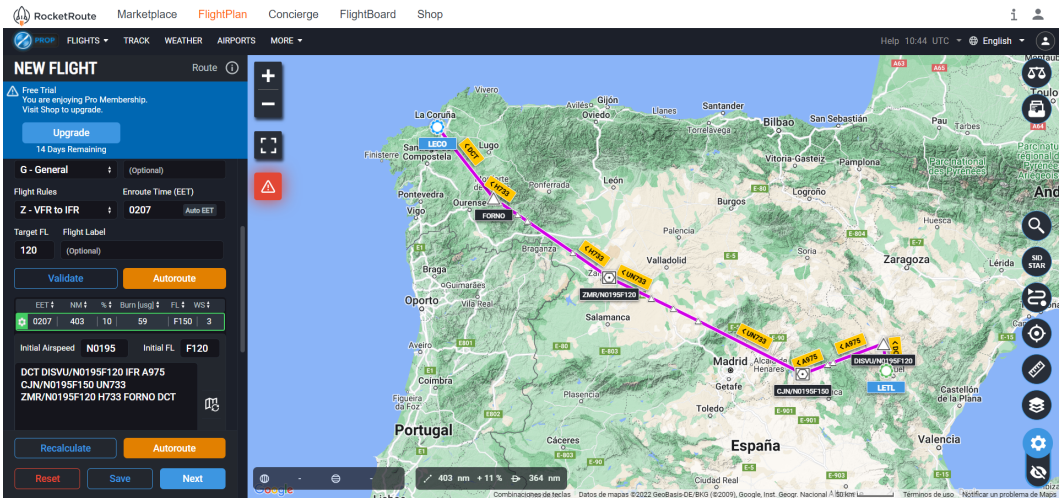


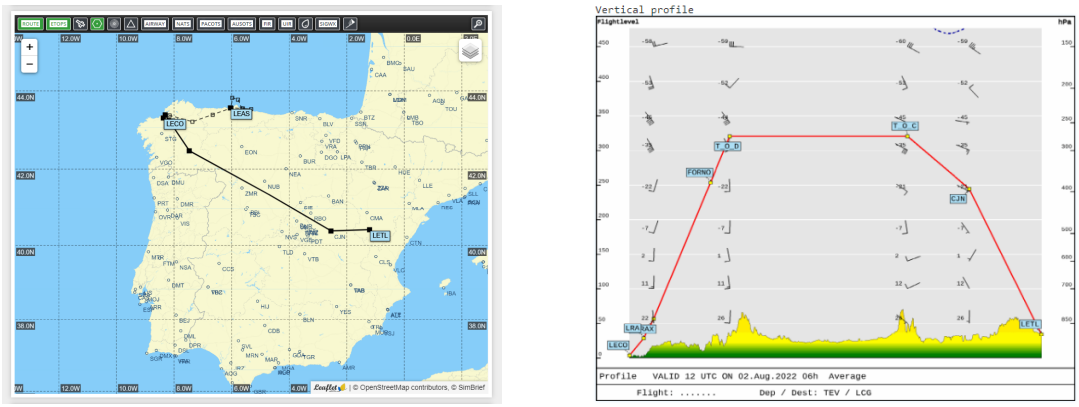
Figura 2.1: Ejemplo de ruta generada por RocketRoute

Además, cuenta con una gran variedad de opciones para exportar las rutas generadas, facilitando así múltiples posibilidades para el procesamiento por parte de software externo. Sin embargo, al tratarse de un programa de pago, se ha preferido optar por otras opciones de carácter gratuito que se adapten mejor a los requisitos de la app. Otra de las razones que hacen preferible otro tipo de software es que el amplio abanico de posibilidades que ofrece esta herramienta supera el alcance impuesto en lo relativo a las características de la ruta generada por la aplicación. Asimismo, la generación y descarga de rutas en segundo plano resulta una tarea compleja, de igual manera que ocurre con el siguiente candidato.

2.1.2. SimBrief

Simbrief es la herramienta gratuita de generación de planes de vuelo instrumentales por excelencia. Con este servicio es posible generar planes de vuelo precisos que se asemejan con gran exactitud a los producidos por software profesional.

Esta herramienta cuenta con gran cantidad de utilidades que describen al detalle el plan de vuelo generado. No solo es posible obtener una ruta acorde a bases de datos de navegación actualizadas y a la meteorología actual, sino que se obtiene información específica para cada aeronave relativa a parámetros fundamentales como su performance, sus pesos, sus limitaciones y su consumo. Además, no solo genera un diseño 2D de la ruta (Figura 2.2a), sino que ofrece un perfil vertical de ruta realista que incorpora la intensidad de los vientos y el perfil de terreno a lo largo del recorrido (Figura 2.2b).



(a) Representación de la ruta

(b) Perfil vertical de ruta

Figura 2.2: Ejemplo de ruta generada mediante SimBrief

Se trata de un software desarrollado por NaviGraph [8], una empresa especializada en la distribución y desarrollo de cartas aeronáuticas profesionales a nivel mundial, enfocadas al sector de la simulación 3D de vuelos.

Las rutas generadas desde Simbrief permiten la descarga de un informe completo que contiene una enorme cantidad de información relativa tanto a la ruta como a la actuación del avión, pasando por gran variedad de detalles de carácter meteorológico. Sin duda se trata de un servicio que ofrece resultados muy completos que podrían ser implementados en la aplicación desarrollada (como se recoge en el Capítulo 7). Sin embargo, el formato de los datos de salida requiere un esfuerzo de procesamiento mayor para poder acoplar sus resultados al código base. Asimismo, la tarea de automatizar la salida de datos desde Simbrief se presenta compleja en comparación con la que se tiene a la hora de utilizar la herramienta seleccionada: RouteFinder.

2.1.3. RouteFinder

Se trata de la opción elegida para implementar en el presente proyecto. Tanto por su fiabilidad como por su sencillez, se ha posicionado como la opción más viable para la extracción de rutas desde bases de datos de aeronavegación.

Este servicio web [9] consiste un planificador rápido de rutas para vuelo instrumental IFR (del inglés *Instrumental Flight Rules*) con algunas herramientas asociadas para ayudar a los pilotos y a los operadores de vuelo en su trabajo diario. Tanto la base de datos de aeronavegación como las tecnologías empleadas son proporcionadas por

la compañía ASA srl [10], un proveedor italiano de servicios de planes de vuelo por ordenador para pilotos privados y operadores de aviones comerciales.

La cobertura de los datos ofrecida por RouteFinder es mundial, pero el sistema tiene disposiciones especiales para ayudar a trabajar con rutas IFR en Europa.

Este servicio se encuentra en línea desde 1997. En su origen, este estaba destinado a proporcionar rutas aéreas básicas para el entretenimiento personal con simuladores de vuelo para ordenador. Sin embargo, a día de hoy es lo suficientemente flexible como para ser una buena ayuda para la confección rápida de vuelos y la creación de rutas IFR válidas.

La Figura 2.3 muestra el menú de opciones que ofrece la versión gratuita, el cual permite elegir el aeródromo de origen y destino, así como una serie de parámetros relativos a la ruta (versión de la base de datos, uso de SIDs o STARs o nivel de vuelo, entre otras opciones).

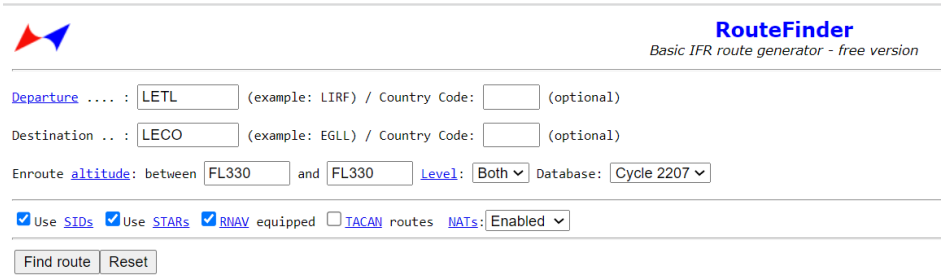


Figura 2.3: Interfaz de la versión gratuita de RouteFinder

Una vez generada la ruta, se muestra un breve informe de la misma (Figura 2.4). De este informe se procesa en la aplicación tanto la tabla con cada uno de los puntos de la ruta como el código de ruta mostrado en la última línea.

Cabe destacar que la versión gratuita de RouteFinder no genera el perfil de altitudes de la ruta. Sin embargo, debido a que dentro de la aplicación el usuario puede modificar una serie de parámetros de vuelo que pueden afectar al mismo, resulta interesante llevar a cabo un procesamiento interno para generar un perfil vertical propio.

RouteFinder - http://rfinder.asalink.net
Basic IFR route generator - ***not for actual navigation***
Commercial usage is forbidden. Automated queries: only by prior arrangement.
(C)1997-2021 ASA srl - Italy

NAT: Eastbound track message identification is 214
NAT: Westbound track message identification is 214
Computed route from **TERUEL** (LETL, LE) to **A CORUNA** (LECO, LE): 15 fixes, 571.1 nautical miles

Cruise altitude between FL330 and FL330
LETL (0.0nm) -DCT-> **GARVU** (112.5nm) -UN10-> **VASUM** (119.3nm) -UN10->
ALEPO (124.1nm) -UN10-> **NOLSA** (129.7nm) -UN10-> **PPN** (150.2nm) -UP152->
PODUX (163.0nm) -UP152-> **BAGAS** (178.1nm) -UP152-> **BAPOR** (223.0nm) -UP152->
BADRU (227.1nm) -UP152-> **NENEM** (251.4nm) -DCT-> **INCEF** (368.5nm) -DCT->
LOTEE (432.1nm) -A5-> **MEGAT** (535.1nm) -STAR-> **LECO** (571.1nm)

Details:

ID	FREQ	TRK	DIST	Coords	Name/Remarks
LETL		0	0	N40°24'43.00" W001°13'03.00"	TERUEL
GARVU		340	113	N42°10'15.99" W002°04'40.00"	GARVU
VASUM		26	7	N42°16'18.99" W002°00'39.99"	VASUM
ALEPO		26	5	N42°20'36.99" W001°57'48.00"	ALEPO
NOLSA		26	6	N42°25'39.00" W001°54'27.00"	NOLSA
PPN	112.3	26	21	N42°44'02.00" W001°42'07.00"	PAMPLONA
PODUX		322	13	N42°54'04.00" W001°52'53.99"	PODUX
BAGAS		322	15	N43°05'54.00" W002°05'41.00"	BAGAS
BAPOR		322	45	N43°40'59.99" W002°44'14.99"	BAPOR
BADRU		321	4	N43°44'11.00" W002°47'48.00"	BADRU
NENEM		321	24	N44°03'05.00" W003°09'00.99"	NENEM
INCEF		320	117	N45°31'06.00" W004°57'38.00"	INCEF
LOTEE		216	64	N44°39'31.99" W005°50'12.00"	LOTEE
MEGAT		229	103	N43°29'56.00" W007°35'46.99"	MEGAT
LECO		252	36	N43°18'07.00" W008°22'38.00"	A CORUNA

Tracks are magnetic, distances are in nautical miles.

LETL DCT **GARVU** UN10 **PPN** UP152 **NENEM** DCT **INCEF** DCT **LOTEE** A5 **MEGAT** STAR **LECO**

[Back]

Figura 2.4: Ejemplo de ruta generada por RouteFinder

2.2. Aplicaciones Android para la generación de rutas aéreas

De igual modo que se ha investigado acerca de los servicios web disponibles para la generación de rutas reales extraídas desde bases de datos de navegación reales, se ha llevado a cabo una búsqueda similar dentro del mercado de aplicaciones Android. Como se ha adelantado, si bien se han encontrado diversas apps para la generación y representación de rutas de acuerdo a bases de datos de navegación, no se ha conseguido dar con ninguna que incorpore la simulación de aeronaves que recorran esas rutas. De cualquier forma, se procede a exponer varios ejemplos de aplicaciones que permiten al usuario generar y visualizar rutas reales.

2.2.1. Flight Planner

Flight Planner [11] ofrece un servicio similar a la web Route Finder, con la particularidad de que representa la ruta sobre un mapa (Figura 2.5) y ofrece la posibilidad de exportarla para su uso en diferentes simuladores de vuelo. Se trata de una app sencilla y fácil de usar, lo que junto a la fiabilidad de las rutas generadas la posiciona como una de las aplicaciones con mejor valoración dentro de este segmento.

Sin duda se trata de una aplicación cuya interfaz y funcionamiento se asemejan en gran medida al objetivo del presente proyecto. Sin embargo, de igual manera que ocurre con las apps que se exponen a continuación, únicamente se centra en la propia ruta, sin extender sus funcionalidades a la simulación.

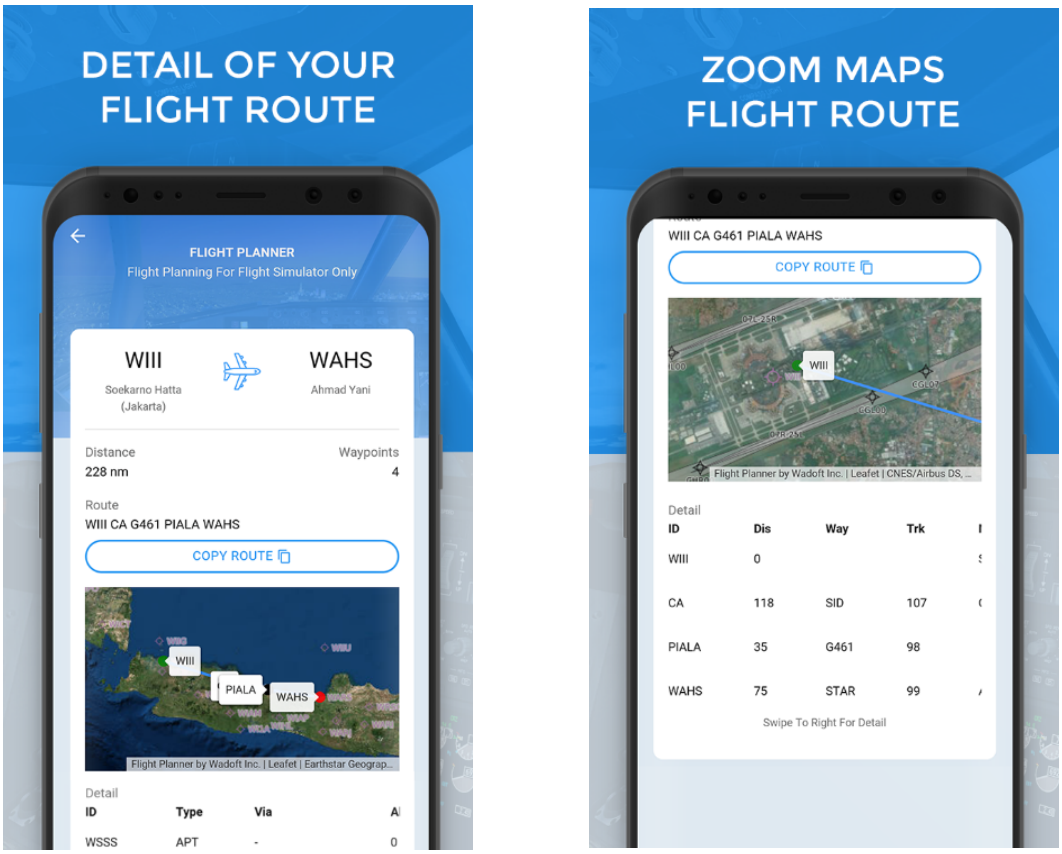


Figura 2.5: Ejemplo de ruta generada por Flight Planner

2.2.2. RocketRoute FlightPlan

Además de su servicio web, RocketRoute tiene su propia aplicación móvil para dispositivos Android [12], la cual posee gran variedad de prestaciones.

Esta aplicación, destinada principalmente al uso profesional, permite tanto la creación de rutas personalizadas entre dos aeródromos como la generación de rutas mediante el procesamiento de bases de datos de navegación (Figura 2.6). Además, la personalización de las rutas ofrece el usuario una gran versatilidad, puesto que este puede elegir libremente cualquier localización del mapa como punto de ruta o bien elegir entre alguno de los puntos de ruta reales procesados desde bases de datos.

Se trata de un servicio aplicable tanto para vuelos VFR como IFR, que ofrece a los pilotos la opción de navegar la ruta en tiempo real gracias al uso de localización GPS. Adicionalmente, RocketRoute FlightPlan incluye muchas otras prestaciones como cálculos de consumo de combustible, consulta de su base de datos de aeródromos y acceso a cartas de navegación aeronáutica e información meteorológica actualizada.

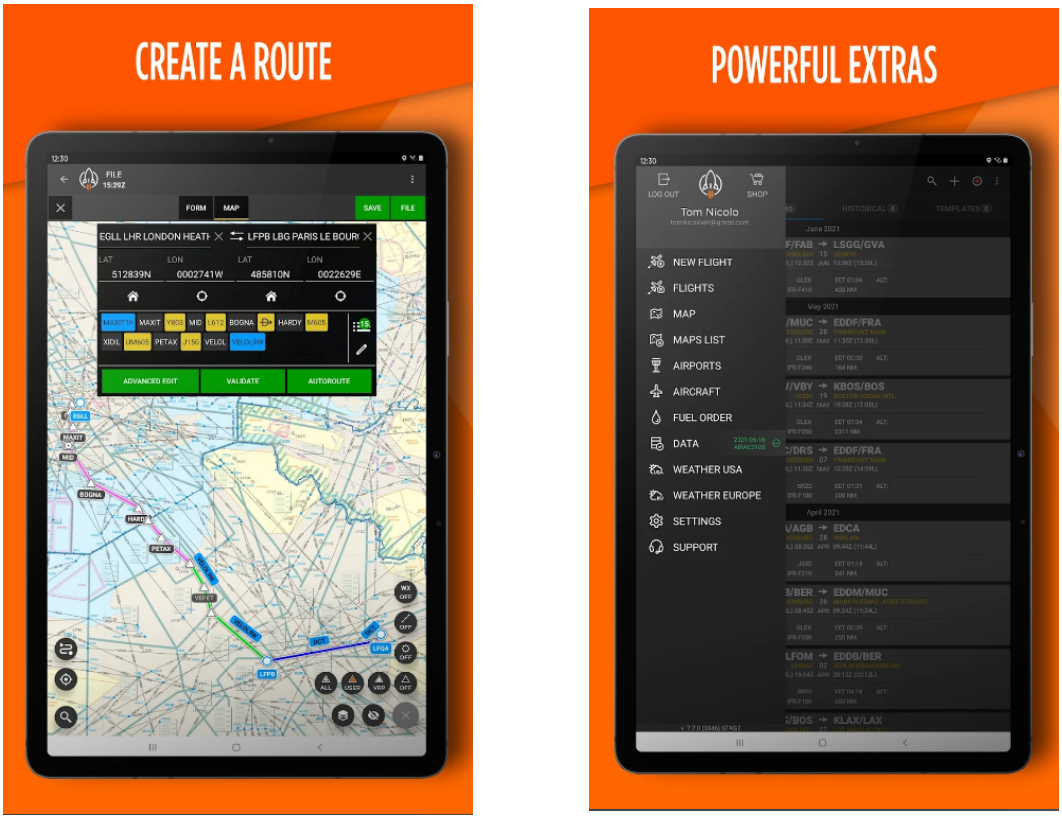


Figura 2.6: Creación de un plan de vuelo con RocketRoute

2.2.3. SkyDemon

Divelements Limited, compañía al cargo de la app SkyDemon [4], define su software como el número uno a nivel mundial en términos de planificación y navegación VFR. Esta herramienta de planificación de rutas, permite realizar todas las tareas relacionadas con la elaboración del plan de vuelo, incluyendo la integración de previsiones meteorológicas, NOTAMs e información de aeródromos, además de facilitar las tareas de cálculo de combustible, carga y centrado.

Con más de 100000 descargas en la plataforma de distribución oficial de Android y la valoración media más alta dentro de su segmento, SkyDemon se sitúa como una de las aplicaciones Android favoritas para los pilotos que lleven a cabo vuelos VFR. Durante la navegación (Figura 2.7), el piloto es informado en tiempo real sobre las posibles restricciones de vuelo (por limitaciones de espacio aéreo, meteorología u otros aviones), los desvíos inesperados y los posibles cambios de ruta. Sin duda se trata de una herramienta de navegación muy completa cuyo fuerte es la navegación en tiempo real de las rutas generadas.

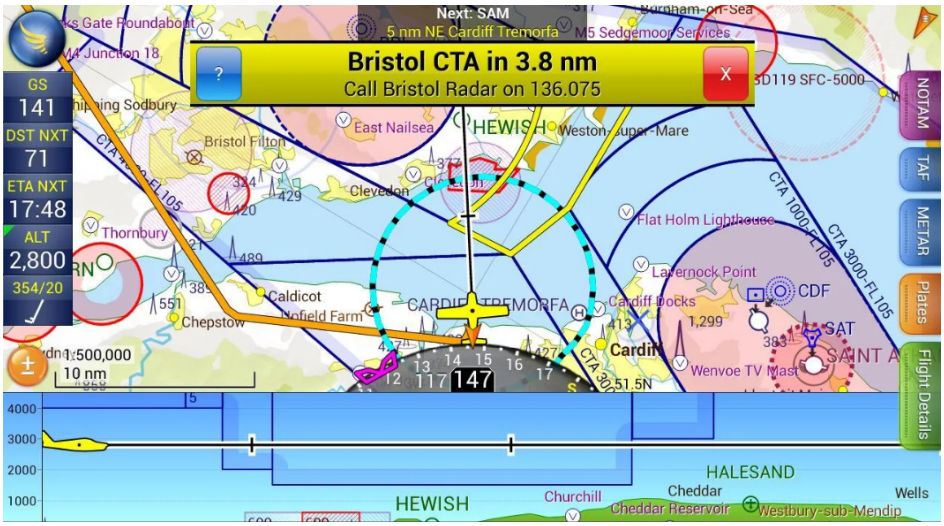


Figura 2.7: Ejemplo de navegación VFR con SkyDemon

3. Contexto tecnológico

En el presente capítulo se describen los lenguajes, servicios, tecnologías y entornos de desarrollo utilizados para llevar a cabo la aplicación. Como se ha anticipado, se trata de una app desarrollada en lenguaje Java para dispositivos Android. Además de estos dos conceptos, será necesario describir una serie de servicios, librerías y otro software que ha sido necesario para el desarrollo de la app.

3.1. Java

Además de dar nombre a una plataforma de computación, Java [13] es conocido por tratarse de un lenguaje de programación de propósito general capaz de acometer todo tipo de proyectos y ejecutarse en múltiples plataformas, siendo a día de hoy uno de los más populares y con mayor número de aplicaciones en el sector de la programación.

Java es un lenguaje de programación orientado a objetos y concurrente. Una de sus principales virtudes es que se trata de un lenguaje multiplataforma, lo que permite que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo. Es decir, el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra. Esto conlleva la necesidad de ejecutar sus programas en una máquina virtual de Java (JVM, del inglés *Java Virtual Machine*) tras haber sido compilados a *bytecode* (un código intermedio más abstracto que el código máquina).

La JVM es una máquina virtual de proceso nativo (capaz de ejecutarse en una plataforma específica), la cual interpreta y ejecuta instrucciones expresadas en bytecode. Esto permite al lenguaje Java su portabilidad en los distintos sistemas operativos, puesto que una vez compilado a bytecode podrá ser compilado en la JVM correspondiente para cada uno de ellos (Figura 3.1).

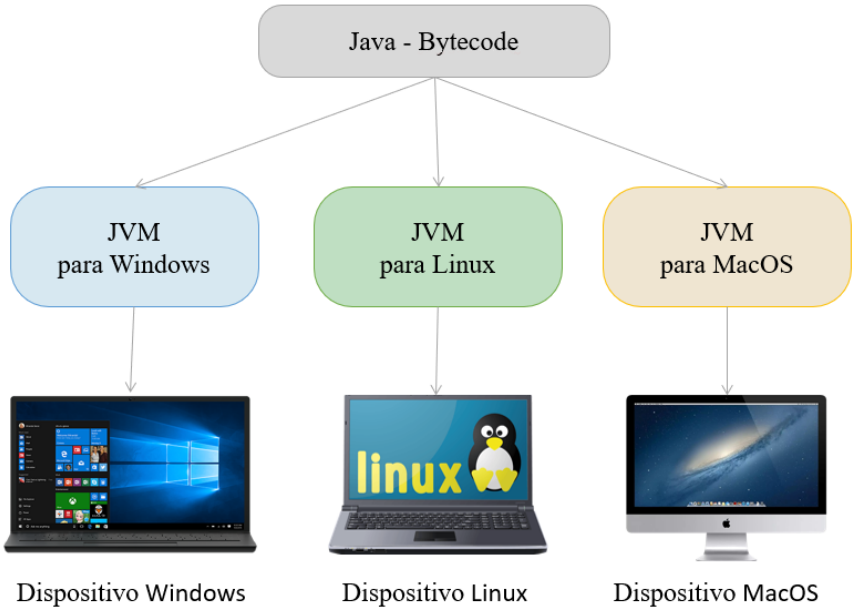


Figura 3.1: Arquitectura general de un programa en ejecución en una Máquina Virtual Java

Con Java se pueden llevar a cabo prácticamente todo tipo de proyectos, desde aplicaciones o servicios web a aplicaciones de escritorio de consola o interfaz gráfica, pasando por el desarrollo de aplicaciones móviles. Este último punto es en el que se centra el presente proyecto. Y es que Java es el lenguaje utilizado para el desarrollo nativo de aplicaciones Android, lo que ha hecho que su popularidad se haya incrementado notablemente los últimos años y que su demanda profesional sea elevada.

