Universitat Autònoma de Barcelona

# *App* de Viajes

**UFly**

*(multiplataforma)*

Memoria del Trabajo Fin de Grado

Gestión Aeronáutica

realizado por

Hamet Lamin Jalil Beida

Roger Villalba i Fuentes

dirigidos por

Ernesto Emmanuel Santana

**Escola d'Enginyeria**

Sabadell, *02 Juliol de 2020*

El sotasignat, ***Ernesto Emmanuel Santana***

Professor de l'Escola d'Enginyeria de la UAB,

CERTIFICA:

Que el treball què correspon aquesta memòria ha estat realitzat sota la seva direcció per en ***Hamet Lamin Jalil Beida*** i en ***Roger Villalba Fuentes.***

I per tal que consti firma la present.

SANTANA CRUZ ERNESTO EMMANUEL - Y0145546Q

Firmado digitalmente por SANTANA CRUZ ERNESTO EMMANUEL - Y0145546Q
DN: cn=SANTANA CRUZ ERNESTO EMMANUEL - Y0145546Q, c=ES
Fecha: 2020.07.02 17:52:54 +01'00'

**Signat:**

Sabadell, *02 de Juliol de 2020*

**HOJA DE RESUMEN** **TRABAJO FINAL DE GRADO DE LA ESCUELA DE INGENIERÍA**

| | |
|---|---|
| *App* de viatges *UFly (multiplataforma)*<br>*App* de viajes *UFly (multiplataforma)*<br>Travel *App UFly (multiplatform)* | |
| **Autores:** Hamet Lamin Jalil Beida y Roger Villalba Fuentes | **Fecha:** *Julio 2020* |
| **Tutor:** Ernesto Emmanuel Santana Cruz | |
| **Titulación:** Grado en Gestión Aeronáutica | |


**Palabras Clave**

- Català: Aplicació, *Flutter,* estacionalitat, multiplataforma, programari
- Castellà: Aplicación, *Flutter,* estacionalidad, multiplataforma, software
- Anglès: App, *Flutter,* seasonality, multiplatform, software



**Resumen del Treball Fin de Grado**

- Català:

  Les aplicacions mòbils suposen una gran eina per satisfer necessitats que beneficien als seus usuaris. Davant d'una situació on les companyies aèries son capaces d'oferir als seus clients ofertes personalitzades per tal de maximitzar el seu benefici, gràcies a eines com les cookies, es planteja el desenvolupament d'una aplicació multiplataforma, on siguin els propis usuaris, els que estableix el seu pressupost per tal d'aprofitar la estacionalitat que experimenten els preus dels bitllets d'avió.

- Castellà:

  Las aplicaciones móviles suponen una gran herramienta para satisfacer necesidades que benefician a sus usuarios. Ante una situación donde las compañías aéreas son capaces de ofrecer a sus clientes ofertas personalizadas a fin de maximizar su beneficio, gracias a herramientas como las cookies, se plantea el desarrollo de una aplicación multiplataforma, donde sean los propios usuarios, los que establezcan su presupuesto para aprovechar la estacionalidad que experimentan los precios de los billetes de avión.

- Anglès:

  Mobile applications are increasingly present in our day to day. They are a great tool to meet needs that benefit its users. Faced with a situation where airlines are able to offer their customers personalized offers in order to maximize their profit, thanks to tools like cookies. The development of a cross-platform application, where the users are those that set their budget in order to take advantage of the seasonality experienced by airline ticket prices.

# Resumen

En la actualidad, existe un sector que destaca por encima de los demás. Este es sin duda el sector tecnológico, pues ha revolucionado todos los áreas de negocios, servicios, medicina, logística, comercio, turismo, banca, etc. También forma parte de la vida de las personas, generando una gran dependencia en nuestra vida cotidiana, las nuevas generaciones nacen con un móvil, una tablet o una consola en la mano y todo esto nos lleva a una sociedad cada vez más dependiente de la tecnología.

En el sector tecnológico, existen diferentes nichos, áreas o especializaciones, los más destacados hoy en día y que han creado un mercado estable de billones de dólares en la última década, son: el mercado de las *Apps*, *Software* de gestión empresarial (ERPs), inteligencia artificial (AI) y el ecommerce.

Nosotros, Roger y Hamed, hemos realizado este TFG para así introducirnos en este maravilloso mundo tecnológico, mediante la creación de una *App* multiplataforma *(iOS* y *Android)*. Una que tuviera relación directa con el sector aeronáutico, aportando así valor a dicho sector.

En este documento vamos a explicar y detallar de la mejor manera posible, todo el proceso necesario para la creación de una *App*. Con el fin de contextualizar y resumir brevemente nuestra *App* podríamos decir que se trata de una aplicación gratuïta, dirigida a cualquier persona que viaje en avión, facilitando información sobre los vuelos en tiempo real, información de ciudades, aeropuertos y con una propuesta de valor totalmente innovadora.

# Abstract

Currently, there is a sector that stands out above the rest. This is undoubtedly the technological sector, as it has revolutionized all areas of business and services, like medicine, logistics, commerce, tourism, banking, etc. It is also part of people's life, generating a great dependence on our daily lives, the new generations are born with a mobile phone, a tablet or a console in their hands and all this leads us to a society increasingly dependent on technology .

In the technology sector, there are different niches, areas or specializations, the most prominent and which have created a stable market of billions of dollars in the last decade, are: the *Apps* market, Business management software (ERPs), artificial intelligence (AI) and ecommerce.

We, Roger and Hamed, have carried out this TFG, in order to get introduced to this wonderful technological world. Creating an *App* from scratch, but not any *App*, one that had a direct relationship with the aeronautical sector, being able to add value to that sector .

In this document we will explain and detail in the best possible way, the entire process of creating a multiplatform App, that is, for both Android and iOS, the two main operating systems for mobile devices. In order to contextualize and briefly summarize our *App*, we could say that it is a free application, addressed to anyone traveling by plane, providing information of flights in real time, information of cities, airports and with a totally innovative value proposition.

# Agradecimientos

Antes de empezar con este trabajo queremos hacer una mención especial a aquellas personas, recursos y herramientas sin las cuales no habría sido posible la construcción de un proyecto de esta magnitud. Por ello se merecen aparecer en este trabajo.

Dicho esto, agradecer a **Fernando Herrera**, profesor de *Udemy* cuyos cursos:

- ***Flutter:*** *Guía completa de desarrollo para iOS*
- ***Android Flutter:*** *Diseños profesionales y animaciones*

Estos nos ayudaron a comprender y familiarizarnos con un lenguaje de programación, cuya aparición en el mercado del desarrollo de aplicaciones es muy reciente *(Dart)*. Por otro lado también debemos agradecer a **Andrea Bizzotto**, otro mentor que hemos tenido durante el desarrollo del proyecto, también a través de la misma plataforma, con su curso:

- ***Flutter & Firebase:*** *Build a Complete App for iOS & Android*.

Estas son las dos figuras principales que nos han ayudado de forma pasiva, puesto que sus cursos eran de pago a través de videos prácticos y explicativos.

Así mismo también debemos mencionar la gran cantidad de creadores de contenido que se encuentran en la plataforma de **Youtube** y que hemos tenido acceso sin coste alguno, así como también información suministrada por los proveedores de servicios como ***Google***, ***Firebase***, ***Node***, ***Flutter***, y todas las guías de las distintas APIs, las aplicadas en el proyecto y las que lamentablemente dejamos atrás puesto que no satisfacen las necesidades de nuestro proyecto.

Finalmente agradecer a la comunidad de ***GitHub*** y ***StackFlow***, pues nos han permitido amenizar la agonizante tarea de lidiar con errores, algo que inevitablemente debe afrontar cualquier programador.

De todo corazón gracias por permitirnos ir más allá.

*Hamet Lamin Jalil Beida*
*Roger Villalba Fuentes*

# Índice General

Resumen ..... 4
Agradecimientos ..... 6
Índice de Figuras ..... 9
Índice de Tabla ..... 11
Glosario ..... 12

**1. Introducción ..... 13**
1.1. Objetivos ..... 13
1.2. Propuesta de valor ..... 14
1.3. Motivación ..... 15
1.4. Alcance ..... 16
**2. Análisis del Mercado ..... 18**
2.1. Análisis sector Aeronáutico ..... 18
2.2. Análisis de las Apps ..... 21
**3. Estructura del proyecto ..... 29**
**4. Marco Conceptual ..... 33**
4.1. APIs ..... 34
4.2. Lenguajes ..... 35
4.3. Herramientas y recursos ..... 38
4.3.1. Flutter ..... 38
4.3.2. Google Cloud Platform ..... 45
4.3.2.1. Cloud Functions ..... 45
4.3.2.2. Cloud Storage ..... 47
4.3.2.3. Firebase ..... 47
4.3.3. Visual Studio Code ..... 51
4.3.4. Atom ..... 53
**5. Metodología ..... 54**
5.1. APIs empleadas ..... 55
5.2. Lenguajes empleados ..... 57
5.3. Modelado de Datos ..... 59
5.4. Planificación temporal ..... 61
**6. Desarrollo ..... 65**
6.1. Front-end ..... 66

6.1.1. Arquitectura ..... 66
6.1.2. UI ..... 69
6.1.3. UX ..... 70
6.1.4. Gestión del Estado ..... 71
6.1.5. Arbol de *widgets* ..... 74
6.1.6. Packages ..... 74
6.2. Back-end ..... 76
6.2.1. Arquitectura BD ..... 76
6.2.1.1. Collections & Documents ..... 77
6.2.2. Funciones *Cloud* ..... 79
6.2.2.1. Funciones *User* ..... 79
6.2.2.2. Sistema de notificaciones ..... 82
**7. Uso de la aplicación ..... 87**
**8. Puesta en producción ..... 99**
8.1. Testing ..... 99
**9. Futuras implementaciones ..... 103**
9.1. Posibilidades de negocio ..... 105
**10. Conclusiones ..... 107**
**11. Referéncias ..... 108**
11.1. Artículos y páginas web ..... 108
11.2. Videos y tutoriales ..... 110
**12. Anexos ..... 118**
12.1. Respuestas JSON ..... 119
12.2. Ejemplos de Código ..... 129
12.3. Firestore ..... 140
12.4. Cloud Functions ..... 144
12.5. Modelos Dart ..... 183
12.6. Scripts ..... 192

**Índice de Figuras**


**Figura 1.** Aplicación *Aena* ..... 24
**Figura 2.** Aplicación *Expedia* ..... 25
**Figura 3.** Aplicación *eDreams* ..... 25
**Figura 4.** Aplicación *Kayak* ..... 26
**Figura 5.** Kayak *departures* ..... 26
**Figura 6.** Calendario búsqueda *Hopper* ..... 26
**Figura 7.** Notificaciones *Hopper* ..... 27
**Figura 8.** Notificaciones *UFly* ..... 27
**Figura 9.** Comparativa de *Apps* ..... 28
**Figura 10.** Esquema de gestión interna del equipo ..... 30
**Figura 11.** Pilares investigación, desarrollo y testeo ..... 31
**Figura 12.** API *information exchange (esquema)* ..... 34
**Figura 13.** Ejemplo manejo de estado ..... 40
**Figura 14.** Ejemplo de los dos tipos de estado ..... 41
**Figura 15.** Ejemplo de flujos con el patrón BLOC ..... 43
**Figura 16.** Recursos Flutter *template* ..... 44
**Figura 17.** UI Flutter *template* ..... 44
**Figura 18.** GCP Tools ..... 45
**Figura 19.** Las posibilidades que ofrece Firebase ..... 48
**Figura 20.** Firestore, estructura almacenamiento ..... 50
**Figura 21.** Captura del editor de código VSC ..... 51
**Figura 22.** Captura del editor de código ATOM ..... 53
**Figura 23.** Diagrama Global Gantt ..... 62
**Figura 24.** Diagrama Gantt con distribución cargas de trabajo ..... 63
**Figura 25.** UFly - *FlightSearch* ..... 70
**Figura 26.** Evolución gestión estado *UFly* ..... 71
**Figura 27.** Arquitectura general *UFly* ..... 76
**Figura 28.** Parámetros creación Tópico ..... 83
**Figura 29.** *Topics* creados ..... 85
**Figura 30.** UFly - *RegisterPage* ..... 87
**Figura 31.** UFly - *HomePage* ..... 88
**Figura 32.** UFly - *FlightSearch, Cities* y *Airports* ..... 89
**Figura 33.** UFly - *FlightSearch* - resultados ..... 89
**Figura 34.** UFly - *Scroll CityDetalle* ..... 90

**Figura 35.** UFly - *CityDetalle, versión móvil* ..... 90
**Figura 36.** UFly - *AirportDetalle (FlightDetalle* y *AirportInfo)* ..... 91
**Figura 37.** UFly - *SearcUsers*, *Search Results* y *PeopleProfile (3 estados)* ..... 92
**Figura 38.** UFly - *Topics* ..... 94
**Figura 39.** UFly - *Notifications Push* ..... 94
**Figura 40.** *Create Topic* ..... 94
**Figura 41.** Borrar suscripción ..... 94
**Figura 42.** Compra billete ..... 94
**Figura 43.** Rutas hacia la venta de billetes ..... 95
**Figura 44.** Notificación onMessage ..... 95
**Figura 45.** Notificación onLaunch y onResume ..... 96
**Figura 46.** UFly - *ProfilePage* y *Icons Navigation Routes* ..... 96
**Figura 47.** UFly - *ProfilePage* ..... 97
**Figura 48.** UFly -*Settings*, *EditProfile* y *CreditCard* ..... 98
**Figura 49.** UFly - *PendentToAcceptPage* ..... 98
**Figura 50.** Errores - no *responsive UI* ..... 101
**Figura 51.** UFly - *Hotels* y *Taxis* ..... 103
**Figura 52.** UFly - Mejorar sistema seguimiento ..... 104

**Indice de Tablas**


**Tabla 1.** Visión global Sector Aéreo Español *(2018)* ..... 19
**Tabla 2.** Evolución tráfico aéreo español *(Enero - Octubre 2019)* ..... 20
**Tabla 3.** Descargas de *Apps* del sector ..... 27
**Tabla 4.** Elementos de un lenguajes ..... 35
**Tabla 5.** Lenguajes de programación más utilizados ..... 36
**Tabla 6.** Resumen *Widgets* y ejemplos ..... 39
**Tabla 7.** *Cloud Functions*, tipos y uso ..... 46
**Tabla 8.** Resumen APIs empleadas ..... 55
**Tabla 9.** Resumen CF empleadas ..... 56
**Tabla 10.** Lenguajes utilizados e IDEs ..... 58
**Tabla 11.** Modelado de datos Firestore *UFly* ..... 59
**Tabla 12.** Visión general del desarrollo por bloques ..... 65
**Tabla 13.** Carpetas y archivos creados para *UFly* ..... 67
**Tabla 14.** Consideraciones de *UI* ..... 69
**Tabla 15.** Conceptos Provider ..... 72
**Tabla 16.** Técnicas de estado empleada ..... 73
**Tabla 17.** Paquetes empleados en *UFly* ..... 74
**Tabla 18.** Esquema Colecciones *UFly* ..... 77
**Tabla 19.** Esquema Documentos *UFly* ..... 78
**Tabla 20.** Agrupación CF *UFly* ..... 79
**Tabla 21.** CF sistema de gestión usuarios ..... 80
**Tabla 22.** CF Sistema de Notificaciones ..... 84
**Tabla 23.** Recepción *push notifications* ..... 85
**Tabla 24.** Estados *PeopleProfile* ..... 92
**Tabla 25.** Apartados de las notificaciones ..... 93
**Tabla 26.** Ventajas y desventajas entre VD y FD ..... 100
**Tabla 27.** Estrategias para monetizar una *App - UFly* ..... 105

**Glosario**

| | |
|---|---|
| ORG | Origen |
| DEST | Destino |
| ETD | Estimated Time Departure |
| ETA | Estimated Time Arrival |
| ATD | Actual Time Departure |
| ATA | Actual Time Arrival |
| BD | Base de Datos |
| SQL | Structured Query Language |
| NoSQL | Not Only SQL |
| CF | Cloud Functions |
| TS | Typescript |
| JS | Javascript |
| API | Application Programming Interface |
| GCP | Google Cloud Platform |
| UI | User Interface |
| UX | User Experience |
| SaaS | Software As A Service |
| Async | Asynchronous |
| SDK | Software Developer Kit |
| AS | AviationStack |
| TFG | Trabajo Fin de Grado |
| POO | Programación Orientada a Objetos |
| VSC | Visual Studio Code |
| cmd | Comando *(ejecutado por consola)* |
| IDE | Integrated Development Environment |
| ERP | Enterprise Resource Planning |
| AI | Artificial Intelligence |
| MIT | Massachusetts Institute of Technology |
| uid | User Identifier |
| CA | Comunidad Autónoma |
| IATA | International Air Transport Association |
| OACI | Organización de Aviación Civil Internacional |
| DDoS | Distributed Denial of Service |
| OS | Operative System |

# 1. Introducción

## 1.1. Objetivos

Nuestro principal objetivo es poder ofrecer una solución práctica y tangible confeccionando una aplicación multiplataforma para facilitar información sobre *salidas, llegadas, billetes de vuelo, ciudades*, aeropuertos, etc. Ofreciendo a nuestros usuarios la posibilidad de adquirir billetes de vuelo que atiendan única y exclusivamente a sus necesidades, a través de un sistema de notificaciones.

Para poder cumplir con este objetivo, hemos creído conveniente establecer todo un seguido de objetivos secundarios que nos ayuden a lograrlo.

**Los objetivos secundarios:**

- Facilitar el acceso a información aeroportuaria.
- Crear un servicio propio,  útil y que aporte valor al usuario final.
- Comprender el alcance y las dificultades del mundo de las *Apps*.
- Aprender nuevas herramientas para el diseño y creación de *Apps*.
- Comprender y estudiar los distintos proveedores de almacenaje de datos en la nube, *Cloud*.
- Planificar cada uno de los pasos para la confección de una aplicación..
- Mejorar nuestras habilidades para el desarrollo *front-end* y *back-end*.
- Detallar el uso del producto/servicio creado.
- Realizar un análisis de ***UFly*** y sus futuras implementaciones.
- Investigar distintas formas que permitan monetizar una aplicación.

## 1.2. Propuesta de valor

Con el desarrollo de este TFG pretendemos crear una aplicación que permita a los los usuarios del sector aeronáutico no estar constantemente pendientes del fluctuante mundo de los precios de los billetes de avión.

Cómo sabemos, el sector aeronáutico, al ser **estacional** experimenta notables variaciones en el precio de los billetes de vuelo. De esta forma las compañías aéreas logran ofrecer un precio u otro en función de la demanda que presenta el mercado. A su vez, estas también hacen uso tecnologías que todos conocemos bajo el nombre de *cookies,* que les permiten saber cuántas veces y en qué lugares has consultado precios, para así maximizar su beneficio, aquí es dónde **UFly** expone su mayor atractivo, las notificaciones.

Nuestros usuarios disponen de un apartado en el menú de la *App* para crear de forma fácil una alerta que les permita ser notificados cuando el precio de un billete para una ruta determinada en una fecha o intervalo se encuentre por debajo del precio marcado por el usuario, facilitando así la adquisición del billete que se desea comprar.

Para crear una alerta introducen una ciudad de origen y una de destino, una fecha concreta o un intervalo y el precio máximo que están dispuestos a pagar. Toda esta información se almacena en la base de datos y se muestra en el apartado **Topics**. La *App* busca constantemente los parámetros de cada tópico y si encuentra un vuelo que cumpla con las condiciones impuestas por el usuario, se le notifica inmediatamente a través de una notificación ***push***, que le lleva directamente a la página de compra del billete.

Para concluir la propuesta de valor, nos gustaría hacer referencia a una cuestión que surgió durante la lluvia de ideas:

*"¿Cuánto tiempo más aguantaremos los clientes, cediendo nuestro gran poder como consumidores a las empresas?"*

Esta frase evidencía que desde hace mucho tiempo el consumidor ha tenido que adaptarse a las empresas y a sus ofertas. Con nuestra *App* ha llegado el momento de romper con esta tendencia, y dejar que sean los propios usuarios los que establezcan las normas del juego.

## 1.3. Motivación

Este proyecto nace de la necesidad de construir una herramienta que aporte valor al sector y a la sociedad, con el conocimiento adquirido a lo largo del grado en Gestión Aeronáutica. Otro elemento motivador para la confección de este proyecto era la posibilidad de aprender y emplear las herramientas de desarrollo de *Software* más potentes de la actualidad e inevitablemente del futuro, pues nos dirigimos a una sociedad donde el recurso más importante son los datos, su comprensión y su uso.

Como detallamos en el análisis del sector *(punto 2.1)*, hoy en día las compañías aéreas son capaces de crear ofertas personalizadas, por ello queremos que las personas viajen a donde quieran, cuando quieran y al precio que deseen. Esta premisa nos motivó a crear un producto/servicio que ofreciera dicha posibilidad, aportando al sector aeronáutico una solución a través de la tecnología.

Desde los inicios del grado, hemos creado un buen equipo de trabajo, no tan solo eso sino que a ambos integrantes nos apasiona la programación y todas las posibilidades que dicha tecnología nos puede brindar.

Al plantear un TFG de estas características, decidimos tomarnoslo como un reto y demostrar si éramos capaces de crear una *App* multiplataforma, totalmente desde cero, siendo esta funcional y vinculada con el transporte aéreo de pasajeros

## 1.4. Alcance

El desarrollo de esta aplicación móvil abarca todo un conjunto de necesidades, más allá de las intrínsecamente funcionales. Puesto que hoy en día el valor de una aplicación no se mide tan sólo a través de su funcionalidad, también se tiene en cuenta el diseño, experiencia de usuario y otros factores que analizaremos en este proyecto.

El alcance abarcado por este TFG contempla los siguientes aspectos:

**Funcionalidad**

- Mostrar en relación de un ID de vuelo, ORG, DEST, ETD, ETA, ATD, ATA, terminales *(origen y destino)*, puertas de embarque y cinta de recogida de equipajes.
- Posibilitar el almacenaje de la información anterior respecto a un vuelo.
- Mostrar para un aeropuerto, las salidas y llegadas, pudiendo acceder a la información del ID de vuelo.
- Facilitar la búsqueda de aeropuertos *(código IATA/nombre de la ciudad)*.
- Mostrar información relativa al aeropuerto *(País, ciudad, CA, código IATA, código OACI, tlf. contacto, código postal, página web, mapa circundante)*.
- Posibilitar la búsqueda de billetes de vuelo en función de ORG, DEST y fecha deseada.
- Mostrar información relevante sobre diferentes aspectos de las ciudades, permitiendo obtener una visión global de estas.
- Posibilitar el almacenaje de la información anterior respecto a una ciudad.
- Introducir aspectos de *Social Network:*
    - *Búsqueda de usuarios y visionar su perfil*
    - *Gestión del perfil y solicitudes de seguimiento*
- Crear alertas para notificar al usuario, cuando el mercado alcance el precio deseado para una ruta (*ORG-DEST-PRECIO-RANGO DE FECHAS*).
- Dotar a la aplicación de un sistema consistente de Notificaciones para avisar del hallazgo del punto anterior.

**Diseño**

- Confeccionar UIs coherentes *(temática uniforme)*.
- Crear un diseño *responsive (adaptable al dispositivo en que se use)*.
- Brindar una experiencia al usuario agradable y limpia.

No podemos obviar un aspecto que por desgracia ha limitado el desarrollo de este proyecto, y es el hecho de que nos encontramos ante un sector paralizado en gran medida por lo que al transporte de personas concierne. Puesto que estamos viviendo un período de crisis sanitaria que por desgracia afecta a todo el mundo.

El **covid-19** ha paralizado todos los medios de transporte, afectando especialmente al sector aeronáutico, debido a su papel como pieza clave de la globalización. Partimos de la base que la situación actual es algo anormal e imprevista, de forma que cuando iniciamos el desarrollo del proyecto esta situación no representaba un aspecto a considerar.

El factor pandemia por desgracia ha influenciado al desarrollo del mismo de distintas formas. No sólo porque nuestra muestra de testeo se ha visto reducida de forma dramática, sino también porque muchos de los proveedores de información del sector se han visto obligados a cerrar. Esto nos ha obligado a adaptarnos de forma rápida y ágil a cambios en las APIs.

En cuestión de días, los servidores de 2 APIs *(providers)* cayeron y dejaron de responder a nuestras peticiones *(requests)*. Como veremos más adelante en el apartado de APIs, estas constituyen un aspecto fundamental en torno al cual nosotros construimos la aplicación. Aunque en favor de la verdad, esta problemática nos obligó a crear un *Back-end* mucho más consistente y con contemplación de errores, a su vez la arquitectura *Cloud* nos permite implementar cambios en el código sin necesidad de que la aplicación sufra modificaciones y por tanto los usuarios no se vean obligados a realiza la descarga de actualizaciones innecesarias, derivadas de errores estructurales en el código.

Por último, gracias a la implementación de esta nueva estructura disponemos de una capacidad de adaptación rápida ante cambios o pérdidas de proveedores de información.

# 2. Análisis del Mercado

## 2.1. Análisis del Sector Aeronáutico

El sector aeronáutico es un mercado global, por lo que puede verse afectado por una gran cantidad de variables como por ejemplo el país en el que lo analizemos, la época *(estación)* en la que se centre, el precio del petróleo, la disponibilidad y variedad de compañías aéreas que existan en la región, etc.

Como podemos observar no es un mercado fácil de analizar de manera específica, aunque existen un conjunto de rasgos que son comunes como por ejemplo los fabricantes de aeronaves *(Airbus, Boeing)* no son los únicos, aunque sí que son los dos gigantes del sector.

Otro rasgo común es la estacionalidad a la que se ven sometidos los precios de los billetes de avión. Por eso, para poder entender qué es y qué implica esta estacionalidad debemos primero entender el concepto..

La estacionalidad hace referencia a las fluctuaciones o cambios de ciertas variables de forma regular en el tiempo y que pueden predecirse, pues se repiten de forma periódica. En nuestro caso este fenómeno se ve claramente reflejado en las variaciones de precio que experimentan los billetes de avión.

Esto se debe a varios factores, aunque el principal es la variación de la demanda. En general, la gente acostumbra a viajar en verano, por ello los billetes suelen ser más baratos en invierno. Aunque lo cierto es que hoy en día este fenómeno tan evidente hace unos años, está cambiando. Pues ahora las compañías disponen de una herramienta que les permite conocer los intereses de sus clientes antes tan siquiera de que estos consulten sus webs. Esto se debe a las *cookies*.

Las *cookies* son archivos creados por los sitios web dónde se almacenan ciertos datos que se transmiten entre el emisor y el receptor. Por ejemplo si hacemos uso de *Internet* el rol de emisor corresponde al servidor dónde está almacenada la página web, suministrando normalmente tres tipos de archivos *(.html, .js, .css).* El rol de receptor corresponde al navegador que se esté empleando en el momento de consulta de la web *(p.e. Google Chrome, Opera, Firefox,...).*

Gracias a esta herramienta se crea un perfil sobre el usuario en cuestión, en el cual se almacenan los trayectos en los que ha mostrado interés, de forma que cuando accedes a diferentes páginas para consultar precios para tu viaje, estos ya disponen de tus intereses pudiendo mostrar el precio que crean conveniente basado en los precios que ya has visto.

Estas *cookies* recolectan las "huellas digitales" que dejas en tu computadora o teléfono inteligente, cada vez que realizas una búsqueda pudiendo así de esta manera rastrear tu actividad e intereses.

Nos encontramos entonces ante un mercado no tan sólo estacional sino dónde las compañías disponen de la tecnología y recursos suficientes como para crear una oferta personalizada, permitiéndoles maximizar el margen de cada una de sus ventas. Lo que deja inevitablemente al usuario del transporte aéreo a merced de estas compañías para poder realizar sus viajes.

Es aquí donde nuestro proyecto pretende hacer hincapié, permitiendo al usuario dejar de estar a merced de estas compañías y ofreciéndoles la posibilidad de confeccionar ofertas, basadas en su presupuesto y no delegando la decisión a un algoritmo que aprovecha información desconocida para el usuario aeronáutico.

Para poder obtener una imagen global del sector aeronáutico hemos confeccionado la siguiente tabla, basada en cifras del estado español *(ejerc. 2018)*:

| Aspecto | Valor | Fuente |
|---|---|---|
| **Facturación** | 11.838.000.000 € *(2018)*<br>*67 % export / 33% import* | TEDAE |
| **Aportación PIB** | 1 % (total) 6,1 % (industrial) | |
| **Volumen Negocio** | 9.029.000.000 € *(2018)* | |
| **Crecimiento** | ↑ 5,88 % *(respecto 2017)* | |
| **Evolución** | ↑ 440 % *(desde 2000)* | |
| **Inversión I+D+i** | 1.065.420.000 € *(2018)* | |
| **Empleo** | 43.265 empleados | |

**Tabla 1.** Visión global Sector Aéreo Español *(2018)*

En la tabla anterior podemos apreciar el volumen y tamaño del sector. A continuación presentaremos otra tabla, aunque esta estará centrada en el análisis de la evolución del tráfico aéreo comercial español. De esta forma comprenderemos mejor el segmento de mercado al que nuestra aplicación irá destinada *(en España)*.

| Aspecto | Valor | | Fuente |
|---|---|---|---|
| **Segmentación** | 36,6 M pax *(doméstico)*<br>132,3 M pax *(UE)*<br>32,9 M pax *(extra-UE)* | 18 %<br>66 %<br>16 % | Ministerio de Transportes, Movilidad y Agenda Urban |
| **Pasajeros *2019*** | 201.500.000.000 pax | | |
| **Pasajeros *2018*** | 193.600.000.000 pax | | |
| **Crecimiento** | ↑ 4,1 % *(respecto 2018)* | | |
| **Crecimiento Compañías traf. Doméstico** | **Vueling**<br>**Volotea**<br>**Ryanair**<br>**Air Europa**<br>**Grupo Binter**<br>**Iberia Express**<br>**Iberia**<br>**AirNostrum**<br>**Norwegian** | ↑ 10,3 %<br>↑ 10,0 %<br>↑ 8,90 %<br>↑ 8,80 %<br>↑ 8,80 %<br>↑ 7,70 %<br>↑ 6,30 %<br>↑ 1,60 %<br>↓ 39,30 % | |
| **Crecimiento Compañías traf. UE** | **Jet2.com**<br>**EasyJet**<br>**Vueling**<br>**Lufthansa**<br>**Iberia**<br>**Ryanair**<br>**Transavia**<br>**Grupo Tuifly**<br>**Norwegian** | ↑ 9,10 %<br>↑ 7,40 %<br>↑ 6,60 %<br>↑ 6,40 %<br>↑ 6,30 %<br>↑ 6,20 %<br>↑ 0,70 %<br>↓ 5,30 %<br>↓ 5,60 % | |
| **Crecimiento Compañías traf. extra - UE** | **Ryanair**<br>**American Airlines**<br>**Norwegian**<br>**Turkish Airlines**<br>**Iberia**<br>**Air Europa** | ↑ 34,70 %<br>↑ 15,10 %<br>↑ 14,50 %<br>↑ 10,70 %<br>↑ 9,40 %<br>↑ 9,20 % | |

**Tabla 2.** Evolución tráfico aéreo español *(Enero - Octubre 2019)*

## 2.2. Análisis del Sector de las *Apps*

El año pasado los usuarios consumidores de aplicaciones, generaron ingresos superiores a los 101.000 millones de dólares a las diferentes tiendas de aplicaciones como Google Play y Apple Store superando en un 75% el ingreso obtenido por las mismas en 2016. Así lo reveló la casa de análisis *App Annie*, que se especializa en medir el mercado de descargas de aplicaciones y consumo de *software* a nivel mundial. En su estudio *The State of Mobile 2019*, la compañía detalla que el año pasado en todo el planeta se descargaron cerca de 194.000 millones de aplicaciones *(Hernández , 2019).*

Existe una gran variedad de aplicaciones con temáticas muy distintas, por lo que hay dos factores clave a contemplar a la hora de crear una aplicación:

- Sector, segmento o nicho
- De pago o gratuita

Si intentamos encontrar el patrón que tienen en común las aplicaciones más descargadas, podemos observar que están especializadas en un sector concreto, (*comunicación, ventas, información, etc.)*.

Gran parte de ellas son gratuitas, de esta manera su alcance es mayor y existen menos barreras de entrada para que la *App* sea aceptada por el usuario final. También brindan una experiencia social y compartida, esto es un fenómeno fácilmente evidenciable en plataformas como *Instagram, Facebook, Whatsapp, Tik Tok,* etc.

En este mercado, cada vez más competitivo, existen gigantes que llevan años dominando e invirtiendo grandes cantidades de dinero en el sector. Un claro ejemplo de esto es la red social Facebook, que gracias a su apuesta constante por la innovación y la mejora de procesos en sus productos, ha conseguido permanecer en la cima de las empresas tecnológicas. Facebook tiene como estratégia comprar todas las *Apps* que podrían llegar a convertirse en una competencia sólida. Tal y como sucedió con *Whatsapp e Instagram*.

Podemos evidenciar que es un mercado en el que cuesta mucho hacerse un espacio, ya que por muy buena, bonita y funcional que sea una aplicación, si sus

desarrolladores y/o promotores no disponen de los recursos necesarios para promocionarla es muy difícil poder competir con gigantes como Facebook.

Hoy en día las aplicaciones constituyen una de las fuentes más fáciles y globales de suministrar servicios a los usuarios, pues como ya hemos apuntado anteriormente estás son gratuitas y están al alcance de todo el mundo, con tan sólo tres *clicks*.

Aún así, somos muy conscientes que nuestros recursos son muy limitados, nuestro equipo tan sólo dispone de dos integrantes, nuestra experiencia es casi nula y no disponemos de presupuesto alguno.

En parte, es posible la confección de una *App* gracias a grandes compañías como Google, que ofrecen a los programadores la oportunidad de crear servicios de forma cuasi gratuita, puesto que su mentalidad es que los desarrolladores paguen a medida que su aplicación se use *(Pay as you grow)*, esto facilita la construcción de proyectos como este.

Esta facilidad supone una gran ventaja, pues la infraestructura necesaria para la creación es casi nula, así como la inversión económica inicial. Aunque más adelante veremos que para que la aplicación realmente funcione a nivel de producción, es necesaria una cierta inversión que permita pagar a los proveedores de información, para poder aumentar el número de *requests* mensuales así como también aumentar los límites de consultas y capacidad de almacenamiento de nuestra base de datos.

Las *Apps* nos permiten realizar acciones de forma fácil y rápida, acciones como: realizar un cálculo gracias a la calculadora, realizar una fotografía y compartirla al instante con cientos, miles e incluso millones de personas en todo el mundo, realizar compras, leer libros, ver videos, leer notícias, aprender, etc.

Una ventaja para los usuarios y un aspecto que nosotros como desarrolladores debemos de tener muy en cuenta, pues se ha convertido en una necesidad impuesta por el mercado, es la especial atención a los detalles y a la que en este trabajo trataremos bajo el nombre de *User Experience (UX)*.

Hoy en día una aplicación debe de ser atractiva para el usuario, y no sólo funcional, puesto que la funcionalidad es algo que para una aplicación está presupuesto. La funcionalidad es una y puede ser la misma para distintas

aplicaciones pero una forma de destacar es presentar un diseño distinto del resto de aplicaciones que sirven al mismo propósito. De forma que la estrategia de diferenciación de nuestro producto/servicio no se basa tan solo en el valor añadido sinó también en un diseño bonito y distinto.

Una de las mayores **ventajas** de las *Apps* es la facilitación y agilización de las operaciones de compraventa. En general esto radica en la simplicidad de su uso, así como la inmediatez y cercanía en el acceso a la información o a determinados servicios.

Las aplicaciones móviles también han aparecido en la gestión interna de empresas pues permite acceder tanto a documentos y herramientas de la empresa, como a servicios de mensajería y correo electrónico que facilitan la comunicación de los empleados, mejorando el flujo de información y el cumplimiento de objetivos. En este sentido, las *Apps* favorecen el crecimiento de la productividad en el día a día de las empresas y un ahorro en costes de mantenimiento.

Comentar que no todo son ventajas por lo que al desarrollo de aplicaciones concierne, existen una serie de **desventajas**. Las aplicaciones requieren tiempo y dedicación, pese a que el desarrollo de las mismas no sea garantía de éxito y eso es debido a una de las desventajas más notables, la difícil aceptación de la misma por parte de los usuarios.

Los estándares de calidad son altos y no es fácil ofrecer todo lo que el cliente necesita o requiere en las primeras iteraciones del código, más aún si es el primer producto o solución que presentas al mercado.

Para obtener una mayor visibilidad, puesto que una aplicación una vez creada es igual que una página web, una isla desierta, si no la presentas y la promocionas, esta no existe. Difícilmente la aplicación ganará visibilidad por sí sola, para ello existen las tiendas de aplicaciones.

Aunque para poder publicar las aplicaciones en dichos portales es necesario seguir procedimientos concretos, por ejemplo en iOS, el propio sistema operativo, es el encargado de gestionar su tienda de aplicaciones, debe revisar la aplicación que se quiere publicar, de forma que estos den el visto bueno.

Por último pero no menos importante debemos considerar que las aplicaciones ocupan espacio de almacenamiento, así como también hacen uso de la batería. Por lo que estos dos indicadores pueden producir numerosas desinstalaciones de la aplicación, que como hemos comentado no resulta fácil obtener dichas descargas, por lo que que son indicadores a los que debemos prestar atención para evitar puntos de fuga de nuestros futuros usuarios.

Para poder publicar en estos dos mercados de aplicaciones, es necesario disponer de las licencias correspondientes, que te certifican como desarrollador de iOS y de Android. Ambas **certificaciones** son anuales y de pago, con diferencias económicas notables.

- Desarrollador iOS: 99 €/año
- Desarrollador Android: 25 €/año

Ahora ya podemos ver que los costes iniciales de lanzamiento de una aplicación nos son nulos, puesto que existen unos mínimos para empezar a publicar los productos/servicios confeccionados.

Una vez analizado el mercado global de las *Apps* y presentado las ventajas y desventajas de estas, procederemos a realizar un análisis de las *Apps* existentes a fecha de hoy, que suponen una competencia directa para el producto que estamos desarrollando.

*App Aena*

Comprende desde el seguimiento de cambios producidos en un avión hasta el pago y facturación del servicio de parking. Esta aplicación ofrece un servicio integral para el usuario. El único contra de esta aplicación es que su utilidad se ve limitada a vuelos con salida o llegada a aeropuertos del estado español.

Esta aplicación ofrece información en tiempo real, puesto que cuenta con una gran BD a nivel estatal.

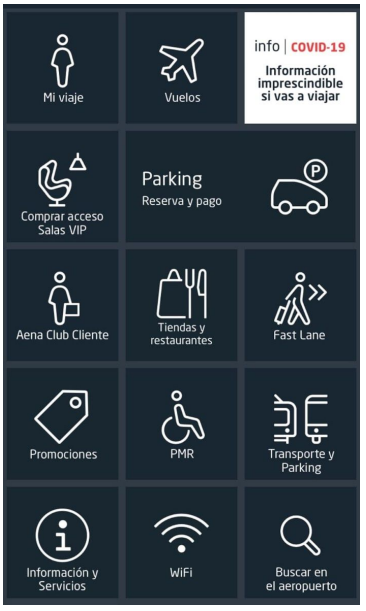



**Figura 1.** Aplicación *Aena*

### *App* Expedia

A continuación analizaremos uno de los comparadores de vuelos más conocidos y arraigados en el sector. Nos permite, como podemos ver en la siguiente figura, encontrar alojamiento, vuelos, paquetes de ofertas, alquiler de vehículos y a la vez actividades a realizar en el lugar de destino.

Poder encontrar toda esta información y adquirirla, supone un gran incentivo para los usuarios, facilitandoles todos los elementos indispensables para realizar un viaje.


**Figura 2.** Aplicación *Expedia*

La aplicación de Expedia es un claro referente a tener presente, pues su oferta resulta muy atractiva para el público que nosotros queremos alcanzar. Expedia, no dispone del servicio que nosotros si brindamos como la creación de notificaciones.

### *App* eDreams

Esta aplicación es muy similar, por no decir casi idéntica a la de Expedia. Con la diferencia que esta nos ofrece la posibilidad de comprar billetes de tren, autobús, y reservar en ciertos restaurantes de forma previa al viaje. Estamos ante otra *App* popular con ofertas y servicios complementarios, pues satisfacen las necesidades de los usuarios. La *App* no dispone de servicio de notificaciones.

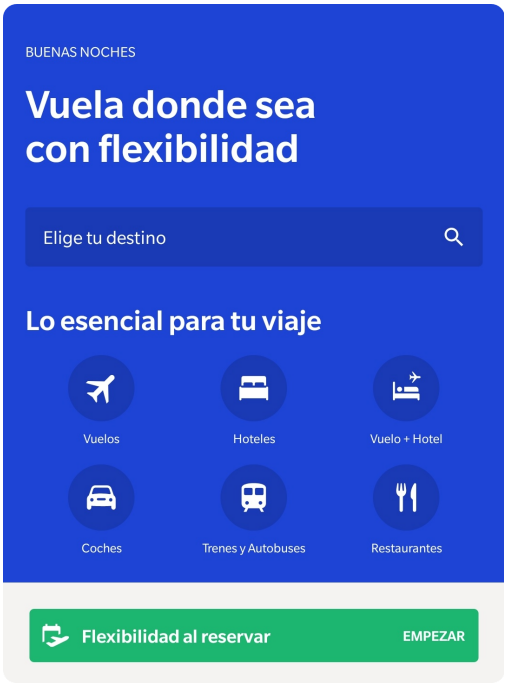



**Figura 3.** Aplicación *eDreams*

*App* Kayak

Esta aplicación ofrece servicios que nosotros también hemos incluido en nuestro proyecto, como por ejemplo la consulta de salidas y llegadas de los vuelos acordes a un aeropuerto.

En la figura 4 podemos apreciar que la estructura principal entorno a la cual se articula la *App* no dista mucho de las analizadas anteriormente. Incluye buscador de vuelos, hoteles, automóviles e incluso paquetes *(combinación de estos elementos)*.

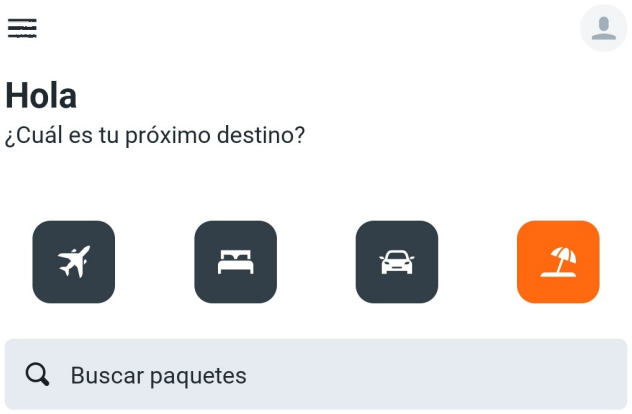



**Figura 4.** Aplicación *Kayak*

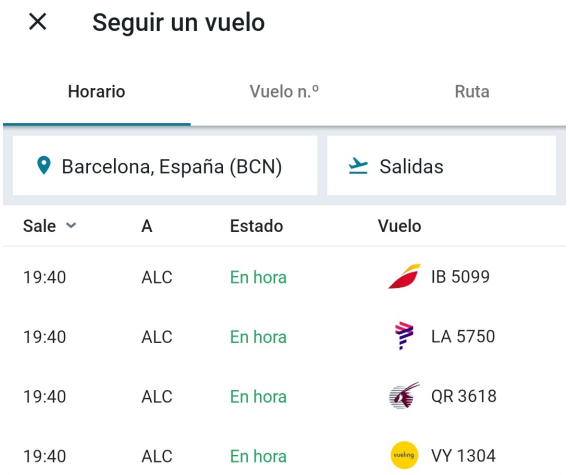



**Figura 5.** *Kayak departures*

*App* Hopper

Esta aplicación pese a no ser tan conocida como las mostradas anteriormente, sí que cuenta con un sistema de notificaciones, de forma que puedas vigilar la variación del precio de un vuelo. Sin embargo esto tan sólo puede aplicarse a un día concreto, mientras que nuestra aplicación es capaz de hacerlo en un rango de fecha y a un precio determinado por el usuario, de forma que no reciba notificaciones hasta que el vuelo encontrado se adapte a sus necesidades/deseos.

Un aspecto interesante de esta aplicación es la diferenciación que ofrecen en el calendario cuando un usuario está buscando un vuelo.

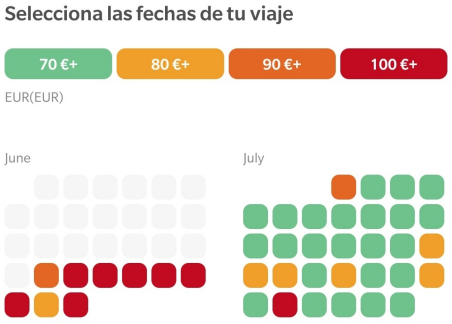



**Figura 6.** Calendario búsqueda *Hooper*

En la siguiente figura podemos comparar el sistema de notificaciones entre *Hopper* y nuestra *App*.


**Figura 7.** Notificaciones *Hopper*

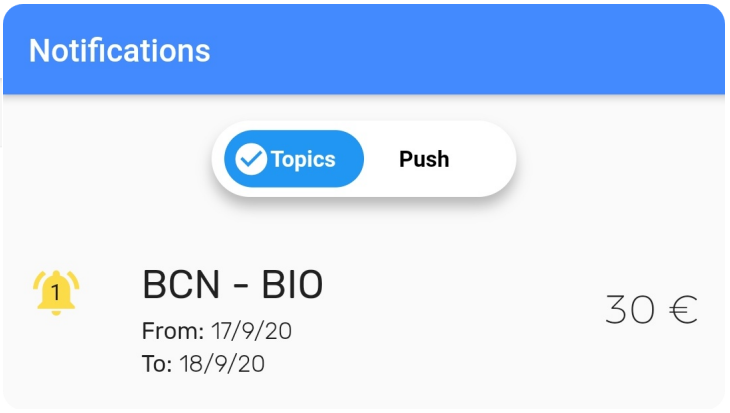



**Figura 8.** Notificaciones *UFly*

Por lo que respecta al sistema de notificaciones de Hopper se crea un hilo referente a dicha "vigilancia", mientras que nosotros tan sólo mostramos un listado de todas las notificaciones generadas una vez se establece el tópico al cual se quiere prestar atención.

Para finalizar con el análisis del mercado de las *Apps* mostraremos el número de descargas de las aplicaciones anteriores, para ver la aceptación de los usuarios en función *(App)*.

| Aplicación | Descargas |
|---|---|
| Aena | + 500.000 |
| Expedia | + 10.000.000 |
| eDreams | + 5.000.000 |
| Kayak | + 10.000.000 |
| Hopper | + 10.000.000 |

**Tabla 3.** Descargas de *Apps* del sector

Los números hablan por sí solos, tras analizar las principales aplicaciones que suministran un servicio similar al desarrollado y disponibles en el *Play Store* podemos obtener una idea general sobre cuáles son las similitudes que existen entre ellas.


**Figura 9**. Comparativa de *Apps*

Tras presentar a grandes rasgos el mercado de las *Apps* y analizadas aquellas que guardan cierta similitud con el servicio que nosotros planteamos en el desarrollo de este TFG, podemos constatar que nuestra aplicación no dista mucho de las otras ofertas que existen actualmente en el mercado.

Cabe destacar que tan sólo hemos analizado aquellas que hemos considerado más consolidadas y con mayor experiencia en el sector. De todas formas existen una gran cantidad de aplicaciones que no aparecen en el análisis y que sirven al mismo propósito. Esto denota la gran competitividad del mercado y no tan sólo en esta área, sino que este fenómeno es extrapolable a cualquier nicho o segmento de mercado de las aplicaciones móviles.

Podemos pues observar la importancia que supone aportar una propuesta de valor innovadora, que permita diferenciarse del resto, ya sea a nivel de funcionalidad, uso o diseño.

# 3. Estructura del proyecto

Para poder realizar un **proyecto** de tal magnitud, en el tiempo indicado y realizar un estudio post desarrollo, decidimos estructurar planificar los tiempos y cargas de trabajo bajo unos ciertos criterios que comentaremos a continuación.

Para poder maximizar la eficiencia y optimizar procesos de aprendizaje e implementación, decidimos que cada uno de los integrantes se centraría en una área concreta del proyecto. Aunque ambos conocemos todos los aspectos relacionados con ambas partes y hemos sido de ayuda en todos los puntos del trabajo, decidimos que proceder bajo la dinámica de **especialización**, de esta forma podríamos alcanzar mayores logros, aunque luego estos tuvieran que ser compartidos, comprobados y explicados.

Nos gusta pensar que hemos estado trabajando de forma *async* (veremos más adelante en este trabajo que implica a nivel conceptual) puesto que es un paradigma de la programación "paralela" que permite que se ejecuten de forma simultánea líneas de código, en este caso no hablamos de código sino de cargas de trabajo.

Para poder sintetizar dichas cargas y no entrar en detalle de forma previa a la explicación, resumimos que *Hamet Lamin Jalil Beida* dedicó gran parte de su tiempo a entender y aplicar conceptos de programación en Dart al apartado front-end del proyecto. En cambio *Roger Villalba Fuentes* se centró en desarrollar la estructura de la BD y crear las *Cloud Functions*, en definitiva el back-end.

Destacar que tras contemplar todas las fases de desarrollo del proyecto, no existe una parte fácil y una difícil, pues cada una de ellas presentan numerosas dificultades que entre la ayuda y colaboración de ambos hemos superado. Una de las dinámicas que más hemos usado ha sido la programación conjunta de objetivos, con reuniones de seguimiento, casi diárias, para comprobar dichos avances en los objetivos o en su defecto presentar los problemas abordados y buscar nuevos enfoques que quizá no se habían planteado previamente.

El proyecto como tal, se base en la iteración de 3 procesos:

- **Diseño** *(concepción, idea, principales pantallas y propuesta valor)*
- **Desarrollo** *(uso de herramientas para lograr objetivos establecidos)*
- **Testeo** *(presentar avances, análisis y test, replanteamiento o finalización)*


**Figura 10.** Esquema de gestión interna del equipo

A parte de estas tres existen fases previas e intermedias, sin embargo no constituyen por sí solas un bloque fundamental del proyecto. Por lo que referencia a nuestra metodología se puede aplicar un esquema muy parecido al mostrado de forma gráfica en la figura 10.

Las fases de diseño y testeo se realizaron de forma conjunta.

En la fase de **diseño** se establecieron los objetivos a alcanzar y definimos la dirección que se iba a tomar para el desarrollo.

La fase de **desarrollo** se realizó de forma individual, cada uno afrontó sus problemas en el área/campo de estudio.

Finalmente en la fase de **testeo** se compartían los resultados y conclusiones obtenidos en cada línea de trabajo. De esta forma alguien ajeno a la tarea *(ajeno en el sentido estricto de la palabra pues existía un contexto y una idea sobre lo que se estaba evaluando)* podía aportar valor, constatar puntos de fallo o aspectos de mejora.

En definitiva, nos permitía obtener una visión más amplía del servicio y de los componentes que estábamos desarrollando, así como también crear una metodología de trabajo y motivarnos de forma mutua y constante.

Para poder detallar con mayor precisión y obtener una visión general de todo lo que implica nuestro proyecto, a continuación se muestra un diagrama que identifica los tres pilares clave del proyecto.


**Figura 11.** Pilares investigación, desarrollo y testeo

Para poder comprender mejor estos tres pilares de la figura anterior debemos contemplar las dimensiones de estos campos, más adelante detallaremos cada uno de ellos, y su valor para nuestro proyecto.

Cabe destacar que la investigación es sumamente relevante, pues nos permite trazar o más bien delimitar el camino que deberemos recorrer durante el desarrollo. Podemos ver que existe un tercer elemento que no resulta un pilar como tal, sino que permite cerciorarse de que ambos pilares *(investigación y desarrollo)* han sido consolidados. Todos los campos fueron sometidos a numerosas oleadas de comprobaciones para asegurarnos de:

1. Comprendemos cómo beneficia al proyecto
2. Conocíamos las limitaciones de de uso
3. Cerciorarse del correcto funcionamiento de las herramientas implementadas

En definitiva, estos pilares permiten garantizar la consistencia de nuestra propuesta, así como también consolidar las conceptos aprendidos y maximizar esfuerzos en aquellos que aún no éramos capaces de implementar.

Destacar que los pilares son muy importantes para comprender el trabajo realizado. Al tratar campos realmente amplios, para comprender el proyecto hemos creído conveniente estructurar nuestro proyecto a partir de la explicación de los pilares, por ello el trabajo ha sido estructurado de la siguiente manera :

- Marco Conceptual *(Punto 4)*
  - *Introducción teórica de las herramientas usadas en el desarrollo, para su contextualización y comprensión. (Resumen Investigación).*
- Metodología *(Punto 5)*
  - *Establecer la relación entre herramientas y elementos/servicios confeccionados. (Puente entre Investigación y Desarrollo).*
- Desarrollo *(Punto 6)*
  - *Entender a través del trabajo realizado los elementos/servicios creados para la App. (Resumen Desarrollo).*
- Puesta en Producción *(Punto 7)*
  - *Comentar puntos de fallo y errores destacables del proyecto, dificultades afrontadas y sus soluciones. (Resumen Testing).*

# 4. Marco Conceptual

A grandes rasgos en este apartado del trabajo haremos un barrido para detallar las principales herramientas y recursos que hemos empleado en el desarrollo del proyecto. Analizaremos su utilidad y mencionaremos sus principales características. Abordaremos desde conceptos básicos hasta especificaciones en lenguajes, plataformas o recursos disponibles de los distintos proveedores.

Antes de empezar debemos tener claros, los distintos roles que existen en el mundo de las aplicaciones.

***Front-end*:** Es la parte de la aplicación que interactúa con los usuarios, conocida como el lado del cliente. Básicamente es todo aquello que vemos en la pantalla al acceder a la *App*: botones, colores, iconos, teclado, animaciones, transiciones, textos, mensajes, alertas, navegación, etc. En definitiva todo la programación dedicada a lo que uno ve e interactúa con, dicho de otra manera un desarrollador front-end se encarga de la **UI** y **UX,** interfaz de usuario y experiencia del mismo.

***Responsive UI:*** Define la creación de aplicaciones con la capacidad de adaptarse a todo tipo de pantallas. Para testear la *App* se compila la UI en un dispositivo ya sea virtual o físico, un diseño *responsive* es aquel que se adapta al dispositivo del usuario, independientemente de su dispositivo.

***Back-end*:** Esta parte engloba toda la lógica que se encuentra por detrás de la *App*, es la parte invisible para el usuario, este no sabe ni de su existencia y por eso se le denomina a menudo “el lado del servidor”. Se encarga de la toma de los datos, almacenaje en la BD, procesar la información y entregar los resultados a los usuarios. Los desarrolladores Back-end deben tener amplios conocimientos en *frameworks,* tipos de base de datos, entender y saber trabajar con múltiples lenguajes.

A continuación procederemos a analizar las APIs, que son y cuál es su principal utilidad, detallaremos cuales son los lenguajes principales necesarios para poder entender el código que conforma la aplicación, puesto que planteamos una aplicación multiplataforma, con gran parte de su servicio en el *Cloud*. También analizaremos los *frameworks* empleados y sus correspondientes *SDKs*.

## 4.1. APIs

Las **APIs** *(Application Programming Interfaces),* son herramientas que permiten a los desarrolladores obtener servicios integrados. Podríamos definir API como los pilares alrededor de los cuales se articulan las aplicaciones, esto es debido a que la mayoría de las APIs que podemos encontrar se alimentan de grandes BBDD (Bases de Datos). De esta forma la aplicación puede nutrirse directamente de servicios externos, creados por terceros, sin la necesidad de crearlos desde cero. Como por ejemplo, realizar una entrada masiva de datos que bien podría tardar días, meses e incluso años.

Por ello las aplicaciones podrían ser entendidas como componentes de estas grandes bases de datos que atienden a unas necesidades concretas. Es por este trabajo previo que la mayor parte de las APIs son de pago, pues la recolección, mantenimiento y flujo constante de datos debe amortizarse por parte de los desarrolladores y las empresas propietarias de los servidores.

En definitiva lo que debemos entender, es que las APIs serán las encargadas de suministrarnos los datos que procesaremos y mostraremos en nuestra aplicación.

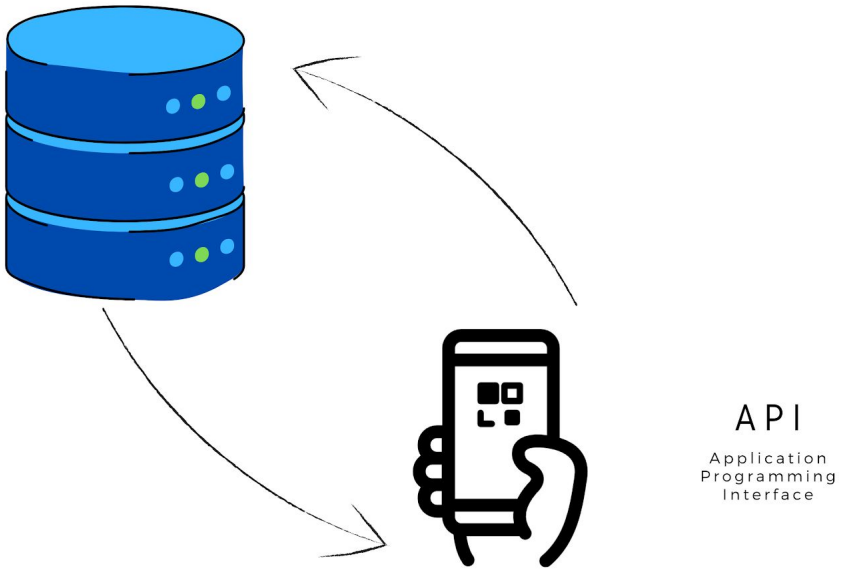



**Figura 12.** API *information exchange (esquema)*

## 4.2. Lenguajes

Un lenguaje de programación es un lenguaje formal, cada uno con un conjunto de reglas gramaticales bien definidas, que ofrecen la capacidad de escribir *(programar)* una serie de instrucciones o secuencias de órdenes en forma de algoritmos con el fin de poder controlar el comportamiento lógico/físico de una computadora. El conjunto de órdenes escritas o más comúnmente conocidas como código, constituyen un programa.