A modo de síntesis, algunas de las características principales del lenguaje Java son:

- Lenguaje simple pero potente
- Portable o multiplataforma
- Orientado a objetos
- Distribuido
- Multihilo
- Dinámico

3.2. Android

Android [14] es un sistema operativo (SO) de código abierto desarrollado por Google, basado en el núcleo Linux. Aunque se trate de un SO diseñado específicamente para dispositivos móviles, en los últimos años se está potenciando su desarrollo para televisiones, relojes inteligentes y automóviles, entre otros dispositivos.

Este SO de código abierto permite el desarrollo de aplicaciones a terceros (desarrolladores ajenos a Google). Con este fin se proporcionan un conjunto completo de APIs (del inglés *Application Programming Interfaces*) y herramientas de desarrollo, depuración, compilación y emulación. Asimismo, Google ofrece su propio sitio enfocado a ayudar a los programadores en el desarrollo de sus apps Android [15].

Android se basa en el concepto de máquina virtual de Java. Sin embargo, teniendo en cuenta sus limitaciones, Google decidió crear una nueva específica para su SO: la máquina virtual Dalvik. Esta máquina virtual está optimizada para su uso en aplicaciones Android, ahorrando la máxima memoria posible mediante el uso de registros y la delegación de operaciones al Kernel. A pesar de que no haga uso de la JVM, contiene la mayoría de las librerías propias del lenguaje Java, incluyendo además un conjunto de librerías en C/C++ necesarias para diversos componentes propios de Android.

Algunas de las principales características de este SO son:

- Plataforma abierta
- Adaptable a todo tipo de hardware
- Portabilidad asegurada
- Incorpora gran cantidad de servicios (GPS, navegador, bases de datos...)
- Buen nivel de seguridad
- Optimizado para trabajar con baja potencia y poca memoria
- Alta calidad de sonido y gráficos

3.3. XML

A la hora de desarrollar la parte gráfica de una aplicación Android, y más concretamente utilizando el entorno de desarrollo nativo Android Studio, resulta fundamental familiarizarse con el lenguaje XML.

XML, siglas del inglés *Extensible Markup Language*, puede definirse como un lenguaje de marcado que define un conjunto de reglas de codificación de documentos en un formato que es legible tanto por la máquina como por el hombre. Algunas de sus características son:

- Es el lenguaje utilizado por Android para el desarrollo de sus interfaces gráficas
- Es basado en texto
- Puede definir otro lenguaje de marcado
- Su estructura es autodescriptiva
- Proporciona una estructuración lógica del documento
- Permite intercambio de información entre varias plataformas

3.4. Librería OSMDroid

El pilar fundamental de una aplicación de simulación de vuelos 2D es ciertamente un mapa sobre el cual llevar a cabo la visualización de las aeronaves. A la hora de decidir qué proveedor de mapas era preferible para llevar a cabo el proyecto, se planteó en primer lugar el uso del servicio de mapas nativo Google Maps. Sin embargo, tras valorar otras opciones de acceso libre, se decidió optar por implementar librerías que hiciesen uso de la plataforma de mapas OpenStreetMap [16]. En concreto, las librerías utilizadas en el proyecto son OSMDroid [17] y su ampliación OSMBonusPack [18]. Uno de los principales motivos de esta decisión fue la posibilidad de tener mayor independencia de Google, puesto que se trata de software libre. Además, a lo largo de la asignatura Sistemas de Gestión de Vuelo por Computador se había hecho uso de librerías de OpenStreetMap para Windows, por lo que se partía de un conocimiento previo.

OpenStreetMap es un proyecto colaborativo para crear mapas editables y libres para todo desarrollador o público en general que quiera hacer uso de ellos. Los mapas se crean utilizando información geográfica publicada con una licencia de contenido abierto, la cual es capturada con dispositivos GPS móviles, ortofotografías y otras fuentes que son generadas por voluntarios de todo el mundo. Se trata de un proyecto de mapeo comunitario abierto que no abarca la totalidad del terreno a nivel mundial, puesto que se inició desde cero en el año 2004 y todavía se encuentra en desarrollo.

OSMDroid es una librería que reemplaza a la clase MapView (v1 API) de Android, la cual utiliza servicios de Google Maps. Esta incluye un sistema modular de proveedores de texturas con soporte para numerosas fuentes en línea y fuera de línea y soporte de

superposición para el trazado de iconos, así como el seguimiento de la ubicación y el dibujo de formas. Asimismo, esta librería presenta un amplio abanico de clases para la representación de elementos en el mapa, tales como iconos, marcadores, campos de texto, líneas, polígonos, etc. Se trata de una librería que, a pesar de ser completamente gratuita, abre la puerta a realizar gran variedad de aplicaciones de manejo de mapas, como es el caso del presente proyecto.

Por último, cabe destacar el uso de la librería OSMBonusPack, la cual amplía las funcionalidades ofrecidas por OSMDroid. Entre otras mejoras, esta librería permite agrupar marcadores del mapa en clústeres o grupos, lo cual ha resultado realmente útil en este proyecto para agrupar la gran cantidad de aeródromos representados en el mapa.

3.5. SQLite

A la hora de almacenar rutas aeronáuticas en la aplicación para que el usuario pueda simular aviones sobre ellas sin necesidad de conexión a internet, se ha decidido generar una base de datos haciendo uso de SQLite.

SQLite [19] es un sistema de base de datos relacional (RDBMS, del inglés *Relational Database Management System*) que funciona como un servidor propio e independiente. Se trata de un software gratuito y de dominio público, que permite almacenar los datos en un único archivo, sin hacer uso de un servidor externo. No precisa de configuración de puertos, tamaños o ubicaciones, lo que junto a su simplicidad, rapidez y usabilidad contribuye a la facilidad de su uso. Se recomienda para aplicaciones o programas independientes que no requieran mucha escalabilidad, como es el caso del presente proyecto. Por todo ello, se considera a SQLite como una tecnología muy adecuada para aplicaciones móviles.

En cuanto al concepto de RDBMS, una base de datos relacional se puede describir como un modo de recopilar información de manera organizada en tablas, que guardan relaciones comunes entre sí por medio de claves. Cada fila de la tabla es un registro con ID único, a lo que se denomina clave. Por su parte, las columnas contienen atributos de datos, y cada registro por lo general tiene un valor para cada uno del los atributos, facilitando el establecimiento de relaciones entre los datos de la tabla. En el caso concreto del proyecto, cada una de las filas de la tabla de las bases de datos es un punto de una ruta, que por medio de un ID conforma una ruta completa junto con aquellas filas que comparten esa misma clave.

3.6. Software utilizado

A continuación se proceden a describir los principales programas o entornos de desarrollo empleados para desarrollar la aplicación.

3.6.1. Android Studio

Para llevar a cabo el desarrollo de la app se ha optado por hacer uso de Android Studio, el cual permite desarrollar tanto en lenguaje Java como en Kotlin. Este último lenguaje se encuentra en auge debido a la apuesta de Google por su utilización en las aplicaciones Android [20], aunque Java sigue siendo a día de hoy el más utilizado a nivel mundial en este ámbito [21]. A pesar de que el uso de Kotlin se valorase como una opción interesante, se prefirió el uso de Java debido a los conocimientos previos adquiridos en este lenguaje.

Android Studio [15] es el entorno de desarrollo (IDE) oficial para la plataforma Android. Desde su surgimiento en 2014, reemplazó a Eclipse como el IDE oficial para desarrollo de aplicaciones Android. Es un IDE basado en el IntelliJ IDEA, otro entorno de desarrollo creado por JetBrains. Su interfaz se muestra en la Figura 3.2.

Algunas de sus principales características son:

- Soporte para la construcción basado en Gradle
- Renderizado de vistas en tiempo real
- Refactorización específica de Android
- Editor de diseño con vista previa de los cambios realizados directamente en los archivos xml
- Control de versiones mediante Mercurial, Git, Github o Subversion
- Herramientas de Lint para identificar problemas de rendimiento, usabilidad y compatibilidad de versiones
- Variedad de marcos de trabajo y herramientas de prueba
- Inspección de código en tiempo real para depuración de errores

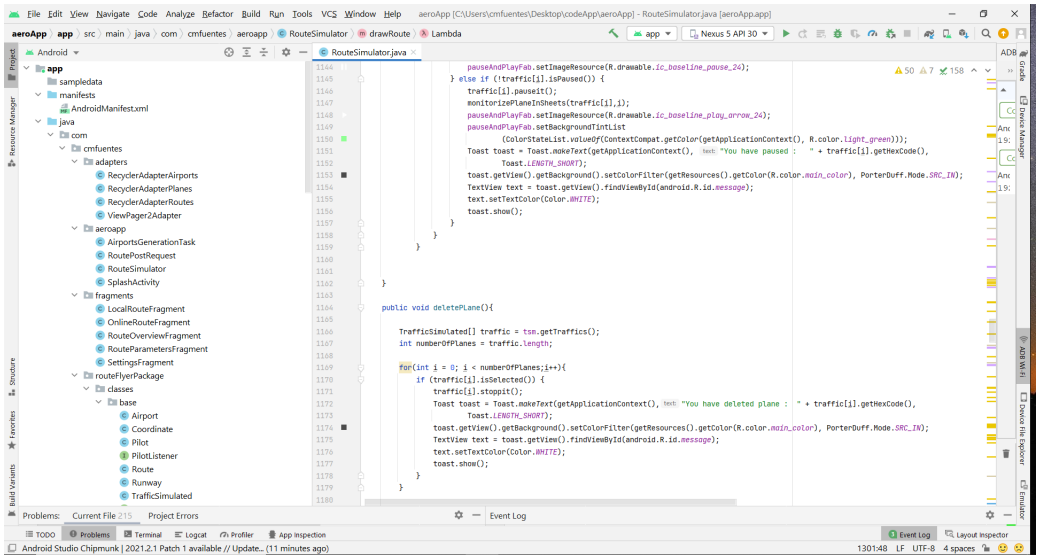


Figura 3.2: Interfaz de Android Studio

Como se ha expuesto, el soporte para la construcción de las aplicaciones está basado en Gradle [22], una herramienta que permite la automatización de compilación de código abierto, la cual se encuentra centrada en la flexibilidad y el rendimiento. Este instrumento de construcción multiplataforma está basado en Apache Ant y Apache Maven. La declaración de dependencias de los proyectos se lleva a cabo mediante un lenguaje basado en Groovy, en lugar de hacer uso de XML (cuyo uso se delega a la interfaz gráfica de usuario).

3.6.2. GIMP

Desde el inicio del desarrollo de la aplicación se ha apostado por invertir esfuerzos en que la aplicación presentase un diseño atractivo. Con el fin de lograr este objetivo, además de hacer un buen uso de gran variedad de componentes ofrecidos por Android, ha resultado fundamental cuidar la estética y calidad de las imágenes e iconos utilizados. Esto abarca aspectos como la creación de logos e iconos propios, la edición y redimensionado de imágenes y la generación de vectores (gráficos vectoriales bidimensionales en formato de lenguaje de marcado [23]) para incluir en los distintos menús de la aplicación. Gran parte de los iconos utilizados durante el desarrollo, los cuales han sido extraídos de fuentes de dominio público, han precisado de una edición previa a su incorporación en la app. Para llevar a cabo estos procesos ha resultado fundamental el uso de la aplicación de escritorio GIMP.

GIMP [24] (de sus siglas en inglés *GNU Image Manipulation Program*) es un software de edición de imágenes digitales en forma de mapa de bits. Se trata de un programa libre y completamente gratuito.

Este permite el tratamiento de imágenes mediante capas, con el objetivo de poder modificar cada una de ellas de manera independiente a las demás. Cada capa tiene sus propias características, como por ejemplo su propia visibilidad y grado de transparencia. A pesar de tratarse de un software gratuito, incluye una extensa paleta de herramientas que ofrece posibilidades de edición muy variadas. Además, soporta gran cantidad de formatos y es capaz de exportar las imágenes procesadas en aquellos más utilizados a nivel mundial, como pueden ser PNG, BMP, JPG, GIF, PDF, etc.

La Figura 3.3 muestra un ejemplo del aspecto que tiene la interfaz gráfica de su aplicación de escritorio. En su interior se encuentra uno de los iconos más utilizados en la aplicación: el icono de aeródromo, el cual se ha elaborado partiendo de la imagen del avión que aparece en su interior.

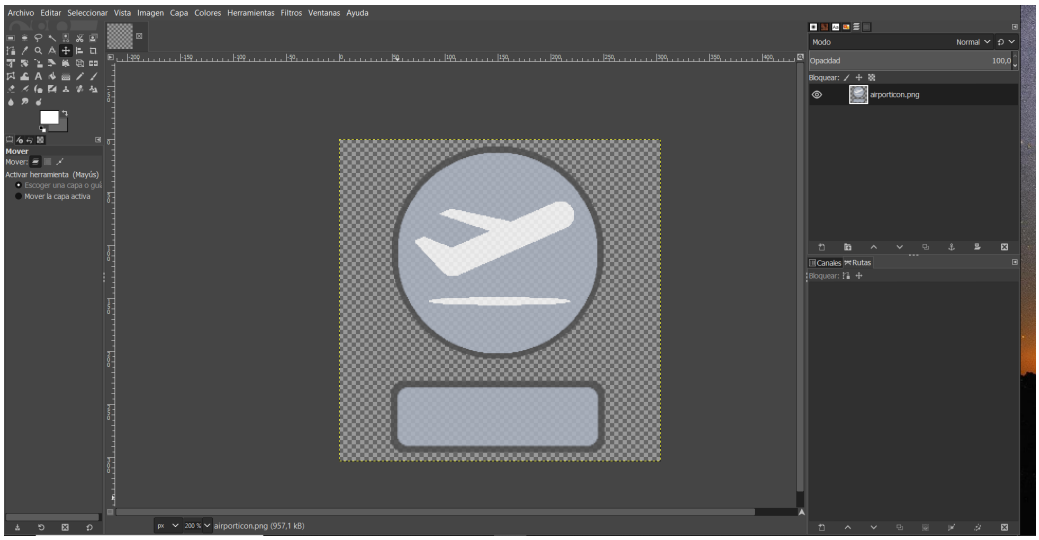


Figura 3.3: Interfaz de GIMP

3.6.3. RouteFinder

Como se ha anticipado en el Punto 2.1.3, uno de los servicios fundamentales utilizados para poder llevar a cabo un simulador de vuelos sobre rutas aéreas realistas es la versión gratuita de la web RouteFinder.

Para llevar a cabo la generación de rutas en este proyecto se ha hecho uso únicamente de la versión gratuita [9]. Esta permite configurar los aeródromos de origen y destino mediante sus códigos OACI, así como algunas de las características propias de la ruta (Figura 2.3).

3.6.4. GenyMotion

Resulta evidente que uno de los elementos fundamentales para desarrollar en Android es un dispositivo que funcione con este SO, para así poder testear las aplicaciones y depurar el código en busca de fallos. Si bien lo ideal es trabajar con un dispositivo físico, existe la posibilidad de utilizar un emulador que haga el papel de este. A lo largo del proyecto se ha hecho un uso frecuente del dispositivo móvil personal del desarrollador. Sin embargo, ha resultado fundamental también el uso de un emulador, destinado a tareas tales como la mejora del carácter responsivo de la app. A pesar de que Android Studio tienen la opción de utilizar sus propios emuladores, se ha preferido el uso de un emulador externo que se ejecute de manera más fluida en el computador. En concreto, el software elegido ha sido GenyMotion.

GenyMotion [25], considerado como uno de los emuladores más rápidos de Android, aprovecha la arquitectura x86 para ejecutar de forma fluida y rápida distintos dispositivos Android. Está basado en las máquinas virtuales x86 optimizadas para ejecutarse sobre VirtualBox, y permite ejecutar todo tipo de aplicaciones y juegos en ordenadores Windows, Mac o Linux. La Figura 3.4 muestra el dispositivo emulado con GenyMotion del cual se ha hecho uso durante el desarrollo de la app, un dispositivo Android 9 (API 28) con dimensiones de pantalla 1080x1920 píxeles y densidad de 480 dpi.

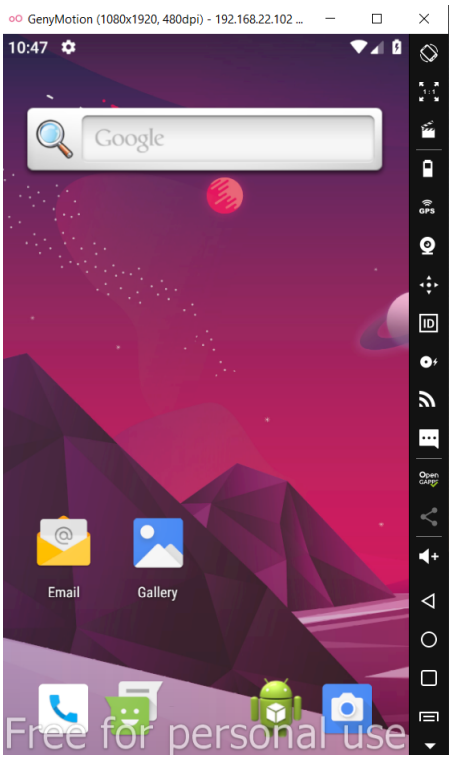


Figura 3.4: Dispositivo emulado con GenyMotion

4. Prestaciones y diseño de la aplicación

En el presente Capítulo se proceden a abordar las prestaciones y el diseño de la aplicación RouteFlyerDroid. Este comienza por la exposición de todas las funcionalidades que presenta la app, para dar paso posteriormente a la explicación de las principales pantallas y menús de navegación de su interfaz gráfica. Esta parte del proyecto se centrará en la exposición de la aplicación desde un punto de vista práctico, mostrando todas sus características y sus funcionalidades, sin entrar en aspectos técnicos de su desarrollo. Esto último se abordará en el Capítulo 5, donde se expondrá la estructura que se ha seguido para el desarrollo del código y de los elementos gráficos de la aplicación. Asimismo, se omite también la explicación detallada de las instrucciones de uso de la aplicación, lo cual se abordará en el Manual de Usuario expuesto en el Anexo A.

4.1. Funcionalidades

El objetivo de esta sección es explicar detalladamente cuáles son las principales funcionalidades de la aplicación, destacando aquellos aspectos que merezca la pena resaltar por su utilidad o bien porque hayan supuesto una mejora relevante durante el desarrollo de la misma. Estas se recogen de manera sintetizada en la Figura 4.1.

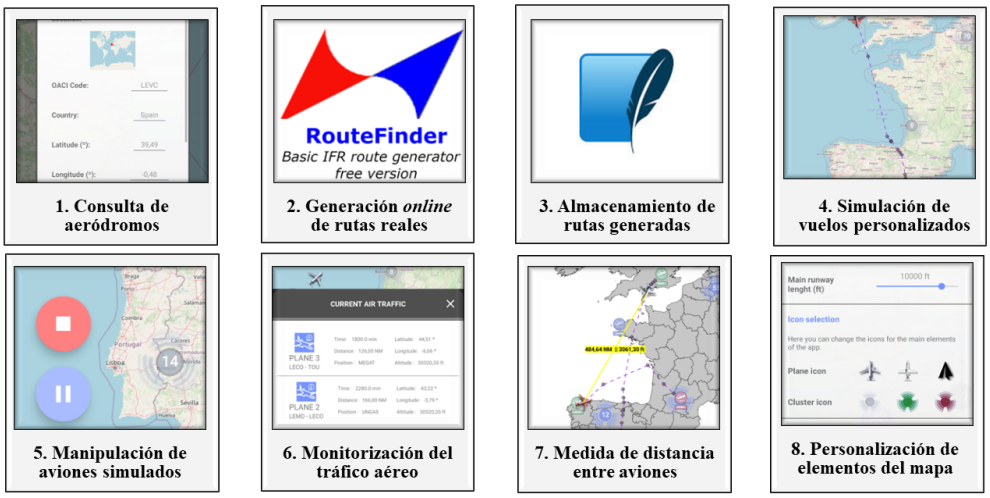


Figura 4.1: Principales funcionalidades de la aplicación

A continuación se procede a describir detalladamente en que consiste cada una de las funcionalidades. Puede resultar preciso señalar que estas no se exponen en orden de importancia para el desarrollador, sino más bien siguiendo el orden lógico que el usuario podría seguir cuando manipule la aplicación.

4.1.1. Visualización y consulta de aeródromos

A pesar de no tratarse de la funcionalidad principal de la app, la visualización y consulta de parámetros de aeródromos alrededor del mundo puede considerarse como uno de los principales atractivos que ofrece. A través del procesamiento de la base de datos *NaviGraph AIRAC Cycle 2205* [8], la aplicación almacena información de un total de más de 14.000 aeródromos.

Estos aeródromos son accesibles de dos formas: desde un listado que contiene la totalidad de los aeródromos procesados de la base de datos de navegación (accesible mediante el Elemento 2.2 de la Figura 4.5), o de manera gráfica desde el mapa, el cual contiene representados solamente aquellos cuya longitud de pista principal supere un valor establecido por el usuario. La finalidad de este filtro por longitud es la de no saturar el mapa al representar un exceso de iconos de aeródromos extraídos de la base de datos.

La información de los aeródromos que se brinda incluye parámetros fundamentales de los mismos y de sus pistas, tal y como se muestra la Sección 4.2.

4.1.2. Generación de rutas mediante petición HTTP a RouteFinder

El objetivo primordial impuesto a la hora de desarrollar la app ha sido el de poder simular aviones que siguieran rutas extraídas desde bases de datos de navegación, dando así a las simulaciones una mayor aproximación a la realidad. Para ello, y de acuerdo con los métodos presentes en el código base, se decidió extraer las rutas desde la versión gratuita de la web RouteFinder. Como se ha expuesto en la Sección 3, esta versión de RouteFinder no ofrece la posibilidad de configurar las rutas con elevada precisión. Sin embargo, debido a mayor simplicidad con respecto a otras webs similares, y la mayor facilidad en el procesamiento de la ruta generada, se ha presentado como la mejor opción para implementar en una aplicación de este calibre. Resulta preciso subrayar que las rutas generadas por RouteFinder están destinadas a la navegación IFR, por lo que los planes de vuelo simulados en la aplicación se corresponden con las reglas de vuelo instrumental.

Cabe destacar que el procesamiento de rutas desde las bases de datos de navegación ha estado desde un primer momento fuera del alcance del proyecto, puesto que se trata de un trabajo de programación que escapa del tiempo establecido para el desarrollo de la app. Es por ello que, tal y como se ha adelantado en el Capítulo 2, desde un principio se optó por extraer rutas que ya hayan sido procesadas. Consecuentemente, esto implica hacer uso de software de terceros, como en este caso RouteFinder.

En una primera aproximación al objetivo de simular rutas reales, la descarga de rutas desde la web se hacía de manera manual, almacenando los archivos de texto en formato .txt dentro de la aplicación. Sin embargo, a medida que se fue desarrollando el software se valoró la posibilidad de automatizar la descarga de rutas desde la web con el fin de ampliar el abanico de posibilidades de la app. Finalmente, esta propuesta se llevó a cabo enviando solicitudes POST [26] al formulario HTML del servidor de RouteFinder y procesando su respuesta, convirtiendo de este modo el proceso manual en un proceso automático en segundo plano. Esta mejora se implementó haciendo uso de la librería de código abierto OkHttp [27]. Sin duda, se trata de una de las funcionalidades de mayor envergadura de la app, puesto que acoplar el software de simulación con un servicio de terceros que puede procesar prácticamente cualquier ruta abre un amplio abanico de posibilidades en la simulación. Y, sobre todo, el hecho de que este proceso se pueda llevar a cabo de manera automática puede resultar realmente cómodo y atractivo para el usuario.

Si bien es cierto que se trata de una prestación verdaderamente interesante para el usuario, esta está condicionada por la necesidad de conexión a internet, así como por la disponibilidad del servidor de RouteFinder en el momento de petición de una nueva ruta. Para evitar que la simulación dependa por completo del buen funcionamiento de la red y del estado de servidor de procesamiento de rutas, la app incluye una base de datos con rutas generadas previamente. Esta funcionalidad se expone en el siguiente punto.