Todos los lenguajes tienen elementos comunes, como podemos ver en la siguiente tabla. De forma que pese a que cambie la gramática o la sintaxis, los elementos suelen ser los mismos, a través de las cuáles un programador puede construir un programa, que acabe convirtiéndose en un producto o servicio.

| Elemento | Definición | Ejemplos |
|---|---|---|
| **Variables** | Espacios de memoria destinados al almacenamiento de datos específicos. Existen diferentes tipos de datos almacenables. | *String, int, float, bool* |
| **Condicionales** | Permiten evaluar condiciones en ciertos puntos del código. | *if, else, else if* |
| **Bucles** | Ejecutan las líneas de código incluidas en ellos de forma iterada hasta que se cumpla cierta condición/condiciones. | *For, while* |
| **Funciones** | Permiten desmenuzar el código evitando reescribir la misma orden cada vez que se quiera ejecutar.. | *print('Hello World');* |

**Tabla 4.** Elementos de un lenguaje

Los lenguajes se pueden clasificar en estáticos o dinámicos, en estos primeros se debe inferir de forma previa a la compilación el tipo de variable que se almacenará. En cambio en un lenguaje dinámico, el valor de cada uno de los tipos de variables se infiere de forma automática en tiempo de ejecución.

Para poder verlo con mayor detalle, C y C++ son lenguajes estáticos, mientras que Java o Dart, son lenguajes dinámicos, en este último existe de hecho un tipo de variable llamada *dynamic* que permite directamente declarar variables sin necesidad de indicar que tipo de información almacenará.

A continuación hemos confeccionado una tabla donde se listan algunos de los lenguajes más conocidos actualmente.

| Lenguajes de programación más utilizados | |
|---|---|
| **Front-end** | **Back-end** |
| HTML | PYTHON |
| CSS | PHP |
| JAVASCRIPT | NODE.JS |
| jQUERY | GO |
| SWIFT | TYPESCRIPT |
| REACT | C# |
| DART | JAVA |
| ANGULAR | DJANGO |
| REDUX | RUBY |

**Tabla 5.** Lenguajes de programación más utilizados

Algunos de estos lenguajes están diseñados de forma expresa para el desarrollo en sistemas operativos concretos, como por ejemplo Swift *(plataforma iOS)* o para ámbitos concretos como Matlab para cálculo, álgebra, física o investigación.

**DART**

Un lenguaje que pese a crearse en 2011, aún no ha alcanzado su máximo esplendor. Es un lenguaje que junto a un *SDK* (*Flutter*) que veremos en más detalle en el apartado de SDKs, nos permite un desarrollo multiplataforma, en otras palabras no permite desarrollar aplicaciones para sistemas operativos distintos *(iOS y Android)*. Así como también para aplicaciones web, sin la necesidad de replicar el mismo código *(parte lógica)* en tres lenguajes distintos, uno para cada plataforma.

Destacar que está respaldado por una de las mayores compañías tecnológicas del mundo *(Google)*, lo cual nos asegura una cierta mantenibilidad y estabilidad.

**TypeScript**

Es un lenguaje tipado que permite encontrar errores antes de su compilación, pues es estático a diferencia de JavaScript que es dinámico.

Una de las principales ventajas que ofrece TypeScript es la creación de *Scripts* que se ejecutan como si hubieran sido desarrollados en JavaScript. Para simplificarlo, creas código de forma segura, luego, antes de ejecutarlo, con el comando:

> ***node*** myFile.js

Debemos considerar que el archivo creado es .ts (*TypeScript*) y debemos convertirlo a JavaScript antes de ejecutarlo, esto añade un paso al desarrollo aunque tan sólo hay que ejecutar en el directorio en el que se encuentre el archivo que queremos convertir de ***.ts → .js***, el siguiente comando:

> ***tsc*** myFile.ts

Tras la ejecución del *cmd* dispondremos, en el mismo directorio del fichero myFile.js, listo para ser ejecutado. Destacar también que existen otras formas de sincronizar ambos ficheros para no tener que estar haciendo esta conversión por cada cambio que hagamos y queramos comprobar. Aunque hay que sincronizarlos previamente, de forma que la conversión se ejecute de forma automática por cada cambio que hagamos en el fichero .ts .

## 4.3. Herramientas y recursos

En este apartado aprovecharemos para hablar sobre los *SDKs,* el *Cloud Environment* y los distintos editores de textos *(IDEs)* investigados.

Antes de detallar cómo los hemos empleado y para qué, debemos entender que son y que nos permiten estas herramientas para la confección de aplicaciones.

### 4.3.1. Flutter

Los **SDK**s *(Software Developer Kits)*, están formados por un conjunto de herramientas de desarrollo de *software,* que permiten a un programador/desarrollador crear una aplicación informática para un sistema concreto. Un SDK permite la comunicación entre el lenguaje (código) y una API, también puede comunicarse con determinados sistemas embebidos.

La mayor parte de los SDKs cuentan con su propia documentación así como también una comunidad que soluciona problemas, comenta aspectos interesantes y relevantes, que permiten a todo el conjunto de desarrolladores que trabajan con dicho SDK mejorarlo, añadirle capacidades y encontrar errores.

**Flutter** es un *kit* creado por Google, basado en Dart, que posibilita el desarrollo multiplataforma, así como el uso de un conjunto de *Widgets* que nos permiten diseñar de forma rápida el front-end.

Para poder entender los aspectos básicos de este SDK, primero debemos repasar ciertos conceptos claves.

Flutter está basado en el uso de ***Widgets***, cuando en Flutter decimos que todo son *Widgets* es porque literalmente así es, el programa como tal *(App)* es un *widget* formado por *widgets*.

Para poder entender mejor este fenómeno, debemos comprender que un simple texto, un botón, una imagen, una pantalla o un icono, en Flutter son *widgets*. Es más existen *widgets* que nos permiten colocar múltiples *widgets* en ciertas posiciones/ubicaciones de la pantalla.

En la siguiente tabla se muestra a *grosso modo* los tipos de *widgets* que se pueden encontrar en Flutter, con una pequeña descripción y algunos ejemplos de estos.

| Tipo | Utilidad | Ejemplos |
|---|---|---|
| Estructura *App* | *Widgets* que constituyen elementos básicos de una aplicación y de su navegación | *AppBar, Drawer, Scaffold* |
| Botones | *Widgets* que permiten mezclar interfaz *(diseño)* y funcionalidad *(lógica)*. | *ButtonBar, FlatButton, FloatingActionButton* |
| *Input* y selectores | Estos *widgets* nos permiten la entrada de datos y modificación de los mismos. | *TextField, Switch, Slider, DateTimePicker* |
| Paneles, alertas y cuadros de diálogo | Permiten la comunicación entre la aplicación y el usuario. | *AlertDialog, ExpansionPanel, SnackBar, SimpleDialog* |
| Informativos | Nos permiten informar al usuario de que es lo que está sucediendo en la aplicación. | *Card, CircularProgressIndicator ,Icon, ToolTip* |
| Diseño | Permiten estructurar los *widgets*, algunos permiten incluir otros widgets y otros ya disponen de una distribución concreta. | *Divider, ListTile, Column, Row* |

**Tabla 6.** Resumen *Widgets* y ejemplos

Un término muy importante en la programación es el manejo del estado. En Flutter existen diferentes técnicas, para poder entender más adelante cuáles hemos empleado en nuestro desarrollo debemos antes, comprender que es y que implica la gestión del estado.

El *State Management*, manejo del estado, es la metodología empleada para el gestionar el flujo de información, persistencia de datos o simplemente comportamientos de la UI. La verdad es que no existe una definición estandarizada para este concepto, cada desarrollador la interpreta e implementa a su manera, aún así se persigue el mismo objetivo, poder manejar los cambios que se generan en la interfaz de usuario con la interacción del mismo.

Para entender de una forma más sencilla en qué consiste, procederemos a explicarlo mediante un ejemplo.

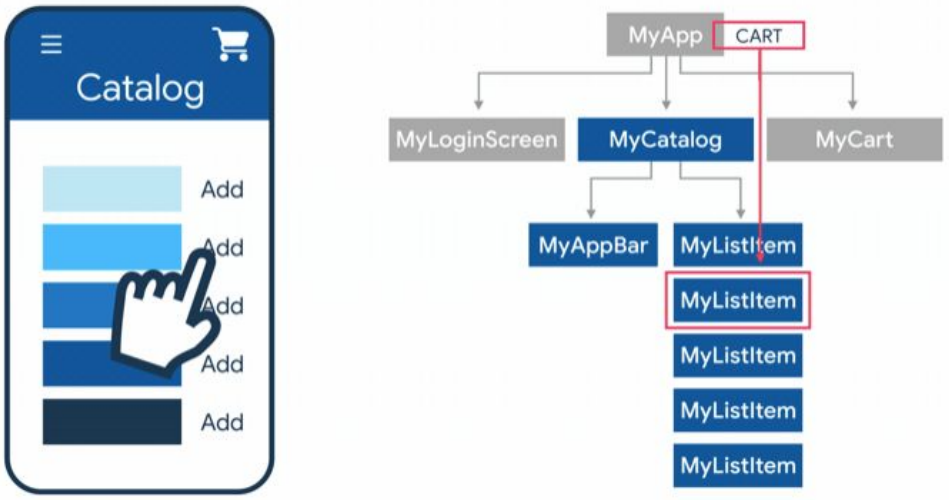



**Figura 13.** Ejemplo manejo de estado. Fuente. Flutter Documentation (2017)

Como se puede observar en la figura 13, disponemos de una pantalla UI de login, una de Catálogo con un estado inicial contenido en un *AppBar* y 5 productos y también disponemos de una pantalla de carrito con un estado inicial, vacía.

Ahora bien, el usuario se encuentra en la pantalla del catálogo y realiza la acción de añadir un producto, por tanto inmediatamente necesitamos que se actualice la pantalla del carrito, pasando de un estado inicial de vacío a disponer del producto seleccionado.

El *State Management,* permite que una *App* pase de ser estática, a ser dinámica, o también conocida como *reactive*. En el SDK que empleamos existen diversas maneras de gestionar el estado de la aplicación, algunas fáciles aunque limitadas y otras más complejas aunque su abanico de posibilidades es mucho más amplio.

Bien, ahora que tenemos una idea más clara de en qué consiste la gestión del estado, veamos la importancia que tiene, y cómo de imprescindible es para cualquier *App* o página web. Antes de detallar cada una de las técnicas, debemos diferenciar los dos tipos de *widgets* que existen en Flutter, el ***StatelessWidget*** (sin estado, estático) y el ***StatefulWidget*** (con un estado en dicho *widget*). Este último también se conoce como estado efímero, estado de la interfaz de usuario o estado local. Este se emplea para manejar un único *widget*, dónde las otras partes del árbol de *widgets* no requieren acceso al estado *(cambios)* de este. Por tanto no se necesita utilizar técnicas de gestión de estado, todo lo que se necesita es un *StatefulWidget*.

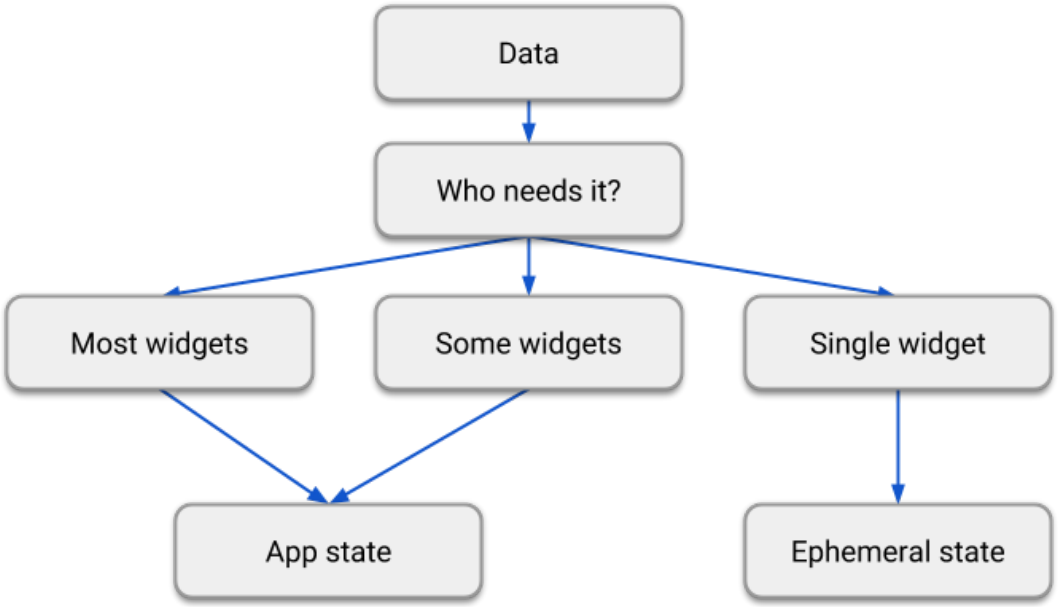



**Figura 14.** Ejemplo de los dos tipos de estado.

Por otra parte, existe el estado de la aplicación, que representa un estado superior, uno que que queremos compartir con otros *Widgets* (otros componentes de la aplicación).

Algunos de los elementos que denotan una gran necesidad de disponer de estado, son por ejemplo, las preferencias de usuario, información de inicio de sesión, notificaciones en una aplicación de redes sociales, el carrito de compras en aplicaciones de comercio electrónico o el estado de artículos leídos y no leídos en una *App* de noticias, entre otros.

No existe una regla clara y universal para distinguir si un *widget* debe ser efímero o formar parte del estado de la aplicación. Muchas veces se empieza con un estado efímero, pero a medida que la aplicación crece, acaba pasando a formar parte del estado de la *App*.

Una forma fácil de visualizar esta necesidad es a través de elementos como el FutureBuilder(), un tipo de *widget* que permite que un *widget* o conjunto de ellos no se construya hasta que una *Promise* se haya completado *(complete, reject o error)*. Puesto que no se puede mostrar una información si aún no se dispone de ella. Por ello el estado es importante, y en función de este, debemos construir una pantalla u otra. Por ejemplo elegir entre *LoadingPage* o *DisplayInformationPage*.

A continuación vamos a realizar una pequeña descripción de cada una de estas técnicas de gestión del estado que se pueden encontrar y utilizar en Flutter.

***SetState*:** En *Flutter* existen dos tipos de widgets, como ya hemos comentado los *Stateless* y los *Stateful*, los primeros son estáticos como los *Icons*, *IconButton*, *Text,* etc. En cambio los segundos son dinámicos, es decir cambian su apariencia respondiendo a los eventos desencadenados por la interacción de usuario, un ejemplo de estos sería un *Checkbox*, un *Slider* o un *TextField*. El estado de estos *widgets* se almacena en un objeto de tipo *State*, diferenciando así el estado de la apariencia. Cuando el estado cambia, el objeto *State* llama al setState(), haciendo que este vuelva a dibujar el *widget*, esta vez con el nuevo estado.

**BLOC:** *Business Logic Component*, esta técnica de manejo de estado en Flutter, fue creada por Paolo Soares y Cong Hu y presentada en la conferencia de Dart en 2018, actualmente es la técnica que más recomendada por Google. Aunque no es tan sencilla de implementar como Provider, eso sí es más consistente y ofrece

acceso a la información desde un cualquier parte a todo el proyecto. Hay que instalar su *Package* en el ***Pubspec.yaml***. Es una técnica basada en *Streams*, es decir los datos fluyen del Bloc a la UI o viceversa, en forma de flujos.

Al utilizar Bloc, creamos clases que nos ayudan a centralizar la lógica de los estados y sus correspondientes cambios, producidos en la aplicación, de esta forma, estas clases serían las encargadas de recibir las acciones o eventos que se producen en la *App* y que modifican el estado.

Estas mantienen el estado en memoria y son responsables de comunicar a todas las partes implicadas (pantallas o *widgets*) los cambios que se han producido en el estado. De forma que en la declaración de la UI tan solo declaramos los *widgets* que aparecen en función del estado de una variable, que se ve modificada por el BLOC a través de eventos.

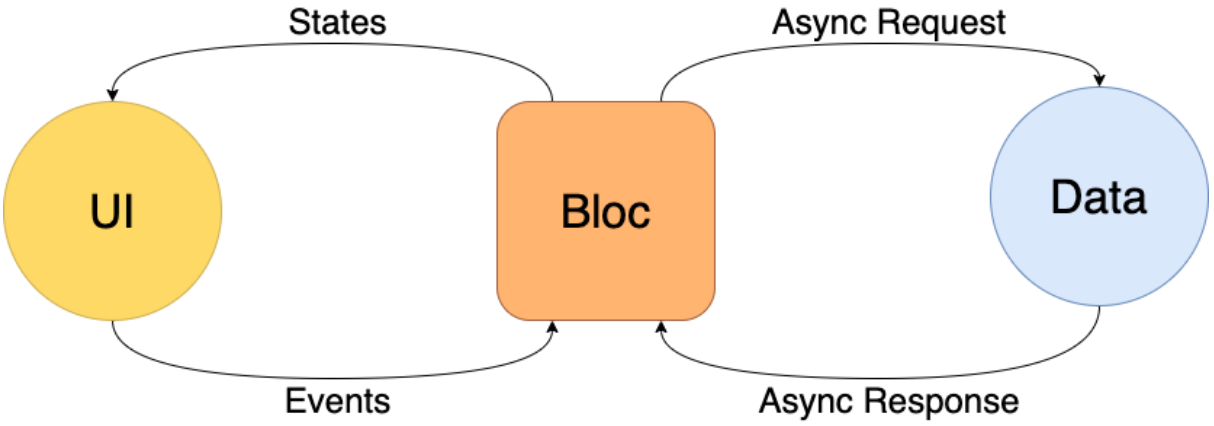



**Figura 15.** Ejemplo de flujos con el patrón BLOC. Fuente: Google Documentation.

No es una técnica sencilla, al menos las primeras veces, pues pese a su nombre no parece lógica, al menos para nuestro bagaje en programación, pues lo más parecido que habíamos empleado eran los apuntadores y en este caso está basado en el misma idea, aunque se debe tener una idea muy clara de que es un evento, que es un estado y tener presente que los eventos y estados se gestionan en el BLOC. La UI tan sólo atiende al estado. En esta también se declaran los accionadores de los eventos.

Cuando creamos un proyecto en Flutter, tras todas las instalaciones necesarias, debemos ejecutar el siguiente comando a través de la consola.

> ***flutter*** create my_app_name

De esta forma le estaremos ordenando al SDK que nos cree un proyecto, con una plantilla de *App* básica. A continuación mostraremos los ficheros que crea y una captura de la aplicación. Una vez ejecutado el comando y a través de consola se nos indica que todo ha sido creado de forma correcta.

Ahora deberemos abrir nuestro IDE para poder ver todos los recursos creados, si queremos ver qué es lo que a nivel visual se ha creado, en consola, deberemos, en el directorio en el que se encuentra nuestro proyecto recién creado, ejecutar el siguiente comando:

> ***flutter*** run

De esta forma compilamos el código *template* que nos ofrece Flutter por defecto.

En la figura 16 podemos observar todo un conjunto de ficheros/archivos que permiten el desarrollo de una *App* multiplataforma. El archivo principal de la aplicación es el **main.dart**.

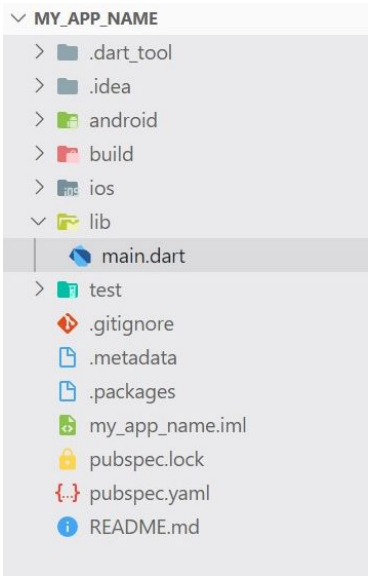



**Figura 16.** Recursos Flutter *template*

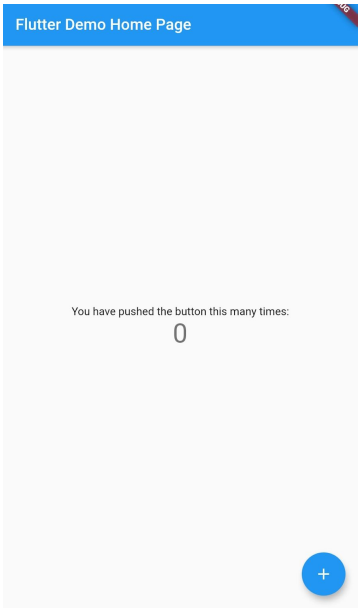



**Figura 17.** UI Flutter *template*

El código correspondiente a la figura anterior, se puede encontrar en el anexo, apartado 12.2.1 Ejemplos de código.

### 4.3.2. Google Cloud Platform

Google ofrece un gran abanico de herramientas para facilitar el desarrollo de aplicaciones, no tan solo móviles sino también web. De entre todas, tan sólo enfatizamos en tres de ellas, pues la oferta de servicios es realmente amplia y para el desarrollo de este TFG tan solo debemos entender las que aparecen en la siguiente figura con un marcador azul.

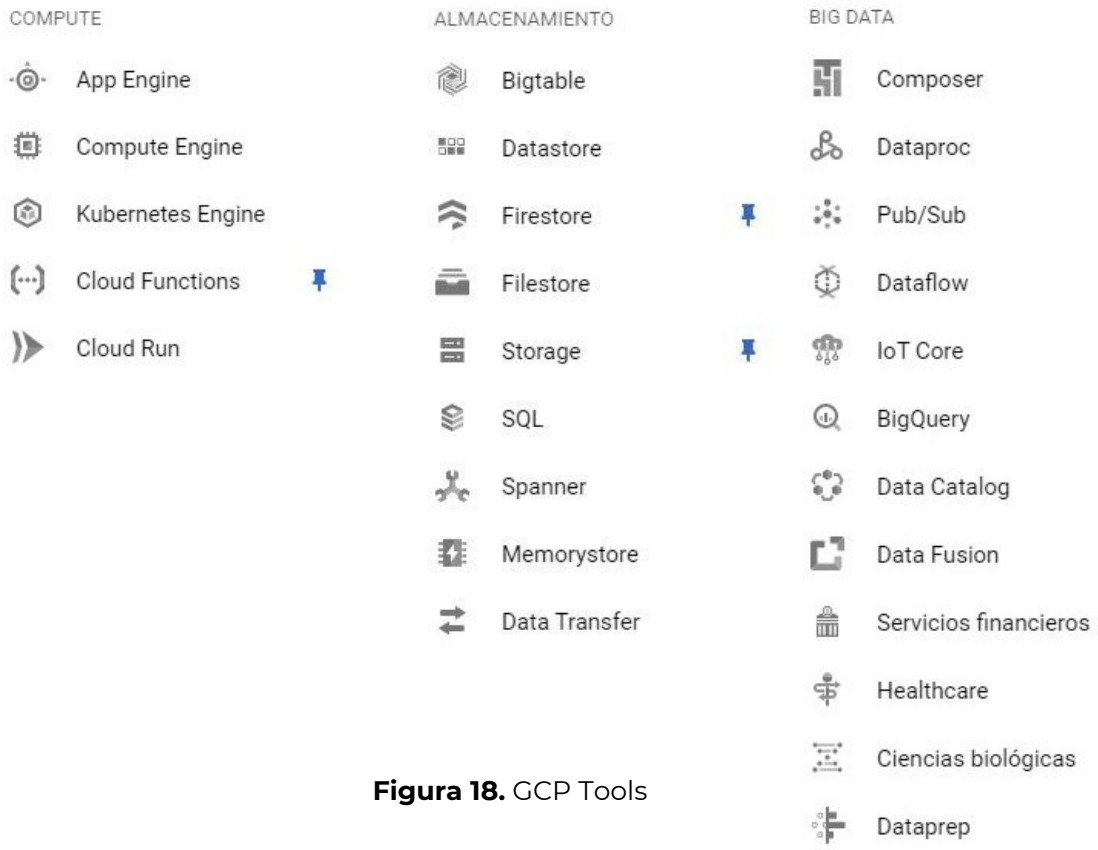



**Figura 18.** GCP Tools

#### 4.3.2.1. Cloud Functions

Las CF son una herramienta sumamente útil pues permite subir trozos de código al *Cloud* y ejecutarlos bajo demanda. En otras palabras permite construir funciones que se pueden alcanzar ya sea a través de requests *(http/s)* o restringir dichas *requests* a que tan sólo puedan ser lanzadas desde la *App*.

Como ya hemos comentado anteriormente las CF funcionan a modo de API, la única diferencia es que podemos limitar el alcance de estas, restringiendo así el acceso a nuestra base de datos.

Podemos distinguir 3 tipos distintos de CF:

| Tipo | Descripción | Ejemplos Anexo |
|---|---|---|
| **onRequest** | Son accesibles desde cualquier lugar, puesto que reaccionan *(ejecutan el código)* a través de una URL. Cualquier persona o dispositivo, entrando la dirección exacta, con los parámetros necesarios puede obtener respuesta de una función Cloud. | **12.2.2** |
| **onCall** | Tan sólo son accesibles desde una aplicación concreta, para ello es necesario disponer de un archivo JSON que contenga las claves necesarias para que la CF pueda cerciorarse de que dicha *request* se está realizando desde una aplicación con el acceso correspondiente. A su vez, obtener información sobre el identificador único de cada usuario, controlando quién pide que. Limitando el acceso a ciertos recursos de la *App*. Algo fácilmente visible, cuando se ofrece un servicio *premium*, permisos especiales, etc. | **12.2.3** |
| **Automatic Trigger** | No son accesibles de forma directa, sino que atienden a ciertos sucesos de la base de datos, estas se activan de forma automática. Por ejemplo, cuando un usuario se autentifica, podemos automáticamente crear un objeto *user* en la BD, gracias a la función **.onCreate()**. De esta forma cada vez que se autentifique un usuario creamos un objeto que contenga la información de dicho usuario, así como también asignarle cierta información que usará más adelante en la *App*. | **12.2.4** |

**Tabla 7.** *Cloud Functions*, tipos y uso

#### 4.3.2.2. Cloud Storage

Cloud Storage es un servicio de GCP que permite el almacenamiento de ficheros. Uno podría pensar, para qué necesitamos Cloud Storage si ya disponemos de una Base de Datos *(Firestore),* la respuesta es sencilla, *Firestore* tan solo permite el almacenamiento de datos del tipo *String, bool, int, double, Array y Map*. En otras palabras no ha sido diseñado para almacenar imágenes. Comentar que Cloud Storage permite no sólo almacenar imágenes sino también ficheros.

A continuación enumeramos algunas de las ventajas que nos ofrece Cloud Storage:

- Permite optimizar los precios y el rendimiento de todas las clases de almacenamiento *(Administración del ciclo de vida de los objetos)*.
- Almacenamiento seguro y duradero, dispone de capacidad para resistir errores e interrupciones. Como por ejemplo, ataques DDoS *(ataque a un servidor desde muchos ordenadores a la vez, para inutilizar un servicio).*
- Se puede emplear para entregar/suministrar contenido, *Data lakes* y copias de seguridad.
- Fácil integración de sus servicios a través de una API que suministra Google, de forma que se puede emplear en CF y en *Source Code*.

#### 4.3.2.3. Firebase

Firebase, una de las **plataformas** para desarrollo móvil más famosas hoy en día, adquirida por Google en 2014. Cuenta con un servicio de BD llamado **Firestore** que permite almacenar datos en el *Cloud*.

Antes de que existieran este tipo de plataformas, para crear una *App* se necesitaba: un equipo especializado en front-end, uno en back-end y uno encargado de juntar estos dos. Pues cada bloque empleaba sus propios lenguajes.

Desde que aparecieron plataformas como Firebase, se ha facilitado en gran medida la creación, gestión y control del back-end de una aplicación.

Esto permite la aparición de una nuevo término, el **FullStack,** dónde una sola persona es capaz de manejar el front-end y el back-end. Estas plataformas ofrecen un desarrollo más rápido, accesible y optimizado. Surgen de la necesidad de proveer una **API** para guardar, sincronizar y leer datos en tiempo real de la nube (*Cloud*).

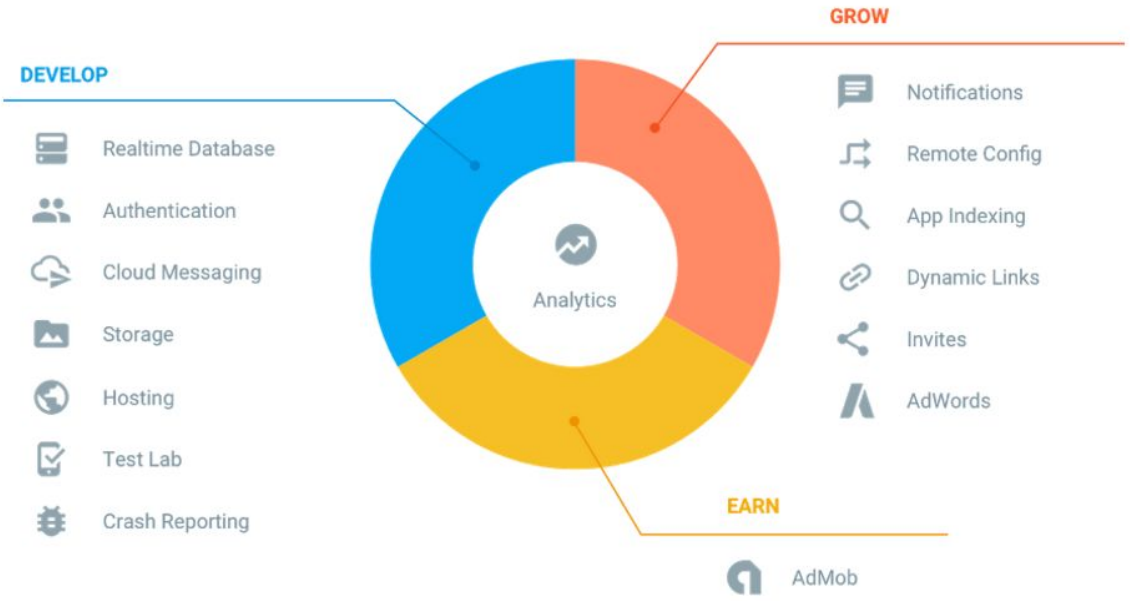



**Figura 19.** Las posibilidades que ofrece Firebase

Como se puede observar en la figura anterior, sus características principales se dividen en 4 grupos:

**Analíticas:** Se incluyen dentro del servicio gratuito de la plataforma para medir el comportamiento de los usuario en la *App*. También cuenta con un panel de métricas personalizable.

**Desarrollo:** Basado en la optimización, reduciendo el tiempo de desarrollo, existen varias posibilidades entre las cuáles destacan: el almacenamiento *Cloud*, configuración remota, detección de errores y el testeo de la *App*.

**Crecimiento:** Permite gestionar el aumento de usuarios de la aplicación, con la posibilidad de captar nuevos usuarios mediante las funcionalidades de invitación o notificación.

**Monetización:** Firebase integra una herramienta llamada *AdMob* que permite a los desarrolladores monetizar su *App* mediante la configuración de distintos elementos publicitarios *(pantallas, videos,...)*.

A la hora de aprender e implementar, Firebase pone a disposición de los desarrolladores documentación de calidad, tutoriales y ejemplos. Su equipo de desarrollo está muy activo en la comunidad de **Github** y **Stack Overflow**, ofreciendo soporte a muchos usuarios.

A pesar de ser una herramienta muy potente, no todo es bueno pues no es completamente gratis. La plataforma dispone de 2 planes pensados para las necesidades de cada usuario.

El primer plan ***'Spark'*** es gratis pero tiene ciertas limitaciones como el espacio de almacenamiento o las conexiones simultáneas, comentar que este plan no permite realizar llamadas a APIs que no sean propias de Google.

El segundo plan es el ***'Blaze'*** y es de pago por uso, pagas por lo que utilizas en cada momento. Este plan comprende todo lo que se incluye en él gratuïto, de forma que si el gasto es inferior al que Firebase ofrece por defecto con el plan *'Spark'*, no se realizará ningún cargo.

Firebase ofrece el servicio Firestore que permite crear, configurar, controlar y gestionar tu propia BD *Cloud*.

Destacar que el tipo de base de datos *(Firestore)* no se trata de la típica base de datos relacional, también conocida como BD SQL, sino que se trata de un BD NoSQL, este modelo de base de datos no se basa en almacenar los datos en tablas y luego relacionarlas a través de claves primarias, sino que el almacenamiento se realiza a través de estructuras clave-valor, un claro ejemplo y simple de este tipo de estructura es un objeto JSON, para poder gestionar mejor los conjuntos de información, Firestore emplea una estrategia de uso basada en *Documentos* y *Colecciones*.

**Colecciones**

Una colección es la suma de uno o más documentos.

**Documento**

Un documento es la agrupación de datos tipo clave-valor a través de un identificador único. Este documento puede recoger uno o más datos del tipo clave-valor.

En la siguiente figura podemos apreciar con mayor detalle esta agrupación que emplea Firestore y que pretende simplificar el uso de su servicio de BD.

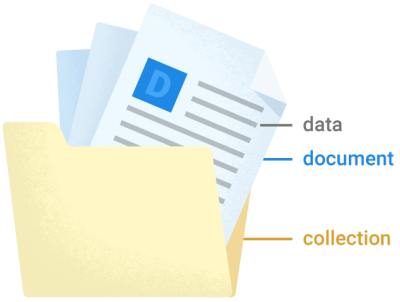



**Figura 20.** Firestore, estructura almacenamiento

En la figura anterior se puede apreciar la forma en la que se agrupan los datos en Firestore. En el anexo, apartado (12.3 Firestore), se pueden encontrar ejemplos de las *Collections* y *Documents* de ***UFly***.

Para finalizar con el apartado de Firebase nos gustaría destacar que no sólo dispone de Firestore *(servicio de BD noSQL que empleamos en este TFG)* sino que también cuenta con una herramienta llamada Realtime Database, podríamos decir que es la versión anterior a Firestore, aunque la facturación del servicio Realtime DB no se realiza a través de *reads, writes* y *deletes*.

En RTDB se almacenan objetos JSON y la información no se almacena de forma estructurada como en Firestore.

### 4.3.3. Visual Studio Code

Se trata de un editor de código creado por Microsoft y anunciado el 29 de abril del 2015 en la conferencia *Build*. El 28 de noviembre de ese mismo año fue lanzado bajo la Licencia **MIT** y su código fuente publicado en Github. Es un **IDE** multiplataforma soportado en Windows, macOS y Linux. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código *(autocomplete)*, fragmentos y refactorización *(format document)* de código.

VSC combina dos características fundamentales, la ligereza y la potencia. Tiene soporte nativo para JavaScript, TypeScript y Node.js. Dispone de un enorme ecosistema de complementos *(extensions)* que permite trabajar con casi cualquier lenguaje de programación, C, C#, Java, Python, PHP, GO, Dart, así como también SDKs cómo es el caso de Flutter, aunque también se podrían incluir otros como Angular, React, Vue.js.

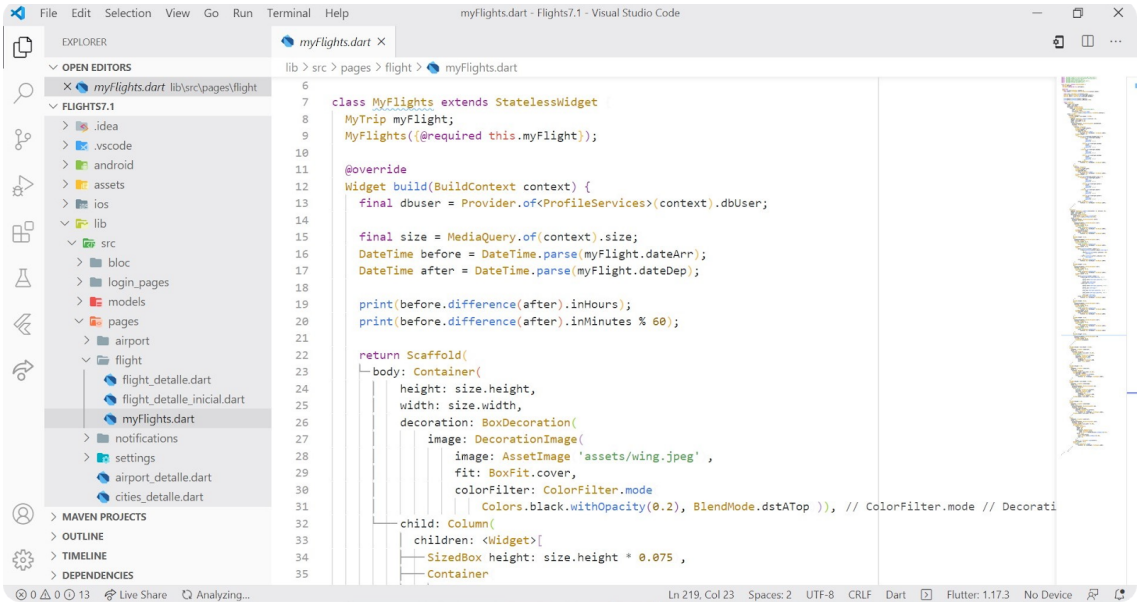



**Figura 21.** Captura del editor de código VSC

La interfaz de usuario es muy sencilla y parecida a otros editores de código como Sublime Text, Atom o el mismo Visual Studio tradicional. Se compone de una barra lateral, un editor, una barra de estado y un panel inferior.

**Barra lateral:** se encuentra a la izquierda como se puede observar en la figura 21, en este apartado encontramos una sección para el manejo de los archivos, un buscador, un *Source Control* de las versiones, un *debug* que se encarga de

compilar el código, extensiones que se pueden añadir al **IDE**. Los tres últimos corresponden a instalaciones realizadas por nosotros, en ellas encontramos un apartado para testeo (crear *Scripts* que comprueben ciertos aspectos de nuestro código), el **SDK** de **Flutter** (nos permite rápidamente obtener una visión del árbol de *widgets* del archivo que tengamos abierto), y por último el *Live Share* que permite compartir código como si de un documento de Google Drive se tratara.

**El Editor:** Se encuentra en la parte central y es el más importante, puesto que es el apartado más empleado del IDE. Aquí es donde se editan los archivos a través de código. Al abrirlo se despliega el código correspondiente al archivo, pudiendo editar o eliminar parte de él.

**Barra de Estado:** Se encuentra en la parte inferior del editor. Aparece información útil, como el número de líneas de código y el número de caracteres que conforman el archivo.

**Panel Inferior:** Desde la barra de estado se puede desplegar un panel donde se muestran otros paneles que recogen información sobre la depuración, errores y avisos. También disponemos de una consola y una terminal, ambas integradas en el IDE, que permiten ejecutar *Scripts,* aplicaciones o programas sin la necesidad de abrir una ventana de comandos, navegar hasta el directorio en el que se encuentren y allí ejecutar el archivo deseado.

Una característica muy interesante de este IDE, es que se puede modificar la disposición de estas áreas mencionadas anteriormente, es decir se puede adaptar a los gustos de cada uno, pudiendo aumentar el espacio de un área en concreto o cambiarla de posición, e incluso eliminarla del área de trabajo, reduciendo así el contenido visual del editor, permitiéndonos centrar nuestra atención en aquellas tareas que queremos realizar.

Si combinamos *ctrl + shift + p* accedemos a opciones rápidas del IDE. Cuando el archivo es muy grande y dispone de muchas líneas de código aparece un minimapa que se encuentra en la parte superior derecha del *framework*. Este te permite obtener una idea de la magnitud del archivo y navegar de forma rápida e intuitiva a través del código, pudiendo encontrar errores de forma visual.

### 4.3.4. ATOM

Cómo ya hemos mencionado, Atom es también un IDE gratuïto que facilita la integración con GitHub. Pues permite sincronizar una cuenta de dicha plataforma y que los cambios que se produzcan en el editor se cuelguen de forma automática. Algo muy útil si se está trabajando en distintas líneas de desarrollo y se quiere acceder a ellas o consultar la última versión disponible.

A continuación mostramos el IDE Atom. Cómo podemos observar su estructura resulta muy parecida a VSC, por lo que no entraremos a detallar los elementos que lo conforman.

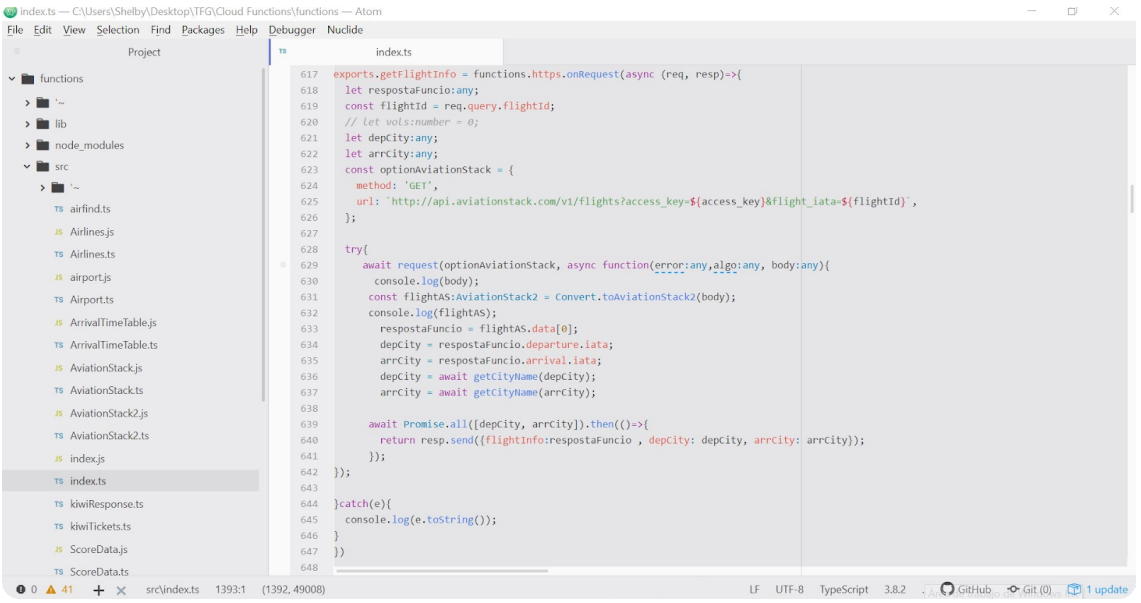



**Figura 22.** Captura del editor de código Atom

Finalmente comentar que este editor de texto fue creado por la conocida plataforma de GitHub y se puede emplear en Windows, Linux y MacOs.

Soporta lenguajes como: C/C++, C#, XML, SQL, JSON, JavaScript, Python, entre muchos otros, que se pueden añadir a través de SDKs, como por ejemplo Dart.

# 5. Metodología

En este apartado comentaremos las decisiones estratégicas en relación a la creación de la aplicación.

A través de la explicación metodológica podremos establecer la relación entre el marco conceptual y el desarrollo de la *App*. Inicialmente realizamos estudios exploratorios, para conocer e introducirnos en la temática, las distintas posibilidades, herramientas, recursos, etc.

A medida en que íbamos acotando el camino, nuestros estudios se convirtieron en una mezcla entre exploratorios y experimentales, testeando las posibilidades que ofrecía cada tecnología *(herramienta, framework, SDK, etc)*, aspectos a considerar, ventajas y contras de cada una de ellas.

Llegados a este punto nuestros estudios eran más analíticos, cuantas entradas existen en los portales de dudas, existencia o no de documentación, utilidad, limitaciones, etc. De la misma forma debíamos comprobar que realmente éramos capaces de utilizar las herramientas y entender su funcionamiento, para poder así alcanzar las metas propuestas.

Pese a intentar dejar constancia de muchos de los estudios realizados, algunos de ellos no han perdurado en el tiempo debido a la ausencia del mismo. Puesto que nos adentrabamos en terrenos desconocidos, y lo realmente era importante para nosotros, era ser capaces de crear y presentar un prototipo final, útil y tangible, dejar constancia de todas y cada una de las decisiones y conceptos aprendidos no era viable pues muchos eran aprendidos en vano, por lo que al desarrollo de la *App* respecta.

Debido a la profundidad y alcance de toda la tecnología empleada, tan sólo explicaremos aquellas características indispensables para comprender el trabajo realizado. Puesto que por sí solas pudieran llegar a ser dignas de su propio TFG, ya sea por nivel de complejidad como herramientas, frameworks o recursos.

En este apartado citaremos y detallaremos las APIs y lenguajes empleados, explicaremos y mostraremos ejemplos del modelaje de datos y finalmente presentaremos la planificación realizada para el proyecto.

## 5.1. APIs empleadas

La información con la que cuenta nuestra *App* se puede entender a partir de la siguiente tabla, donde se listan y resumen todas las APIs empleadas.

| Proveedor | Servicio | Resumen | *Requests (gratis)* |
|---|---|---|---|
| **Aviation Stack** | Información vuelo | Dado un ID nos suministra información relativa al vuelo, como *hora de llegada y salida*, *fecha, estado, terminal, etc.* | 500 *mensuales* |
| | Vuelos llegadas | Dado el código IATA del aeropuerto nos facilita las llegadas de este. | |
| | Vuelos salidas | Dado el código IATA del aeropuerto nos facilita las salidas de este. | |
| | Información Aerolíneas | Dado el código IATA de una aerolínea nos facilita información como *tamaño flota, fecha de creación, sede, etc.* | |
| **AirportInfo** | Información Aeropuerto | Dado el código IATA de un aeropuerto nos suministra información como *web, latitud, longitud, teléfono, etc.* | *ilimitadas* |
| **Teleport** | Información Ciudad | Dado el nombre de una ciudad, nos facilita, *país, población, moneda, descripción etc.* | *ilimitadas* |
| | Caracteristicas Ciudad | Dado el nombre de una ciudad nos provee un valor acorde a ciertas características de la ciudad, como *contaminación, aceso a internet, habitaje, etc.* | |
| **Pexels** | Imagenes Ciudad | Dado el nombre de una ciudad, nos facilita imágenes de esta. | 20.000 *mensuales* |
| **Kiwi** | Billetes de vuelo | Dado un *ORG*, un *DEST* y una fecha nos facilita el acceso a *precio, disponibilidad, link de compra, etc.* | *ilimitadas* |

**Tabla 8.** Resumen de APIs empleadas

A partir de la información/servicio que nos proporcionan los distintos proveedores, listados en la tabla anterior, nos han permitido crear *Cloud Functions* para adaptar la información suministrada a nuestras necesidades, creando así nuestros propios objetos.

Estas funciones *(CF)* podrían considerarse como nuestra propia API, puesto que funcionan como *end-points*, a los que nuestra aplicación pregunta y obtiene una respuesta. De hecho para comunicarnos con las APIs externas, al usar estas, API keys que son sensibles de ser mal empleadas por usuarios "maliciosos" decidimos que serían las CF las encargadas de ocultar dichos valores sensibles. A continuación se muestra una tabla donde aparecen las CF confeccionadas según el proveedor.

| Proveedor | CF | *Input* | *End-Point* | Anexo |
|---|---|---|---|---|
| Aviation Stack | getFlightInfo | *String* FlightID | https://cloudfunctions.net/getFlightInfo?flightId=KL1958 | **12.1.1** |
| | getArrival TimeTable | *String* Airport | https://cloudfunctions.net/getArrivalTimeTable?airport=BCN | **12.1.2** |
| | getDepartre TimeTable | *String* Airport | https://cloudfunctions.net/getDepartureTimeTable?airport=BCN | **12.1.2** |
| | getAirlineInfo | *String* Airline | https://cloudfunctions.net/getAirlineInfo?airline=Vueling | **12.1.3** |
| AirportInfo | getAirportInfo | *String* Airport | https://cloudfunctions.net/getAirportInfo?airport=BCN | **12.1.4** |
| Teleport | getCityData | *String* City | https://cloudfunctions.net/getCityData?city=New+York | **12.1.5** |
| kiwi | getKiwiFlight Tickets | *String* orig, *String* dest, *String* date | https://cloudfunctions.net/getKiwiFlightTickets?orig=BCN&dest=BIO&day=07&month=07&year=2020 | **12.1.6** |

**Tabla 9.** Resumen CF empleadas

Con la creación de las CF de la tabla 9, no sólo conseguimos *end-points* empleables para cualquier aplicación, sino que también aseguramos nuestro código de forma que el producto/servicio final sea aún más consistente y robusto, previendo futuros problemas.

Tras analizar las APIs externas que usamos en nuestro código y ver las distintas respuestas que somos capaces de generar, obtenemos una imagen más clara de qué tipo de información disponemos en nuestra aplicación. De la misma manera también podemos entender la importancia que suponen las APIs para una aplicación, puesto que sin ellas no podríamos brindar estos servicios a nuestros usuarios.

Finalmente destacar que algunas APIs, como hemos visto en la tabla 8, requieren suscribirse a un plan de pago para poder así, aumentar el número de *request*, puesto que la *App* puede y debe escalar en función de sus necesidades.

## 5.2. Lenguajes empleados

Nosotros como estudiantes hemos aprendido POO. En concreto los lenguajes que más hemos utilizado han sido C y C++. Aunque aprovechando la oportunidad que se nos brindaba de realizar un trabajo durante todo un semestre. Tras analizar todos los lenguajes existentes en la fase previa al trabajo, encontramos un lenguaje, Dart, que nos posibilitaba el desarrollo multiplataforma.

Para la confección de las CF se nos exigía el uso de JavaScript o TypeScript. En cuyo caso, investigamos ambos y no tardamos en darnos cuenta de que la cuestión no era uno u otro, sino elegir entre un lenguaje estático (TypeScript) o dinámico (JavaScript). Al disponer de cierta experiencia con lenguajes estáticos, decidimos emplear TS.

Para este proyecto, queríamos aprovechar las ventajas que nos ofrecía Dart, aunque para ser completamente sinceros, la elección vino derivada de la necesidad de emplear dicho lenguaje, para hacer uso del SDK de Flutter.

A continuación se muestra una tabla resumen de los lenguajes empleados:

| Lenguaje | Características | IDE | Anexo |
|---|---|---|---|
| Dart | - Lenguaje dinámico<br>- POO<br>- Código abierto<br>- Estructurado<br>- Flexible<br>- Con herencia simple<br>- Soporte de interfaces<br>- Tipado opcional de datos | *VSC* | **12.2.1** |
| TypeScript | - Lenguaje estático<br>- POO<br>- Permite incluir<br>- Node.js<br>- Compilado *.js*<br>- Inyección dependencias<br>- Código Abierto | *ATOM* | **12.2.2** |

**Tabla 10.** Lenguajes utilizados e IDEs

## 5.3. Modelado de Datos

El modelado de ***UFly*** puede dividirse según el apartado que estemos contemplado. En la siguiente tabla podemos apreciar de los distintos apartados, los principales objetos. También concretamos en que sección del anexo encontrar un ejemplo para la creación de un modelo de cada apartado.

| Apartado | Objetos principales | Anexo |
|---|---|---|
| BD Firestore | Subscriptions | **12.3.1** |
| | Topics | **12.3.2** |
| | User | **12.3.3** |
| | AllUsers | **12.3.4** |
| Cloud Functions | **User** | **12.4.1** |
| | AviationStack2 | |
| | Airport | |
| | ScoreData | |
| | KiwiTicket | |
| Source Code | **dbUser** | **12.5.1** |
| | FlightData | |
| | OurAirport | |
| | CityInfo | |
| | KiwiTickets | |
| BD Dispositivo | **GoSoon** | **12.5.2** |
| | MyTrips | |

**Tabla 11.** Modelado de datos Firestore *UFly*

El modelado de datos inicial, fue pensado para disponer de 2 objetos básicos de consulta, aeropuertos y usuarios. Tras confeccionar varios *Scripts* con los que manipular aeropuertos de distintas APIs conseguimos diseñar uno *(anexo, apartado 12.6.1 fireAirports.ts)* capaz de incorporar más de 6.000 aeropuertos. Incluimos todos en un único documento, para poder ahorrar *reads* en la facturación de Firebase.

Aunque nos dimos cuenta que Firestore no funcionaba bien, puesto que no le resultaba fácil cargar tal cantidad de datos, por lo que trazamos otra estratégia.

Para solventar dicho problema, convertimos los 6.000 aeropuertos en un JSON con 6.000 objetos (aeropuertos) contenidos en un *Array*. Como este JSON pasaría a formar parte de nuestro *Source Code,* y no queríamos disponer de toda la información, puesto que con ciertos parámetros teníamos suficiente, decidimos crear una guía para poder usarla a modo de sugerencia para el usuario.

A continuación se muestran dos elementos contenidos en ***allAirports.json***.


```
[{"Anadyr":{"cityName":"Anadyr","iata":"DYR","firebase":"airports1"}},
{"Bintulu":{"cityName":"Bintulu","iata":"BTU","firebase":"airports1"}}]
```


Como podemos observar la estrategia era simple, crear una guía que nos permitiera obtener de forma rápida el código IATA del aeropuerto, así como el nombre de la ciudad en la que se encuentra. Y en caso de querer ampliar dicha información, guardar la ubicación del objeto aeropuerto a través del campo firebase. Puesto que el JSON es sumamente extenso, tan solo hemos incluido las dos primeras entradas a modo de resumen.

Comentar que para poder manipular los objetos que obtenemos de las distintas APIs, debíamos crear un objeto adaptado a la respuesta para poder acceder a los campos de cada una de estas. Si observamos el anexo, apartado 12.4.2.1. *Index.ts,* nos daremos cuenta que existen un conjunto de funciones al inicio de dicho fichero que apelan todas al mismo lexema, convertir, *(p.e. Conversion, Convert, etc.),* estas funciones realizan el cambio del *body* de la *request* a objetos interpretables por las CF de forma que nos permitan su manipulación.

El *Source Code*, se encarga de interpretar, manipular y lo más importante de todo, mostrar la información procesada al usuario final.

Estas conversiones las podemos encontrar en la carpeta ***lib > models*** aquí se estipulan todos los modelos/classes que se emplean en el código fuente.

Finalmente comentar que en el *source code* también hemos creado classes que atienden a un servicio local, a través de un package *(sqflite)* con el cual se realizan operaciones de guardado y eliminación de una BD local. El lenguaje que se emplea para manipularlos *(guardar/eliminar)* es SQL. Podemos encontrar el código correspondiente de la clase `GoSoon` en el anexo, apartado 12.5.2 Modelo GoSoon.

Para concluir con el modelado de datos, destacar que todos los objetos han ido evolucionando en función de nuestras necesidades. También han sido modificados debido a problemas con APIs derivados de la pandemia global del covid-19.

## 5.4. Planificación temporal

Para poder alcanzar todos los objetivos y metas de este proyecto, la planificación ha sido un aspecto crucial para el correcto desempeño de este TFG.

Para poder mostrarlo de forma visual, hemos confeccionado dos diagramas que ejemplifican nuestra planificación temporal.

El primero, muestra a grandes rasgos el desarrollo del trabajo, en cambio el segundo, pretende mostrar con mayor exactitud cada una de las tareas llevadas a cabo.


**Figura 23.** Diagrama Global Gantt

De la figura anterior podemos observar que hay tareas que aparecen de forma esporádica, como p.e. *testeo*, pues se realizó al final de cada una de las fases de programación y toma de decisiones. A finales de Marzo se observa un testeo, eso es debido a que testeamos las herramientas y conceptos que íbamos a aplicar.

Cómo podemos ver, el desarrollo de la parte práctica finaliza semanas antes de la entrega del trabajo, aunque en este diagrama no aparecen las horas dedicadas a cada una de las macro tareas, la dedicación era cuasi completa. Puesto que nos dedicamos a tiempo completo al desarrollo de este proyecto.

Ahora procederemos a presentar un diagrama de *Gantt* más detallado, en el que no sólo aparecen las tareas realizadas, sino que aprovecharemos también para generar una idea de la distribución de las cargas de trabajo.