4.1.3. Acceso a rutas locales y almacenamiento de nuevas rutas en memoria

A la hora de iniciar una nueva simulación, se ofrecen dos opciones para la selección de la ruta: elegir entre una de las rutas preestablecidas desde la base de datos interna, o bien hacer una petición al servidor de RouteFinder para obtener una ruta personalizada, tal y como se ha expuesto en el punto previo.

Esta base de datos local permite elegir en un principio una de las rutas preestablecidas que se han generado como muestra. Sin embargo, uno de los puntos fundamentales de la creación de una base de datos es que esta pueda ir creciendo a medida que se

generan nuevas rutas desde la web. Es por ello que la aplicación incorpora la funcionalidad automática de almacenar en la base de datos toda ruta que se genere enviando peticiones a la web, para posteriormente mostrarla como opción cuando se quiera lanzar una nueva simulación.

Ciertamente, se trata de una prestación que retroalimenta las posibilidades del usuario, dado que puede repetir rutas generadas previamente. Además, esta característica brinda al usuario cada vez mayor independencia de la conexión a internet a medida que este vaya generando las rutas que desee.

4.1.4. Simulación de vuelos. Configuración de los parámetros de vuelo

Pese a que el resto de funcionalidades que se exponen en esta sección pueden resultar de gran interés y nutren a la aplicación de mayores posibilidades de uso, el pilar sobre el que se fundamenta el software es la simulación de aviones que siguen una determinada ruta.

Obviando las hipótesis asumidas en el código y la simplificación de ciertas variables a la hora de diseñar las rutas, los aviones simulados buscan tener un comportamiento lo más parecido al de una aeronave real. A pesar de tratarse de un código enfocado al ámbito académico más que al segmento profesional de la simulación de aeronaves, el código base contempla todas las ecuaciones físicas que rigen el movimiento de los aviones y su navegación a través de rutas reales. Es por ello que las simulaciones llevadas a cabo por la aplicación presentan resultados que se aproximan en gran medida a las actuaciones de aviones reales para rutas similares, ofreciendo distancias y tiempos de vuelo similares.

Uno de los principales atractivos de la simulación es que se pueden parametrizar gran cantidad de variables de vuelo de la aeronave previamente a su lanzamiento, así como parámetros temporales relativos a la propia simulación. Esto incluye algunos parámetros de vuelo como: la altitud de crucero; velocidades independientes durante los tramos de ascenso, crucero y descenso; ratios de ascenso y descenso independientes y el tipo de navegación deseada.

Por último, resulta fundamental que las aeronaves simuladas puedan comunicarse con el usuario tanto de manera gráfica como numérica. Esta dualidad se logra tanto mediante su representación en el mapa como a través de la visualización de diversos parámetros de vuelo del avión seleccionado, todo ello acorde a los parámetros temporales de la simulación

4.1.5. Manipulación de las aeronaves simuladas: pausa y borrado

Una vez el usuario haya iniciado una o varias simulaciones, la aplicación permite manipular cada uno de los aviones. En concreto, el usuario tiene la capacidad de pausar y reanudar una simulación, así como de eliminar aviones del tráfico aéreo generado.

Esto ofrece al usuario la posibilidad de controlar el tráfico aéreo según desee, tanto para visualizar los parámetros de un avión en un momento concreto como para sincronizar dos simulaciones y relacionarlas según le convenga, entre otros ejemplos de uso.

4.1.6. Monitorización del tráfico aéreo

Una de las características con mayor potencial del código base es la posibilidad de simular múltiples aviones (cada uno con su propia configuración) de manera síncrona. Esto abre las puertas a la monitorización de un mapa de tráfico aéreo, que en el caso de esta app se fundamenta en su visualización gráfica en el mapa y en la consulta numérica de los parámetros de todos los aviones en vuelo.

A través de un menú desplegable, tal y como se expondrá en la sección de diseño de la interfaz (Sección 4.2), se ofrece la posibilidad de consultar el estado de las aeronaves simuladas de manera síncrona, además de manipular la selección de aeronaves desde el propio menú.

Indudablemente, la simulación simultánea de aeronaves abre un amplio abanico de posibilidades en el desarrollo de la app, destacando aquellas relativas a la interacción entre las mismas. Si bien es cierto que algunas propuestas de interacción planteadas a lo largo del desarrollo se han preferido dejar como trabajos futuros, se ha podido explotar esta interacción mediante una opción de medida de distancia entre aeronaves. Esta funcionalidad se expone en el siguiente punto.

4.1.7. Medida de distancia entre aviones

Siempre y cuando el mapa de tráfico contenga más de un avión, se habilitará el modo multiselección o modo de medida de distancias entre aviones. Este modo permite seleccionar dos aviones con el fin de visualizar la distancia entre ellos, tanto en su componente horizontal como vertical. Una vez seleccionadas las dos aeronaves, se muestra en el mapa una línea que las une y que sigue un código de color relativo a su cercanía, acompañada del valor de las distancias horizontal y vertical entre ambas.

Si bien la multiselección podría ser aplicable a otros elementos gráficos como aeropuertos o puntos de ruta, se ha preferido optar por destinar esta funcionalidad única-

mente a la medida de distancia entre aviones, pues se trata de una herramienta que va en la línea que sigue el proyecto sobre la monitorización de tráfico aéreo. Sin embargo, de cara a futuras versiones de la app, se podría valorar la implementación de la medida de distancias entre otros elementos del mapa.

4.1.8. Personalización del aspecto de los elementos del mapa

Con la finalidad de aportar una mayor personalización de la app se incluye también la opción de manipular distintos ajustes relativos a la representación de elementos en el mapa. Concretamente, se permite configurar la representación de los aeródromos, así como elegir los iconos para los aviones y los propios aeródromos. Se busca conseguir así una experiencia más libre por parte del usuario, además de poner el foco en el potencial de personalización de la aplicación.

4.2. Interfaz gráfica

En el apartado anterior se han expuesto las principales funcionalidades de la aplicación de forma teórica. En esta sección se procede a exponer de una manera gráfica el diseño de la app, mostrando de forma general todas las pantallas, menús y pestañas que ofrece.

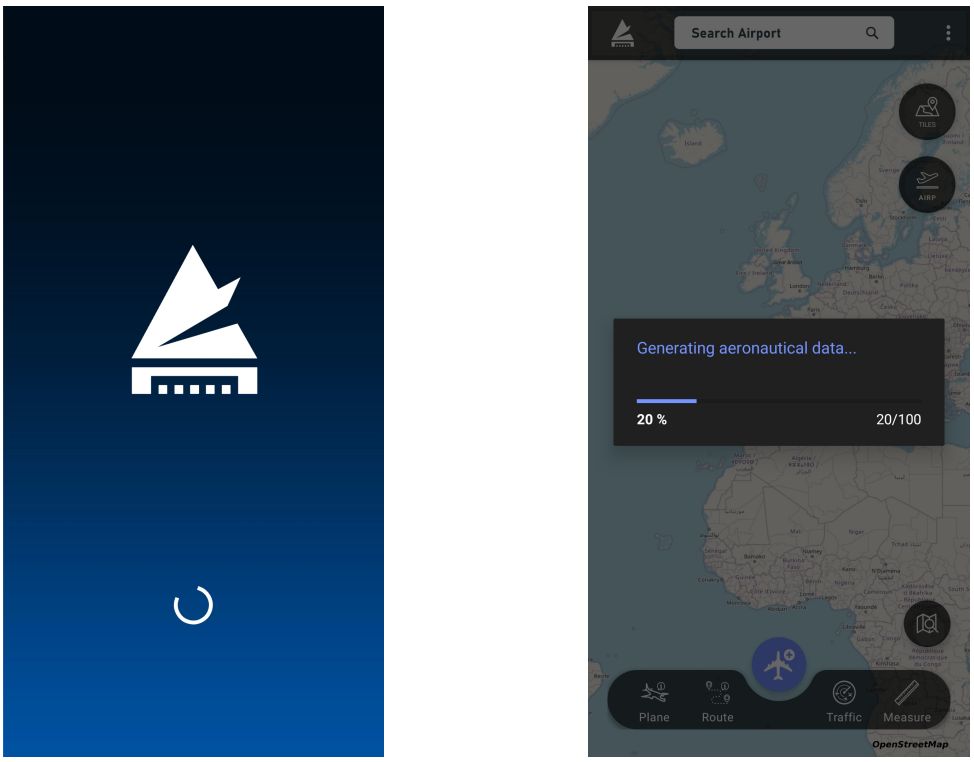
Desde que se inició el proyecto se ha buscado que la estética de la aplicación se asemejase a la que se puede observar en aplicaciones de carácter profesional. A pesar de carecer de conocimientos sobre el desarrollo de aplicaciones móviles al inicio del proyecto, se ha procurado invertir tiempo en el aprendizaje y desarrollo de una interfaz de usuario atractiva y fácil de usar. Pese a que se pueda valorar una optimización de los componentes y elementos utilizados en el diseño de la aplicación para mejorar el rendimiento de la misma, se considera que se le ha podido dar la apariencia que se pretendió en un principio.

4.2.1. Pantalla de bienvenida y carga de aeródromos

La primera pantalla mostrada al usuario al inicio de la aplicación es una pantalla de bienvenida o *splash screen*, la cual muestra el logo de la aplicación (Figura 4.2a). El objetivo de esta pantalla es darle un aspecto más profesional a la aplicación, contribuyendo a mejorar la parte estética de la misma.

Tras la pantalla de bienvenida se muestra también un diálogo de carga de datos (Figura 4.2b). Este diálogo de carga, que se muestra durante unos pocos segundos, informa al usuario sobre el progreso en la carga de la estructura de datos de aeródromos. Una

vez se hayan cargado todos los datos, se da paso a la pantalla principal de la aplicación.



(a) Pantalla de bienvenida **(b)** Diálogo de carga de aeródromos

Figura 4.2: Pantallas de bienvenida y carga de datos

4.2.2. Pantalla principal

La pantalla principal (Figura 4.3) fundamenta su diseño en una barra de navegación inferior flotante (Figura 4.4), donde se encuentran los principales controles de la aplicación. El botón flotante del centro de la barra de navegación (Elemento 1.3) permite generar nuevas simulaciones. El resto de botones de la barra posibilitan al usuario la consulta de parámetros de las aeronaves simuladas. En primer lugar, los dos primeros botones (Elementos 1.1 y 1.2) permiten consultar los parámetros de vuelo del avión seleccionado y las características de su ruta. Por otro lado, los botones señalados como 1.4 y 1.5 permiten monitorizar el tráfico aéreo actual y activar el modo de medida de distancia entre aeronaves, respectivamente.

Figura 4.3: Interfaz principal de la aplicación

Figura 4.4: Barra de navegación inferior

Por otra parte, la barra superior de la aplicación (Figura 4.5) da acceso a la pantalla de información de la app (Elemento 2.1), la base de datos de aeródromos generada (Elemento 2.2) y la pantalla de ajustes del mapa (Elemento 2.3), respectivamente.

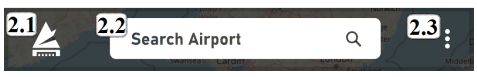


Figura 4.5: Barra de navegación superior

Por último, mediante los botones flotantes que se reparten en lado derecho de la app se permite al usuario, en orden descendente: cambiar la textura del mapa, ocultar/mostrar la capa de aeródromos y manipular los controles de zoom (En orden, Elementos 3,4 y 5 de la Figura 4.3).

En los próximos puntos se procede a exponer las pantallas y menús que ofrece la aplicación, a las que se accede mediante los controles mencionados en la pantalla principal.

4.2.3. Opciones de consulta de aeródromos

Tal y como se ha expuesto previamente, durante el inicio de la aplicación se genera un vector de aeródromos extraído desde la base de datos de navegación de NaviGraph [8], que contiene más de 14.000 aeródromos.

Estos aeródromos se pueden consultar en un listado (Figura 4.7b) que se despliega pinchando en el centro de la barra superior de la app, el cual permite realizar una búsqueda filtrando por país, nombre del aeródromo o código OACI (un identificador de 4 dígitos de acuerdo con la normativa de la Organización Internacional de Aviación Civil). Asimismo, los aeródromos se pueden consultar en el propio mapa (Figura 4.7a). Sin embargo, se ha considerado necesario filtrar la representación de los aeródromos por la longitud de la pista principal (por defecto fijada en 9000 pies), con la finalidad que solo un porcentaje de los mismos sea representado, evitando que el renderizado del mapa se vea afectado por un exceso de elementos gráficos.

En cuanto a la representación de los aeródromos en el mapa, cabe destacar que se han asociado en grupos denominados *clusters*, los cuales agrupan los iconos de los aeródromos en un radio de un determinado número de píxeles (por defecto, 400 píxeles). Además, cabe destacar que cada aeródromo está representado con una imagen en mapa de bits (*bitmap*) único, puesto que este muestra el código OACI de cada uno de ellos. Todo ello se puede observar claramente en la Figura 4.7a, donde el Elemento 1 es un clúster que agrupa 3 aeródromos cercanos y el Elemento 2 es un único aeródromo que queda fuera de cualquier clúster (concretamente el aeropuerto español de Vitoria). Esta herramienta permite que la representación de los aeródromos resulte más estética y no sature el mapa, permitiendo mostrar cada uno de ellos a medida que se aumenta el zoom, tal y como se ejemplifica en la Figura 4.6 para el aeropuerto de Madrid-Barajas.

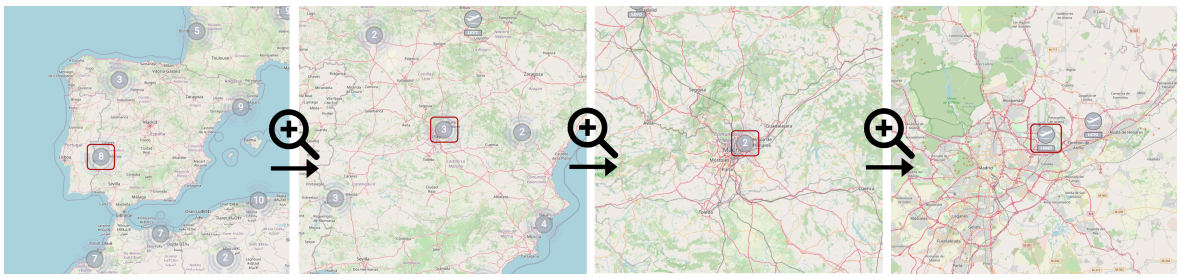
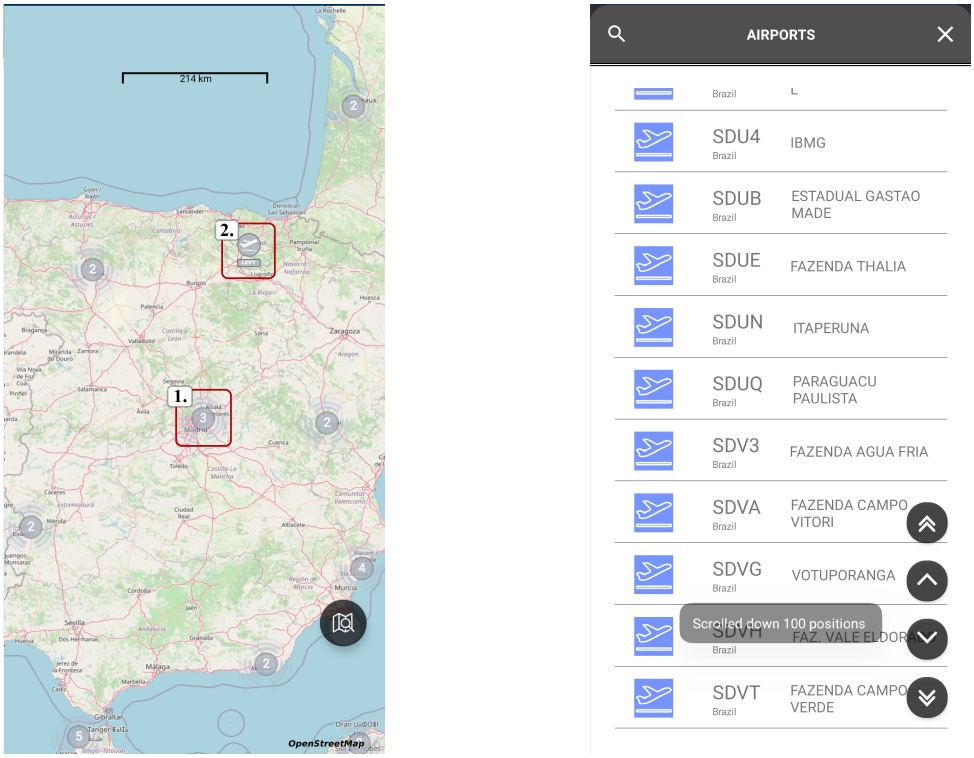


Figura 4.6: *Clustering* de aeródromos según el zoom para el aeropuerto Madrid-Barajas



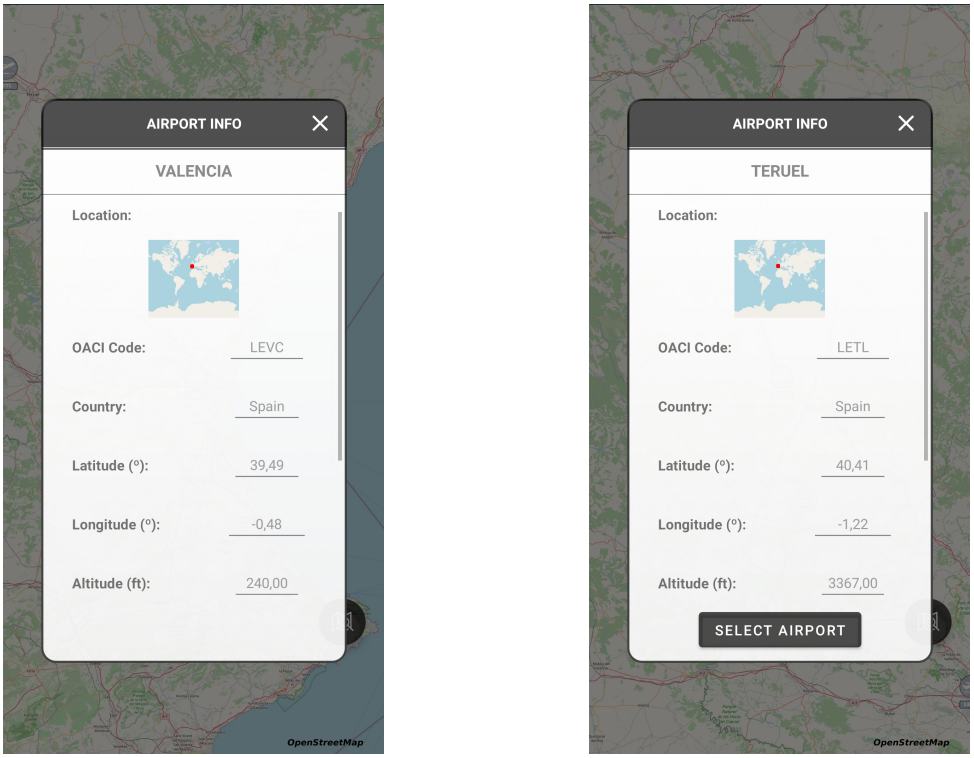
(a) Consulta de aeródromo desde el mapa **(b)** Consulta de aeródromo desde el listado

Figura 4.7: Consulta de aeródromos mediante mapa o listado

4.2.4. Diálogos de consulta de aeródromo

Ya sea pulsando un aeródromo desde el mapa o bien seleccionándolo desde el listado desplegable, la aplicación permite al usuario acceder a un diálogo que muestra los principales parámetros del aeródromo (Figura 4.8). Estos parámetros incluyen: nombre, código OACI, país, posición y características de su pista o pistas.

El diálogo de aeródromo puede presentarse bien únicamente como una pestaña de consulta de parámetros (Figura 4.8a) o bien aparecer con la opción de seleccionar ese aeródromo para iniciar una nueva ruta (Figura 4.8b), siempre que se acceda a este a través del menú de generación de nuevas rutas, que se presenta en el próximo punto.



(a) Diálogo de aeródromo: No seleccionable **(b)** Diálogo de aeródromo: Seleccionable

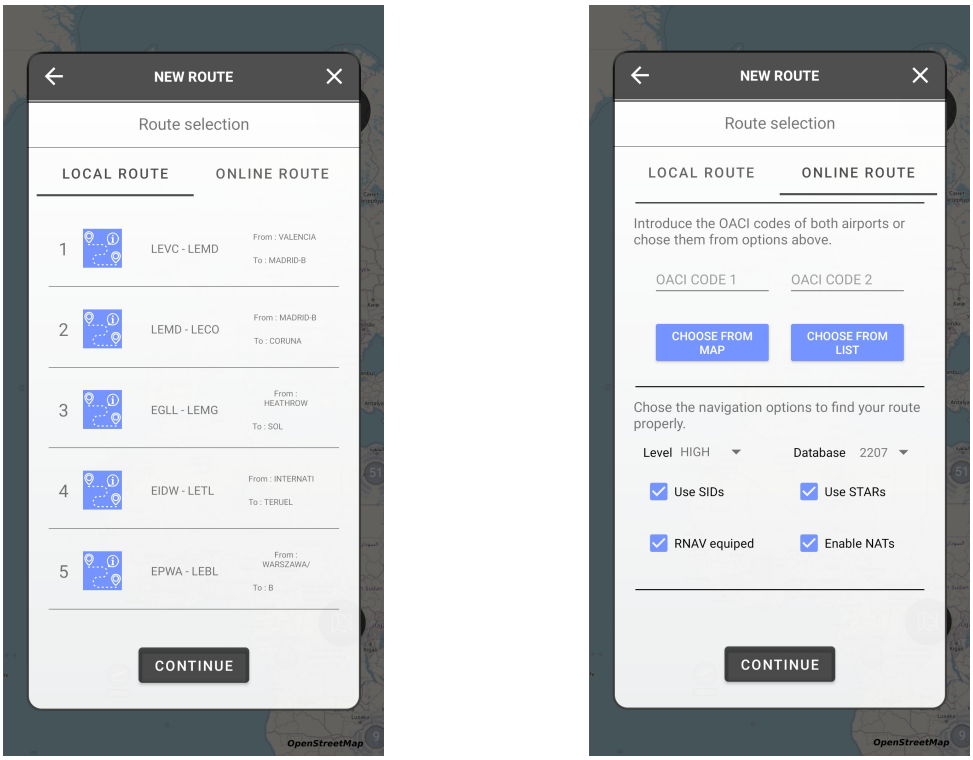
Figura 4.8: Diálogos de consulta de parámetros de aeródromo

4.2.5. Menús de generación de una nueva simulación

Cuando se pulsa en el botón flotante central de la aplicación (Elemento 1.3 de la Figura 4.4), se accede a los menús o diálogos de generación de una nueva simulación. Estos menús constan de 4 pasos: la selección de la ruta, la configuración de los parámetros del avión y la simulación, la revisión general de los parámetros de la ruta generada y finalmente la visualización de la ruta en el mapa (Figura 4.9)

Figura 4.9: Pasos a seguir para iniciar una nueva simulación

En primer lugar, se debe seleccionar la ruta que se desea recorrer. Para ello se ofrecen dos opciones: elegir entre una de las rutas que se encuentran almacenadas en la base de datos local (Figura 4.10a), o bien generar una nueva ruta entre los dos aeródromos que se desee. En el caso de que se opte por la segunda opción, se deberán seleccionar los dos aeródromos que se deseen mediante tres opciones: la introducción manual de sus códigos OACI, su selección desde el mapa o su selección desde el listado de aeródromos. Además, se deberán configurar una serie de parámetros relativos a la aeronavegación que permiten especificar que tipo de ruta se desea. Cabe destacar que una vez la nueva ruta sea generada, esta será guardada automáticamente como ruta local y estará disponible para posteriores simulaciones en el listado de rutas locales.

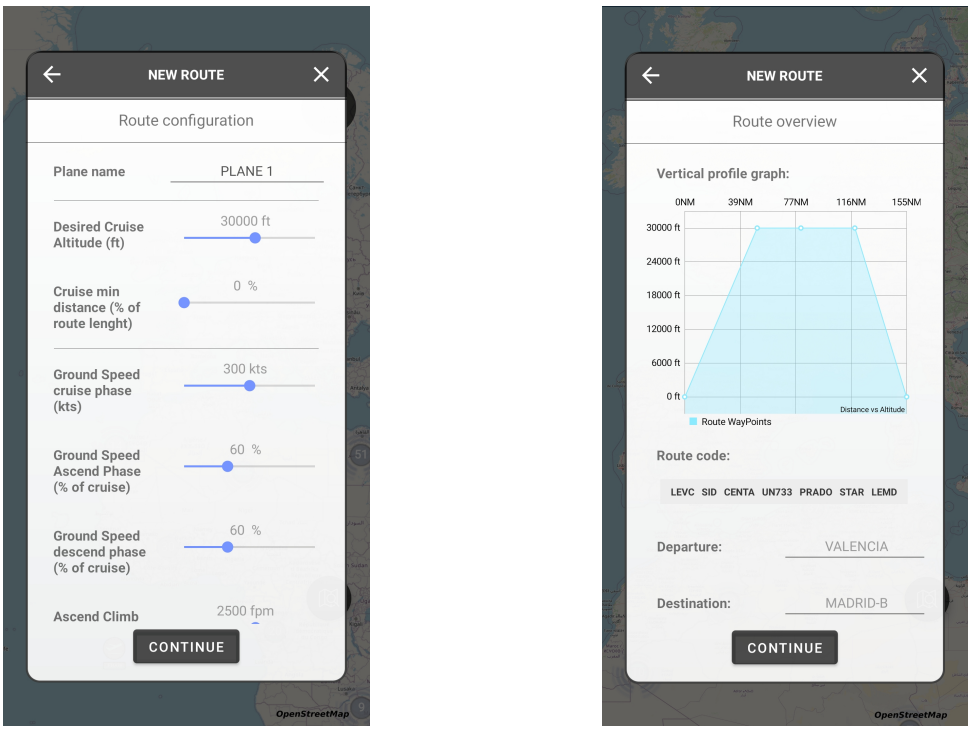


(a) Selección de ruta local **(b)** Selección de ruta online

Figura 4.10: Menú de selección de nueva ruta

Una vez se ha seleccionado la ruta deseada y esta se ha procesado correctamente, se procede a la configuración de los parámetros del avión simulado mediante un listado desplazable (Figura 4.11a). Estos parámetros incluyen tanto aquellos parámetros físicos relativos al desempeño del avión (altitud de crucero deseada, velocidades y ratios de ascenso/descenso) como algunos relativos a la propia simulación (periodo de simulación y tiempo de espera del hilo).

Tras el procesamiento de la ruta y la configuración de las variables de la simulación, la aplicación ofrece un resumen de la ruta generada (Figura 4.11b). Este resumen incluye tanto los parámetros principales de la ruta (origen, destino, longitud, tiempo estimado, número de puntos de ruta...) como un gráfico del perfil de altitud de la ruta y un campo de texto con el código aeronáutico de la misma.

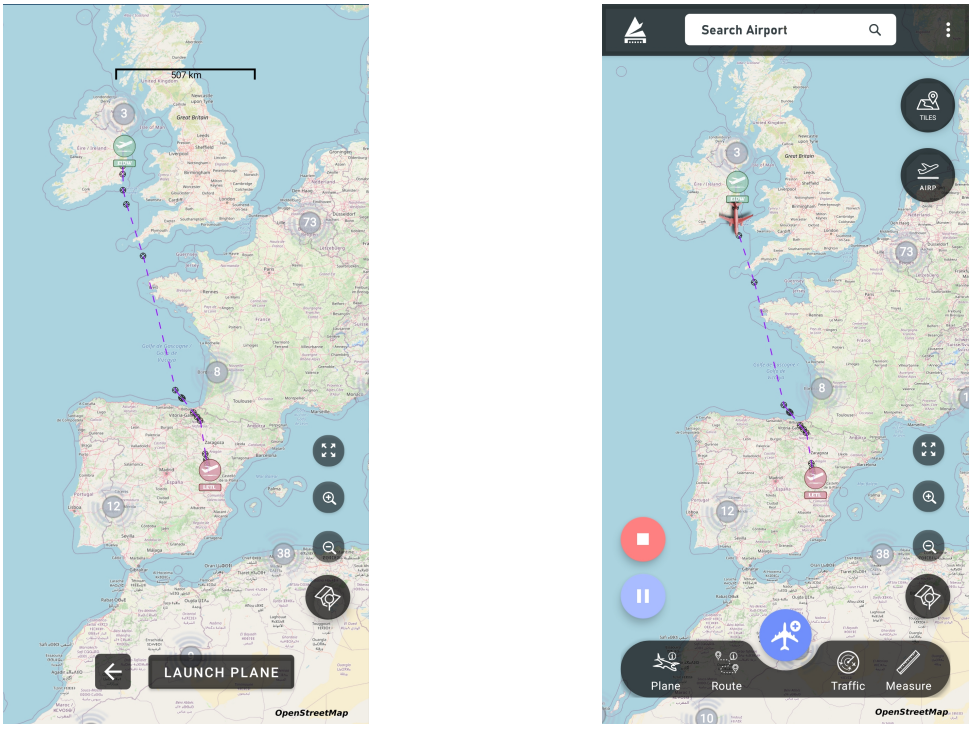


(a) Menú de configuración de parámetros del avión simulado

(b) Menú de resumen de las características de la ruta generada

Figura 4.11: Menús de creación de una nueva simulación

Finalmente, tras continuar en el menú de resumen de ruta, se accede de nuevo al mapa donde se ofrece una representación de la ruta generada (Figura 4.12a). Sobre la ruta se destacan los aeródromos origen y destino, así como los puntos de ruta, pudiéndose consultar información de todos ellos. En esta pantalla se permite al usuario iniciar la simulación o bien retornar a las pantallas anteriores para consultar o modificar parámetros de la ruta. Una vez lanzado el nuevo avión, la interfaz recobra su aspecto inicial (Figura 4.12b) y las pestañas de consulta de los parámetros del avión se habilitan. Estas pestañas permiten monitorizar parámetros de vuelo del avión seleccionado, tal y como se expone en el siguiente punto.



(a) Representación de ruta en el mapa previa al lanzamiento

(b) Nueva simulación iniciada. Pantalla principal.

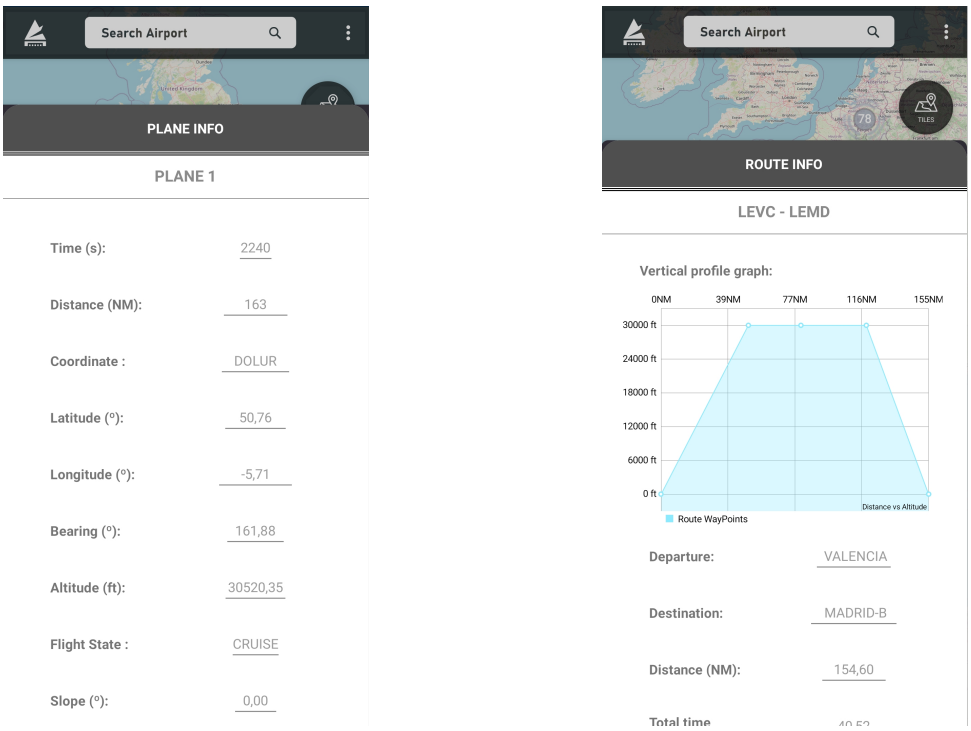
Figura 4.12: Fase final de la generación de una nueva simulación

4.2.6. Pestañas de monitorización de parámetros de avión y su ruta

Una vez lanzada una o varias simulaciones, la aplicación permite al usuario seleccionar uno de los aviones simulados. La selección del avión posibilita abrir las pestañas de monitorización de sus parámetros de vuelo y su ruta, además de cambiar su icono y representar su ruta sobre el mapa (Figura 4.12b).

La pestaña de monitorización de parámetros de vuelo del avión (Figura 4.13a) (accesible desde el Elemento 1.1 de la Figura 4.4)) muestra variables fundamentales como la posición de la aeronave en las 3 dimensiones, su rumbo, el tiempo de vuelo, la distancia recorrida, la última coordenada alcanzada y aquellas relativas a la navegación vertical.

Por su parte, la información que se muestra en la pestaña de información de la ruta del avión permanece constante a lo largo de la simulación, puesto que se trata de parámetros fijados previamente. Esta pestaña desplegable (accesible desde el Elemento 1.2 de la Figura 4.4)) expone parámetros de la ruta que sigue el avión seleccionado, mostrando un resumen similar al ofrecido en el menú de creación de ruta (Figura 4.11b).



(a) Pestaña de monitorización de parámetros del avión seleccionado

(b) Pestaña de consulta de información de la ruta del avión seleccionado

Figura 4.13: Pestañas de información del avión seleccionado

4.2.7. Pestaña de monitorización de tráfico aéreo

Tal y como se ha destacado en la sección previa, una de las funcionalidades más potentes de la aplicación es la simulación simultánea de varios aviones, los cuales poseen configuraciones de simulación totalmente independientes. Resulta de gran interés poder observar sus parámetros de manera síncrona, lo cual es posible mediante la pestaña de monitorización de tráfico aéreo (Figura 4.14). Esta pestaña (accesible desde el Elemento 1.4 de la Figura 4.4) muestra el listado actual de aviones que se encuentran volando, acompañado de una monitorización de algunos de sus principales parámetros. Además, esta pestaña permite también la selección de un avión pulsándolo desde el listado, lo que automáticamente lo destaca en el mapa acompañado de su ruta.

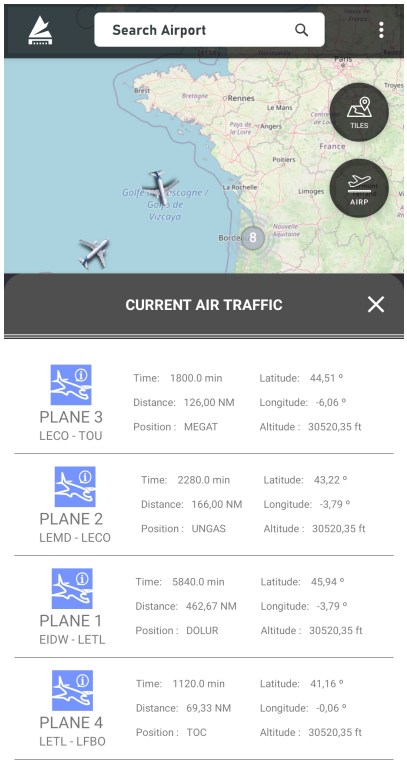


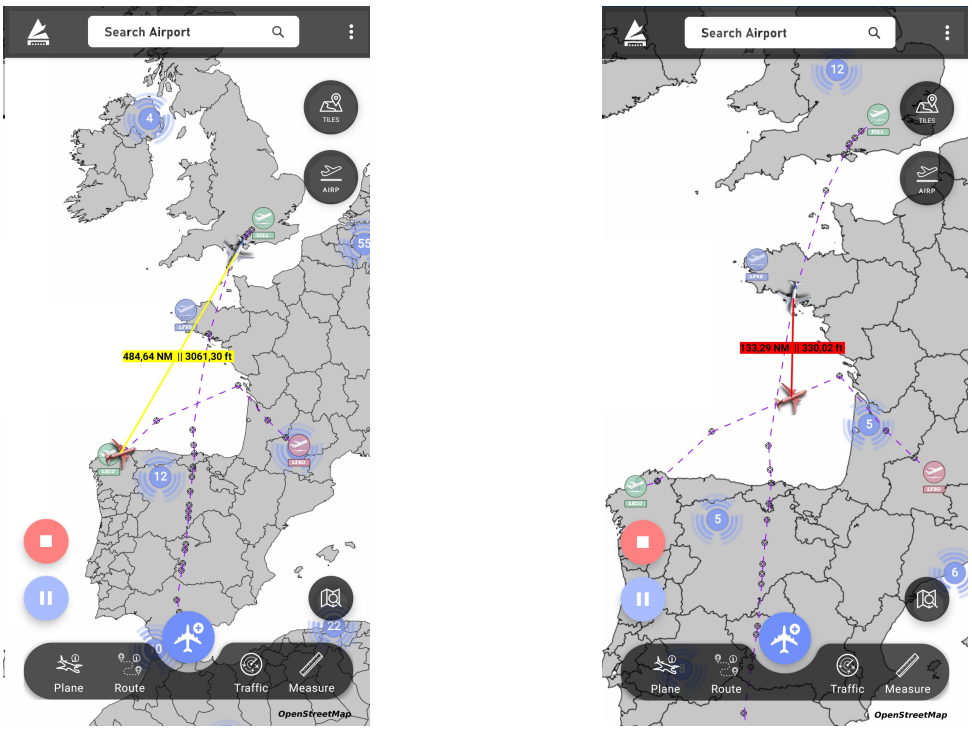
Figura 4.14: Pestaña de monitorización del tráfico aéreo

4.2.8. Modo multiselección. Medida de distancia entre aviones

Como se ha expuesto en el apartado de funcionalidades de la app, siempre que el mapa de tráfico contenga más de una aeronave, el usuario tendrá la posibilidad de activar el modo multiselección o modo de medida de distancia entre aeronaves. Esta funcionalidad es accesible desde el cuarto botón de la barra de navegación inferior (Elemento 1.5 de la Figura 4.4).

En el caso de que haya menos de 2 aviones en el mapa, este botón mostrará un aviso para indicar que se precisa de 2 o más aviones para su uso. Asimismo, si el modo de medida se encuentra activo y en ese instante deja de cumplirse la condición mencionada, automáticamente se regresará al modo normal de la aplicación.

Como se observa en la Figura 4.15, se ha querido diferenciar el modo multiselección del estado normal de la app. Para ello se ha dado un aspecto diferente al mapa mediante el cambio de su textura y los colores de algunos elementos. Esta figura muestra un ejemplo de cómo el código de colores de la línea de medida va acorde a la distancia entre ambas aeronaves, siguiendo una progresión desde verde hasta rojo a medida que estas se aproximan en el mapa.



(a) Modo de multiselección de aviones. Medida 1

(b) Modo de multiselección de aviones. Medida 2

Figura 4.15: Modo multiselección. Medida de distancias

4.2.9. Pantallas de ajustes del mapa e información de la app

Como se ha adelantado, con el objetivo de dotar a la aplicación de una mayor personalización se incluye la ventana de ajustes (Figura 4.16), a la cual se puede acceder desde la esquina superior derecha de la pantalla (Elemento 2.3 de la Figura 4.5). Dentro de esta pantalla se han incluido únicamente opciones relativas a la representación de elementos aeronáuticos en el mapa. Concretamente, el usuario puede manipular, por un lado, el radio que abarca cada uno de los clústeres de aeródromos, y por otro, la longitud de la pista principal de aeródromo a partir de la cual este se representa o no en el mapa. Esto permite al usuario decidir por completo la manera en la que los aeródromos se representan, teniendo en cuenta que la manipulación de estos ajustes puede afectar al renderizado del mapa. Además, se incluye también la opción de cambiar los iconos de los aviones y de los propios aeródromos.

Pulsando en el lado contrario de la barra superior de la aplicación (Elemento 2.1 de la Figura 4.5) se puede acceder a la pantalla de información de la app (Figura 4.17),

la cual contiene tanto datos de contacto del autor como advertencias a cerca del uso de la misma.

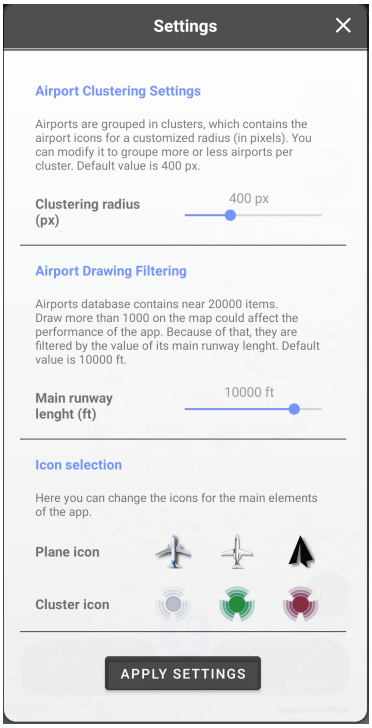


Figura 4.16: Pantalla de ajustes gráficos del mapa

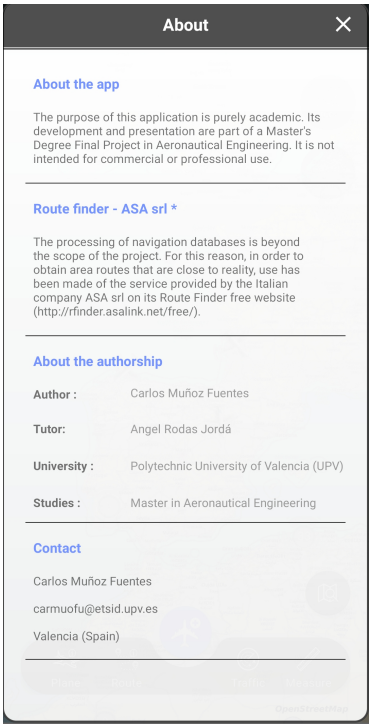


Figura 4.17: Pantalla de información de la aplicación y contacto

5. Trabajo realizado

En este capítulo se procede a exponer de manera general el trabajo llevado a cabo para desarrollar la aplicación. Se comienza con un apartado dedicado a la exposición de conceptos técnicos que han resultado esenciales para el desarrollo de misma. Posteriormente, se explica la estructura que se ha adoptado para desarrollar el código del proyecto, así como el funcionamiento de las clases principales y la algorítmica que hay detrás de las prestaciones fundamentales de la app. Finalmente, se incluye un breve resumen de los principales elementos implicados en la creación de la interfaz gráfica.

5.1. Fundamentos técnicos del desarrollo de la aplicación

El objetivo de esta sección es formular una introducción a algunos de los elementos de desarrollo que han resultado fundamentales para llevar a cabo la aplicación. A continuación, se enuncian brevemente ciertos fundamentos teóricos, que se acompañan con información concreta relativa al trabajo realizado.

5.1.1. Inyección de dependencias

La inyección de dependencias es una técnica muy utilizada en programación, cuyo uso destaca en el desarrollo de aplicaciones Android. En Android Studio, las dependencias permiten incluir bibliotecas, tanto internas como externas, en el proyecto de Android. Concretamente, las dependencias se incluyen en el fichero *build.gradle*, correspondiente al sistema de automatización de construcción de código Gradle, visto en la Sección 3.6.

Durante el desarrollo de este proyecto se han implementado gran variedad de dependencias, siendo muchas de ellas propias de la biblioteca de compatibilidad de Android (AndroidX) y algunas externas a esta. Dentro de todas ellas cabe destacar algunas librerías externas que han resultado fundamentales para implementar las funcionalidades que ofrece la app. Estas se exponen brevemente a continuación.

Librerías OSMDroid y OSMBonusPack

Dentro de las librerías externas de las que se ha hecho uso, la que posee mayor importancia es, sin lugar a dudas, la librería OSMDroid. Junto con su ampliación OSMBonusPack, conforman la base de la interfaz gráfica de la aplicación. Debido a su importancia dentro del proyecto, se ha dedicado una sección dentro del Capítulo 3 para definir resumidamente ambas librerías. Sin embargo, no se ha entrado al detalle de su uso dentro de la aplicación. Con este fin, se exponen resumidamente las principales clases de la librería de las que se ha hecho uso en el presente proyecto:

- **MapView**. Se trata del mapa, con el que actúan todas las demás clases de la librería.
- **ItemizedIconOverlay**. Esta clase dibuja una capa con iconos ampliamente configurables de manera individual, de tipo OverlayItem. Es utilizada para generar la capa de aviones.
- **Marker**. Se trata de un único icono personalizable, con opciones interesantes alternativas a las que ofrece la opción anterior. Se usa para representar los aeródromos y los puntos de ruta.
- **PolyLine**. Esta clase permite generar líneas personalizables que unen dos puntos geográficos. Se utilizan para la representación de las líneas de ruta, las cuales unen todos los puntos de la misma.
- **RadiusMarkerClusterer**. Es una clase añadida en la extensión OSMBonusPack. Permite agrupar iconos de tipo Marker en clústeres, en función del zoom y de la cercanía de los iconos entre sí dentro del mapa. En este caso, se ha utilizado para la representación de aeródromos. Como bien se ha expuesto en el Punto 4.2.3, esto contribuye a mejorar la estética de su representación, evitando saturar el mapa con iconos.
- **FolderOverlay**. Esta clase permite agrupar las capas que se deseen dentro de otra capa, que únicamente ocupará una posición dentro del vector de capas del mapa. En lo relativo a la app, se ha empleado esta clase para agrupar la capa de aviones y la de medida de distancias, con el fin de mejorar la distribución de los índices de capa. Además, ha resultado fundamental para agrupar las capas de tipo Marker correspondientes a aeródromos y puntos de ruta con las líneas (de tipo Polyline) que unen estos puntos, generando así una única capa correspondiente a una única ruta.

Librería MPAndroidChart

MPAndroidChart [28] es una librería gratuita que permite la generación de todo tipo de gráficos, creada por el desarrollador Phillip Jahoda. Se trata de una herramienta potente y fácil de usar, que presenta una gran variedad de opciones de configuración para los gráficos generados.

En el caso del presente proyecto, se ha hecho uso de esta librería para la representación del perfil vertical de ruta. Haciendo uso de un vector que contiene los dos aeródromos junto a todos los puntos de ruta, se genera un gráfico 2D que muestra la distancia recorrida en millas náuticas frente a la altitud en pies. La Figura 5.1 muestra un ejemplo de un perfil vertical graficado haciendo uso de esta librería. En ella se puede observar también una de las funcionalidades que incluye: la consulta de cada uno de los puntos del gráfico, que en este caso muestra las coordenadas de un punto de ruta.

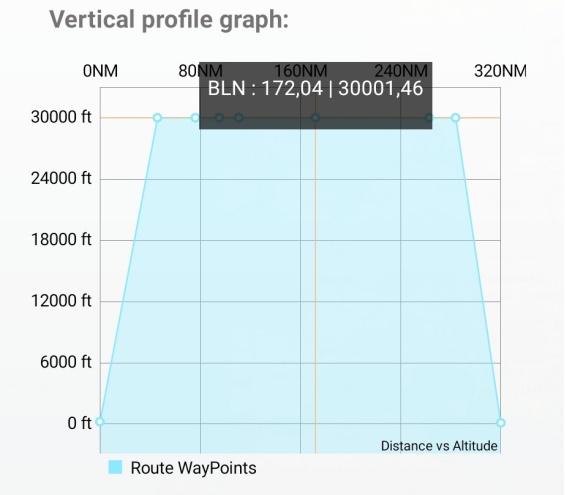


Figura 5.1: Ejemplo de perfil de altitud graficado con MPAndroidChart

Librería OKHttp

OKHttp [27] es un cliente HTTP + SPDY fácil de usar y con amplias prestaciones para la realización de peticiones HTTP. Se trata de un cliente con licencia Apache, soporte HTTP/2, SPDY Y GZIP, que comparte un socket en las peticiones e incorpora cacheo de las mismas, y que por supuesto es compatible con Java y Android.

Como se ha adelantado en la Sección 4.1, esta librería se utiliza para llevar a cabo las peticiones POST al servidor de RouteFinder para la obtención de códigos de ruta. Estas peticiones, las cuales se llevan a cabo en segundo plano, se realizan mediante la

clase fundamental de la librería: OkHttpClient. Esta se encarga de rellenar los datos de formulario de la web HTTP, y procesar la respuesta. Esta respuesta incluye todas las cadenas de texto generadas por el servicio externo, además de un código de respuesta que indica si la petición se ha llevado a cabo adecuadamente. Finalmente, las cadenas de texto se procesan haciendo uso de métodos propios, para obtener finalmente el código de ruta generado.

5.1.2. Componentes básicos de una aplicación Android

Los conceptos de Clase, Método y Atributo son la base del lenguaje Java, puesto que se trata de un lenguaje de programación orientado a objetos. Sin embargo, en el desarrollo de apps para dispositivos Android entran en juego, entre otros elementos, los conceptos de Actividad y Fragmento (*activities* y *fragments*).

Las actividades son, resumidamente, cada una de las pantallas que se crean en una aplicación Android. Una actividad está compuesta, salvo en contadas excepciones, de un parte lógica y una parte gráfica. La parte lógica es aquella destinada al código o "parte programática", encargada de implementar las funcionalidades de la app. Por otro lado, la parte gráfica consiste en un archivo de extensión XML (comúnmente denominado *layout* o contenedor), en el cual se definen los componentes gráficos que se corresponden con la parte lógica y que conforman la interfaz gráfica de usuario (GUI, del inglés *Graphical User Interface*).