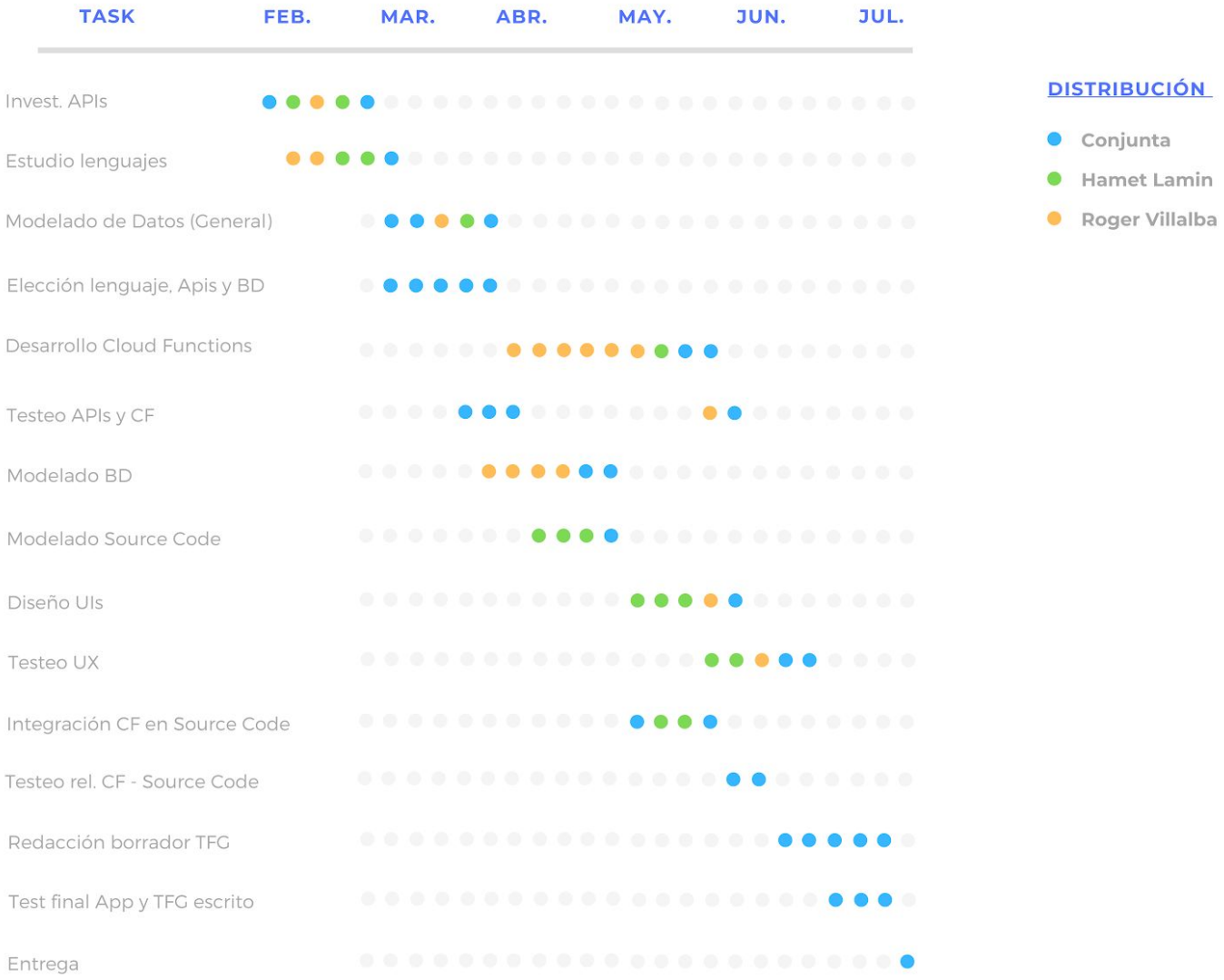



**Figura 24.** Diagrama Gantt con distribución cargas de trabajo

Como se puede ver la distribución de trabajo ha sido muy equitativa, esto es debido a la buena compenetración que comentamos en la introducción del trabajo entre ambos integrantes del equipo.

Tal y como se puede apreciar las tareas solían empezar de forma conjunta, aunque cuando ya estábamos en fase de desarrollo, existe una mayor especialización y algunas tareas, tal como muestra el diagrama empiezan con un único integrante, aunque algo que siempre hemos hecho es acabar las tareas de forma conjunta, no sólo para mostrar los avances realizados sino para ver si existía posibilidad de mejora, o de hacer las cosas de forma distinta o más eficiente.

Finalmente recalcar que las horas dedicadas a este proyecto, superan las 1.000 horas. Debemos apuntar a que es un TFG conjunto y que nos adentramos en un

terreno inicialmente desconocido, nuevos lenguajes, nuevos *frameworks*, mucha investigación para su posterior implementación, mucho ensayo y error, explicarnos conceptos, desarrollar nuevas ideas, etc. Cómo hemos comentado, esta cifra que puede parecer muy elevada. Ha sido extraída a través de un cálculo simple, puesto que no es de exagerar que ha habido días en los que se le hemos dedicado más de 16 horas. Un causante de esto ha sido el estado de alarma que hemos vivido de marzo a junio por la pandemia del Covid-19.

Aunque el volumen de trabajo ha sido elevado, la realización de todas y cada una de las tareas planteadas inicialmente, es debido a una buena planificación inicial y a lo largo del proyecto. Así como también una buena gestión, de una gran colaboración, motivación y ganas de aprender.

# 6. Desarrollo

En este apartado vamos a detallar los elementos, aspectos, funciones y servicios confeccionados para poder presentar ***UFly***.

Como ya hemos comentado, un aspecto clave para llevar a cabo el proyecto, fue la especialización de cada uno de los bloques que se muestran en la siguiente tabla.

| Bloque | Area | Tarea | Anexo |
|---|---|---|---|
| Front-end | Arquitectura Source Code | Modelado de datos en el código fuente | **12.5.1** |
| | UI/UX | Creación de pantallas y diseño *App* | **Fig. 31** |
| | *State Management* | Implementar la form correcta de gestión del estado y comunicación entre pantallas. | **12.5.3** |
| Back-end | Arquitectura BD | Modelado de datos CF, para almacenar en BD. | **12.3** |
| | *Cloud Functions* | Desarrollar todos los *endpoints (propios y con terceros)*, para dar funcionalidad a la *App*. | **12.4.2.2** |
| | Sistema de Notificaciones | Crear un sistema consistente que permita alcanzar la propuesta de valor de nuestra *App*. | **12.4.2.23** |

**Tabla 12.** Visión general del desarrollo por bloques

Para poder comprender el desarrollo del proyecto debemos entender las relaciones que existen entre las distintas áreas, puesto que hacerlo con total detalle dificultaría la comprensión, hemos creído conveniente agrupar el desarrollo de la *App* por bloques, de esta forma obtenemos una imagen global del servicio final creado, sin tener que entrar a detallar todas las relaciones que existen entre todos los puntos de nuestro front/back-end.

## 6.1. Front-end

### 6.1.1. Arquitectura

Para entender de una forma sencilla qué es y que supone una arquitectura en el desarrollo de aplicaciones, podemos resumir su definición, como la estructuración de todos los componentes de la interfaz de usuario para hacer que esta sea reusable, modular, predecible y escalable. Desdel equipo ***UFly*** creemos:

*Te conviertes en un verdadero desarrollador, cuándo los demás entienden tu código*

En nuestro caso, la arquitectura concebida no es de las más avanzadas, aunque desde el primer día hemos considerado este factor De no haber sido así, no se habría podido sacar el proyecto adelante.

Es recomendable estructurar cualquier proyecto, ya que se acostuma a trabajar en equipos y por tanto como mejor estructurado esté el código, más rápido va a ser el desarrollo, y más fácil va ser para los integrantes del proyecto ensamblar todas las piezas que lo conforman.

A medida que el proyecto crece, encontrar un archivo en concreto se hace cada vez más complicado. Una premisa que hemos definido, entre los miembros del equipo, es la agrupación de los archivos según su temática en distintas carpetas.

A continuación mostramos las distintas carpetas que conforman la *App*.

Comentarios :

- El 90% del código de ***UFly*** se encuentra en la carpeta ***lib***.
- En la directorio ***lib > src*** se encuentran todos las carpetas creadas para la aplicación. Cada una de ellas contiene la información que indica su título.
- También existen sub-carpetas que permiten organizar mejor los archivos de la *App*.
- También disponemos de una carpeta ***assets*** que almacena recursos para la aplicación, como imágenes, JSONs, etc

| Carpetas | | | Ficheros |
|---|---|---|---|
| */lib* | | | main.dart |
| /src | /bloc | | - pendent_bloc.dart<br>- people_bloc.dart<br>- profile_bloc.dart |
| | /login_pages | | landing_page.dart<br>- login_page.dart<br>platform_alert_dialog.dart<br>- platform_exceptions.dart<br>- validators.dart |
| | /models | | - airport_detalle_model.dart<br>- airport_model.dart<br>- avatar_reference.dart<br>- category_home.dart<br>- city_model.dart<br>- dbuser_model.dart<br>- flight_model.dart<br>- people_profile.dart<br>- search_user_model.dart<br>- ticket_model.dart |
| | /pages | | - airport_detalle.dart<br>- cities_detalle.dart<br>- home_page.dart<br>- Pendent_To Accept.dart<br>- people_profile.dart<br>- profile_page.dart<br>- ProfileEditMenu_page.dart<br>- search_page.dart |
| | */pages* | /airport | - airport_infrastructure.dart |
| | | /flight | - flight_detalle.dart<br>- flight_detalle_inicial.dart<br>- myFlights.dart |
| | | /notifications | - add_notification_page.dart<br>- mensaje_notification_page.dart<br>- notifications_page.dart |
| | | /settings | - credit_card.dart<br>- information_page.dart |
| | /services | | - airports_service.dart<br>- auth_service.dart<br>- cities_provider.dart<br>- database.dart |

| | | | |
|---|---|---|---|
| | | | - home_services.dart<br>- profile_services.dart<br>- search_provider.dart<br>- ticket_provider.dart |
| | /services | /firebase_storage_service | - build_user_img.dart<br>- firebase_storage.dart<br>- firestore_path.dart<br>- firestore_service.dart<br>- image_picker_Service.dart |
| | | /local_DB | - goSoon_db.dart<br>- trip_db.dart |
| | | push_notifications | - create_topic_Service.dart<br>- push_notifications_provider.dart<br>- topics_services.dart |
| | /utils | | - platform_widget.dart |
| | /widgets | | - airbnb_body.dart<br>- airports_body.dart<br>- airports_body.dart<br>- avatar_widget.dart<br>- CardsTickets_Ida.dart<br>- cities_body.dart<br>- flights_body.dart<br>- flip_card_widget.dart<br>- resultados_tickets.dart<br>- search_tickets_widget.dart<br>- search_user_delegate.dart |
| | /widgets | /calendar | - calendar_pop_up_view.dart<br>- custom_calendar_view.dart |

**Tabla 13.** Carpetas y archivos creados para *UFly*

Como podemos observar, nuestro *Source Code* está compuesto por una gran cantidad de archivos *.dart* creados para la confección de una *App* multiplataforma con todas las funcionalidades y requisitos de diseño presentados en el alcance de este TFG.

### 6.1.2. UI

La interfaz de usuario es el medio con que el usuario puede comunicarse con ***UFly***. Podemos pues imaginar que los conceptos de diseño que queramos implementar estarán vinculados con la confección de todas las UIs.

El principal objetivo de nuestra UI es comunicarse con los usuarios siendo esta el intermediario. En resumen es un medio de generar, comunicar, recolectar y transmitir información.

Para poder confeccionar una aplicación con un diseño coherente y atractivo para los usuarios de ***UFly*** debíamos tener presente ciertos aspectos para lograr dicha coherencia. Pudiendo así presentar un diseño más robusto y hacer énfasis en el *branding* de nuestra marca.

A continuación se muestran los aspecto que hemos contemplado:

| Aspectos | Resumen | Implementación | Ejemplo |
|---|---|---|---|
| **Paleta colores** | Este aspecto comprende la selección de colores para los distintos *widgets* y pantallas | - Fondo App<br>- Color texto<br>- Color botones | **Fig. 31** |
| **Margenes** | Un aspecto que debíamos contemplar para que los *widgets* quedarán separados de forma limpia | - Widgets<br>- Cards<br>- Botones<br>- BorderRadius | **Fig. 32** |
| **Navegación** | Permitir al usuario navegar de forma fácil e intuitiva a través de las distintas UIs | - BottomNavBar<br>- OnTap<br>- MaterialPageRoute | **Fig. 36** |
| ***Responsive*** | *App* adaptable a cualquier dispositivo independientemente del tamaño de su pantalla. | - MediaQuery<br>- Size.height<br>- Size.width<br>- SafeArea | **Fig. 50** |

**Tabla 14.** Consideraciones *UI*

### 6.1.3. UX

La UX o experiencia de usuario se puede definir como el conjunto de elementos relativos a la interacción del usuario con una *App*, dando como resultado una percepción positiva o negativa de dicho producto/servicio. También se puede definir como la vivencia que experimentan los usuarios al navegar, usar o testear la aplicación.

Este apartado está vinculado con el anterior (UI), podemos decir que la UI influye en gran medida en la experiencia del usuario. Ambas son importantes y se deben de tener en cuenta durante el desarrollo.

El resultado de la UI, influye de manera directa a una experiencia de usuario positiva o negativa. Podemos adelantar que disponer de una UX negativa es uno de los factores que más echa para atrás a los usuarios, generando así una gran cantidad de desinstalaciones. En nuestro caso, hemos sido precavidos y hemos considerado este aspecto en todo momento, ya que sino se cuida desde un inicio, cuesta mucho de corregir.

La estrategia que hemos empleado para obtener una buena UX, es centrar el diseño bajo la premisa de que el usuario no sabe qué hacer en la pantalla en la que se encuentra.

Hemos realizado pruebas y tests constantemente en cada sección, para encontrar *bugs* y errores que por pequeños que fueran, puede suponer la diferencia entre el éxito y el fracaso.

Para mostrar tanto la UI como una aproximación de la UX y ejemplificar todos los conceptos comentados en este apartado, la figura 25 muestra una pantalla de ***UFly*** que lo ejemplifica.

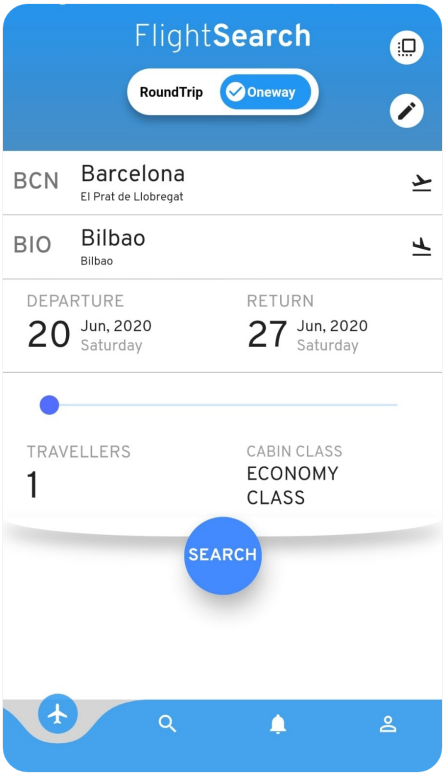



**Figura 25.** UFly - *FlightSearch*

### 6.1.4. Gestión del Estado

Al principio del apartado Front-end, hemos comentado cómo están estructurados los archivos, En este hablaremos de *State Management* conocida también com arquitectura, pero de código, distinta a la del proyecto y un elemento esencial para el funcionamiento de cualquier *App*.

En nuestro caso empezamos utilizando *widgets* con estado efímero, pero llegó un punto en el que ya no podíamos emplearlo, puesto que las funcionalidades que estábamos implementando debían compartir información entre diferentes *widgets*. Nos vimos forzados a realizar el salto que muestra la siguiente figura.

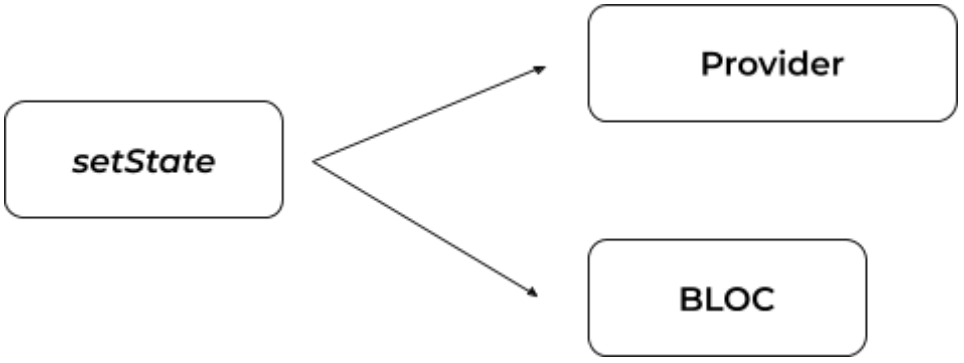



**Figura 26.** Evolución gestión estado *UFly*

El método *setState,* lo hemos utilizado en muchas ocasiones, sobre todo al principio, no es complicado de manejar y tampoco necesita de un *package* externo, pero a medida que iba creciendo el proyecto y el árbol de *widgets*, era muy complicado manejar tantos *widgets* que interactúan con otros a través de este método.

Además cuando empleamos el *setState*, Flutter interpreta que debe redibujar todo el árbol de widgets, esto es útil si nuestra aplicación fuera pequeña, una o dos páginas, pero si la *App* es grande, ya podemos ver que volver a procesarla para aplicar un cambio que afecta tan sólo a un *widget* no es la mejor forma de proceder. Es en este punto en el que aparece la necesidad de empezar a implementar nuevas técnicas como el Provider, un *package* que se instala en el proyecto. Este paquete nos permite manejar el estado de la aplicación sin tener que preocuparnos por las llamada de retorno o *InheritedWidgets*. Para emplearlo debemos comprender los conceptos que mostramos en la siguiente tabla.

| Conceptos | Descripción | Anexo |
|---|---|---|
| Change Notifier | Una clase que se encuentra en el SDK de *Flutter* y nos permite encapsular el estado de la *App*, proporcionando notificaciones de los cambios que se han realizado a sus *'listeners'*. Para *Apps* simples con uno o dos *ChangeNotifier/s* se puede manejar el estado, pero para aplicaciones más complejas que disponen de varios modelos, se requiere de varios *ChangeNotifiers*. | 12.2.5 |
| Change Notifier Provider | Proviene del paquete *Provider* y es un *widget* que proporciona una instancia de un *ChangeNotifier* a sus descendientes, es decir este se coloca siempre por encima de todos los *widgets* que necesitaran acceder a él, en nuestro caso la mayoría de los *ChangeNotifierProvider* creados los hemos colocado en lo más alto del árbol de *widgets*, *main.dart*. Evitando así posibles problemas en futuras implementaciones. Para poder llamar más de una instancia, se tiene que llamar primero al *Multiprovider* y dentro de este se colocan todos los *ChangeNotifierProvider*. | 12.2.6 |
| Consumer | Una vez creada la clase extendiendo *ChangeNotifier* y tenemos instanciado el *ChangeNotifierProvider* en el árbol de *widgets*. Podemos conocer los cambios que se generan en la clase pudiendo redibujar tan sólo ciertos *widgets* sin necesidad de dibujar de nuevo todo el árbol. Para ello debemos estar escuchando en todo momento a través del *widget* consumer *(*basado en *Streams)*. | 12.2.7 |

**Tabla 15.** Conceptos Provider

En nuestro caso para no estar declarando en cada *widget* un *consumer* y redibujar la pantalla entera cuándo muchas veces lo que cambia es un simple botón o un mensaje , lo que hemos hecho ha sido utilizar el *Provider.of*, para llamarlo desde cualquier *widget* y reconstruir sólo aquello que nos interesaba.

Cabe decir que ***Provider*** es la técnica de manejo de estado que más hemos utilizado en la *App*. Otra que también nos hemos visto obligados a emplear aunque no tanto como la anterior es el **BLOC**.

En nuestro caso hemos utilizado esta técnica en el manejo del perfil de los usuarios, para las solicitudes pendientes de aceptar por el usuario y también para los perfiles de otros usuarios. Podíamos haber empleado Provider, pero también queríamos experimentar y probar nuevas técnicas y empezar aprender BLOC, ya que es la recomendada por Google.

Tras ver, analizar y estudiar videos, cursos y documentaciones, fuimos capaces de crear un BLOC muy sencillo. En favor de la verdad, es cierto que acostumbrarse es tedioso, aunque una vez entiendes la idea, es muy práctico y aunque pueda parecer paradójico es sumamente sencillo. Además permite al desarrollador separar la lógica de la UI. Algo muy útil si se trabaja en equipos grandes donde existen por un lado los programadores de *back-end* y los diseñadores o programadores de *front-end*.

Para concluir con el apartado sobre la gestión del estado, enumeramos las distintas técnicas empleadas para la construcción de ***UFly***.

| Técnica | Resumen | Anexo |
|---|---|---|
| *setState* | Primera aproximación a la gestión de estado. Empleado tan sólo en *widgets* básicos | 12.2.8 |
| Provider | Empleado para transmitir información necesaria en distintas ramas del árbol de *widgets* | 12.2.9 |
| BLOC | Utilizado para generar cambios en la UI en función de eventos, como por ejemplo *fetchDataAPI* | 12.5.3 |

**Tabla 16.** Técnicas de estado empleadas

### 6.1.5. Árbol de *Widgets*

En nuestro caso, disponemos de un árbol de *widgets* muy grande y a decir verdad cuándo empezamos con el proyecto es algo que no teníamos ni idea de cual iba a ser su extensión. Tampoco es que nos supusiera un gran problema, pero si lo hubiéramos controlado desde el principio, hubiéramos podido reducirlo, haciendo que se reutilicen varios *widgets* con tan solo pequeñas modificaciones.

Otra cosa por lo que es importante controlar el árbol de *widgets*, es cuando se empieza a utilizar el Provider. Esto sí nos llegó a causar problemas ya que al instanciar algún ChangeNotifierProvider, lo hicimos por debajo de algunos *widgets* que lo iban a necesitar más adelante y al estar instanciado por debajo estos no lo podían consumir. Aquí fue donde tuvimos que investigar todo el tema del árbol de *widgets* y volver hacer la instancias de manera correcta.

### 6.1.6. Packages

A continuación se muestran todos los paquetes que hemos utilizado en el proyecto, como se puede observar, la tabla confeccionada comprende, el nombre del paquete, una breve descripción de su funcionalidad y la versión empleada en la aplicación final.

| Paquete | Descripción | Versión |
|---|---|---|
| `google_fonts` | Ofrece diferentes tipos de letra. | `^0.3.2` |
| `intl` | Ayuda a parsear la hora, según región. | `^0.16.1` |
| `path_provider` | Permite encontrar rutas habituales del OS. | `^1.6.11` |
| `cupertino_icons` | Ofrece una gran biblioteca de iconos. | `^0.1.2` |
| `provider` | Técnica para el manejo de estado. | `^4.1.3` |
| `curved_navigation_bar` | Utilizado para la navegación del menú. | `^0.3.2` |

| | | |
|---|---|---|
| flip_card | Utilizado para dar efecto al *HomePage* | ^0.4.4 |
| equatable | Simplifica establecer igualdad entre clases | ^1.1.1 |
| animate_do | Utilizado para animar algunos widgets | ^1.7.2 |
| http | Utilizado para hacer llamadas a *endpoints* | ^0.12.1 |
| percent_indicator | Utilizado en el diseño de indicadores de *City* | ^2.1.3 |
| flutter_slidable | Permite crear elementos deslizables | ^0.5.4 |
| firebase_auth | Posibilita la autentificación en la *App* | ^0.14.0+5 |
| google_sign_in | Registrarse e iniciar sesión con Google | ^4.4.4 |
| carousel_slider | Utilizado para la presentación de imágenes | ^2.1.0 |
| cloud_functions | Permite implementar llamadas .onCall() | ^0.4.2+3 |
| image_picker | Permite seleccionar la imagen de la galería | ^0.6.5+2 |
| firebase_storage | Permite subir contenido al Cloud Storage | ^3.1.5 |
| cloud_firestore | Permite realizar *reads*, *writes* y *deletes* | ^0.13.5 |
| firebase_messaging | Necesario para la implementación de las *Notificaciones* | ^6.0.13 |
| url_launcher | Necesario para abrir la página de compra | ^5.4.7 |
| flare_flutter | Utilizado para aplicar animaciones | ^2.0.3 |
| flutter_bloc | Necesario para implementar el BLOC | ^4.0.0 |
| flutter_map | Utilizado para implementar el mapa de *Airport* | ^0.9.0 |
| latlong | Necesario para implementar el mapa | ^0.6.1 |
| path_drawing | Utilizado para dibujar la UI del *Profile* | ^0.4.1 |
| sqflite | Necesario para la BD local | ^1.3.0+1 |

**Tabla 17.** Paquetes empleados en *UFly*

## 6.2. Back-end

Como ya hemos visto en el apartado front-end empleamos el SDK Flutter (Google), para el back-end también utilizaremos otras herramientas que ofrece esta empresa. A continuación analizamos aspectos de nuestro back-end como la arquitectura, las funciones *Cloud* diseñadas, agrupadas por temática.

### 6.2.1. Arquitectura BD

Llegados a este punto ya disponemos de una imagen global de todas las herramientas que conforman ***UFly***. En la siguiente figura se muestra un diagrama que pretende resumir y simplificar todas las relaciones que existen entre los distintos elementos que dan forma a la aplicación.

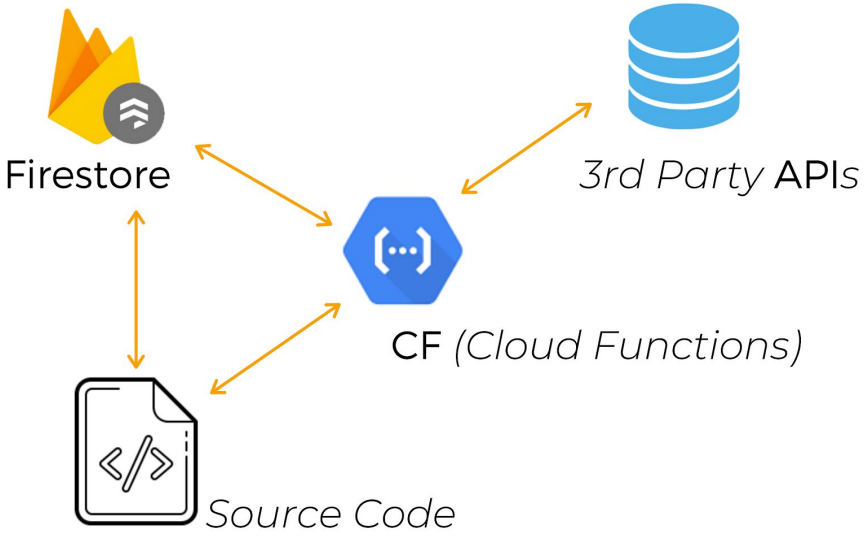



**Figura 27.** Arquitectura general *UFly*

Muchas veces como usuarios creemos que el núcleo de una plataforma es aquello que podemos ver, tocar e interactuar. El diagrama de la figura anterior deja claro cuál es el pilar en torno al cual ***UFly*** puede ofrece su servicio, las *Cloud Functions*. Pues son las encargadas de reclamar, procesar y transmitir la información, entre todos los componentes que dan forma a la *App*.

Destacar que gracias a las ***Firebase Security Rules*** podemos regular y restringir el acceso a ciertos documentos y/o colecciones. Para poder analizar las funciones que manejan la información de nuestra BD debemos presentar cual es la agrupación que hemos confeccionado para almacenar la información.

#### 6.2.1.1. Colecciones y Documentos

Como ya hemos introducido en el marco conceptual, *Firestore (BD UFly)*, se articula en función de documentos y colecciones. Para poder comprender la arquitectura, a continuación hemos creado una tabla en la que aparecen las colecciones con un breve resumen, el esquema empleado para el título de los documentos y un ejemplo en el que se puede apreciar cómo se ven en *Firebase Console (panel de control BD)*.

| Colecciones | Titulo Doc. | Resumen Col. | Anexo |
|---|---|---|---|
| **Subscriptions** | ORG-DEST-PRECIO-FECHA_INICIAL-FECHA_FINAL | Engloba todas las suscripciones creadas por los usuarios de la *App* | **12.3.1** |
| **Topics** | ORG-DEST-PRECIO-FECHA_INICIAL-FECHA_FINAL | Engloba todos los tópicos creados por los usuarios de la *App* | **12.3.2** |
| **users** | uid *(usuario)* | Engloba todos los usuarios de la *App* | **12.3.3** |
| **allUsers** | *UsersDoc* | Almacena una guía de consulta para encontrar usuarios en la *App* | **12.3.4** |

**Tabla 18.** Esquema Colecciones *UFly*

En la siguiente tabla mostramos un resumen de la información importante que debe almacenar cada documento, así como también un ejemplo en formato JSON, que nos permita ver cómo podríamos acceder a la información contenida en un *Document*.

| Document ID | Resumen Doc. | Anexo |
|---|---|---|
| ORG-DEST-PRECIO-FECHA_INICIAL-FECHA_FINAL *(subscriptions)* | Almacena el *fcmToken* y el *uid* de los usuarios suscritos a un tópico | 12.3.5 |
| ORG-DEST-PRECIO-FECHA_INICIAL-FECHA_FINAL *(topics)* | Almacena todos los mensajes en relación a un tópico. | 12.3.6 |
| uid *(usuario)* | Almacena toda la información relacionada con el usuario. | 12.3.7 |
| usersDoc | Almacena el *uid*, *nickName* y la *url* de la imágen para realizar un búsqueda rápida de usuarios | 12.3.8 |

**Tabla 19.** Esquema Documentos *UFly*

Para evitar incongruencias, duplicaciones o desestructuraciones hemos creado un seguido de CF que nos permitan de forma automática *(Automatic Triggers)* garantizar la coherencia de los datos y la estructura de nuestra información.

### 6.2.2. Funciones *Cloud*

Ya hemos visto que las CF son un pilar fundamental en el back-end de nuestra aplicación, a continuación procederemos al análisis de las distintas funciones. Las hemos agrupado según su finalidad.

| Finalidad | Resumen | Apartado |
|---|---|---|
| **Gestión APIs *(terceros)*** | Permiten consultar información externa, no incluida en *Firestore* | **5.1** |
| **Gestión Usuarios** | Nos permite añadir, modificar, eliminar, información respecto de un usuario *(uid)* | **6.2.2.1** |
| **Gestión Notificaciones** | Nos permite añadir o eliminar suscriptores a un tópico, crear o eliminar un tópico. También son las encargadas de entregar las notificaciones al usuario | **6.2.2.2** |

**Tabla 20.** Agrupación CF *UFly*

#### 6.2.2.1. Funciones *User*

Son todas aquellas funciones que crean, modifican o se encargan de la gestión de usuarios. También incluye la búsqueda de otros usuarios, modificar la imágen de perfil, obtener la lista de solicitudes de seguimiento, etc.