Los fragmentos son componentes que funcionan dentro del ámbito de una actividad. Su objetivo es ampliar parte de la lógica que se utiliza para navegar entre pantallas o actividades. Estos fragmentos representan el comportamiento de una porción de la GUI asociada a una actividad. Por consiguiente, si una actividad finaliza, todos los fragmentos ligados a ella también lo hacen.

Las definiciones expuestas se pueden ver perfectamente reflejadas en el uso que se ha hecho de estos elementos en el desarrollo de RouteFlyerDroid. En la aplicación, la GUI principal y toda su lógica pertenece en su mayoría a una única actividad. Sin embargo, para desarrollar muchos de los menús o ventanas de navegación se ha hecho uso de fragmentos. Estos fragmentos complementan la lógica de la actividad principal en aspectos tales como la generación de una nueva ruta, además de incluir sus propias vistas o *layouts*. Todo esto se expondrá con más detalle en la próxima sección.

5.1.3. Programación multihilo

Uno de los conceptos fundamentales a comprender cuando se lleva a cabo por primera vez el desarrollo de una aplicación Android es el sistema de jerarquías que siguen los hilos. Resulta fundamental conocer que, por defecto, todos los procesos de la aplicación

se llevan a cabo en el hilo principal. Este hilo es el encargado de trabajar sobre la GUI, enviando los eventos a los widgets de la misma, así como facilitando la comunicación entre cada uno de los componentes. Esto quiere decir que se debe prevenir la ejecución de otros procesos en este mismo hilo, evitando a toda costa que cualquier proceso pesado se lleve a cabo en él, puesto que esto puede suponer que la aplicación deje de responder (lo que se conoce como ANR, del inglés *Application Not Responding*). Para lograr que estos procesos se ejecuten en segundo plano, Android ofrece gran variedad de posibilidades, entre las que se procede a destacar aquellas que se han implementado en el presente proyecto.

Threads y Runnables

El primer contacto con el sistema de jerarquías para los hilos de ejecución de Android tuvo lugar a la hora de adaptar el código base (el cual se expone en la siguiente sección) a un entorno de desarrollo para Android. El código en cuestión, desarrollado en Java, se encontraba diseñado para su ejecución en otro entorno de desarrollo (concretamente se estaba ejecutando en NetBeans IDE). En ese entorno, la jerarquía era diferente, y los hilos de ejecución se podían lanzar de forma simultánea sin afectar a la interfaz de usuario. Sin embargo, no es conveniente lanzar un nuevo de hilo de ejecución dentro de la lógica de una actividad de Android, puesto que este se ejecutará automáticamente en el hilo principal. La solución a esto pasa por lanzar o iniciar ese *thread* fuera del código principal. Esto se llevó a cabo mediante la implementación de la interfaz Runnable, permitiendo llamar a los procesos de simulación desde dentro de la propia lógica de sus clases.

Handler

Una vez ejecutado un *thread* en segundo plano, usualmente el usuario puede encontrarse con la necesidad de interactuar con ese proceso, bien recibiendo información o modificando algunos aspectos del mismo. Sea cual sea la interacción, esta no es posible de manera directa si el *thread* está ejecutándose en segundo plano. Por ejemplo, dentro de la actividad principal, no se pueden utilizar directamente las interfaces (*listeners*) de las clases de simulación para comunicarse con los aviones simulados, puesto que estas se han ejecutado en otro hilo. Para ello es necesario utilizar alguna de las clases que ofrece Android, como puede ser la clase Handler, la clase Service o la clase AsyncTask. Dado que las últimas dos clases citadas hubiesen supuesto una modificación considerable de las clases de simulación, se decidió hacer uso de un Handler para poder recibir información de los hilos secundarios (los aviones simulados).

De forma sintetizada, un Handler actúa de puente entre un hilo secundario y el hilo principal de la aplicación, tal y como ilustra la Figura 5.2. Instanciando un objeto tipo Handler en la actividad principal y ejecutando su método *.post()* dentro de las

interfaces de comunicación, se consigue recibir la información de las aeronaves.

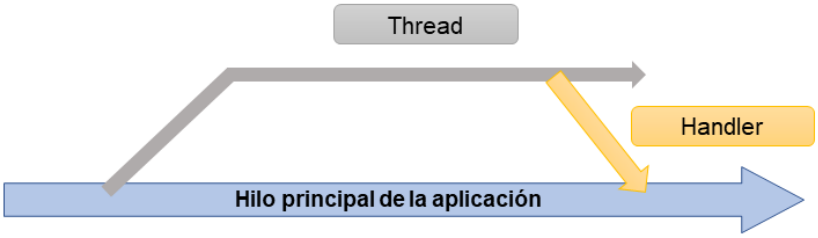


Figura 5.2: Funcionamiento de la clase Handler

En resumen, la clase Handler ha sido la elegida para llevar a cabo la comunicación con los hilos de simulación de aviones. Junto con la implementación de la interfaz Runnable, que permite que las simulaciones se ejecuten en segundo plano, se ha conseguido una correcta adaptación del paquete base a un entorno de desarrollo Android.

AsyncTask

Como se ha anticipado, una de las opciones más comunes para implementar procesos en segundo plano es la clase AsyncTask. Se trata de una de las clases más utilizadas para este fin, debido a la gran variedad de funcionalidades que permite implementar. A diferencia de la utilización de la combinación Thread (o Runnable) + Handler, esta opción permite la actualización del estado del proceso secundario haciendo uso de métodos propios de la clase. Estos métodos, los cuales se pueden consultar en la documentación [29], se muestran en el esquema de funcionamiento de la Figura 5.3.

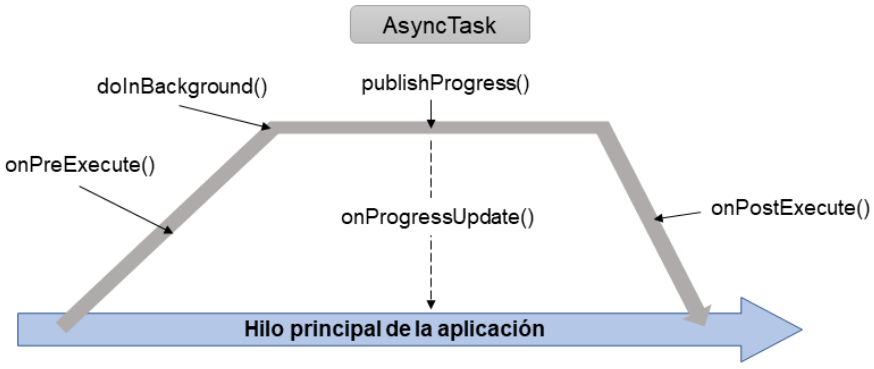


Figura 5.3: Funcionamiento de la clase AsyncTask

La utilización de esta clase se considera ideal para procesos que duren pocos segundos, como operaciones de red que no requieran la descarga de grandes cantidades de datos. En lo relativo al proyecto, se ha empleado en dos cometidos: la realización de peticiones HTTP en segundo plano a la web RouteFinder para extraer códigos de ruta, y el procesamiento de la estructura de datos de aeródromos al inicio de la aplicación. Sin duda se trata procesos cortos y no muy costosos, a los que se ajusta perfectamente la utilización de esta clase frente a otras alternativas, como por ejemplo el propio Handler o el uso de Service.

5.1.4. Bases de datos SQLite

Como se ha expuesto en la Sección 3.5, para llevar a cabo el almacenamiento de rutas en la memoria local se ha hecho uso de un sistema de gestión de bases de datos relacional SQLite. Para poder trabajar con este tipo servicio ha sido necesario implementar dos librerías propias de Android: SQLiteOpenHelper y SQLiteDatabase. La primera de ellas está destinada a la creación de la base de datos y el control de sus versiones, mientras que la segunda ofrece los métodos necesarios para la manipulación de la base de datos.

En lo relativo a las clases propias en las que se ha hecho uso de las librerías expuestas, cabe destacar el desarrollo de dos clases que dan acceso a una completa interacción con la misma. Estas clases propias son MySQLiteOpenHelper (no confundir su nombre con MySQL, otro sistema de gestión), destinada a la creación de la misma; y MyDatabase, que contiene todos los métodos necesarios para incluir, consultar y eliminar elementos de la misma. Ambas son clases estáticas contenidas en el paquete *utils*, que se expondrá en la sección siguiente.

Entrando al detalle en la estructura de la base de datos, se ha optado por distribuir cada uno de los puntos de ruta en una línea distinta, siendo las columnas atributos de ese punto de ruta. De este modo, y mediante un identificador de ruta, se ha conseguido generar una base de datos de fácil consulta y manipulación.

5.2. Estructuración del código

A nivel puramente organizativo, el código de la aplicación se encuentra dividido en seis paquetes diferenciados, los cuales agrupan clases con características similares. La organización de los paquetes se puede observar en la Figura 5.4, extraída desde el apartado de directorios de Android Studio.

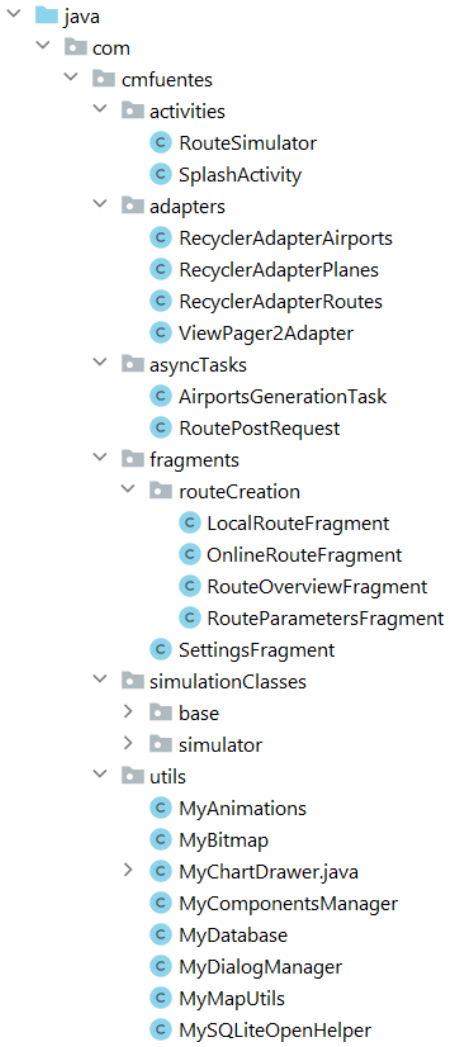


Figura 5.4: Organización de las clases de la aplicación

Dentro de esta estructuración por paquetes, cabe destacar en primer lugar los correspondientes a las actividades y los fragmentos. Concretamente, la aplicación hace uso de una única actividad (además de la SplashActivity) y un total de 5 fragmentos. Esta actividad principal, denominada en este caso RouteSimulator, es el núcleo del código de la aplicación, pues prácticamente todas las demás clases interactúan directamente con ella. La Figura 5.5 muestra un diagrama general de organización del código, donde se destaca la interacción de la clase principal (RouteSimulator) con el resto de los paquetes generados.

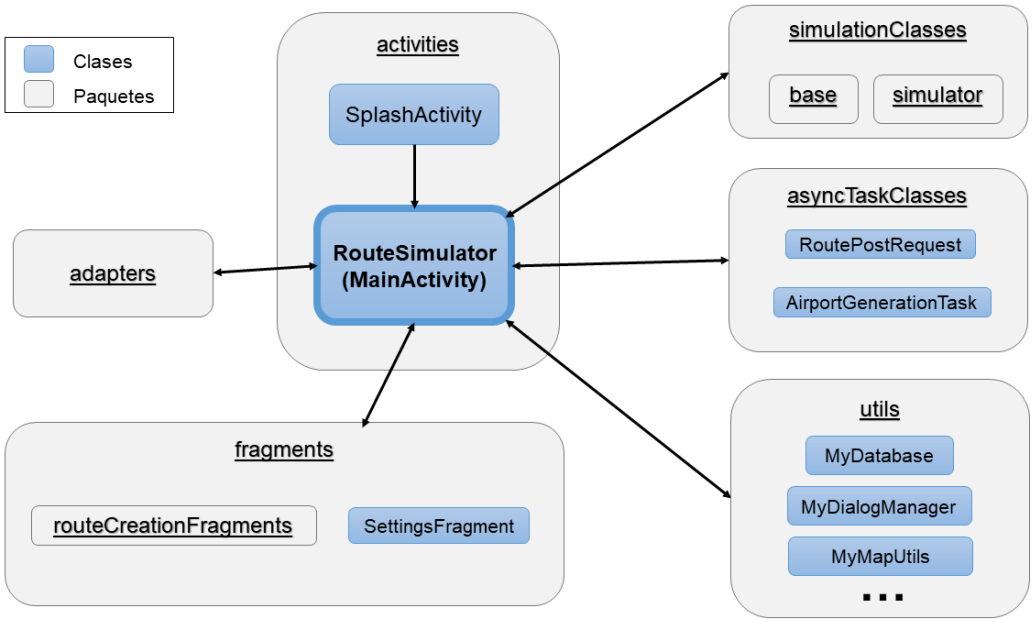


Figura 5.5: Diagrama general de los paquetes y clases principales

En los siguientes puntos de esta sección se procede a explicar con mayor o menor detalle, según la importancia de las mismas dentro del código, las clases que contienen cada uno de los paquetes.

5.2.1. Actividades

El paquete de actividades (Figura 5.6) contiene tanto la actividad principal, que en este caso se denomina RouteSimulator, como una *splash activity* o actividad de bienvenida, la cual se muestra durante unos segundos antes de dar paso a la pantalla principal. Dado que esta última carece de utilidad real, se considera a lo largo del texto que el código se basa en una única actividad.

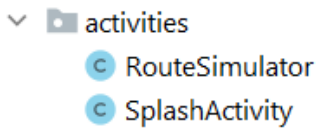


Figura 5.6: Paquete de actividades

La clase RouteSimulator, que extiende a la clase AppCompatActivity, es el núcleo de la aplicación. Además de contener sus propios métodos inherentes a la actividad,

contiene todos los métodos relativos a la implementación de la simulación de vuelos en el mapa. Estos métodos se podrían dividir en diversas categorías según su utilidad. Una buena forma de agrupar el tipo de métodos que contiene la clase RouteSimulator es la que se muestra en la Figura 5.7 (se obvian los métodos propios de la actividad, como el *onCreate()*).

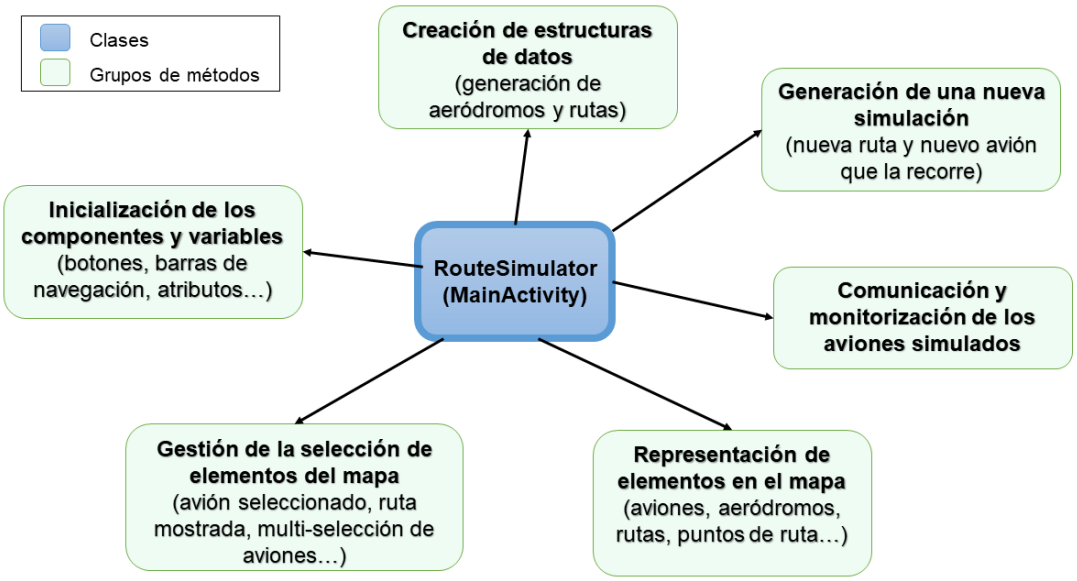


Figura 5.7: Esquema general de los métodos que contiene la clase RouteSimulator

Como se puede observar en el esquema, la actividad principal contiene todo lo relativo a la implementación del paquete de simulación de vuelos dentro del entorno gráfico de una aplicación de mapas para dispositivos Android. Evidentemente, se trata de una simplificación del contenido de la actividad, puesto que el objetivo de este punto no es detallar cada uno de los métodos que contiene. En la siguiente sección se procederá a explicar con más detalle la algorítmica de los métodos más importantes en el desarrollo de la app.

Cabe destacar que esta actividad contiene toda la interfaz gráfica principal que se mostraba en la Figura 4.3. Como se ha comentado, esta interfaz basa su diseño en un mapa, sobre el cual actúan gran parte de los métodos descritos.

5.2.2. Fragmentos

Como se ha adelantado, se ha hecho uso de una actividad principal, la cual contiene gran parte de la lógica de la aplicación. Además de esta actividad, se ha hecho uso de cinco fragmentos, los cuales se muestran en la Figura 5.8. Su utilización está destinada a pantallas o menús que precisan interacción o bien procesamiento de información introducida por el usuario.

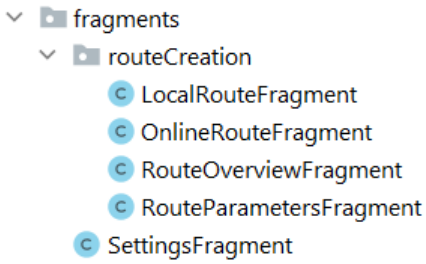


Figura 5.8: Paquete de fragmentos

Concretamente, estos fragmentos se han utilizado para dos cometidos: los menús de creación de nueva ruta y el menú de ajustes; tal y como se esquematiza en la Figura 5.9. Junto al esquema, se incluye una a continuación una breve descripción de cada uno de estos fragmentos.

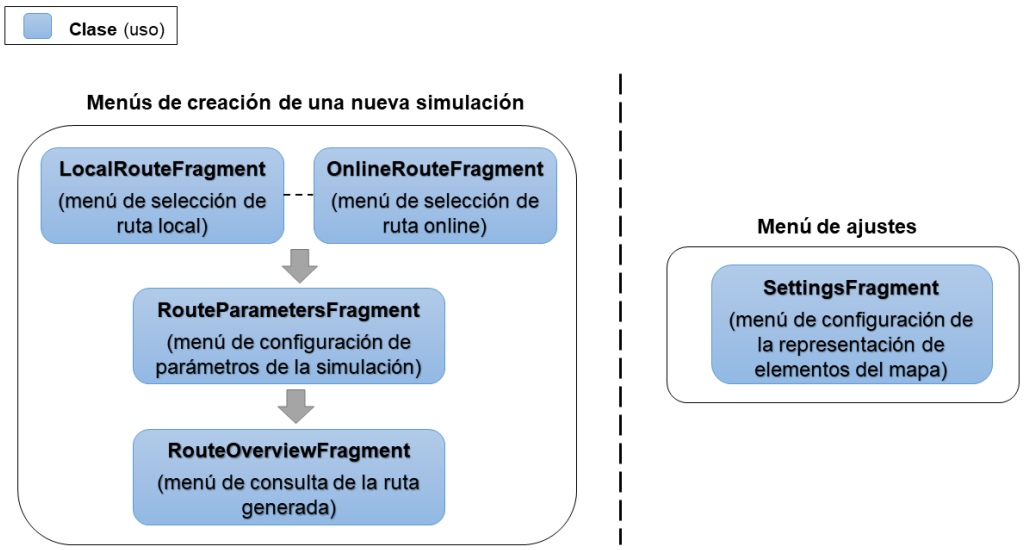


Figura 5.9: Esquema de utilización de los fragmentos

- **LocalRouteFragment** y **OnlineRouteFragment**. Se encargan de procesar la información relativa a la selección de la ruta. Su interfaz se mostraba en la Figura 4.10.
- **RouteParametersFragment**. En él se procesan los parámetros de ruta elegidos por el usuario (Figura 4.11a).
- **RouteParametersFragment**. Es el encargado de mostrar el resumen de parámetros de la ruta generada (Figura 4.11b).
- **SettingsFragment**. Muestra el menú de ajustes (Figura 4.16) y se encarga de procesar las variables relativas a la representación de elementos en el mapa.

5.2.3. Paquete de simulación de vuelos

Como se ha mencionado a lo largo de la memoria, el presente proyecto parte de un código base destinado a la simulación de vuelos. Las clases que componen este paquete se exponen en la Figura 5.10.

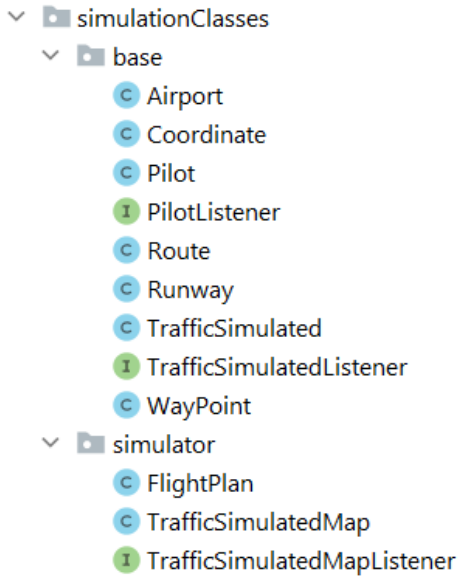


Figura 5.10: Paquete de simulación de vuelos

Como se puede observar en su estructura, el código de simulación se divide en dos paquetes diferenciados: el paquete *base* y el paquete *simulator*. El primero de ellos contiene todas las clases básicas destinadas a la simulación de vuelos, las cuales representan elementos fundamentales como: el piloto, el avión, la ruta o las coordenadas a seguir. Por su parte, el paquete *simulator* contiene clases que posibilitan sacar todo el potencial a las clases básicas, pues permiten la simulación simultánea de varios planes

de vuelo y su monitorización conjunta (Figura 5.12).

A continuación se procede a mencionar brevemente el cometido de cada una de las clases básicas implicadas en el código. Esto se acompaña de un diagrama genérico de las mismas (Figura 5.11).

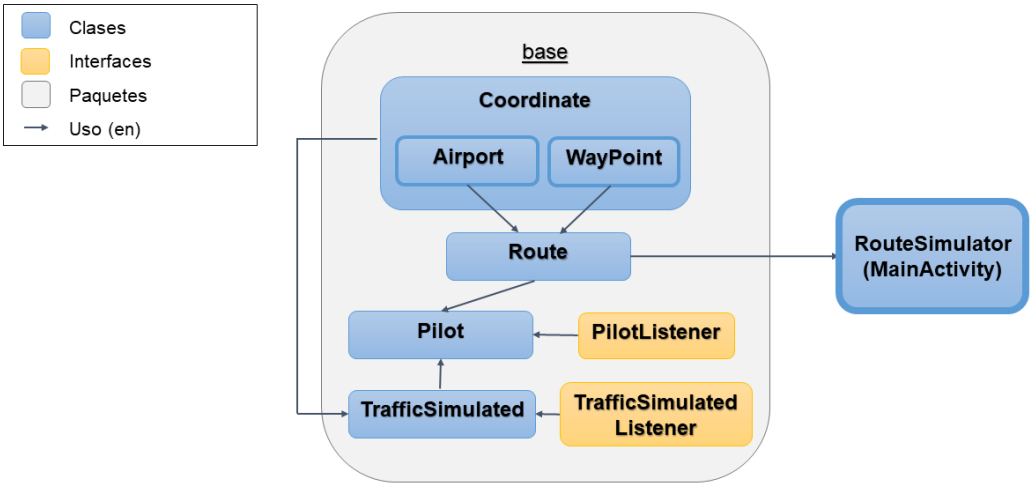


Figura 5.11: Esquema general del las clases *base* del paquete de simulación de vuelos

- **Coordinate**. Es la clase sobre la que todas las demás se asientan. Representa coordenadas de latitud, longitud y altitud e incorpora métodos para calcular parámetros y relaciones entre coordenadas.
- **WayPoint y Airport**. Son subclases de la clase coordenada que representan puntos de ruta y aeródromos respectivamente.
- **Route**. Está compuesta por dos aeródromos y diversos puntos de ruta, es la ruta que recorrerá el piloto.
- **Pilot**. Es un *thread* que controla el avión dándole consignas de vuelo. Haciendo uso de la ruta a recorrer se encarga de suministrar las coordenadas objetivo al avión y comprobar que las siga.
- **TrafficSimulated**. Es el avión que vuela las rutas. Se trata de un *thread* cuyo único objetivo es mantener actualizada su coordenada según le mande el piloto. Clase de muy bajo nivel.
- **PilotListener y TrafficSimulatedListener**. Son las interfaces que permiten que piloto y avión manden información de su estado o posición.