A continuación resumimos las funciones relacionadas con la gestión y tratamiento de usuarios en una tabla para poder suministrar una pequeña explicación de su funcionalidad y uso.

| CF | Resumen | Anexo |
|---|---|---|
| `onUserWelcome()` | `// AUTOTRIGGER`<br>Cuando un usuario se autentifica por primera vez *(Register)*. Esta nos permite crear el objeto usuario dentro de ***Collections > users.***<br>Permite saber el tipo de *auth (email o Google Account)*. | 12.4.2.2 |
| `onUserGoodbye()` | `// AUTOTRIGGER`<br>Cuando un usuario es eliminado, ya sea por voluntad propia, o lo eliminemos nosotros *(admins)*. | 12.4.2.3 |
| `onAvatarChanged()` | `// AUTOTRIGGER`<br>Cuando un objeto, en este caso siempre serán imágenes, se sube al Cloud Storage. Pudiendo así obtener la dirección *url* en la que almacenamos el recurso. También permite cambiar la imagen contenida en la lista de *allUsers*. | 12.4.2.4 |
| `searchUser`<br>`(String userToFind)` | `// ONCALL`<br>Recibe como parámetro el nombre del usuario *(nickName)* que se quiere buscar. Devuelve la información de la lista *allUsers*.<br><br>**Ej**. Anexo 12.4.3 *JSON response*. | 12.4.2.5 |
| `getUserProfileData`<br>`(String getUid)` | `// ONCALL`<br>Recibe como parámetro el *uid* del usuario del cual se desea obtener su perfil. | 12.4.2.6 |
| `getUserInfo`<br>`(String uid)` | `// ONCALL`<br>Recibe como parámetro el *uid* del usuario. Esta función permite al *Source Code* cosultar, cargar y mostrar la información del usuario que se ha solicitado. | 12.4.2.7 |

| | | |
|---|---|---|
| `updateUserInfo (String uid, String nickName, String userName, String frase, String phoneNumber, String userWeb)` | `// ONCALL` Modifica la información visible de un usuario. Aporta cierto nivel de personalización al servicio ***UFly***. | 12.4.2.8 |
| `sendDeviceToken (String tokenId)` | `// ONCALL` Permite saber en qué dispositivo se encuentra el usuario para poder notificarlo, en relación a un tópico. | 12.4.2.9 |
| `askForFriendship (String future FriendUid)` | `// ONCALL` Añade al usuario en cuestión, el *uid* del usuario que realiza la solicitud a su lista de pendentToAccept. Para que así cuando el usuario vea su lista de solicitudes pendientes, pueda aceptarla o declinarla. | 12.4.2.10 |
| `cancel AskForFriendship (String cancel Friend)` | `// ONCALL` Permite denegar una solicitud de amistad antes de que el otro usuario, pueda aceptar o declinar dicha solicitud. | 12.4.2.11 |
| `acceptFriend (String newFriend)` | `// ONCALL` Procesa todos los cambios que conlleva esta nueva relación. Añadir a *followers* el *uid* del usuario que se ha aceptado (`newFriend`), se elimina este de la lista pendentToAccept, y a su vez se añade en la lista de *following* del `newFriend.` | 12.4.2.12 |
| `declineFriend (String noFriend)` | `// ONCALL` Permite eliminar una solicitud de seguimiento. Debe realizar una operación similar a *acceptFriend()*, aunque ahora eliminando el uid de cada una de las respectivas listas. | 12.4.2.13 |

| | | |
|---|---|---|
| `deleteFriend (String noMore Friend)` | `// ONCALL` Permite eliminar una relación de seguimiento. Elimina `noMoreFriend` de la lista de *following* y el *uid* del usuario de la lista de *followers* de `noMoreFriend`. | 12.4.2.14 |
| `getUsers PendingToAccept (String uid)` | `// ONCALL` De esta forma podemos ver si existen o no nuevas solicitudes. | 12.4.2.15 |

**Tabla 21.** CF sistema gestión usuarios

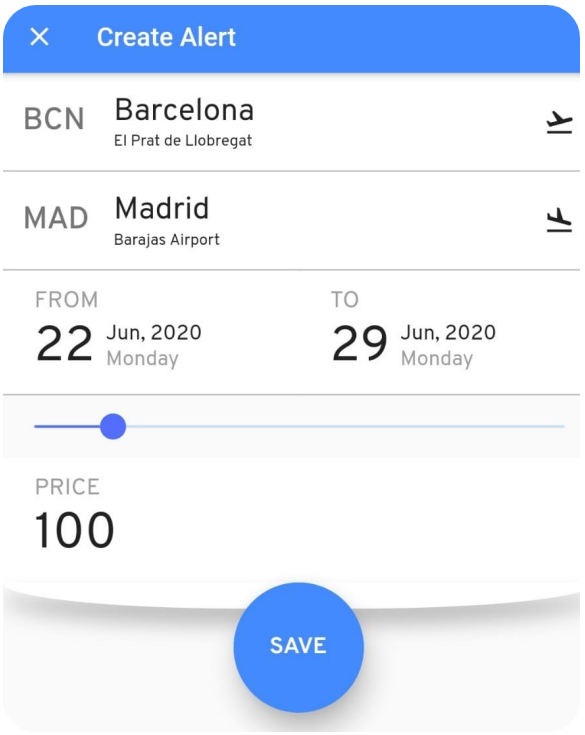
Gracias a este conjunto de CF creadas podemos ofrecer una experiencia social dentro de nuestra *App*.

Sin embargo estas constituyen tan sólo una base para poder ir añadiendo funcionalidades, con la finalidad de mejorar la experiencia. Algunos ejemplos de esta mejora serían: implementar un chat, un *feed*, la posibilidad de compartir ofertas entre seguidores, poder confeccionar paquetes de vuelos a través de las ofertas que actualmente se muestran, etc.

#### 6.2.2.2. Sistema de notificaciones

Ahora procederemos a explicar y detallar las las CF que conforman el sistema confeccionado para la gestión de notificaciones. A la hora de crear un sistema que notifique a los usuarios, debíamos considerar cuáles serían los parámetros a través de los cuáles este pudiera indicarnos sus intereses. Los parámetros que un usuario puede indicar para crear un tópico son los mostrados en la figura 28.

A continuación los listamos para poder comprender cuáles son los datos que debían ser recolectados por el *Source Code* para que luego las CF pudieran procesar los valores y crear un tópico o en su defecto subscribirse a uno existente.




Parámetros creación tópico:

- Origen *(IATA)*
- Destino *(IATA)*
- Fecha Inicio Intervalo
- Fecha final Intervalo
- Precio máximo

Cuando el usuario procede al guardado del tópico, ***UFly*** lanza una *request* a `setNotification()`.

Podemos encontrar esta función y todas las demás en la tabla 22.

**Figura 28.** Parámetros creación tópico

Antes de mostrar la tabla que contiene las CF que permiten a nuestra aplicación articular todas las operaciones necesarias para poder crear el servicio, debemos entender las bases en las que se fundamenta nuestro sistema.

Creamos dos colecciones *(Topics* y *Subscriptions)* en las cuales almacenamos la información necesaria para notificar a nuestros usuarios. Para la creación de ambas colecciones aplicamos el siguiente formato para poder operarlos y referenciarlos. El esquema empleado es el siguiente :

**ORIGEN** - **DESTINO** - **PRECIO** - **FECHA INICIO INTERVALO** - **FECHA FINAL INTERVALO**

A continuación incluimos un ejemplo de cómo nuestras CF interpretan un tópico/suscripción.

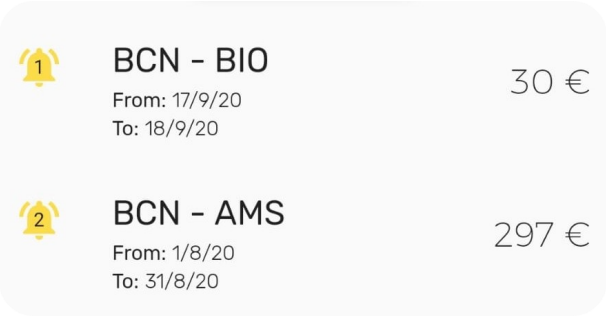
**BCN-LHR-75-10-10-2020-25-11-2020**

De esta forma podíamos rápidamente encontrar si existía o no, alguien que ya hubiera creado un tópico igual al requerido. Una vez se ha creado o encontrado un tópico, la CF *setNotification*(), procede a crear un documento dentro de la colección **subscriptions**, que nos permite almacenar la información del *tokenId* y *uid* del usuario que desea suscribirse, pudiendo saber quiénes deben ser notificados cuando se cumplan dichas condiciones. Finalmente la CF debe modificar el *Array subscriptionsRegister,* pudiendo así disponer de un registro de suscripciones a las que el usuario se ha suscrito.

Para simplificar la explicación del servicio confeccionado, mostramos la tabla que recoge todas las CF involucradas en el sistema .

| CF | Resumen | Anexo |
|---|---|---|
| `setNotification()` | `// ONREQUEST`<br>Permite crear un tópico en la colección **Topics** y suscribirse a él, en caso de que ya exista se suscribe al tópico existente. | 12.4.2.23 |
| `deleteNotification()` | `// ONREQUEST`<br>Cuando un usuario decide eliminar su suscripción a un tópico. Esta CF elimina *fcmToken* y *uid* de la colección **Subscriptions**, en caso de que no existan más suscripciones, elimina el documento *subscriptions* y su tópico correspondiente. | 12.4.2.24 |
| `deliverTopicMessage()` | `// AUTOTRIGGER`<br>Cuando se inserta un nuevo *msg* en un tópico, se dispara automáticamente, generando así las *push notifications*. | 12.4.2.25 |
| `decideToDeleteTopicSubscription()` | `// AUTOTRIGGER`<br>Evalúa si es necesario o no, eliminar un tópico en función del número de suscripciones. | 12.4.2.26 |

**Tabla 22.** CF Sistema de Notificaciones




Una vez el usuario ha creado un tópico, nuestra *App* muestra la información que aparece en la figura 29.

Indicando al usuario cuáles son los tópicos a los que está suscrito.

**Figura 29.** *Topics* creados

En el anexo, apartado 12.2.10. Ejemplo de Código *SubscriptionsRegister,* podemos observar un ejemplo de cómo y qué información almacenamos en la colección **users**.

Hasta el momento hemos visto cómo las CF manipulan el sistema de notificaciones. El siguiente aspecto que debemos tener en cuenta es cómo nuestra *App (Source Code)* gestiona la recepción de las notificaciones. En función de cómo se encuentre la aplicación, mostraremos un tipo de notificación u otra.

A continuación hemos creado una tabla que permite entender cuáles son las distintas formas en las que se gestionan las notificaciones.

| Metodología | Situación | Anexo | Ejemplo |
|---|---|---|---|
| **onMessage** | Cuando se está utilizando la *App*, este método es el encargado de mostrar la alerta. | 12.2.11 | Fig. 44 |
| **onResume** | Cuando el usuario dispone de la *App* inicializada, aunque en el momento en el que se produce la notificación *push,* la aplicación se encuentra en segundo plano. | 12.2.12 | Fig. 43 |
| **onLaunch** | La notificación se muestra cuando el usuario no tiene de la *App* inicializada. | 12.2.13 | Fig. 43 |

**Tabla 23.** Recepción *push notifications*

Ahora ya conocemos los pilares del sistema que dispone ***UFly*** para poder suministrar las notificaciones a nuestros usuarios.

Tan sólo nos falta entender cómo ***UFly*** es capaz de saber cuando un vuelo cumple las condiciones necesarias para ser enviado a nuestros usuarios. Para ello debemos atender a una función contenida dentro de la CF `getKiwiFlightTickets()`, disponible en el anexo, apartado 12.4.2.27.

En ella podremos observar cómo de cada consulta que se haga para un billete, esta CF analiza si existe algún tópico interesado en dicha información, a través de dos funciones, `getLowestPrice` y `publishMessage`.

Pudiendo así, obtener el menor precio del vuelo buscado y así confeccionar un *String* con el formato *(org-dest-precio-fecha)* y analizar si existe o no un tópico interesado en recibir esta información. En cuyo caso se iniciaría un *write* en la colección de **topics**, desencadenando así una série de *automatic triggers* que permitirían al usuario recibir las notificaciones de nuestra plataforma.

# 7. Uso de la aplicación

Finalizado el marco conceptual, funciones *Cloud* y otros aspectos a considerar para entender el funcionamiento de ***UFly***, estamos listos para presentar el producto final desarrollado. Para ello mostraremos y explicaremos a través de *screenshots*, la navegación y que es lo que el usuario puede hacer en cada una en las distintas pantallas de la aplicación.

La pantalla inicial es ***Register***. Existen varias opciones para acceder a la *App*, registrándose con: email y contraseña, cuenta de *Google* o bien acceder de forma anónima. En la siguiente figura se pueden ver todas estas opciones.

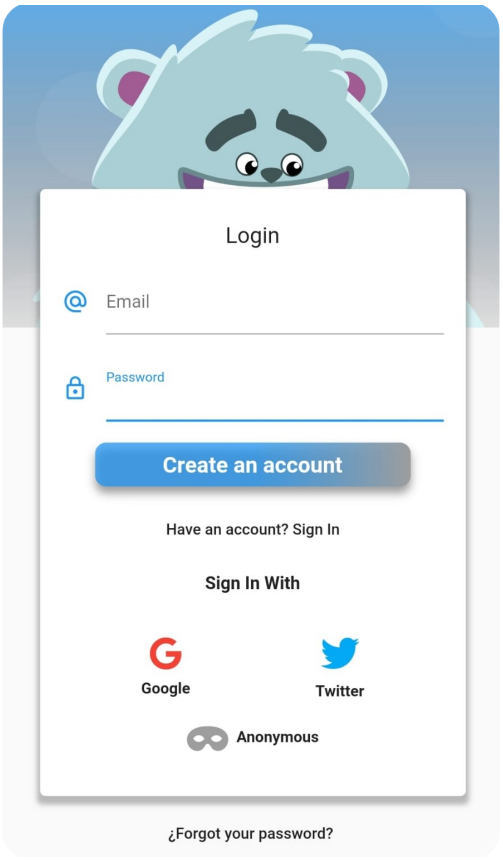



**Figura 30.** UFly - *RegisterPage*

Una vez el usuario se ha registrado o accedido a su cuenta, se encuentra con el siguiente menú principal. Dónde se dispone de una navegación inferior, esta cuenta con cuatro apartados, el que aparece abierto por defecto es el *Home*, en segundo lugar encontramos el *Search* de usuarios, seguido por *Notifications* y finalmente el *Profile*.

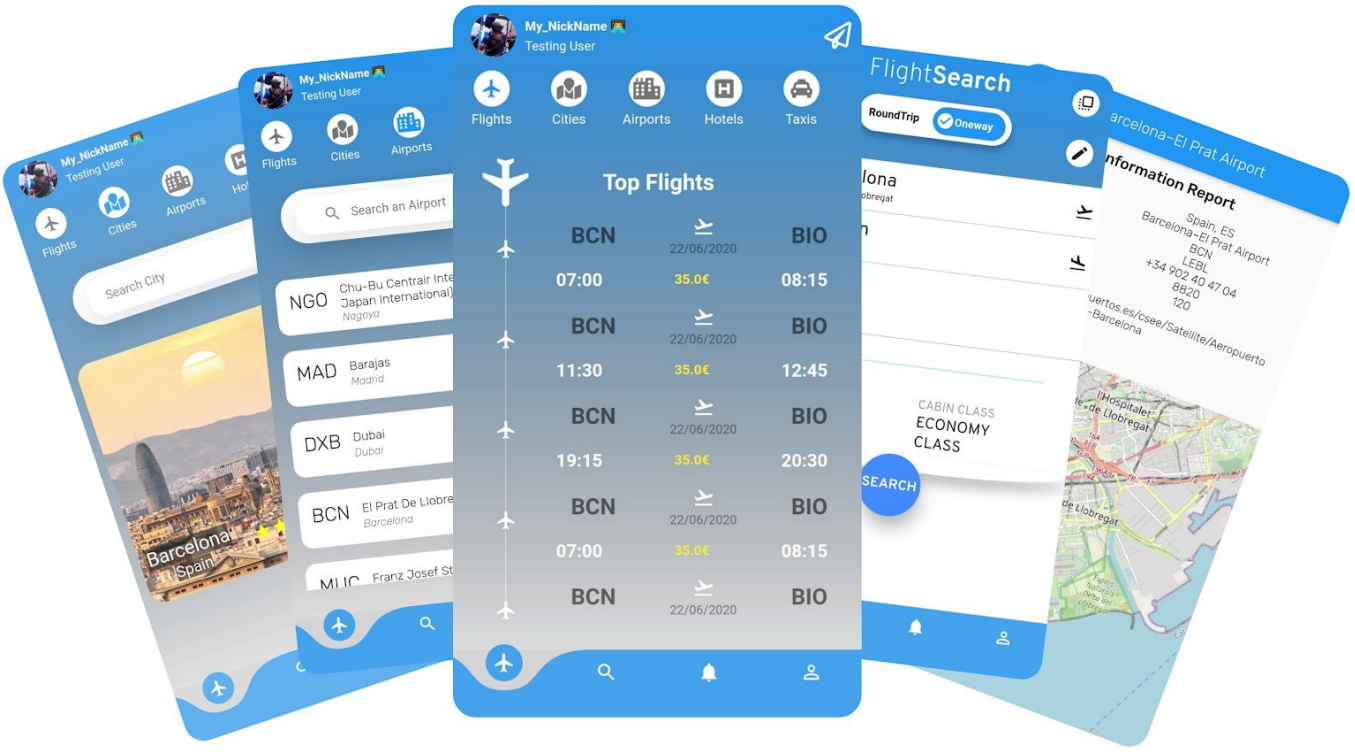



**Figura 31.** UFly - *HomePage*

En el ***HomePage***, nos aparecen una serie de vuelos con un origen y un destino predeterminados y su precio. Si se realiza la acción de clicar en uno de ellos, se produce la navegación hasta la página web de compra.

También se puede hacer un *scroll* para poder visualizar más ofertas. En la parte superior izquierda, encontramos el nombre y la imagen del usuario, si se pulsa nos llevan directamente al *Profile*. El icono del avión de papel nos permite navegar a la búsqueda de vuelos *(FlightSearch)*.

El *HomePage* contiene varios subapartados como *Flights*, *Cities*, *Airports*, *Hotels* y Taxis, estos se emplean para realizar una navegación dentro del *Home*, cada una integrada dentro de un marco común con distintas funcionalidades.

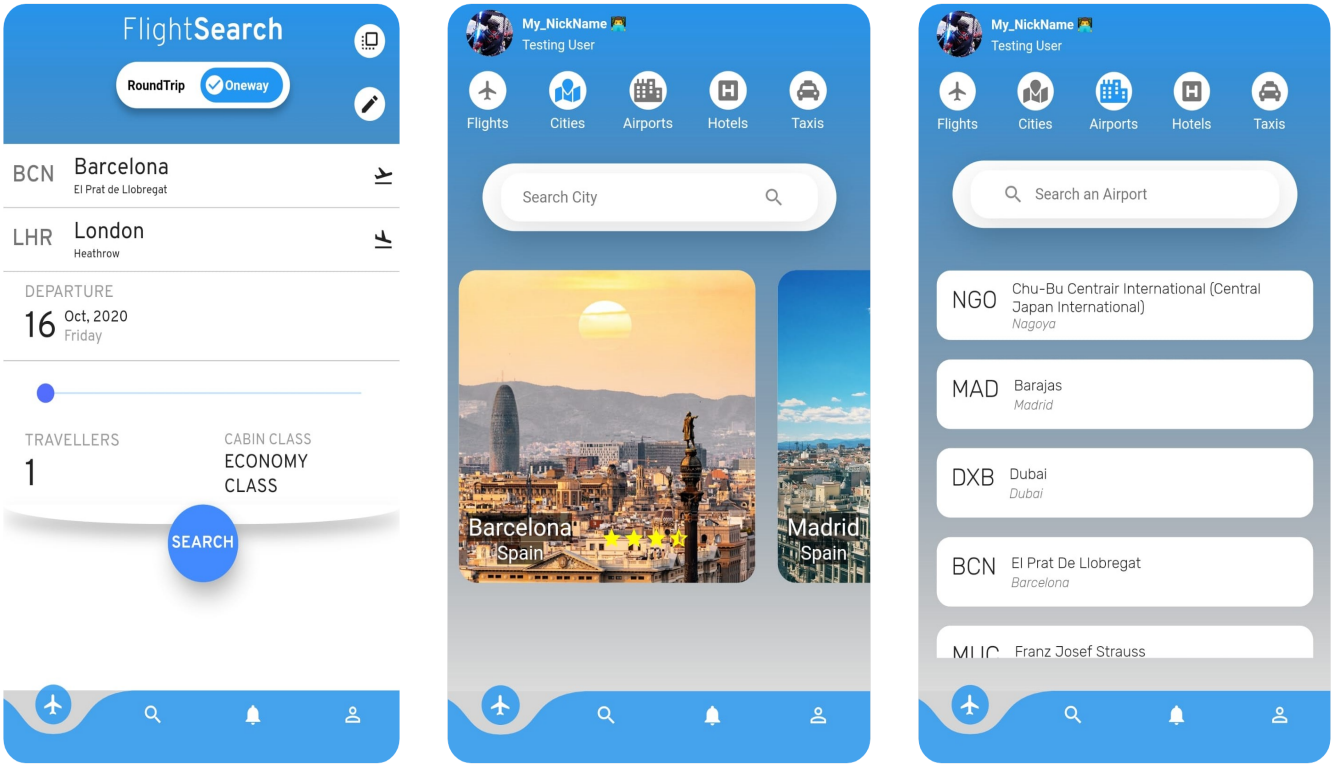
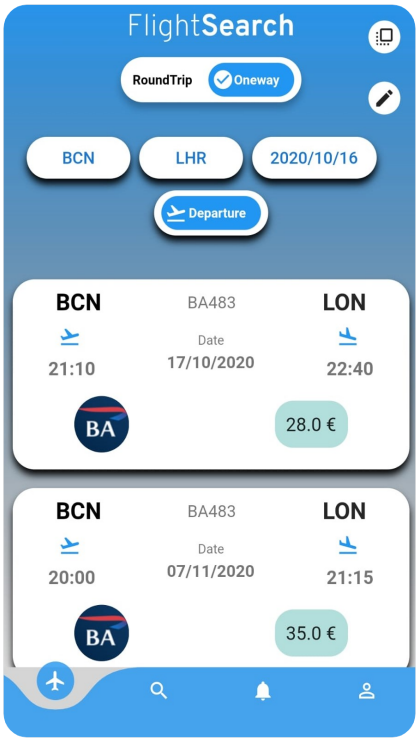



**Figura 32.** UFly - *Flight Search*, *Cities* y *Airports*




***Flight Search:*** En esta pantalla tenemos la posibilidad de realizar la búsqueda de billetes de vuelo, a través de parámetros como:

- Ida/ ida y vuelta
- Origen
- Destino
- Fecha salida/Fecha vuelta
- N° Pasajeros

Para seleccionar un origen o destino se despliega un buscador de ciudades desde la parte inferior. Para elegir la fecha disponemos de un *DateTimePicker*. Al clicar en el *search* nos aparecen los resultados de la búsqueda. Tal como muestra la figura 33.

**Figura 33.** *FlightSearch* - resultados

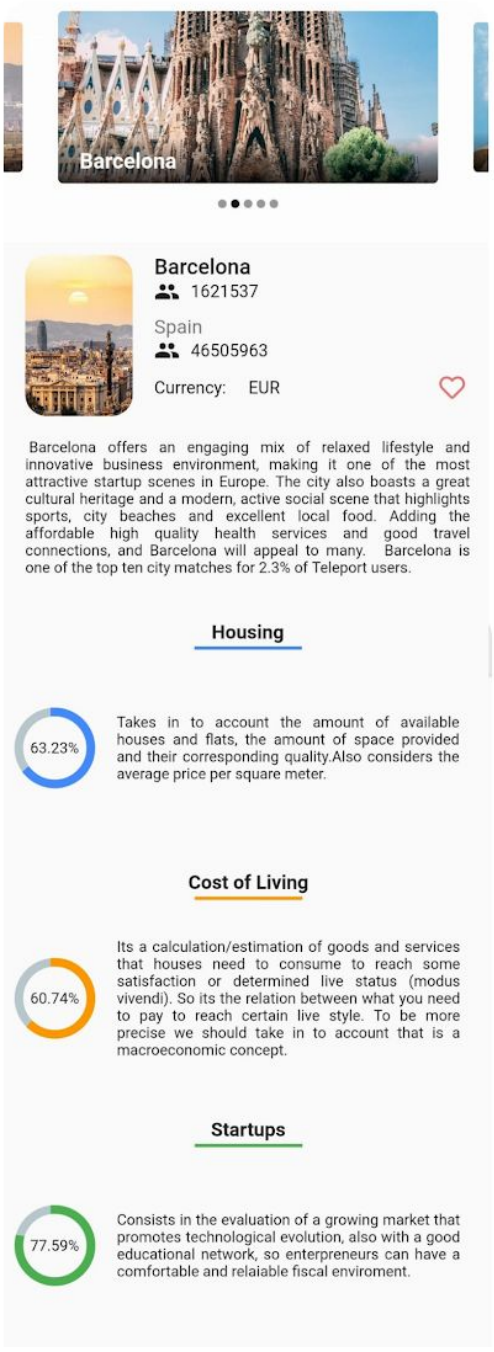



**Figura 34.** UFly - *Scroll CityDetalle*

***Cities:*** Aquí se nos cargan tres ciudades, de forma predeterminada tal y como muestra la figura 32. Disponemos de un buscador de ciudades de todo el mundo, al clicar en cualquier parte de la imagen se nos abre una nueva pantalla que muestra un carrusel de imágenes de la ciudad en cuestión *(figuras 34 y 35)*.

También facilitamos información sobre: país, número de habitantes, divisa, etc.

Podemos realizar un *scroll* vertical y visualizar ratios de diferentes características de la ciudad, como por ejemplo: alojamiento, acceso a Internet, coste de vida, etc.

Si pulsamos sobre el corazón, guardaremos la ciudad como un destino favorito, pudiendo luego acceder a un listado rápido de todas ellas desde nuestro *Profile*.

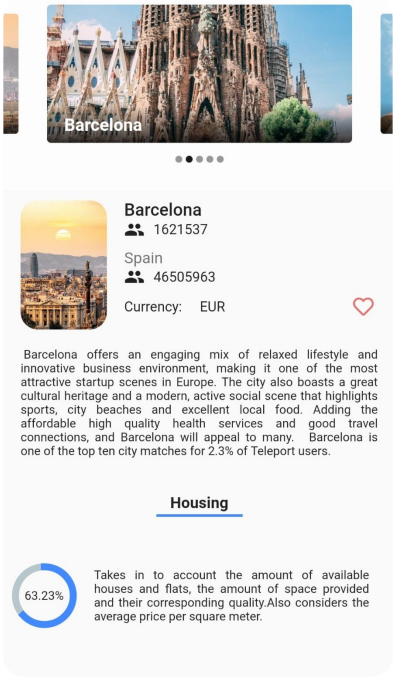



**Figura 35.** UFly - *CityDetalle, versión móvil*

***Airports:*** En esta pantalla disponemos de un buscador similar al de *Cities*. A parte contamos con una lista de los aeropuertos más importantes y conocidos, donde también al clicar en uno se despliega una nueva pantalla.

Esta pantalla nos permite acceder a información relacionada con el aeropuerto, sus llegadas y salidas, de forma que posibilitan el acceso a dicha información con tan sólo introducir el código IATA del aeropuerto en cuestión o en su defecto el nombre de la ciudad en el que se encuentra.