Si bien conviene explicar brevemente el funcionamiento o la finalidad de las clases base, únicamente la clase Route se ha utilizado directamente en la actividad principal (tal y como se muestra en la Figura 5.11). La explicación de esto es que todas estas clases base son instanciadas o implementadas en las clases del paquete *simulator*, por lo que su instancia directa en el código principal de la app no es necesaria. Consecuentemente, las clases que sí se han utilizado directamente en la actividad RouteSimulator son, además de la clase Route, las correspondientes al mencionado paquete *simulator*. Esto se puede observar en el esquema mostrado en la Figura 5.12.

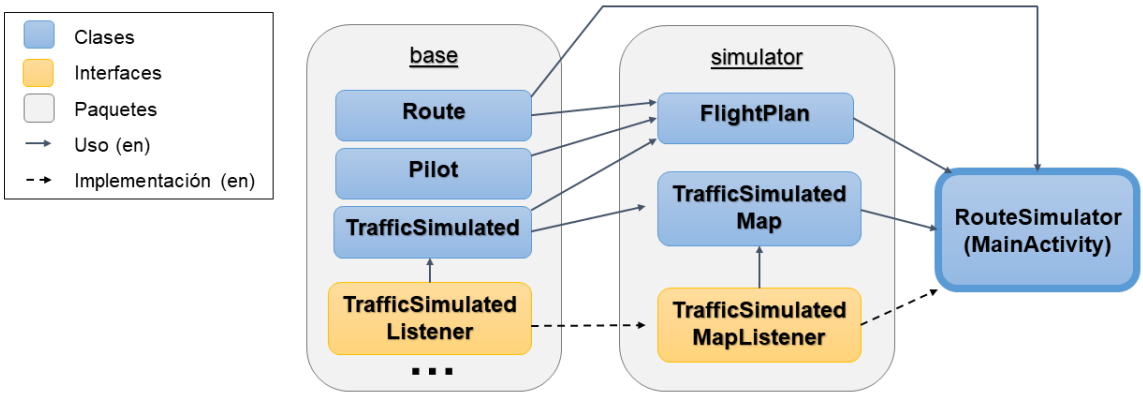


Figura 5.12: Esquema general del las clases *simulator* del paquete de simulación de vuelos

Como se ha adelantado, este paquete permite la creación de mapas de tráfico. Esto implica la simulación simultánea de varios planes de vuelo, los cuales se pueden monitorizar en conjunto. Resumidamente, las funcionalidades de las clases del paquete *simulator* son las siguientes:

- **FlightPlan**. Se trata de un *thread* que engloba las clases básicas fundamentales. Es decir, un plan de vuelo posee características similares a una ruta, pero además incluye un nombre distintivo, así como un avión y un piloto. Este plan de vuelo se puede incluir en un mapa de tráfico, creando así un grupo de aviones que se puede monitorizar de forma conjunta.

- **TrafficSimulatedMap**. Se trata del mapa de tráfico. Es un *hashmap* que contiene todos los aviones cuya simulación se encuentra activa.

- **TrafficSimulatedMapListener**. Es la interfaz que permite la comunicación con el mapa de tráfico. Permite obtener el vector de aviones que se encuentran volando y acceder a todos sus parámetros.

Como se ha adelantado, cabe destacar que este código estaba desarrollado en un principio para su uso académico en ordenador, por lo que fue necesaria una adaptación

del mismo a su funcionamiento en Android. En relación con lo expuesto en la Sección 5.1.3, uno de los cambios iniciales en el código de simulación fue la adaptación de los *threads* (piloto, avión y plan de vuelo) para que estos no se ejecutasen en el mismo plano que el hilo principal. Asimismo, a lo largo del desarrollo de la aplicación, la mayoría de las clases de simulación sufrieron cambios para adaptar su funcionamiento a los procedimientos que se iban implementando. Algunos de estos cambios son, por ejemplo, la inclusión de métodos para el cálculo del perfil vertical dentro de la clase Route, la creación de una nueva clase que representase una pista de un aeródromo (la clase Runway) o la inclusión de nuevos atributos en las clases Airport y TrafficSimulated que facilitasen su interacción con el mapa.

5.2.4. Paquete de procesamiento de datos en segundo plano

Como se ha expuesto en el Punto 5.1.3, la ejecución de hilos o tareas en segundo plano resulta fundamental para la correcta ejecución de un programa de este carácter, y más aún cuando se trata de una aplicación Android. El paquete denominado *asyncTasks* (Figura 5.13) contiene las clases encargadas de procesar información en hilos en segundo plano, las cuales heredan de la clase AsyncTask.

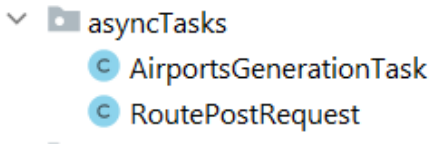


Figura 5.13: Paquete de procesamiento de datos en segundo plano

AirportsGenerationTask

Esta clase es la encargada de procesar la estructura de aeródromos al inicio de la aplicación. Concretamente, esta recoge los métodos necesarios para procesar un archivo .txt que contiene información de más de 14000 aeródromos y generar un listado de objetos de la clase Airport (*ArrayList<Airport>*). Si bien puede parecer poco eficiente realizar este procesamiento cada vez que se inicia la app, se trata de un proceso que apenas dura unos segundos, por lo que se ha preferido a otras opciones como la generación de una base de datos SQL de aeródromos.

RoutePostRequest

Es la clase encargada de realizar las peticiones en segundo plano al servidor de RouteFinder, proceso que se explicaba en el Punto 2.1.3. Haciendo uso de peticiones HTTP POST, termina generando un listado de cadenas de texto (*List<String>*) que se utiliza posteriormente para generar un objeto de clase Route. Este procedimiento se

explica con más detalle en la siguiente sección.

5.2.5. Paquete de útiles

Con el objetivo de mejorar la estructuración y limpieza del código se creó el paquete de útiles (Figura 5.14). Este paquete incluye todas aquellas clases accesorias que ejercen acciones secundarias cuya lógica puede independizarse fácilmente de la clase principal. Se trata de clases que poseen métodos estáticos que incluyen funciones como: animar botones, abrir/cerrar diálogos, mostrar/ocultar componentes, gestionar la base de datos...

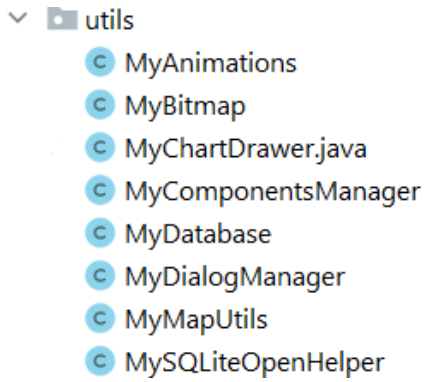


Figura 5.14: Paquete de útiles

La Figura 5.14 recoge las clases del paquete e indica cómo estas interaccionan tanto con la actividad principal como con los fragmentos.

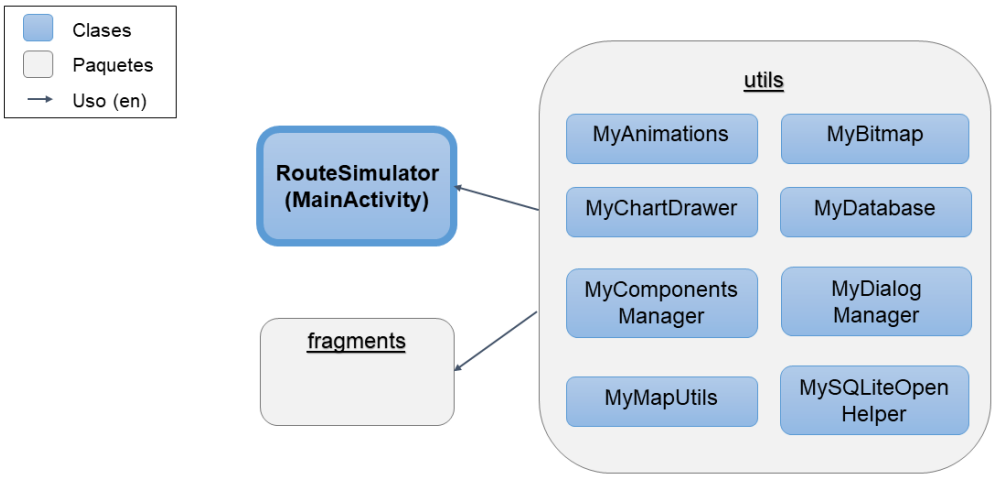


Figura 5.15: Esquema general de utilización del paquete utils

Resumidamente, las clases que componen el paquete *utils* son:

- **MyAnimations**. Contiene los métodos encargados de animar los botones flotantes correspondientes al control de texturas del mapa y el zoom.
- **MyBitmap**. Agrupa los métodos relativos a la manipulación de los mapas de bits. Esto incluye tanto redimensionado y rotación como escritura de texto sobre estos.
- **MyChartDrawer**. En ella se encuentran los métodos de dibujo de gráficas, utilizados para representar los perfiles de altitud.
- **MyComponentsManager**. Su objetivo es agrupar los métodos relativos a la gestión de la visibilidad de los componentes de la interfaz principal (botones y barras de navegación).
- **MyDatabase**. Incluye todos aquellos métodos correspondientes a la manipulación de elementos de la base de datos. Esto involucra tanto a la inicialización de la base de datos, como a la consulta y adicción de nuevos elementos en la misma.
- **MyDialogManager**. Engloba todos los métodos destinados a mostrar u ocultar los diálogos de la aplicación.
- **MyMapUtils**. Incluye métodos correspondientes a acciones secundarias relativas al mapa. Concretamente, gestiona su inicialización y la inclusión de las capas propias de la librería OSMDroid. Incluye también la gestión del minimapa que se muestra en los diálogos de aeródromo.
- **MySQLiteOpenHelper**. Clase encargada de generar la base de datos de rutas.

5.2.6. Paquete de adapters

En lo relativo a estos elementos, cabe destacar que a pesar de que en la estructuración de las clases de la aplicación se haya destinado un paquete únicamente a estos elementos, los *adapters* no se pueden considerar como uno de los principales elementos que forman parte de la aplicación. Por lo que se ha preferido no incluir este concepto previamente en la Sección 5.1, junto a otros elementos fundamentales como las actividades o los fragmentos.

Una buena definición del objeto Adapter o adaptador podría ser la de un puente que une los datos que se quieren mostrar con la vista sobre la cual se quieren mostrar, usualmente en forma de lista. En lo que concierne a la aplicación, se utilizan listados (de tipo RecyclerView) para tres funciones distintas: mostrar la estructura de datos de aeródromos, monitorizar el tráfico aéreo de forma numérica y ofrecer las rutas locales

disponibles. Con el objetivo de mostrar estos listados en una vista se hace uso, respectivamente, de las tres primeras clases que se muestran en la Figura 5.16 (las cuales extienden a la clase Adapter).

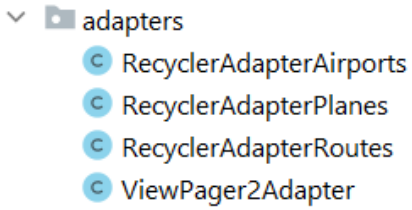


Figura 5.16: Paquete de *adapters*

Por otro lado, la cuarta de las clases que se muestra posee una funcionalidad diferente. Esta es la encargada de gestionar la vista del menú de selección de ruta (Figura 4.10), el cual posee un diseño basado en dos pestañas independientes.

5.3. Algoritmos implementados

En la presente sección se proceden a explicar los procedimientos diseñados para desarrollar algunas de las principales utilidades de la aplicación. Con el fin de ofrecer una visión general de su algorítmica, se acompañan las explicaciones del procedimiento con diagramas de flujo y fragmentos de código. Cabe destacar que las utilidades que se exponen a continuación no se corresponden directamente con las funcionalidades vistas en la Sección 4.1, sino que tratan aspectos más concretos del desarrollo que han resultado clave para la consecución de estas funcionalidades.

5.3.1. Generación de la estructura de aeródromos y base de datos de rutas

Cuando se inicia la aplicación, automáticamente se ejecutan dos procesos en segundo plano cuya finalidad es la generación de las estructuras de datos fundamentales de la aplicación: los aeródromos y las rutas locales. Sin embargo, ambos procesos presentan una lógica diferente en cuanto la forma de procesar y almacenar los datos. La algorítmica de estos procesos se expone en la Figura 5.17.

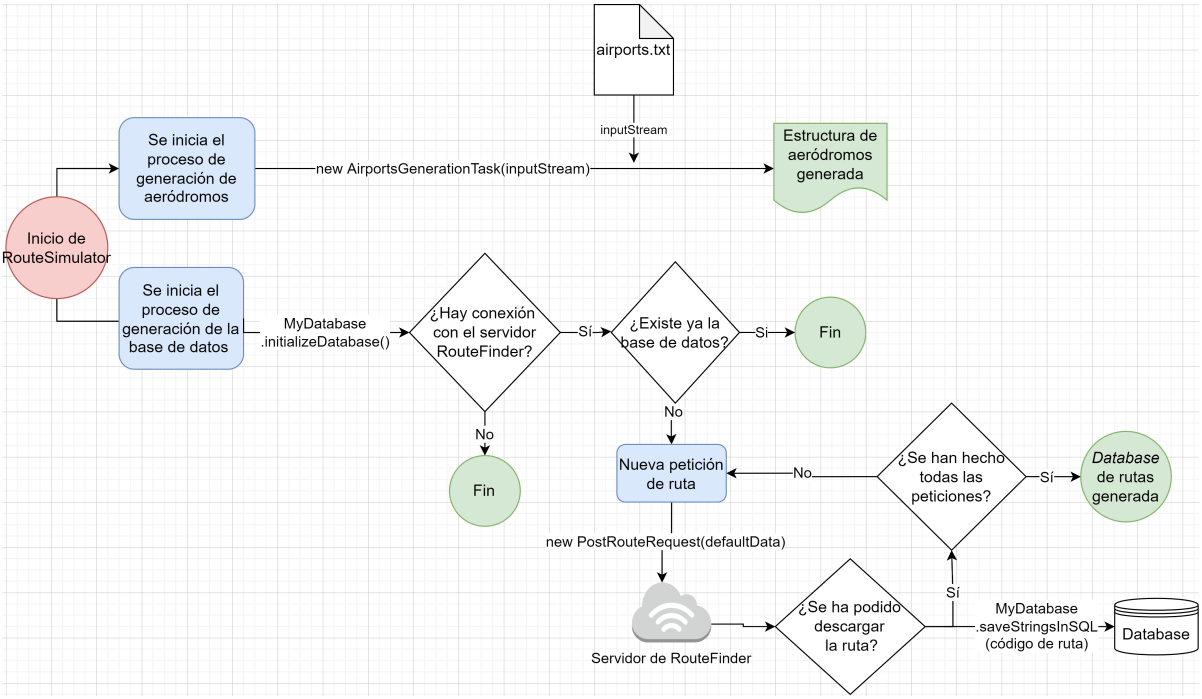


Figura 5.17: Diagrama de flujo del proceso la generación de estructuras de datos

Por un lado, la generación de la estructura de datos de aeródromos se lleva a cabo haciendo uso de la tarea asíncrona AirportsGenerationTask (expuesta en el Punto 5.2.4). Esta recibe datos desde un archivo de texto almacenado en la memoria de la aplicación, los procesa y los almacena en un listado de objetos de la clase Airport, concretamente un ArrayList<Airport>. Cabe destacar que este proceso se lleva a cabo cada una de las veces que se inicia la app, puesto que como se ha adelantado, se trata de un proceso que apenas dura unos segundos y no implica una pérdida de tiempo significativa para el usuario.

Por otro lado, la generación y almacenamiento de rutas se lleva a cabo de distinta manera, haciendo uso del sistema de gestión de bases de datos SQLite. A diferencia del anterior proceso, este solo se lleva a cabo la primera vez que el usuario ejecuta la aplicación en su dispositivo y no se vuelve a realizar en posteriores ejecuciones. Se trata de un proceso más complejo que hace uso de peticiones online al servidor de RouteFinder, mediante la clase RoutePostRequest (expuesta en el Punto 5.2.4), para procesar y descargar rutas que se encuentran previamente definidas. Este proceso precisa de conexión al servidor de RouteFinder, por lo que previamente a su inicio se comprueba la conexión con el fin de asegurar la correcta generación de la base de datos.

5.3.2. Petición y descarga de rutas desde servicio externo

Tanto para llevar a cabo el proceso de generación de la base de datos de rutas (visto en el punto anterior) como para el proceso de generación de una nueva ruta por parte del usuario, es preciso realizar peticiones HTTP POST al servidor de RouteFinder, haciendo uso de la clase RoutePostRequest. Aunque se ha hecho referencia a este proceso en reiteradas ocasiones a lo largo de la presente memoria, no se ha entrado al detalle en su lógica. Con este fin, el diagrama de la Figura 5.18 expone de manera resumida la algorítmica del proceso de petición de ruta.

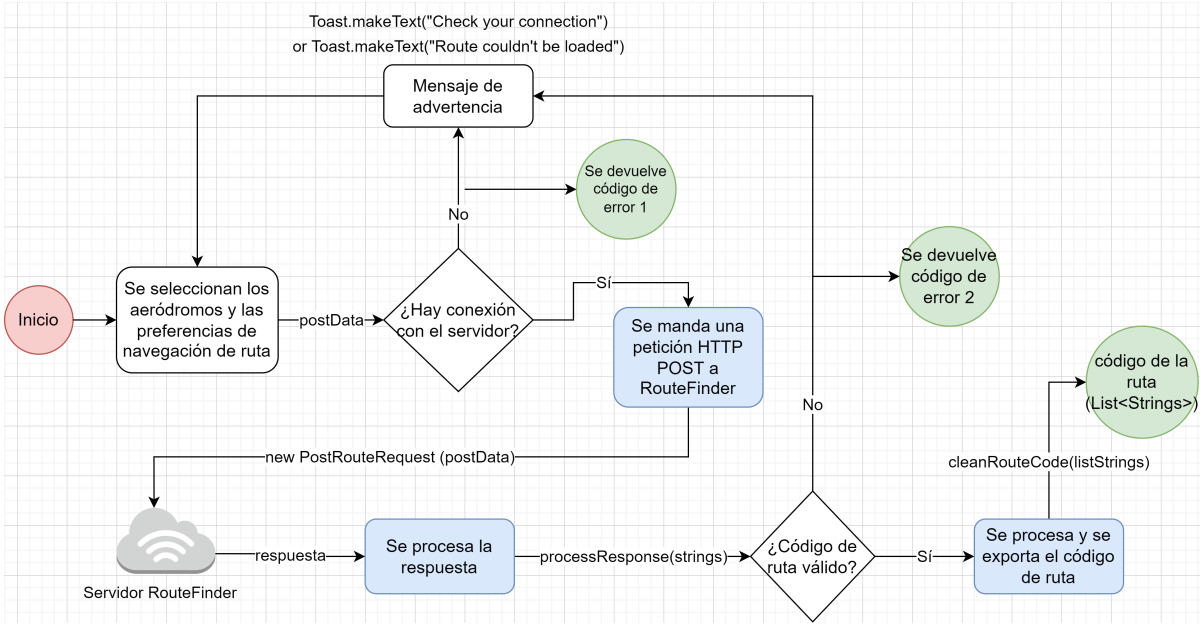


Figura 5.18: Diagrama de flujo del proceso de petición de ruta online

Como se puede observar, el primer paso es seleccionar los datos que se van a introducir en el formulario de solicitud de rutas de RouteFinder (El cual se mostraba en la Figura 2.3). Una vez se hayan agrupado estos parámetros (concretamente haciendo uso de un HashMap) se comprueba en primer lugar el estado de conexión con el servidor de RouteFinder. En el caso de que no haya conexión, se muestra un código de error que indica esta incidencia, y se debería volver a comprobar la conexión para realizar la petición. Una vez comprobada la conexión, se manda la petición POST al servidor de RouteFinder y se procesa su respuesta. En el caso de que no se haya podido procesar la ruta para los parámetros enviados, se mostrará un código de error distinto que indique un resultado erróneo en la petición. Finalmente, si la comprobación del texto obtenido en la petición resulta satisfactoria, se termina procesando este para generar el código final de la ruta (en forma de *List<String>*).

Cabe mencionar que este código de ruta es el que posteriormente se usa en la clase Route para generar objetos de tipo Airport y WayPoint, los cuales definen la ruta a recorrer.

5.3.3. Procedimiento de generación de nueva simulación

En la Figura 4.9 se exponían los pasos a llevar a cabo para generar una nueva simulación. Sin embargo, se trata de una ilustración resumida de los mismos. Poniendo el foco únicamente en los pasos importantes de la lógica del proceso, el diagrama de flujo de la Figura 5.19 busca dar una aproximación más detallada de este proceso en lo que a su implementación se refiere. Es preciso señalar que se omite incluir la posibilidad de que el usuario retroceda en los pasos del procesamiento, pues únicamente saturaría el diagrama sin aportar información relevante.

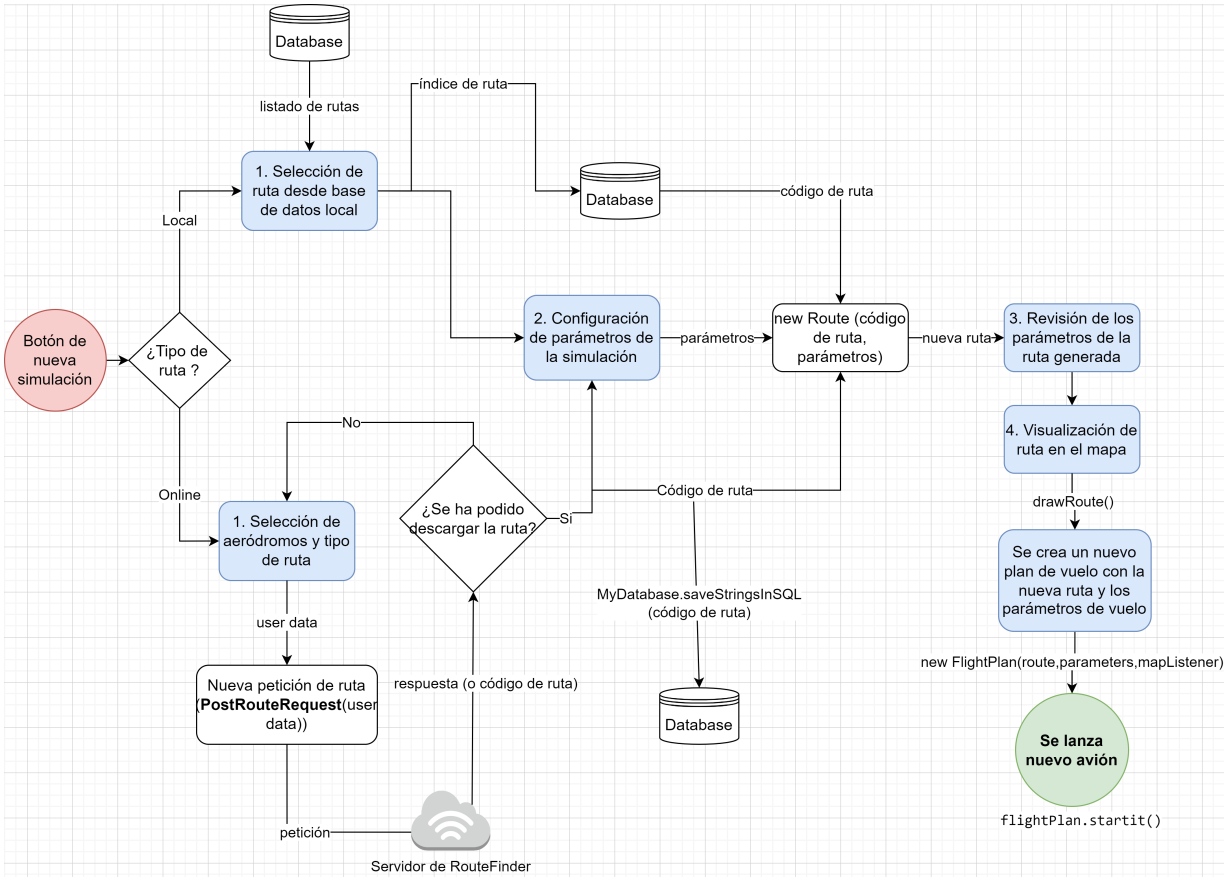


Figura 5.19: Diagrama de flujo del proceso de generación de una nueva simulación

Como se ha reiterado a lo largo del Capítulo 4, existen dos vías para iniciar una simulación: la selección de una ruta local o el procesamiento de una ruta online. El resultado de ambas vías es el mismo: la generación de un código de ruta; el cual incluye todos los datos de los aeródromos origen y destino y de los puntos de ruta a seguir. La primera de ellas, tal y como se observa en el diagrama, implica la conexión con la base de datos local de rutas. Esta interacción con la base de datos implica tanto su consulta, para ofrecer las rutas al usuario en forma de lista (Figura 4.10a); como la extracción del código de ruta, según la ruta seleccionada por el usuario. Por su parte, la generación del código de ruta de forma online implica poseer conexión a internet para mandar peticiones al servidor de RouteFinder, así como conectarse con la base de datos local para guardar automáticamente el código obtenido.