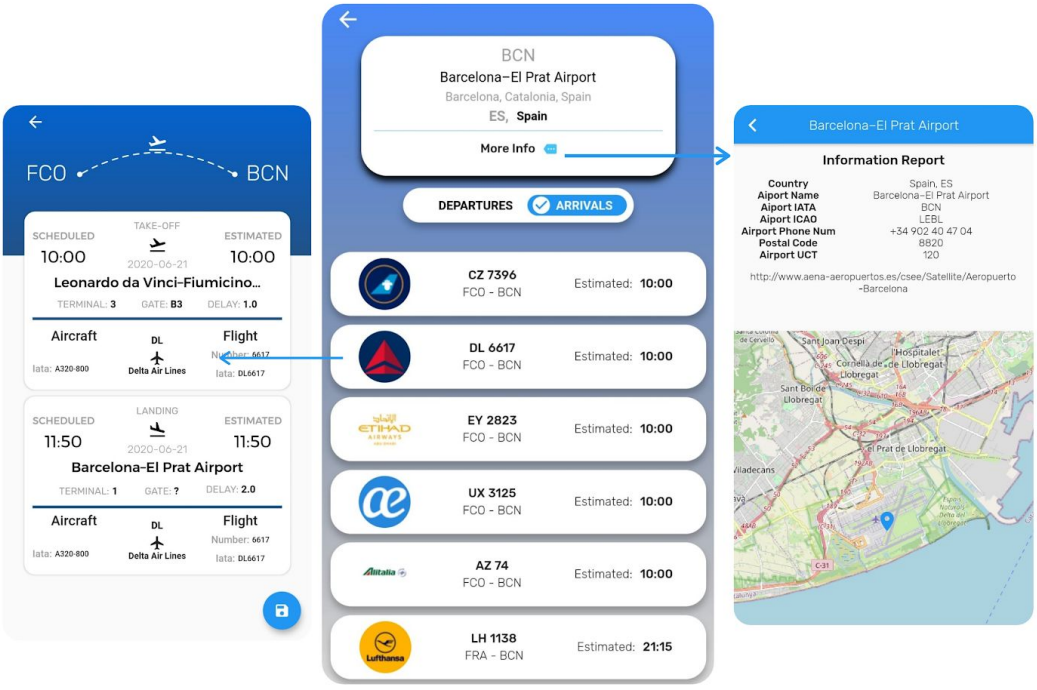



**Figura 36.** UFly - *AirportDetalle (FlightDetalle y AirportInfo)*

Desde esta pantalla, tenemos dos posibilidades, ***More Info*** que nos abre una nueva pantalla con información específica del aeropuerto y un mapa tal y como se muestra en la figura 36.

La otra opción es clicar en un vuelo de los listados en *Arrivals* o *Departures*, abriendo así una nueva pantalla con todos los detalles del vuelo y con opción a guardarlo y poder acceder a él en *Profile*, *MyTrips*.

La siguiente *screen* que podemos encontrar a través de la navegación inferior de la *App* es la del ***Search***. Esta pantalla, nos permite buscar a usuarios que forman parte de la comunidad ***UFly***, visitar su perfil, con la posibilidad de seguir a ese usuario.

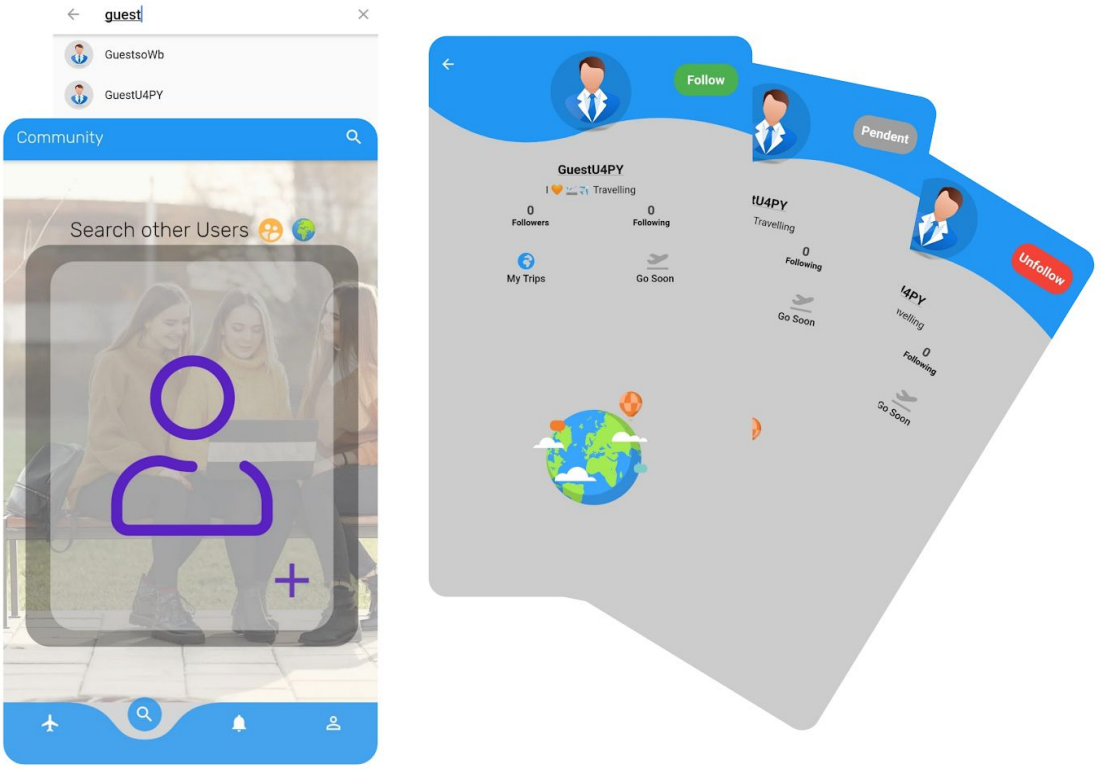



**Figura 37.** UFly - *Search Users, Search Results* y *PeopleProfile (3 estados)*

En la figura anterior podemos apreciar las distintas fases *(estados)* en los que puede estar una solicitud de seguimiento.

| Estado | Descripción |
|---|---|
| *Follow* | Aún no se ha realizado solicitud |
| *Pendent* | Solicitud realizada, a la espera de respuesta |
| *Unfollow* | Finalizar el seguimiento de la cuenta |

**Tabla 24.** Estados *PeopleProfile*

En la sección de *Notifications* encontramos tres apartados, resumidos en la siguiente tabla.

| Apartados | Descripción | Figura |
|---|---|---|
| Tópicos | Muestra los vuelos que hemos establecido para que la *App* nos avise si encuentra algún vuelo que cumpla con los parámetros indicados por el usuario. También existe la posibilidad de eliminarlos (*Figura 41*). | Fig. 38 |
| Push | Lista todas las notificaciones que la aplicación ha suministrado al usuario. Si clicamos encima de una de ellas podremos navegar hacia la página web de compra. En la que podremos corroborar que el precio del billete que ofrece nuestra *App* concuerda con el de compra final (*Figura 42*). | Fig. 39 |
| Add Alert | El botón de añadir, despliega una nueva pantalla para personalizar un tópico al cual el usuario quiere suscribirse y empezar a recibir notificaciones. Existe la posibilidad de establecer una fecha concreta o en su defecto un intervalo. El usuario debe debe seleccionar el precio máximo que está dispuesto a pagar para la ruta *(ORG-DEST)*. | Fig. 40 |

**Tabla 25.** Apartados de las notificaciones


**Figura 38.** UFly - *Topics*

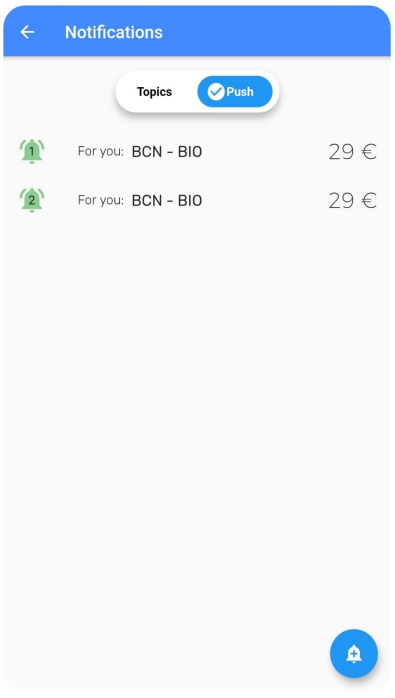



**Figura 39.** UFy - *Notifications Push*

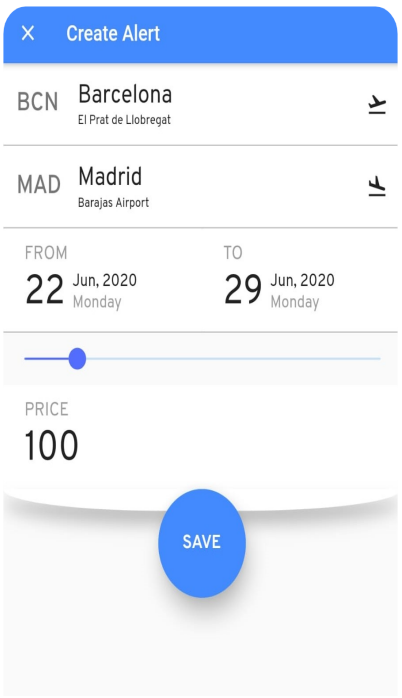



**Figura 40.** Crear *Topic*

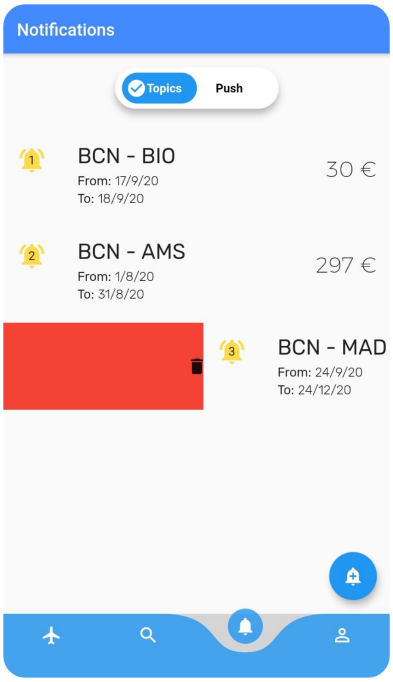



**Figura 41.** Borrar suscripción

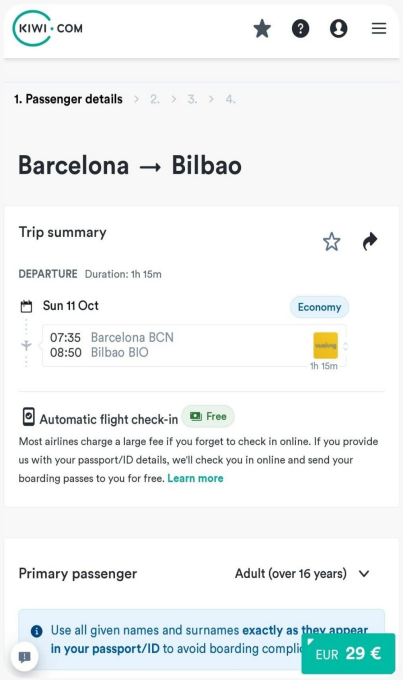



**Figura 42.** Compra billete

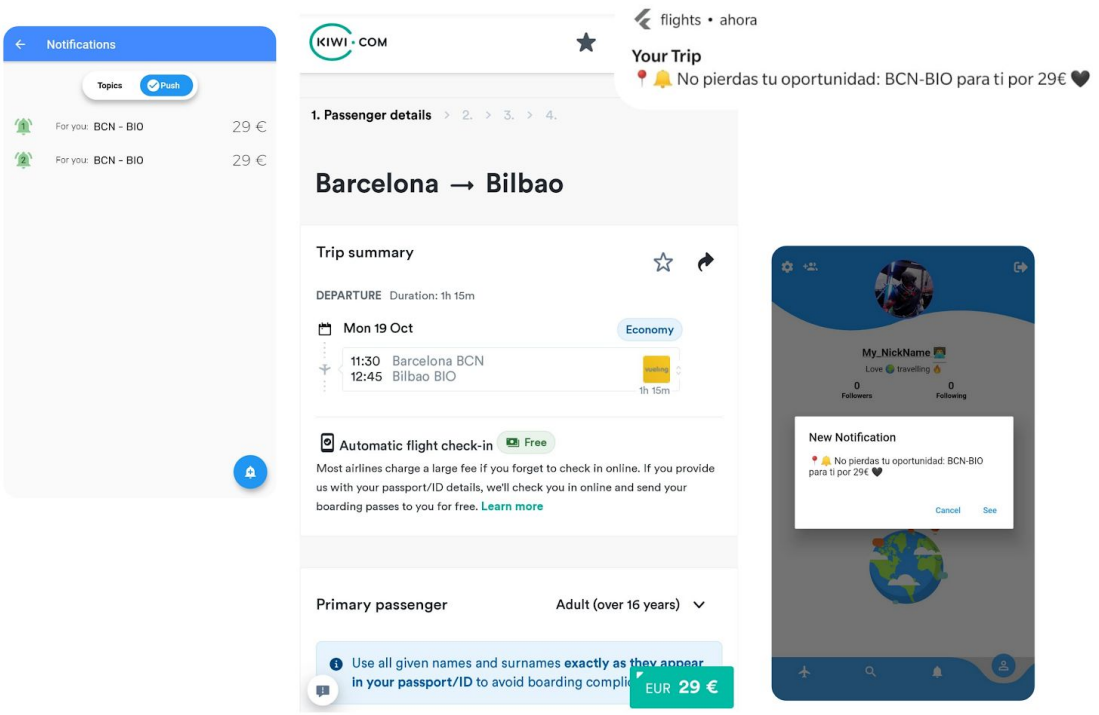
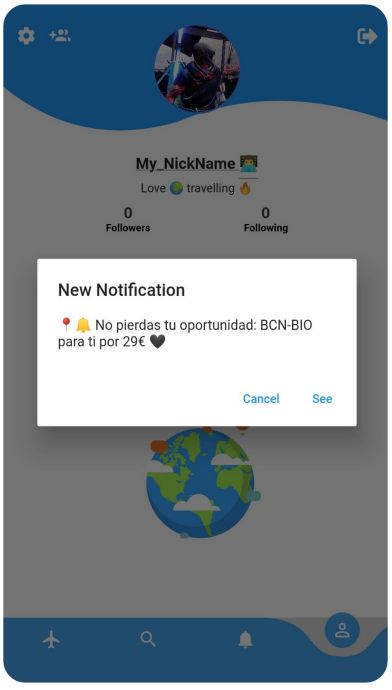



**Figura 43.** Rutas hacia la venta de billetes

En la figura 43 se puede observar un ejemplo de la pantalla de ***Notifications*** - *push* junto a la página de compra de un billete corroborando la consistencia y coherencia del precio comentada.

La información que se muestra en la sección de notificaciones es a modo de resumen, puesto que si la notificación aparece en esta pantalla, sabemos que el usuario ya ha recibido el mensaje, independientemente de que este haya sido procesado a través de **onMessage**, **onResume** o **onLaunch**.




Al llegar una notificación onMessage a la *App*, contamos con dos opciones:

- ***Cancel***, seguir navegando en la App
- ***See***, nos permite navegar a la página Notifications push.

**Figura 44.** Notificación **onMessage**

Si en cambio recibimos una notificación **onLaunch** o **onResume**, se nos mostrará una notificación en la interfaz propia del dispositivo. Al desplegar la barra de notificaciones de este, podremos ver la notificación confeccionada tanto para iOS como para Android.

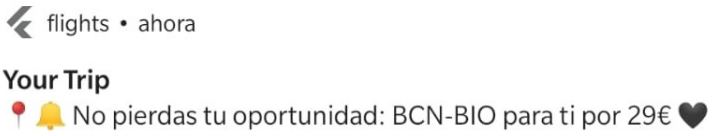



**Figura 45.** Notificación **onLaunch** y **onResume**

Al final de la navegación inferior encontramos el ***Profile*** que dispone de varios apartados. En la parte superior encontramos un icono de *settings* que nos despliega otro menú que mostraremos más adelante, encontramos también un icono de usuarios, que despliega una nueva pantalla, en la que podemos acceder a las solicitudes de otros usuarios y un icono de *logout* de la *App*. También podemos apreciar en el centro superior, la foto del *Profile* que al clicar sobre ella se puede cambiar por una que se encuentre en la galería del dispositivo.

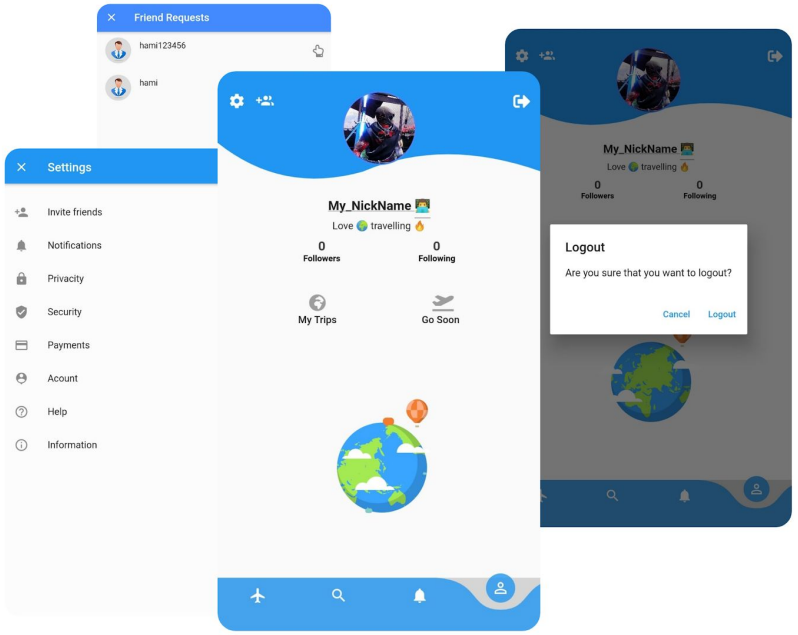



**Figura 46.** UFly - *ProfilePage* y *Icons Navigation Routes*

Debajo de la imagen de perfil se encuentra el nombre de usuario (*NickName*), una frase, el número los *followers* y *following*. Esta página también cuenta con submenús, My Trips y Go Soon *(viajes y ciudades guardados)*.

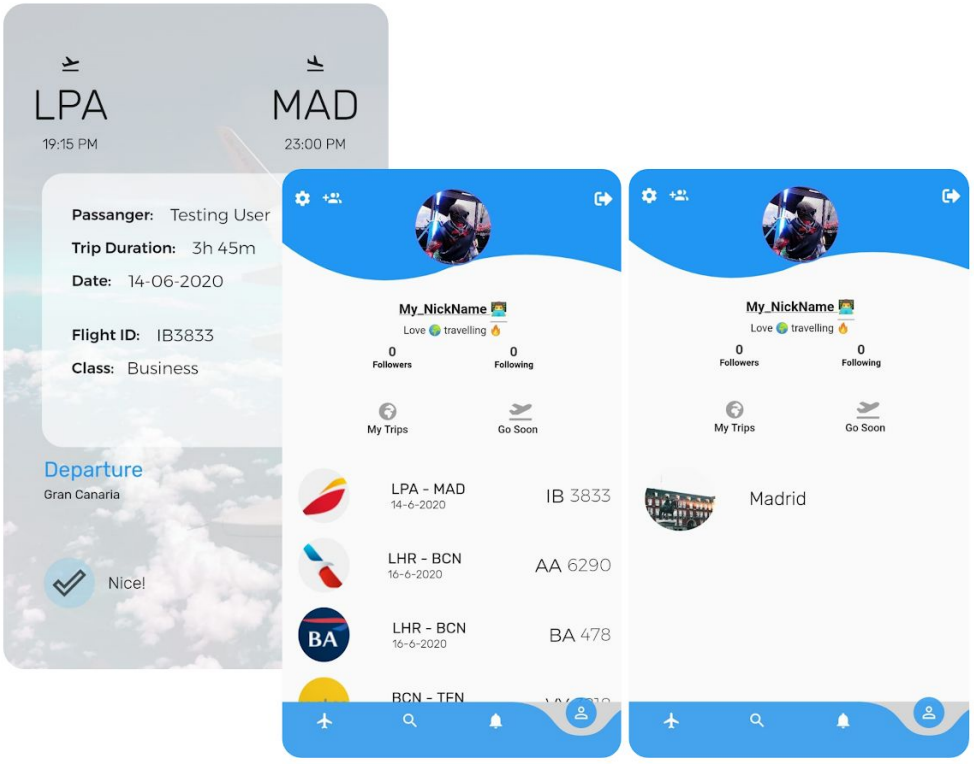



**Figura 47.** UFly - *ProfilePage*

En el apartado de My Trips se encuentra la lista de los vuelos que el usuario ha ido guardando, al clicar en uno se despliega una nueva página que muestra un resumen de los datos de ese vuelo. También está la posibilidad de eliminarlo.

En Go Soon, se encuentra la lista de las ciudades que les hemos dado me gusta en la pantalla de *CityDetalle* mostrada anteriormente.

También existe la posibilidad de eliminarlas de la lista haciendo un slidable hacia la derecha.

Si nos situamos en *SettingsPage* nos despliega un nuevo menú que nos ofrece tres posibilidades, *Profile,* editar el perfil, *Activity* y la propia página de *Settings.*

La página que permite editar la información del usuario, muestra los datos actuales del usuario. En ella se puede modificar cualquier campo y se dispone de un botón para guardar que nos va devolver un mensaje informando al usuario de si los cambios han sido guardado correctamente o no.

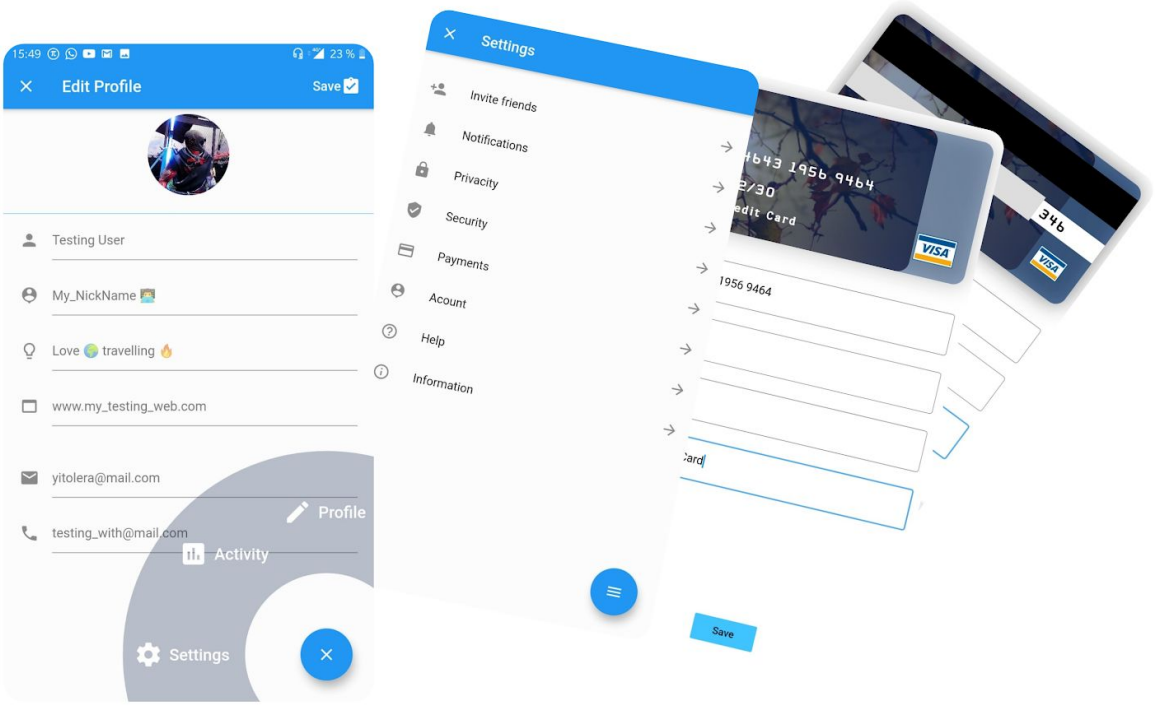



**Figura 48.** UFly - *Settings*, *EditProfile* & *Creditcard*

Actualmente los elementos listados en la pantalla de ***Settings***, muestran una pantalla informativa sobre la aplicación, a excepción la opción de pagos *(Payments)* que permite introducir datos de la tarjeta del usuario, aunque sin almacenar dicha información, sensible, en *Firestore*.

Si navegamos a través del icono de amigos situado en la parte superior izquierda de *ProfilePage*, encontramos las solicitudes de seguimiento que tenemos pendientes de aceptar o declinar, tal y como se muestra en la figura 49.

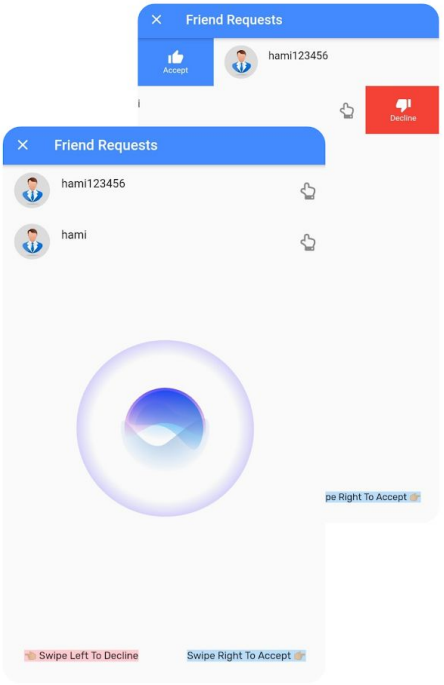



**Figura 49.** UFly - *PendentToAcceptPage*

# 8. Puesta en producción

La puesta en producción de una *App*, constituye el último paso para poder presentar la aplicación al público, a través de una de las tiendas como serían *Play Store (Android)* o *App Store (iOS)*. Sin embargo existe un paso previo sumamente importante, asegurar que el producto/servicio funciona de la forma deseada y que no presenta ningún fallo evitable.

Para ello nosotros hemos realizado gran variedad de tests a lo largo del desarrollo. Hemos procurado solucionarlos a medida que surgían antes de avanzar con el desarrollo de la *App*.

Cuando todo parece estar en orden se deben realizar configuraciones en partes específicas del código para poder hacer el *deploy* para cada plataforma y asegurar que cumpla con las políticas de cada una de las tiendas.

A continuación vamos a explicar algunos ejemplos de errores y puntos fallo más significativos y comunes que hemos afrontado.

## 8.1. Testing

Para el desarrollo, hemos utilizado **VD**s (dispositivos virtuales), tanto para *Android* como para *iOS*. En la fase final de desarrollo se compilaba la *App* en **FD**, dispositivos físicos.

Para ello era necesario hacer el *debug* de la APK en nuestros dispositivos móviles, en ellos realizamos las pruebas y tests de la *App*, emulando ser usuarios finales, anotando todo aquello que no terminaba de funcionar, generaba algún problema en el dispositivo o podría presentar dificultades para la interpretación del usuario, etc. Todo para poder ofrecer una *UX* positiva.

Un aspecto a considerar es el diseño de una *UI responsive*. Inicialmente confeccionamos la aplicación basándonos en nuestros FD. Cuando luego probamos a compilarla en otros dispositivos, que teníamos al alcance, nos dimos cuenta de que estábamos adaptando la *App* a un único dispositivo. Apareció entonces, la necesidad de crear un diseño adaptable a cualquier dispositivo. Antes de detallar cómo conseguimos suministrar dicha adaptabilidad, debemos prestar

atención a la siguiente tabla, donde se estipulan las principales ventajas y desventajas de la compilación de aplicaciones en función del tipo de dispositivo en el que se realice el *debug*.

| *Dispositivo* | Ventajas | Desventajas |
|---|---|---|
| Virtual | - Rapidez en la compilación<br>- Agilidad en el desarrollo<br>- Modificaciones instantáneas<br>- Elección entre distintos dispositivos | - Pérdida de eficiencia *(espacio memoria, capacidad de procesamiento )*<br>- No dispone de funcionalidades como la cámara o galería de un FD |
| Físico | - Eficiencia<br>- *App* tangible<br>- Permite realizar encuestas a otros usuarios<br>- La *App* queda almacenada en el *device* | - Problemas de almacenamiento<br>- Compilación más lenta que en VD<br>- Pérdida de conexión *(volver a compilar)*<br>- Tan sólo disponemos de un único dispositivo *(no se puede comprobar que sea responsive)* |

**Tabla 26.** Ventajas y desventajas entre VD y FD

A lo largo del desarrollo de ***UFly***, hemos experimentado numerosos errores, para solventarlos ha sido necesario que investigar dicho error, probar e implementar las correcciones necesarias.

Al disponer de un elevado número de ejemplos de errores hallados en *test* y pruebas rutinarias a las que hemos sometido a la *App,* tan sólo explicaremos y detallaremos un problema que bajo nuestro criterio, es un aspecto que debe ser considerado de forma previa al desarrollo.

Cómo ya hemos apuntado, la *App* debe poder adaptarse a la pantalla del dispositivo en la que esté corriendo, para lograrlo debemos presentar un diseño *responsive*.

A continuación añadimos una serie de capturas de pantalla de la aplicación, cuando esta se encontraba en fase de desarrollo. Dónde se observan los problemas de adaptabilidad.