Una vez generado el código de ruta, el proceso de generación de una nueva simulación se unifica en una sola vía. Esta pasa por configurar los parámetros de la simulación (tal y como se reflejaba en la Figura 4.11a), que junto al código de ruta generado permiten crear un nuevo objeto Route, el cual es la ruta a navegar por el avión simulado. A continuación se expone el resumen de los principales parámetros de la ruta (Figura 4.11b), seguido de la visualización en el mapa de misma (Figura 4.12a). Finalmente, el último paso a seguir es generar un nuevo plan de vuelo, instanciando un objeto de tipo FlightPlan. Para ello, se hace uso tanto de la ruta (objeto de tipo Route), como de los parámetros configurados en el punto 2 del esquema. Llamando al método de inicio del plan de vuelo, se lanza el nuevo avión (nuevo *thread*) y se termina el proceso.

5.3.4. Representación de aviones en el mapa

Uno de los algoritmos fundamentales desarrollados para la aplicación es sin duda el proceso de representación de los aviones en el mapa. Se trata de un proceso que en un principio supuso cierta dificultad de implementación, pues la librería OSMDroid no está adaptada a la representación de elementos en el mapa que se encuentren constantemente en movimiento. Sin embargo, adaptando el uso de la clase ItemizedIconOverlay se ha conseguido llevar a cabo la representación de los aviones de forma fluida. Además, esta clase incluye su propia implementación de *listeners* para los iconos de la capa, facilitando así la interacción con los aviones. La implementación de esta funcionalidad se expone de manera general en la Figura 5.20.

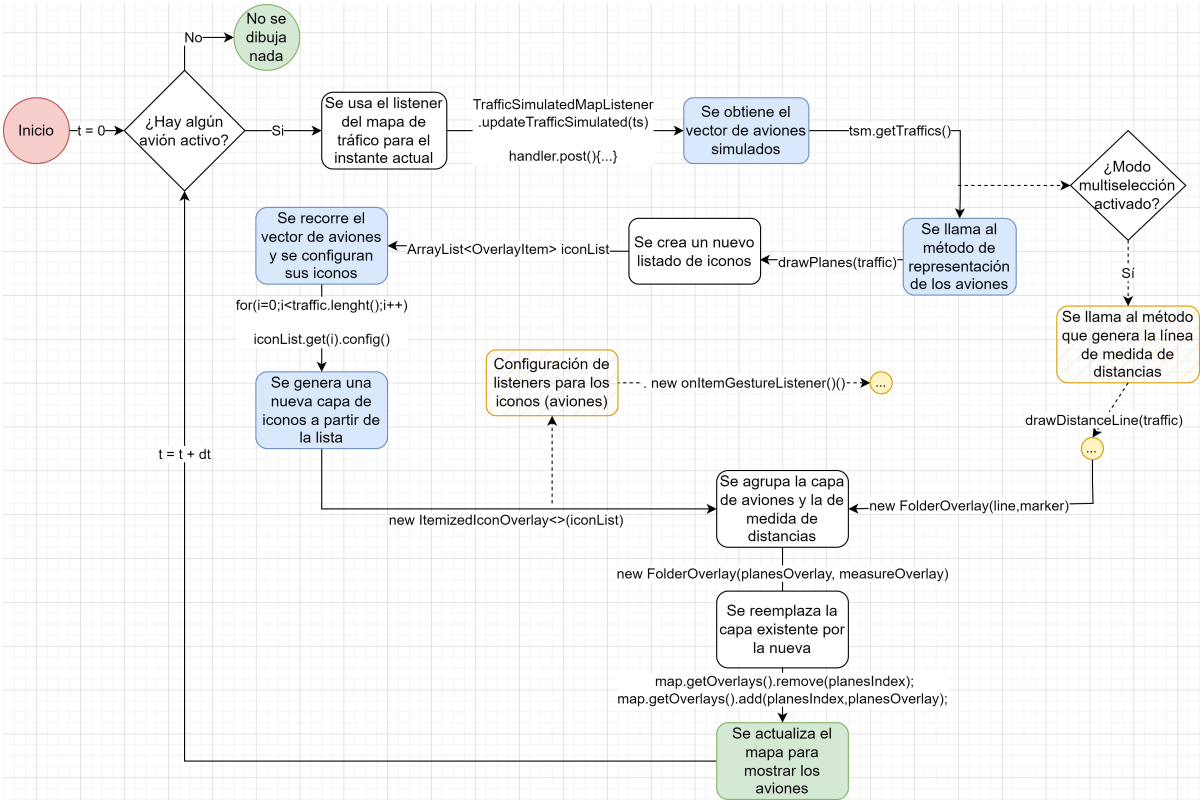


Figura 5.20: Diagrama de flujo del proceso de representación de aviones en el mapa

El proceso de representación de aviones comienza por comprobar si hay alguna simulación activa comunicándose a través de su interfaz (*listener*), pues el método de representación se llama en cada instante que un avión envíe información de su posición. Cuando un avión se comunica con la actividad principal, se obtiene el vector de tráfico aéreo (objeto de tipo TrafficSimulated[]), para pasarlo posteriormente al método encargado de dibujar los aviones en el mapa. En este punto se comprueba si el modo multiselección (o modo de medida de distancias) se encuentra activo, para llamar también al método encargado de dibujar la línea de medida.

El funcionamiento del método que "dibuja" los aviones en el mapa se expone resumidamente en el diagrama expuesto. Este comienza con la creación de un listado de iconos propios de la librería OSMDroid (de la clase OverlayItem), el cual se corresponde con cada uno de los aviones simulados. Seguidamente, se recorre el vector de tráfico para configurar cada uno de los iconos, según su latitud, longitud, altitud, rumbo y estado de selección (seleccionado o no). Este último paso implica tanto situar el icono adecuadamente en el mapa como la manipulación del mapa de bits que lo representa (para lo que se hace uso de los métodos estáticos de la clase MyBitmap). Una vez configurados todos los iconos, se crea una nueva capa de aviones instanciando la clase

ItemizedIconOverlay, la cual lleva implícita la generación de *listeners* para cada uno de los iconos representados (lo que se esquematizará a continuación). Finalmente, la capa de aviones se combina con la capa que muestra la línea de medida de distancias, generando un nuevo conjunto de capas de tipo *FolderOverlay*, el cual se sustituye por su homólogo para el instante temporal anterior. Cabe destacar que el hecho de combinar la capa de aviones y la de medida (la cual estará vacía si no se activa la multiselección) se debe a motivos de organización de las capas.

Selección de aviones

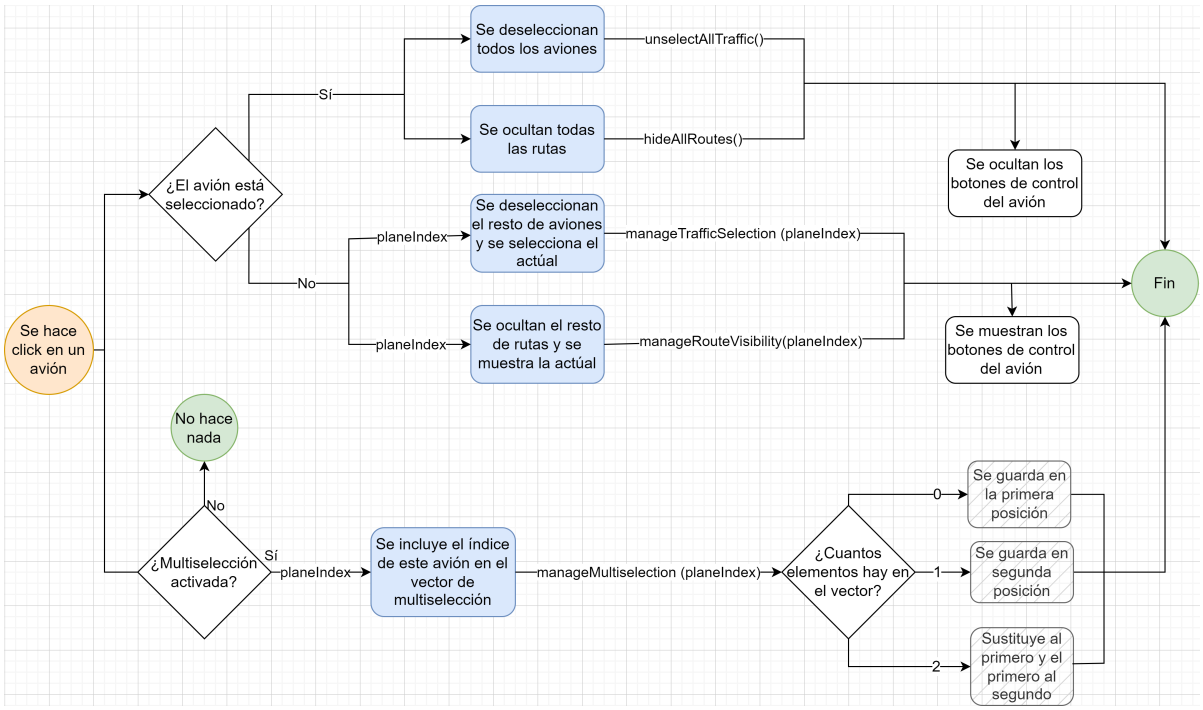


Figura 5.21: Diagrama de flujo del proceso de selección de aviones desde el mapa

En el diagrama de la Figura 5.20 se hace referencia a la configuración de los *listeners* para los iconos de aviones, sin entrar al detalle de su implementación. Por su parte, la Figura 5.21 expone la algorítmica que hay detrás de este proceso, aplicable a la selección de aviones tanto desde el mapa como desde el listado de tráfico (el cual se mostraba en la Figura 4.14). Su lógica es simple: si cuando un avión es seleccionado no se encontraba previamente en ese estado, se marca este como seleccionado y se desmarcan como tal todos los demás. Si, por el contrario, el avión ya estaba seleccionado, simplemente se marca como no seleccionado todo el tráfico aéreo. Esta lógica presenta un comportamiento homólogo en lo relativo a la ruta que sigue el avión seleccionado: se muestra en el mapa cuando este se selecciona y se oculta cuando se marca como no seleccionado.

Por otra parte, la selección de un avión puede implicar otra vía relacionada con el modo multiselección o modo de medida. En el caso de que este se encuentre activo al seleccionar un avión, se gestionará el vector de aviones incluidos en la medida. Esto se hace de forma sencilla mediante un vector de 2 posiciones que contiene los índices de los dos aviones implicados. Al seleccionar un avión con este modo activo, este se almacenará en una de las posiciones del vector, siguiendo la lógica simple que se expone en el diagrama. Este vector se utiliza después para generar la línea de medida de distancia, un proceso que se ha simplificado en el diagrama de la Figura 5.20 y que se expone a continuación con más detalle en la Figura 5.22.

Medida de distancia entre aviones

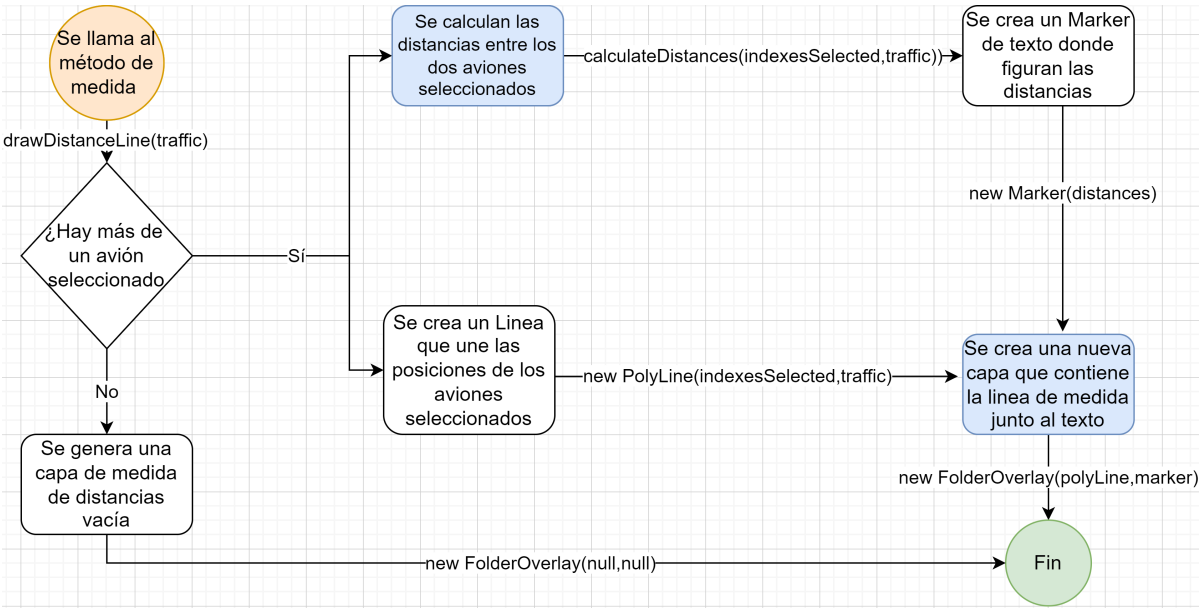


Figura 5.22: Diagrama de flujo del proceso de generación de la línea de medida de distancias

El diagrama de flujo expuesto profundiza con mayor detalle en el proceso de medida de distancias entre aeronaves. Como se exponía en el diagrama general sobre el proceso de representación de aviones (Figura 5.20), la generación de la línea de medida de distancias se lleva a cabo para cada iteración temporal, siempre y cuando este modo se encuentre activo. Dentro de este proceso, se comienza comprobando si el vector de aviones implicados en la medida contiene más de un avión. Si es así, se comienza el proceso de generación de la línea de medida. Por el contrario, si esta condición no se cumple, la capa (FolderOverlay) correspondiente a la línea de medida de distancias no contendrá ningún elemento.

El proceso de generación de la línea que mide las distancias entre los dos aviones seleccionados consta de dos subprocesos, correspondientes a la creación de los dos elementos que componen la línea. Estos elementos son: un Marker de texto que incluye los valores de las distancias calculadas y la propia línea que une los aviones. En el primero de los subprocesos será necesario calcular las distancias, mientras que en el segundo únicamente son necesarias las posiciones de los aviones para el instante correspondiente. Finalmente, ambos elementos son agrupados haciendo uso de un FolderOverlay, que posteriormente se representará junto a la capa de aviones, tal y como se ha expuesto previamente.

5.3.5. Monitorización de parámetros del tráfico aéreo

En lo relativo a los pasos de escucha del tráfico aéreo y extracción del vector de aviones simulados, el proceso de monitorización sigue los mismos pasos que el correspondiente a la representación gráfica de los aviones, tal y como se observa en los primeros bloques del diagrama de la Figura 5.23). Es decir, se extrae el vector de tráfico para cada paso temporal de cualquiera de los aviones y se llama al método en cuestión. Si bien pudiera tratarse de un único método, en este caso se emplean tres métodos distintos, uno para cada proceso de monitorización: información del avión seleccionado (cuya interfaz gráfica se exponía en la Figura 4.13a), información de la ruta que sigue ese avión (Figura 4.13b) e información de todo el tráfico aéreo (Figura 4.14).

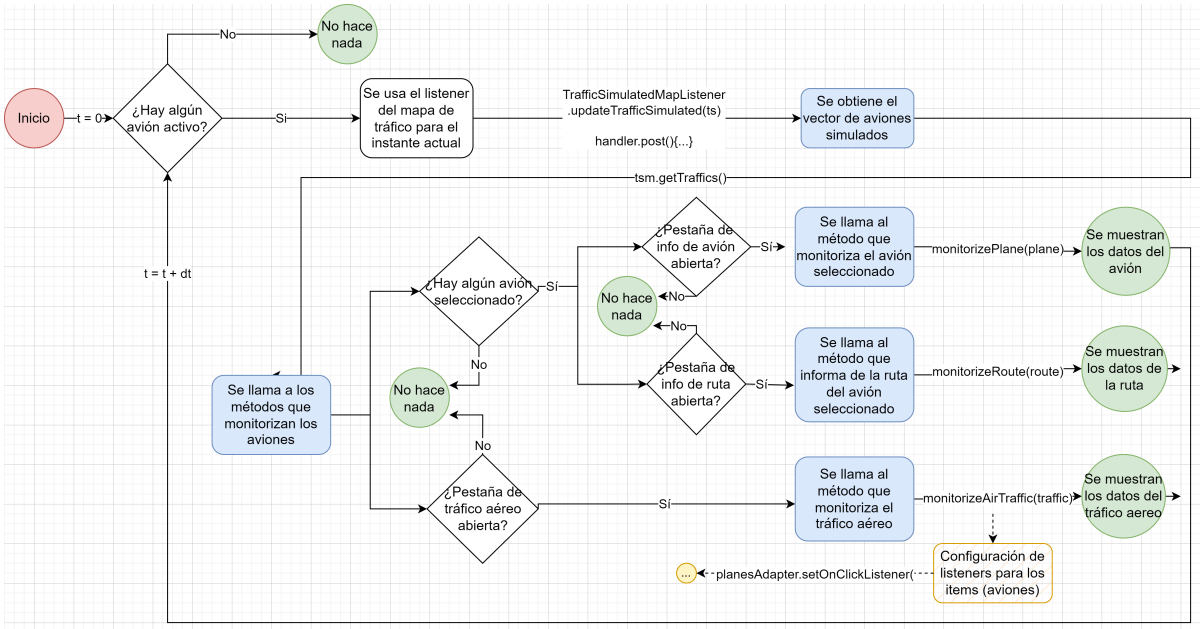


Figura 5.23: Diagrama de flujo del proceso de monitorización de parámetros del tráfico aéreo

Si bien es cierto que el procedimiento presenta un inicio similar al visto en el punto anterior, el proceso de monitorización no siempre se encuentra activo, ya que en este caso las pestañas de información pueden estar abiertas o no. Es por ello que, con el fin de ahorrar en coste computacional, resulta esencial una comprobación previa del estado de las pestañas antes de llamar al método que extrae la información del avión o aviones en cuestión. Si la pestaña se encuentra abierta, entonces se extrae la información; si no, simplemente se omite la monitorización. Asimismo, cabe destacar que tanto la llamada al método que extrae y muestra los parámetros del avión seleccionado como la relativa al método de información de su ruta, se deben hacer únicamente cuando alguno de los aviones esté seleccionado, tal y como muestra el diagrama. Finalmente, cabe destacar que la llamada al método de monitorización del tráfico aéreo implica la configuración de los *listeners* para los elementos del listado de tráfico, unos *listeners* que siguen la lógica exhibida previamente en la Figura 5.21.

5.4. Desarrollo de la parte gráfica

Como resumen al trabajo realizado sobre la parte gráfica de la aplicación se incluye esta breve sección, la cual trata aspectos relativos a los *layouts* y otros componentes fundamentales en el desarrollo de la GUI.

La Figura 5.24 muestra el paquete de *layouts* utilizados para dar forma a las distintas pantallas, pestañas y ventanas que conforman la interfaz de usuario de la aplicación. Como se puede observar, sus nombres inician con el tipo de elemento al que pertenecen. Estos elementos van desde actividades a fragmentos, pasando por otros que se encuentran dentro de estos últimos, entre los que destacan los diálogos o listados (de tipo RecyclerView).

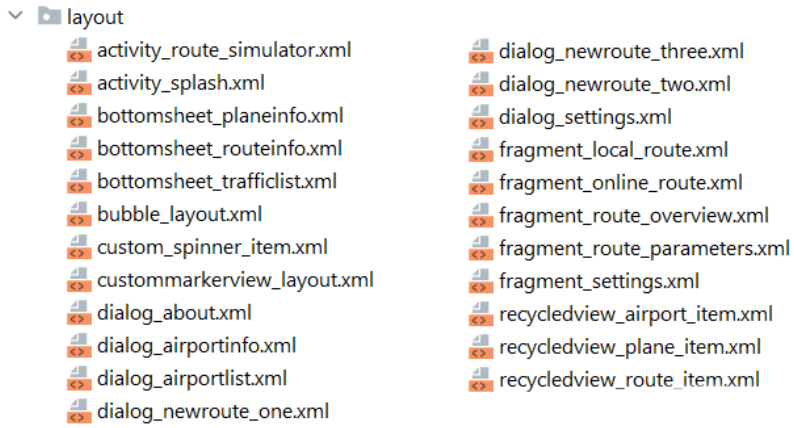


Figura 5.24: Paquete de *layouts*

Algunos puntos fundamentales a destacar sobre el desarrollo de la GUI son:

- En cuanto al tipo de *layouts* utilizados para desarrollar la interfaz, se ha optado en la mayoría de los casos por usar ConstraintLayout, con el fin de generar una GUI responsiva (cuyas proporciones se ajusten a pantallas de distintas dimensiones).
- Los elementos utilizados en la composición de la parte gráfica de la actividad principal son: Toolbar, BottomAppBar, FloatingActionButton, ImageButton y Button. Todos ellos presentan un alto grado de personalización.
- Las distintas ventanas de navegación hacen uso directo de la clase Dialog, mientras que las pestañas inferiores utilizan la subclase BottomSheetDialog. Dentro de los *dialogs* correspondientes a la creación de una nueva simulación se muestran los fragmentos correspondientes.
- La gran mayoría de los iconos que se muestran sobre los botones de las barras de navegación y de los botones tienen formato de vector SVG.
- La gran mayoría de los iconos que se muestran en la app han sido procesados mediante el software GIMP, con el fin de generar símbolos propios que se adaptasen mejor al contexto de la imagen.
- Los listados de aeropuertos, tráfico aéreo y rutas locales se han generado haciendo uso de la clase RecyclerView. Para rellenar sus elementos ha sido necesario el uso de *adapters* (expuestos en el Punto 5.2.6).

6. Conclusiones

En la presente memoria se ha expuesto de manera sintetizada todo un proceso de desarrollo que ha llevado meses de trabajo por parte del alumno. Este proceso ha dado como resultado una aplicación Android de simulación de vuelos sobre rutas aéreas extraídas desde bases de datos de aeronavegación. La última versión de la aplicación cuenta con diversas funcionalidades que pueden resultar de utilidad en la simulación de vuelos, o cuanto menos, presentarse como una herramienta atractiva para todo aquel interesado en el sector de la aeronavegación.

Con este proyecto se ha buscado desarrollar una aplicación diferente a las que se pueden encontrar a día de hoy en el mercado de apps gratuitas para Android, dentro del sector de la aeronavegación. A pesar de tratarse de una aplicación sencilla en comparación con el software profesional que existe en el campo de la aeronavegación, esta incluye funcionalidades que pueden destacarla frente al resto, puesto que desde el principio de su desarrollo se ha puesto el foco en un segmento poco desarrollado: la simulación de vuelos en mapas 2D.

Cabe destacar que desde que se comenzó con su desarrollo, el aspecto estético de la aplicación ha sido uno de los puntos en los que se ha invertido más dedicación. A pesar de carecer de conocimientos previos en este campo, se ha buscado generar una interfaz cuya estética no difiriese en exceso de las aplicaciones desarrolladas por profesionales. Si bien es cierto que la elección y el uso de ciertos componentes posiblemente no cumpla con los cánones marcados en el desarrollo de este tipo de interfaces, se puede concluir que se ha cumplido con los objetivos relacionados con la estética de la app.

En lo relativo al aspecto programático del proyecto, conviene en primer lugar subrayar dos conceptos que han resultado claves en el desarrollo: la adecuada gestión de los procesos (hilos) y el uso de librerías externas, destacando la librería osmdroid. Por otro lado, poniendo el foco en la estructuración del código, se es consciente de que la limpieza y organización del mismo podrían ser mejorables empleando arquitecturas de desarrollo [30], como podrían ser la MVC (del inglés *model-view-controller*), MVP (*model-view-presenter*) o la MVVM (*model-view-viewmodel*). Sin embargo, considerando que el alumno partía desde cero en el desarrollo Android y que se trata de un proyecto de desarrollo unipersonal, se ha omitido hacer uso de modelos de organización establecidos en el sector. Pese a ello, se ha buscado organizar el código de acuerdo a la lógica de cada una de las partes del mismo, de modo que su estructura presente

coherencia y pueda ser interpretado por otros desarrolladores. Finalmente, en cuanto a la algorítmica implementada, conviene remarcar la importancia del proceso de petición de rutas y el propio de la representación e interacción con los aviones, siendo ambos clave en el desarrollo de la app.

Por último, dado que se trata de uno de los motivos principales por los cuales se eligió este proyecto, cabe hacer mención al aprendizaje que se ha llevado a cabo a lo largo del desarrollo de la aplicación, tanto en el lenguaje Java como en el campo del desarrollo para dispositivos Android. Pese a que la base de conocimientos de Java era escasa y la formación en el desarrollo de apps era nula, se ha logrado adquirir los conocimientos suficientes como para poder desarrollar una aplicación funcional con gran cantidad de utilidades y una navegación amigable para el usuario. Pese a que el desarrollo de aplicaciones móviles sea ajeno a los contenidos propios del cuadro lectivo de la Ingeniería Aeronáutica, adquirir conocimientos del sector se valora positivamente, puesto que se trata de un ámbito interesante, a la vez que un sector en auge que va a seguir creciendo durante los próximos años [31].

7. Trabajos futuros

A lo largo del desarrollo del proyecto se han ido tomando decisiones relacionadas con el alcance o las posibles funcionalidades que podía incorporar la aplicación. Tanto por exceder los límites que se plantearon en un principio para el proyecto como por haber optado por dedicar los esfuerzos al desarrollo de otras mejoras o funcionalidades, se han dejado como posibles trabajos futuros o mejoras ciertas ideas de desarrollo. Algunas ideas interesantes podrían ser:

- Desarrollar el código base para generar cálculos de consumo de combustible, así como un perfil vertical más complejo. Para la primera de las ideas se podría hacer uso, por ejemplo, de las ecuaciones propias del modelo de rendimiento de aeronaves BADA [32]. Por otro lado, una buena opción para generar perfiles verticales más complejos podría ser la implementación de procedimientos de aproximación extraídos desde bases de datos de navegación aeronáuticas [8].

- Implementar un servicio externo de generación de rutas más complejo, como por ejemplo SimBrief. Esto podría ofrecer al usuario una mayor cantidad de parámetros de ruta, aunque habría que valorar la complejidad de su implementación frente a la del servicio ofrecido por RouteFinder. Asimismo, se incrementaría la cantidad de información ofrecida por un servicio externo, por lo que la "autoría" de los datos mostrados en la aplicación se reduciría.

- Valorar la posibilidad de realizar un procesamiento interno de las bases de datos de navegación. A pesar de que desde un principio se planteó como una tarea de programación que excedía los límites del proyecto, se podría investigar a cerca de su viabilidad de cara a futuros proyectos o versiones de la app.

- Permitir al usuario crear sus propias rutas personalizadas. Una posible implementación pasaría por guiar al usuario durante la creación de la ruta en función de los puntos que vaya seleccionando, ofreciéndole puntos de ruta o aerovías cercanas de acuerdo con los patrones de aeronavegación establecidos para vuelos reales.

Bibliografía

[1] Flightradar24. Flightradar24 - live air traffic, 2022. URL https://www.flightradar24.com/. 12 Mayo 2022.

[2] RORTOS Company. Airline commander, 2022. URL https://play.google.com/store/apps/details?id=it.rortos.realflight&hl=es&gl=US. 06 Abril 2022.

[3] Inc. Garmin International. Garmin pilot, 2022. URL https://play.google.com/store/apps/details?id=com.digcy.pilot. 27 Febrero 2022.

[4] Divelements Limited. Skydemon, 2022. URL https://play.google.com/store/apps/details?id=aero.skydemon.skydemonandroid. 27 Febrero 2022.

[5] Navigraph. Simbrief, 2022. URL https://www.simbrief.com/home/. 05 Abril 2022.

[6] Google. Google play, 2022. URL https://play.google.com/store/games?hl=es. 15 Abril 2022.

[7] LLC Aircraft Performance Group. Rocketroute, 2022. URL https://www.rocketroute.com/. 05 Abril 2022.

[8] Navigraph. Navigraph: Simulated flight, real navigation, 2022. URL https://navigraph.com/. 05 Abril 2022.

[9] ASA Link Team. Routefinder (free access area), 2022. URL http://rfinder.asalink.net/free/. 28 Junio 2022.

[10] ASA Link Team. Asa link italy. aeronautic information tecnology, 2022. URL http://www.asalink.net/. 05 Abril 2022.

[11] Jozzae. Flight planner - flight planning for flight simulation, 2022. URL https://play.google.com/store/apps/details?id=com.wadoft.flightplanner. 26 Febrero 2022.

[12] RocketRoute Ltd. Rocketroute flightplan, 2022. URL https://play.google.com/store/apps/details?id=com.RocketRoute&hl=es&gl=US. 26 Febrero 2022.

[13] Oracle Corporation. Java | oracle, 2022. URL https://www.java.com/es/. 27 Febrero 2022.

[14] Google. Android - la plataforma que amplía los límites de lo posible, 2022. URL https://www.android.com/intl/es_es/. 22 Febrero 2022.

[15] Android Developers. Android para desarrolladores, 2022. URL https://developer.android.com/. 22 Junio 2022.

[16] OpenStreetMap Team. The openstreetmap project, 2022. URL https://www.openstreetmap.org/. 25 Abril 2022.

[17] OpenStreetMap-Tools for Android. Github: osmdroid, 2022. URL https://github.com/osmdroid/osmdroid. 10 Marzo 2022.

[18] MKergal. Github: osmbonuspack, 2022. URL https://github.com/MKergall/osmbonuspack. 18 Marzo 2022.

[19] MySQL AB. Mysql, 2022. URL https://www.mysql.com. 15 Abril 2022.

[20] Android Developers. Enfoque de prioridad de kotlin en android, 2022. URL https://developer.android.com/kotlin/first?hl=es-419. 02 Marzo 2022.

[21] harkiran78 Geeksforgeeks. Top programming languages for android app development, 2022. URL https://www.geeksforgeeks.org/top-programming-languages-for-android-app-development/. 02 Marzo 2022.

[22] Gradle Inc. Gradle build tool, 2022. URL https://gradle.org/. 22 Febrero 2022.

[23] Android Developers. Svg - cómo agregar gráficos vectoriales de varias densidades, 2022. URL https://developer.android.com/studio/write/vector-asset-studio?hl=es-419. 28 Marzo 2022.

[24] The GIMP Team. Gimp - gnu image manipulation program, 2022. URL https://www.gimp.org/. 02 Marzo 2022.

[25] Genymobile Company. Genymotion – android emulator for app testing cross-platform apps, 2022. URL https://www.genymotion.com/. 05 Marzo 2022.

[26] MDN contributors. Peticiones post - mdn web docs - mozilla, 2022. URL https://developer.mozilla.org/es/docs/Web/HTTP/Methods/POST. 14 Abril 2022.

[27] Inc Square. Overview - okhttp - square open source, 2022. URL https://square.github.io/okhttp/. 15 Abril 2022.

[28] Phillip Jahoda. Mpandroidchart, 2022. URL https://github.com/PhilJay/MPAndroidChart. 10 Abril 2022.

[29] Android Developers. Asynctask | android developers, 2022. URL `https://developer.android.com/reference/android/os/AsyncTask`. 08 Marzo 2022.

[30] Android Developers. Guía de arquitectura de apps, 2022. URL `https://developer.android.com/jetpack/guide?hl=es-419`. 08 Marzo 2022.

[31] Tangram Consulting. Aplicaciones móviles: un sector al alza, 2022. URL `https://tangramconsulting.es/noticias/como-esta-cambiando-el-sector-de-aplicaciones-moviles`. 25 Julio 2022.

[32] A. Nuic, P. Damir, and V. Mouillet. Bada: An advanced aircraft performance model for present and future atm systems. *International journal of adaptive control and signal processing*, 24(10):850–866, 2010.

[33] Remy Webservices UG. Avia maps cartas aeronáuticas, 2022. URL `https://play.google.com/store/apps/details?id=com.mytowntonight.aviamap`. 26 Febrero 2022.

[34] Larry Ellison. Database services | oracle españa, 2022. URL `https://www.oracle.com/es/database/`. 20 Julio 2022.

[35] Glassdoor Team. Glassdoor - busqueda de empleo, 2022. URL `https://www.glassdoor.es`. 02 Agosto 2022.

A. Manual de usuario

En el presente anexo se procede a desarrollar un manual de usuario enfocado a aquellas personas que desconozcan el funcionamiento de la app o quieran consultar alguno de los pasos a seguir durante su utilización.

Las figuras que se exponen continuación (Figuras A.1, A.2 y A.3) muestran la interfaz principal con sus elementos numerados, a los cuales se hace referencia a lo largo del manual.

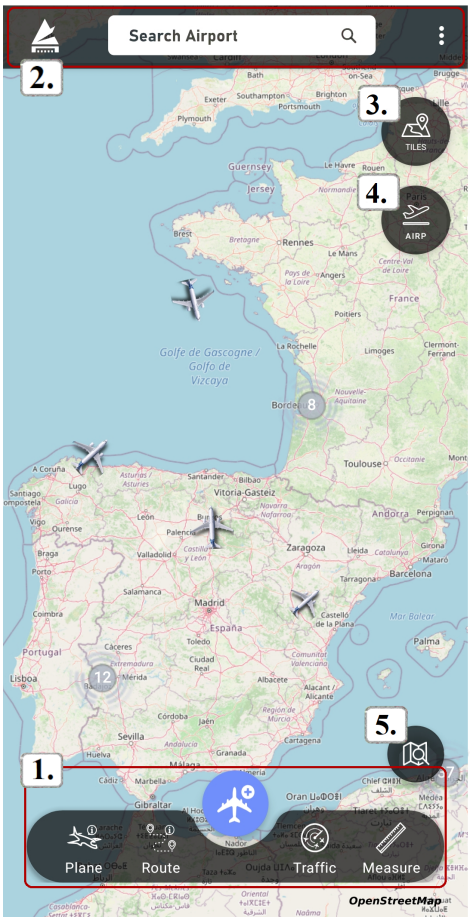


Figura A.1: Interfaz principal de la aplicación

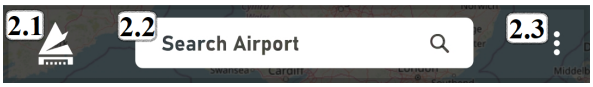


Figura A.2: Barra de navegación superior

Figura A.3: Barra de navegación inferior

Los siguientes puntos explican qué pasos debe seguir el usuario para utilizar la app y hacer uso de todas sus funcionalidades.

CONSULTAR INFORMACIÓN DE AERÓDROMOS

Selección desde listado

- Abrir el listado de aeródromos desde el campo central de la barra de navegación superior (Elemento 2.2 de la Figura A.2).
- Desplazar el listado manualmente o bien filtrar el aeródromo haciendo uso del icono de lupa (🔍) de la esquina superior izquierda. El filtrado de aeródromos se lleva a cabo por código OACI, país o nombre del aeródromo.
- Pulsar en el aeródromo deseado. Se abrirá el diálogo de información de aeródromo y se desplazará el foco a la posición del mismo. Desplazar el listado de información del aeródromo para consultar todos sus parámetros.

Selección desde mapa

- Ampliar el mapa hasta que el icono del aeropuerto deseado sea seleccionable.
- Una vez pulsado el aeródromo se abrirá el diálogo de información de aeródromo y se desplazará el centro del mapa a la posición del mismo. Desplazar el listado de información del aeródromo para consultar todos sus parámetros.

GENERAR NUEVA SIMULACIÓN

- Pulsar en el botón flotante del centro de la barra de navegación inferior (Elemento 1.3 de la Figura A.3). La selección de ruta puede hacerse desde la base de datos de rutas local o bien de forma online.

Selección de ruta local*

- Seleccionar una de las rutas locales disponibles (situándose en la pestaña (LOCAL ROUTE)). En este listado se encuentran las rutas almacenadas por defecto en la app y todas aquellas que se hayan ido generando de forma online.
- Pulsar el botón inferior de continuar.

Selección de ruta online*

- Seleccionar los códigos OACI de los aeródromos origen y destino (situándose en la pestaña (ONLINE ROUTE)). Introducirlos de manera manual o bien seleccionarlos desde el mapa (CHOOSE FROM MAP) o desde el listado (CHOOSE FROM LIST).
- Configurar los 6 parámetros disponibles para el procesamiento de la ruta.
- Pulsar el botón inferior de continuar.

Configuración de los parámetros de la simulación

- En el menú de configuración de la simulación, seleccionar los valores deseados para cada uno de los parámetros. Estos parámetros son:
  1. Altitud deseada para crucero (en pies)
  2. Distancia mínima de la fase de crucero en el caso de que no se pueda alcanzar la altitud deseada (en % de la distancia total de ruta)
  3. Velocidad con respecto a tierra del avión en la fase de crucero (en nudos)
  4. Velocidad con respecto a tierra del avión en la fase de ascenso (en nudos)
  5. Velocidad con respecto a tierra del avión en la fase de descenso (en nudos)
  6. Tasa/velocidad de ascenso (en pies por minuto)
  7. Tasa/velocidad de descenso (en pies por minuto)
  8. Tipo de navegación: ortodrómica o loxodrómica
  9. Periodo de muestreo de la simulación (en segundos)
  10. Tiempo de espera del hilo de simulación (en milisegundos)
- Una vez configurados los parámetros, pulsar en el botón inferior de continuar o bien regresar al menú de selección de ruta.

Revisión de los parámetros de la ruta generada

- En el menú de revisión de ruta, consultar los principales parámetros de la ruta generada. Pulsar en el botón inferior de continuar o bien retornar al menú de configuración de parámetros.

Visualización en el mapa de la ruta generada e inicio de la simulación

– En el mapa que muestra la ruta generada, pulsar en el botón de lanzar avión (LAUNCH PLANE) para iniciar la simulación, o bien retornar a la revisión de los parámetros de ruta pulsando la flecha (←).

SELECCIONAR Y CONSULTAR PARÁMETROS DE AVIÓN SIMULADO

– Seleccionar uno de los aviones simulados.

– Pulsar el icono de información de avión (Elemento 1.1 de la Figura A.3) para consultar los parámetros del avión seleccionado. Estos parámetros varían acorde al tiempo de simulación.

– Pulsar el icono de información de ruta (Elemento 1.2 de la Figura A.3) para consultar los parámetros de la ruta que sigue el avión seleccionado. Estos parámetros son fijos, puesto que la ruta es invariable a lo largo de la simulación.

PAUSAR Y ELIMINAR AVIÓN SELECCIONADO

– Seleccionar uno de los aviones simulados.

– Pulsar el botón morado del lado izquierdo de la pantalla (⏸) para pausar la simulación del avión seleccionado.

– Pulsar el botón verde del lado izquierdo de la pantalla (▶) para reanudar la simulación del avión seleccionado.

– Pulsar el botón rojo del lado izquierdo de la pantalla (■) para eliminar el avión seleccionado.

MONITORIZAR EL TRÁFICO AÉREO

– Pulsar el botón de tráfico situado en la barra de navegación inferior (Elemento 1.4 de la Figura A.3).

– Consultar en el listado los principales parámetros de los aviones cuya simulación se encuentre activa. Pulsar en el listado cualquiera de ellos si se desea seleccionarlo.

MEDIR DISTANCIA ENTRE AVIONES

– Lanzar dos o más simulaciones de aviones.

– Si dos o más simulaciones se encuentran activas, pulsar el botón de medida de la barra de navegación inferior (Elemento 1.5 de la Figura A.3).

– Seleccionar dos aviones, bien desde el mapa o desde el listado de tráfico, para que se represente en el mapa la distancia entre ellos.

CONTROLAR TEXTURAS, VISIBILIDAD DE AERÓDROMOS y ZOOM

– Para cambiar las texturas del mapa, pulsar en el botón flotante superior del lado derecho de la app (Elemento 3 de la Figura A.1: ).

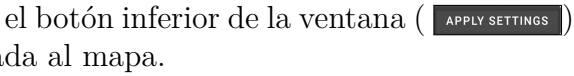
– Para ocultar la capa de aeródromos, pulsar en el botón flotante situado debajo del botón de texturas (Elemento 4 de la Figura A.1: ).

– Para manipular el zoom del mapa, utilizar dos dedos o bien usar los controles de zoom (Elemento 5 de la Figura A.1:).

CONFIGURAR LOS AJUSTES DEL MAPA

– Pulsar en el botón situado en la esquina superior derecha, dentro de la barra de navegación superior (Elemento 2.3 de la Figura A.2). Los parámetros a configurar son:

1. Radio de *clusterización* de aeródromos (en píxeles)
2. Longitud mínima de la pista principal de aeródromo para que este se represente en el mapa (en pies)
3. Icono de representación de los aviones simulados
4. Color del icono de representación de los aeródromos del mapa

– Pulsar en el botón inferior de la ventana (

) para aplicar la configuración seleccionada al mapa.

B. Presupuesto

En el presente apéndice se incluye el presupuesto correspondiente al desarrollo de la aplicación RouteFlyerDroid, un simulador de vuelos 2D sobre rutas áreas reales. El desglose que se procede a exponer contiene tanto la mano de obra necesaria para su desarrollo como el hardware y software utilizado durante los 7 meses de realización del proyecto. Finalmente, se aplicará el 21% de IVA al presupuesto resultante.

B.1. Mano de obra

El presupuesto relativo a la mano de obra se ha calculado de acuerdo con los salarios medios de un desarrollador Android junior sin experiencia y un ingeniero doctor en informática. Si bien es cierto que la aplicación se ha llevado a cabo por un ingeniero técnico aeronáutico, se considera más correcto que este figure en el presupuesto como desarrollador, debido a que las labores llevadas a cabo pertenecen a esta rama.

La contabilización de las horas invertidas por el desarrollador Android se ha realizado de acuerdo a los 13.5 créditos ECTS que le corresponden al Trabajo de Fin de Máster en Ingeniería Aeronáutica. Estos créditos equivalen a un total de 337,5 horas. Se han omitido incluir las horas invertidas en el aprendizaje de las herramientas de desarrollo de aplicaciones móviles, puesto que se asume que previo al inicio del proyecto el desarrollador debería estar instruido en este campo.

Por su parte, dentro de las horas contabilizadas para el doctor ingeniero informático se han tenido en cuenta las reuniones de seguimiento del trabajo y la resolución de dudas de forma asíncrona. Tanto su salario medio aproximado como el del desarrollador Android junior sin experiencia se han extraído de la web de empleo Glassdoor [35]. El total contabilizado se expone en la Tabla B.1.

	Horas (h)	Sueldo (€/h)	Total (€)
Desarrollador Android sin experiencia	337,50	11,00	3712,50
Ingeniero doctor en informática	30,00	35,00	1050,00
Total mano de obra			**4762,50**

Tabla B.1: Presupuesto correspondiente a la mano de obra

B.2. Hardware

En lo relativo al equipo informático empleado, este incluye tanto al ordenador con el que se ha desarrollado el código como al dispositivo móvil personal con el que se han realizado test de la app a lo largo del desarrollo de la misma. El primero de ello es un Acer Aspire A315-54, con un procesador Intel Core i5-10210U de cuatro núcleos a 1.60 GHz, almacenamiento de 256 GB y 8 GB de memoria RAM. Por su parte, el dispositivo móvil es un smartphone Xiaomi Redmi Note 8 con sistema operativo Android 9, 64 GB de almacenamiento interno y 4 GB de memoria RAM. Teniendo en cuenta que ambos dispositivos no fueron utilizados exclusivamente para el desarrollo de este proyecto, se ha tenido en cuenta en este presupuesto su amortización a lo largo de los 7 meses de desarrollo. Para su cálculo se ha aplicado un coste de amortización lineal de acuerdo con el artículo 12.1.a de la Ley de Impuesto de Sociedades para elementos informáticos. Asimismo, se ha utilizado un coeficiente de amortización del 33%, basado en la *Tabla de coeficientes de amortización lineal* que ofrece la Agencia Tributaria. La Tabla B.2 recoge el presupuesto total relativo al uso de hardware.

	Precio inicial (€)	Tiempo de uso (meses)	Total (€)
Acer Aspire A315-54	619,00	7	119,16
Xiaomi Redmi Note 8	179,00	7	34,46
Total hardware			**153,62**

Tabla B.2: Presupuesto correspondiente al uso del hardware

B.3. Software

El software utilizado para el desarrollo tanto de la aplicación como de la presente memoria ha sido: Microsoft Office, LaTeX, Android Studio, GIMP y GenyMotion. A este listado hay que sumar también la licencia de Windows 10 Home, SO con el que se ha trabajado durante los 7 meses. Dentro de todo el software, únicamente precisan de licencia Microsoft Office Y Windows 10 Home, las cuales se adquieren con un único pago inicial. Para calcular su amortización durante el desarrollo del proyecto se ha

hecho uso nuevamente de un porcentaje de amortización del 33%, basado en la *Tabla de coeficientes de amortización lineal* que ofrece la Agencia Tributaria. Por su parte, LaTeX, Android Studio, GIMP y GenyMotion son programas de licencia libre, por lo que no han supuesto costes adicionales para el proyecto. Los resultados relativos al software se exponen en la Tabla B.3.

	Coste inicial (€)	**Tiempo de uso (meses)**	**Total (€)**
Windows 10 Home	145,00	7	27,91
MicrosoftOffice	149,00	7	28,68
Latex	0,00	7	0,00
Android Studio	0,00	7	0,00
GIMP	0,00	7	0,00
GenyMotion	0,00	7	0,00
	Total software		**56,60**

Tabla B.3: Presupuesto correspondiente a las licencias del software

B.4. Presupuesto final

Finalmente, sumando los cargos correspondientes a mano de obra, hardware y software, y aplicando el correspondiente 21% de IVA, se acaba obteniendo un presupuesto total para el proyecto de 6016,98 € (Tabla B.4).

Mano de obra	4762,50 €
Hardware	153,62 €
Software	56,60 €
21% de IVA	1044,27 €
TOTAL	**6016,98 €**

Tabla B.4: Presupuesto total del proyecto

C. Pliego de condiciones

Real Decreto 488/1997, 14 de abril

En este apéndice se proceden a presentar las condiciones para los trabajadores en el empleo de equipos con pantallas de visualización, las cuales se establecen en el Real Decreto 488/1997, del 14 de abril. Estas condiciones hacen referencia tanto a los equipos y el entorno de trabajo como a la interconexión del ordenador y la persona.

C.1. Equipo

Como condición general para todo tipo de equipos, la utilización de los mismos no debe ser una fuente de riesgo para los trabajadores.

En cuanto a la pantalla, esta debe ofrecer una imagen estable, sin fenómenos de destellos, reflejos u otras formas de inestabilidad que puedan molestar al usuario. Asimismo, los caracteres de la pantalla deben estar bien definidos, poseer una dimensión suficiente y estar configurados de forma clara. Adicionalmente, el usuario debe tener la capacidad de ajustar fácilmente la luminosidad y el contraste de la pantalla, adaptándolos a las condiciones del entorno. Por último, se debe trabajar con pantallas orientables e inclinables a voluntad, con el fin de adaptarse a las necesidades del usuario.

En lo relativo al teclado, este debe ser inclinable e independiente de la pantalla, con el fin de permitir al usuario adoptar una postura cómoda que no genere cansancio en brazos o manos. Además, debe haber suficiente espacio delante del teclado para que el usuario pueda apoyar tanto brazos como manos. Por último, se deberá trabajar con un teclado en el que los símbolos de las teclas resalten suficientemente y sean perfectamente legibles desde la posición habitual de trabajo.

Por su parte, la mesa de trabajo debe tener dimensiones suficientes para permitir una colocación flexible de los elementos de trabajo: pantalla/s, teclado, documentos u otro material necesario.

Por último, el asiento de trabajo debe ser estable, proporcionando al usuario libertad de movimiento y procurándole una postura cómoda. Este asiento deberá ser regulable

en altura, además de tener respaldo reclinable que permita ajustar su altura de manera independiente al asiento.

C.2. Entorno

El espacio del puesto de trabajo debe ser suficientemente amplio como para garantizar los cambios de postura y movimientos de trabajo. Además, debe presentar una iluminación que garantice unos niveles adecuados de luz y unas relaciones adecuadas de luminancia entre pantalla y entorno. Se deberán evitar los deslumbramientos u otros reflejos molestos en la pantalla, prestando para ello especial atención a la ubicación del puesto de trabajo dentro del entorno. Se deberá evitar que las fuentes de luz se encuentren en posiciones que provoquen deslumbramiento directo o que produzcan reflejos.

Por otro lado, se deberá tener en cuenta que el ruido que se perciba en el puesto de trabajo no perturbe la atención ni la palabra del trabajador. Esto mismo se aplica en el caso del calor, que se deberá mantener por debajo de los límites que perturben el desempeño del trabajador, así como la humedad del entorno.

Por último, se deberá reducir toda posible radiación nociva a niveles insignificantes desde el punto de vista de la protección de la seguridad y la salud de los trabajadores.

C.3. Interconexión ordenador/persona

Tanto para la elección o modificación de los programas a utilizar, así como de las tareas a realizar por los trabajadores, se deberán tener en cuenta ciertos factores relativos a su utilización. Concretamente, los programas deberán ser fáciles de usar y deberán de estar adaptados tanto a los conocimientos y experiencia del usuario como a la tarea a realizar. Asimismo, los principios de ergonomía deberán aplicarse al tratamiento de la información por parte de la persona.