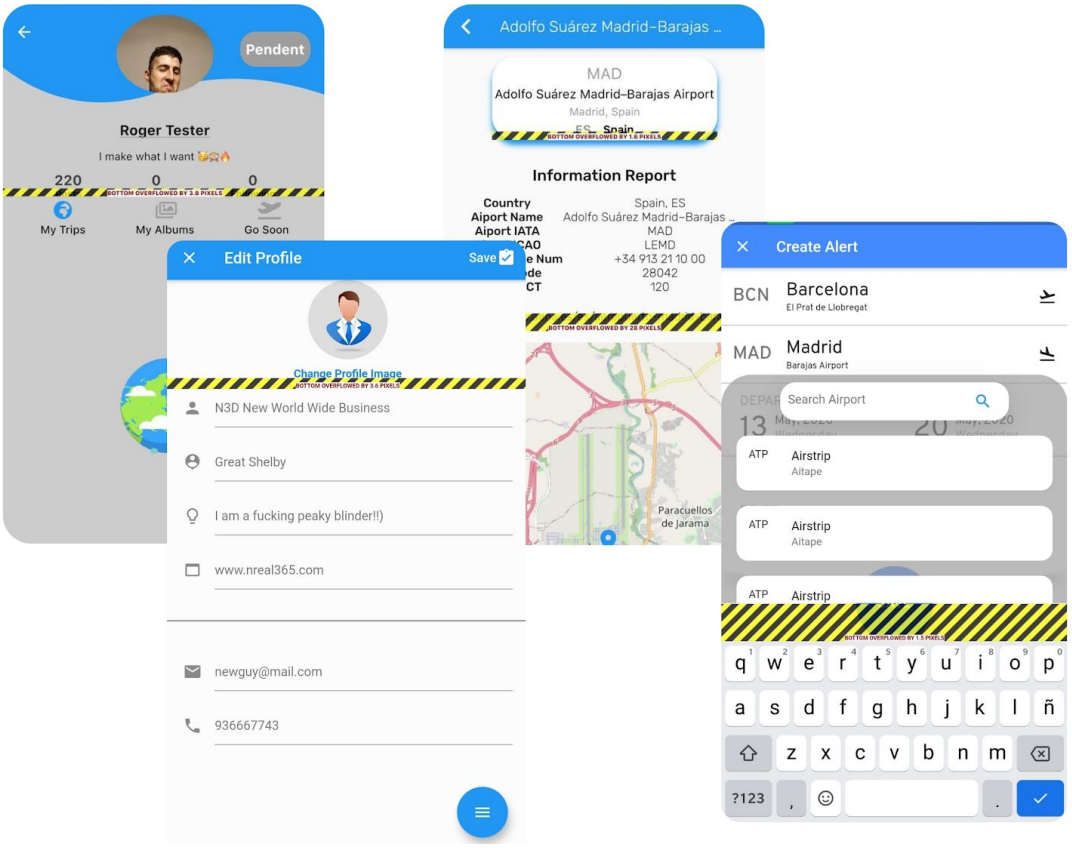



**Figura 50.** Errors - no *responsive UI*

Existen dos maneras distintas de abordar este problema. La que nosotros hemos empleado (una de las soluciones que ofrece Flutter), es confeccionar las pantallas a través de la información que nos provee el dispositivo, instanciando `MediaQuery`.

Esta instancia nos permite construir los *widgets* en función de las medidas del dispositivo en el que esté descargada la aplicación. A continuación se muestra cómo debemos realizar dicha instancia. Pudiendo así disponer de dos atributos, `size.height` y `size.width`.

```
final size = MediaQuery.of(context).size;
```

Una vez realizada la llamada ya disponemos de una variable, `size`, que podemos emplear tal y como se muestra a continuación, dónde establecemos un *Container responsive*, adaptable a cualquier dispositivo.

```
Container(
      margin: EdgeInsets.symmetric(
      horizontal: size.width * 0.35, vertical: 20),
      width: size.width * 0.4, // 40 % anchura del dispositivo
      height: size.height * 0.14, //14 % altura del dispositivo
      decoration: BoxDecoration(
      shape: BoxShape.circle,
       ),
         child: Center(
          child: BuildUserImg(),
   ));
```

Remarcar que estos problemas que experimentó ***UFly*** se deben a no haber contemplado este aspecto desde el inicio del desarrollo de la *App*.

# 9. Futuras implementaciones

La *App* ha sido desarrollada en tan solo 3 meses, pese a disponer de más pantallas y funcionalidades que las planteadas inicialmente, hemos creído conveniente presentarla con nuevos apartados e ideas que han ido surgiendo a lo largo del desarrollo. Estos apartados harán que la *App* sea más completa, robusta y pueda competir con las aplicaciones actuales del mercado.

A continuación vamos a mostrar los apartados incluidos en la *App* para futuras implementaciones.

En la siguiente figura podemos visualizar dos subapartados contenidos en el *HomePage*, *Hotels* y *Taxis*. Nos gustaría que ***UFly*** permitiera las usuarios analizar distintas ofertas de alojamiento, así como también la contratación de servicios como Uber o Cabify, aumentando la utilidad del servicio que brindamos.

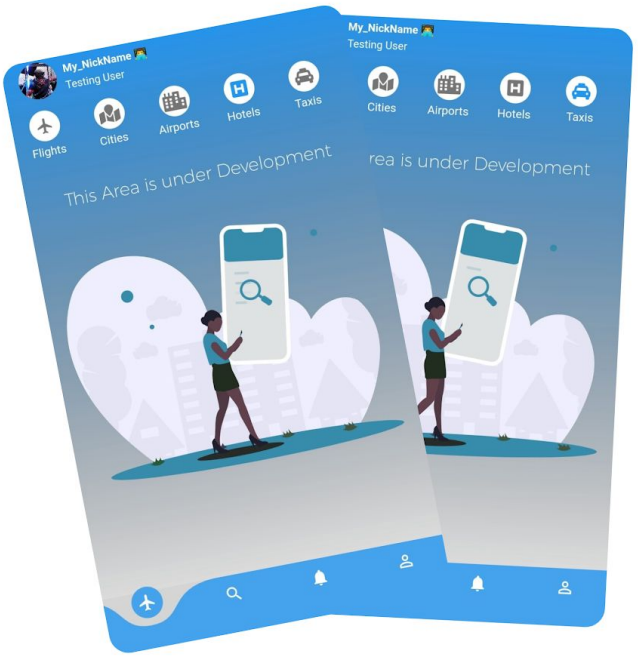



**Figura 51.** UFly - *Hotels y Taxis*

Otra implementación que nos gustaría añadir, es crear un algoritmo que permita a los usuarios encontrar perfiles de interés a los que seguir, o personas con los mismos gustos e intereses. Esto va acompañado de la creación de un *feed* (una

pantalla en la que los usuarios pueden compartir las ofertas que encuentren), permitiéndoles realizar paquetes, pudiendo ofrecer algún tipo de recompensa, económica/digital a nuestros usuarios. Incentivando así su uso de la *App*.

También nos gustaría permitir la creación de grupos de viajeros, ofreciéndoles un *chat* y un canal para compartir y debatir distintas ofertas.

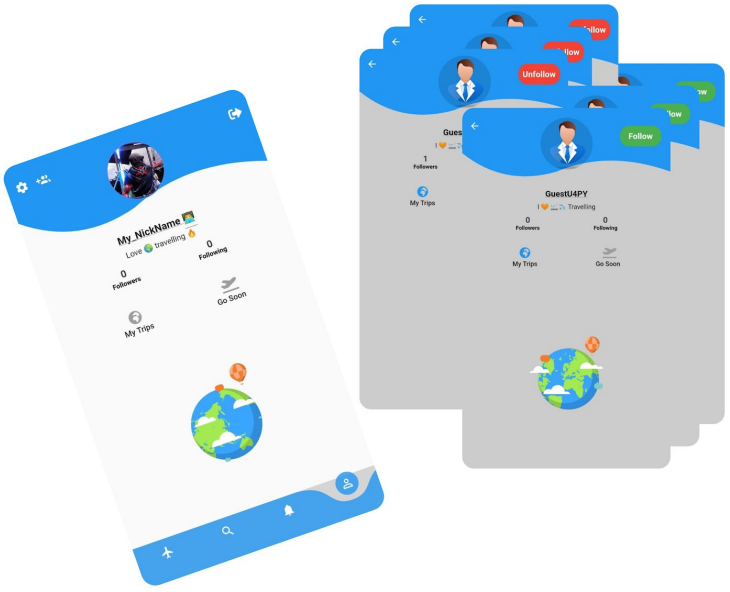



**Figura 52.** UFly - *Mejorar sistema seguimiento*

La mayor parte de las implementaciones futuras, las encontramos en el *Profile*, más concretamente, en el apartado *Settings*.

Nuestra idea es poder ofrecer una experiencia muy personalizada, permitiendo a nuestros usuarios adaptar en la medida de lo posible su experiéncia. Pudiendo así obtener el máximo potencial de nuestras herramientas *(Flutter y Firebase)*.

A continuación listamos algunos de los aspectos de configuración a los que tendrá acceso el usuario y que le permitan a su vez un mayor control sobre su perfil.

- Elegir el color de su Profile.
- Elegir el tema de su aplicación.
- Customizar los vuelos que se cargan en el *HomePage*, según *Go Soon*.

## 9.1. Posibilidades de negocio

Para poder convertir una *App* en negocio, es necesario realizar un estudio previo de las distintas metodologías que existen para rentabilizar una *App*, más allá de la opción básica que es la venta directa de la aplicación como un producto.

Tras dicho estudio hemos creado una tabla que permita de forma rápida comentar las distintas formas en las que es posible monetizar una *App*. También incluimos una reflexión de porque creemos que sería o no buena idea implementar dicha estrategia para ***UFly.***

| Estrategia | Descripción | ***UFly*** |
|---|---|---|
| **Publicidad** | Obtención de ingresos cuando se muestran anuncios a los usuarios que estén usando la *App*. Debemos codificar y establecer el tamaño del anuncio, darlo de alta, introducir la clave que te suministran en servicio de publicidad. Debemos considerar que para un usuario no resulta agradable ver anuncios. | En nuestro caso no se descarta su implementación en un futuro, sin embargo creemos conveniente explorar otros caminos antes de recurrir a la publicidad. |
| **App de pago** | Consiste en la venta de un servicio a un cierto precio. No recomendable, estás obligando a los usuarios a pagar por un servicio que que todavía no han probado y por tanto esto hace muy difícil que obtenga descargas. Esta opción puede considerarse cuando empresa/desarrollador ya dispone de un cierto reconocimiento. | En nuestro caso no sería acertado puesto que creemos más acertado que los usuarios dispongan de un servicio gratuïto, al menos inicialmente. Primero debes mostrar tu servicio, que este sea de utilidad y después el usuario estará más predispuesto a pagar por él. |

| | | |
|---|---|---|
| **Compras *(in-App)*** | Es una de las técnicas más utilizadas hoy en día y no tan solo en *Apps*, sino también en videojuegos. Es una técnica muy efectiva pues permite descargar la *App* sin coste alguno, tampoco dispone de publicidad. Eso sí los servicios o funcionalidades de las que dispone dentro de la *aplicación* son limitadas y por tanto para aumentar estas o desbloquear de nuevas debes pagar. | En nuestro caso esto podría realizarse para poder cambiar los colores del fondo de la App, adquirir más tópicos para poder ser notificado de más rutas, etc. Es interesante, aunque no sería nuestra primera opción puesto que queremos que la *App* sea funcional, atractiva y gratuïta. |
| **Suscripción** | Basado normalmente en un pago regular de una cuota fija, que permite al usuario hacer uso del producto/servicio. Normalmente ofrecen cierta flexibilidad en el pago *(semanal, mensual o anual)*, este tipo de aplicaciones normalmente ofrecen un servicio o contenido concreto. | En nuestro caso esta sería la mejor opción, pudiendo ofrecer paquetes gratuitos mensuales donde se incluya cierta cantidad de Tópicos y Notificaciones, pudiendo aumentar dichas cuantías a través de una pequeña cuota mensual. |
| **Programas de Afiliados** | Se basa en una remuneración según ventas, si vendes un producto a través de una *url* que te suministra el vendedor, recibes una comisión | En nuestro caso sería interesante trabajar con el programa de *partner* de Kiwi, pudiendo obtener un 3% de cada compra. |

**Tabla 27.** Estrategias para monetizar una *App - UFly*

Las técnicas que creemos más convenientes para esta aplicación son: la *publicidad* mediante la configuración en Firebase con anuncios de ofertas. La *suscripción* a través de ofrecer diferentes paquetes de *Topic - Notifications* mensuales. La *afiliación* a través de Kiwi, dónde obtenemos los billetes, pues nos ofrece la posibilidad de afiliarse y poder ganar el 3% de cada venta realizada. A través de un KEY que suministran a sus afiliados o programa de *partner*.

# 10. Conclusiones

El objetivo principal entorno al cual se ha articulado el desarrollo de este proyecto, era confeccionar una aplicación multiplataforma, con una propuesta de valor que aportará un servicio funcional y útil para el usuario del transporte aéreo.

Podemos afirmar, tras su desarrollo y testeo:

- Hemos creado una *App (compatible con iOS y Android)* para usuarios del T.A.
- Hemos facilitado el acceso a la información aeronáutica *(departures/arrivals, aeropuertos, ciudades, vuelos y billetes)*.
- La *App* cuenta con un sistema de Notificaciones que permite al usuario ser alertado cuando un vuelo, se ajusta a sus necesidades *(fecha y precio)*.
- Hemos ganado consciencia del nivel de dificultad que supone aprender nuevos procesos, técnicas y lenguajes para el desarrollo y publicación de una *App*.
- Comprendemos las herramientas esenciales para la creación de *Apps*.
- Hemos estudiado las distintas herramientas empleadas y las más útiles para la confección de *UFly* han sido:
    - Visual Studio Code
    - Flutter
    - Firebase
    - Cloud Functions
- Hemos aprendido a usar plataformas *Cloud* como *Firestore, Storage y CF*.
- Hemos mejorado nuestra habilidad *(facilidad)* para plasmar ideas en código.
- Comprendemos las distintas metodologías para monetizar aplicaciones móviles y web. Las que encajan mejor con nuestro servicio son:
    - Publicidad
    - Suscripción
    - Afiliación

# 11. Referencias

## 11.1. Artículos y páginas web


A. (2018). *Visual studio code.*. Recuperado el 5 de febrero de 2020 de https://www.atareao.es/software/programacion/visual-studio-code/

Apiumhub (2019). *Principales ventajas de usar Typescript.* Recuperado el 3 de febrero de 2020 de https://apiumhub.com/es/tech-blog-barcelona/usar-typescript/

AviationStack (s.f). *Real-Time Flight Tracker & Status API.* Recuperado el 20 de marzo de 2020 de https://aviationstack.com/

Bargueño, M. (2018). *Estacionalidad del sector.* Recuperado el 16 de marzo de 2020 de https://elpais.com/elpais/2018/07/26/album/1532595683_005204.html

BBC News (2018). *Cómo las cookies de tu computadora o teléfono pueden hacer que tus vacaciones te salgan más caras.* Recuperado el 10 de marzo de 2020 de https://www.bbc.com/mundo/noticias-44886758

Belchín, M. (2014). *Dart, el lenguaje de programación web creado por Google.* Recuperado el 2 de febrero de 2020 de https://www.adictosaltrabajo.com/2014/06/23/dart/#:~:text=Dart%20es%20un%20lenguaje%20de,y%20tipado%20opcional%20de%20datos.

Creativa, F. (2020). *Ventajas y desventajas de las aplicaciones móviles.* Recuperado el 12 de marzo de 2020 de https://www.factoriacreativabarcelona.es/blog/ventajas-y-desventajas-de-las-aplicaciones-moviles/

Europa Press (2019). *La industria española de Defensa, Aeronáutica y Espacio cierra 2018 con un 5,8% más de facturación.* Recuperado el 25 de marzo de 2020 de:https://www.europapress.es/turismo/transportes/aerolineas/noticia-industria-defensa-aeronautica-espacio-facturo-58-mas-espana-2018-20190625110432.html

Flightradar (s.f). *Real time flight tracker map.* Recuperado el 25 de febrero de 2020 de https://www.flightradar24.com/

Flutter (s.f). *Material components widgets*. Recuperado el 9 de febrero de 2020 de https://flutter.dev/docs/development/ui/widgets/material

Galán, J.S (2020). *Estacionalidad*. Recuperado el 16 de marzo de 2020 de https://economipedia.com/definiciones/estacionalidad.html

GOB (2019). *Mercado aéreo: Ministerio de transportes, movilidad y agenda urbana*. Recuperado el 20 de marzo de 2020 de https://www.mitma.gob.es/aviacion-civil/estudios-y-publicaciones/estadisticas-del-sector/informes-del-transporte-aereo-en-espana-2019/mercado-aereo-2019

González, G. (2014). *Qué son las cookies de tu navegador y para qué sirven*. Recuperado el 10 de marzo de 2020 de https://blogthinkbig.com/que-son-las-cookies

Hernández, M. (2019). *Industria global de las Apps*. Recuperado el 29 de febrero de 2020 de https://www.forbes.com.mx/industria-de-las-apps-un-mercado-de-mas-de-100000-mdd/

Justo, D. (2018). *Fechas clave para comprar billete avión*. Recuperado el 12 de marzo de 20020 de https://cadenaser.com/ser/2018/04/02/sociedad/1522662930_345238.html

Kiwi (2019). Virtual interlining. Recuperado el 8 de marzo de 2020 de https://partners.kiwi.com/

Pexels (s.f). *Cloudflare*. Recuperado el 9 de marzo de 2020 de https://www.pexels.com/

RapidApi (2020). *Airpot-Info API: How to use the Api*. Recuperado el 20 de marzo de 2020 de https://rapidapi.com/Active-api/api/airport-info/details

Teleport (s.f). *Teleport public APIs*. Recuperado el 22 de marzo de 2020 de https://developers.teleport.org/api/

Wikipedia Contributors (2020). *Kit de desarrollo de software*. Recuperado el 15 de febrero de 2020 de https://es.wikipedia.org/wiki/Kit_de_desarrollo_de_software

Wikipedia Contributors (2020). *C++*. Recuperado el 1 de febrero de 2020 de https://ca.wikipedia.org/wiki/C%2B%2B

Wikipedia Contributors (2020). *Dart: llenguatge de programació*. Recuperado el 2 de febrero de 2020 de https://ca.wikipedia.org/wiki/Dart_(llenguatge_de_programaci%C3%B3)

Wikipedia Contributors (2020). *Entorn integrat de desenvolupament*. Recuperado el 15 de febrero de 2020 de https://ca.wikipedia.org/wiki/Entorn_integrat_de_desenvolupament

Wikipedia Contributors (2020). *JavaScript*. Recuperado el 7 de febrero de 2020 de https://es.wikipedia.org/wiki/JavaScript

Wikipedia Contributors (2020). *TypeScript*. Recuperado el 3 de febrero de 2020 de https://es.wikipedia.org/wiki/TypeScript

Wikipedia Contributors (2020). *Visual studio code*. Recuperado el 5 de febrero de 2020 https://es.wikipedia.org/wiki/Visual_Studio_Code

## 11.2. Videos y Tutoriales

*10 Flutter List VIew and Card Widget*. Bitfumes (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=xEEqCU1id5w

*AnimatedContainer flutter widget of the week*. Google Developers (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=yI-8QHpGIP4

*Animation Basics with Implicit Animations*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=IVTjpW3W33s

*Async/Await flutter in focus*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=SmTCmDMi4BY&t=109s

*Beats parte 2*. Herrera, F. (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=YSfSzYH_pW4

*Bloc library painless state management for flutter*. ResoCoder (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=nQMfaQeCL6M

*Blog App con flutter y firebase storage firestore parte 1*. Bourbon Code (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=CWLvfutxFFQ

*Blog App con flutter y firebase storage firestore parte 2*. Bourbon Code (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=n9UtE4RJ5C8

*Building your first Flutter Widget*. Azaustre, C. (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=WeABVtYYFaw

*Building your first Flutter Widget*. Google Developers (2018).[Video]. Recuperado de https://www.youtube.com/watch?v=W1pNjxmNHNQ

*Conoce el lenguaje Dart*. Desarrollo web (2014). [Video]. Recuperado de https://www.youtube.com/watch?v=TzXOs5mJtsQ&t=750s

*Dart extension methods*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=D3j0OSfT9ZI

*Dart Futures flutter in focus*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=OTS-ap9_aXc

*Dart streams flutter in focus*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=nQBpOIHE4eE&t=370s

*Dart-side and Dark-side (The Boring Flutter Development Show, Ep. 34)*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=y5ByD5su-Ok&t=256s

*Data class & union in one dart package*. ResoCoder (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=ApvMmTrBaFI&t=464s

*Firebase - Rest endpoints with express*. Yogan, R. (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=3LH5QJQKX-Q&t=286s

*Firebase and push notifications in flutter*. FilledStack (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=Lq9-DPKWtIc

*Firebase Cloud Functions - Run Code on Demand*. Academind (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=qcECaRgU4uI

*Firebase Cloud Functions Creating a REST Endpoint with Cloud Functions*. Academind (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=qZ1EFnFOGvE

*Firebase Firestore Chat App: Cloud Functions & FCM (Ep 8) Kotlin Android Tutorial*. Reso Coder (2018). Video]. Recuperado de https://www.youtube.com/watch?v=NybmemQxdts&t=344s

*Firebase RealTime DB vs Firestore*. Esplin, C. (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=TmXct7seeBY

*Flutter Day Livestream Session 3: Dart*. Flutter (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=ZxSyZHq8gUg

*Flutter design tutorial minimal UI part 1*. Codex (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=neXfa4J752A&t=32s

*Flutter E commerce App: UI design, MyKart, time lapse, speed coding, amazon clone, two coders*. TwoCoders (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=N9GGWV8w3og

Flutter in 100 seconds, FireShip (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=lHhRhPV--G0

*Flutter performance tips*. Flutter. (2020) [Video]. Recuperado de https://www.youtube.com/watch?v=PKGguGUwSYE

*Flutter provider advanced firebase data management*. Fireship (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=vFxk_KJCqgk&t=191s

*Flutter provider architecture for state management flutter provider*. FilledStacks (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=kDEflMYTFlk&t=159

*Flutter state management tutorial Provider + ChangeNotifier, Bloc, MobX & More*. ResoCoder (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=jdUBV7AWL2U&t=1273s

*Flutter State Management: setState, BLoC, ValueNotifier, Provider*. Code, A. (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=7eaV9gSnaXw&t=332s

*Flutter the framework of 2020 mobile, web, desktop & beyond.* ResoCoder (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=7mpNF9zWrMc

*Flutter tutorial for beginners #14 expanded widgets*. The netninja (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=zNZvuP8h1vs

*Flutter Tutorial For Beginners Build Your First Flutter App, Flutter App Development*. Eureka (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=9XMt2hChbRo&t=3131s

*Flutter UI tutorial http and provider github following API*. Afgprogrammer (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=VHYvpTwpW-Q&t=605s

*Flutter upload files to firebase storage*. Yogan, R. (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=Y3AtNC684vg

*Flutter: How to start Flutter app on Web, iOS and Android*. Coder Blogger (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=kIOJRIOJXl4&t=627s

*Flutter: Provider, una alternativa al BLoC*. Herrera, F. (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=-KX2rH0qdKA

*GCP vs. Firebase - Functions & Firestore*. Google Cloud Platform (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=zR6CsTLTPsk

*Getting Started with Cloud Firestore with Node.js (Firecasts).* Firebase (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=Z87OZtIYC_0&t=8s

*Getting started with node.js on Google Cloud Functions - console & gcloud cli.* Overson, J. (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=MgivoBkvS5o

*Getting started with node.js on Google Cloud Functions - console & gcloud cli.* Overson, J. (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=MgivoBkvS5o

*Google Cloud Functions: real-time file processing*. Google Cloud Platform (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=rzHm2wu9_LM

*How do Cloud Functions work? Get to Know Cloud Firestore #11*. Fierbase (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=rERRuBjxJ80

*How to use firebase in flutter part 1*. Auth, E. (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=bjMw89L61FI&t=5s

*IDE vs Editor de código ¿Cual es la diferencia?*. GioCode (2017). [Video]. Recuperado de https://www.youtube.com/watch?v=L6twTkvETNs&t=83s

*Implementing complex UI with Flutter*. Szalek, M. [Video]. Recuperado de https://www.youtube.com/watch?v=FCyoHclCqc8&t=1340s

*Isolates and event loops flutter in focus*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=vl_AaCgudcY

*Jake Archibald: in the loop JSConfAsia*. Archibald, J. (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=cCOL7MC4Pl0

*La Lógica de la Programación Orientada a Objetos explicada con Minecraft*. Absolute (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=I848HdWjLMo

*Learn Flutter Clean Architecture & TDD Full Course*. Reso Coder (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=dc3B_mMrZ-Q

*Learn JavaScript Promises (Pt.1) with HTTP Triggers in Cloud Functions (Firecasts)*. Firebase (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=7IkUgCLr5oA&t=358s

*Learn JavaScript Promises (Pt. 2) with a Firestore Trigger in Cloud Functions (Firecasts)*. Firebase (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=652XeeKNHSk&t=162s

*Learn TypeScript in 50 minutes tutorial for beginners*. Codevolution (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=WBPrJSw7yQA&t=3s

*Los 4 principios de la programación orientada a objetos.* Beta Tech (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=tTPeP5dVuA4

*Mixins in dart understand dart & flutter fundamentals.* ResoCoder (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=pJHTyhzt9JU&t=604s

*Nodejs & Flutter, Simple REST API.* Fazt Code (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=SBDI53ECgVk&t=1141s

*Push notifications with firebase messaging flutter.* Milke, J. (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=wjJN1C9UxpY

*Realtime Database triggers (pt. 1) with Cloud Functions for Firebase (Firecasts).* Firebase (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=DglTSNEdl0U&t=368s

*SafeArea flutter widget of the week.* Google Developers (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=lkF0TQJO0bA

*Simple beginners guide to streams flutter and dart stream basics.* FilledStacks (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=53jIxLiCv2

*SnackBar (Flutter Widget of the Week).* Flutter (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=zpO6n_oZWw0

*Starter architecture for flutter & firebase Apps.* Code, A. (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=rMDRgXnBMq0

*State management for dummies.* Baumik, P. (2020). [Video]. Recuperado de https://www.youtube.com/watch?v=gCWU0JRmoq0&t=731s

*States rebuilder zero boilerplate flutter state management.* ResoCoder (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=iIxWMeFnIKA&t=390s

*The Async Await Episode I promised.* Fireship (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=vn3tm0quoqE

*Top 12 Flutter Tips & Tricks.* FireShip (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=FdgDgcrDeNI

*Transform flutter widget of the week*. Google Developers (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=9z_YNlRlWfA

*TypeScript the basics*. FireShip (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=ahCwqrYpIuM

*Typescript vs Javascript*. Awad, B. (2019) [Video]. Recuperado de https://www.youtube.com/watch?v=D6or2gdrHRE&t=1s

*Understanding Cloud Functions: Configuration settings (Firecasts)*. Firebase (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=xu5A1seU6PU&t=9s

*Understanding Cloud Functions: Retries and idempotence (Firecasts)*. Firebase (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=Pwsy8XR7HNE

*Use async/await with TypeScript in cloud functions for firebase firecasts*. Firebase (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=Jr7pDZ1RAUg

*Use TypeScript with Firebase Cloud Functions*. FireShip (2017). [Video]. Recuperado de https://www.youtube.com/watch?v=twmh82lvs1Q

*Using firestore as a backend to your flutter app*. Google Developers (2018). [Video]. Recuperado de https://www.youtube.com/watch?v=DqJ_KjFzL9I&t=265s

*What the heck is the event loop anyway?*. Roberts, P. (2014). [Video]. Recuperado de https://www.youtube.com/watch?v=8aGhZQkoFbQ&t=1008s

*Why flutter uses dart*. Flutter (2019). [Video]. Recuperado de https://www.youtube.com/watch?v=5F-6n_2XWR8

Roger Villalba Fuentes Hamet Lamin Jalil

Sabadell, *2020*

# 12. Anexos

Antes de mostrar lo anexos, comentar que aquí es dónde podremos encontrar la mayor parte del desarrollo de nuestra aplicación. Pues aparece gran parte del código que la conforma. No ha sido fácil organizarlo. Aunque preferíamos redactarlo de nuevo para que así apareciera de forma clara en el memória en vez de realizar capturas de pantalla. De esta forma también disponíamos de un apartado al cual hacer referencia a nuestro propio código sin necesidad de mostrarlo a lo largo de la redacción del trabajo.

Puesto que puede que no todos los lectores de este trabajo estén interesados en leer y comprender nuestro código.

El anexo comprende los siguientes puntos:


**12.1. Respuestas JSON ..... 119**
**12.2. Ejemplos de Código ..... 129**
**12.3. Firestore..... 140**
**12.4. Cloud Functions ..... 144**
**12.5. Modelos Dart ..... 183**
**12.6. Scripts ..... 192**

## 12.1. Respuestas JSON

### 12.1.1. getFlightInfo *response*


```
{
    "flightInfo": {
        "flight_date": "2020-06-07",
        "flight_status": "scheduled",
        "departure": {
            "airport": "Zurich",
            "timezone": "Europe/Zurich",
            "iata": "ZRH",
            "icao": "LSZH",
            "terminal": "1",
            "gate": "A74",
            "delay": null,
            "scheduled": "2020-06-07T11:50:00+00:00",
            "estimated": "2020-06-07T11:50:00+00:00",
            "actual": null,
            "estimated_runway": null,
            "actual_runway": null
        },
        "arrival": {
            "airport": "Schiphol",
            "timezone": "Europe/Amsterdam",
            "iata": "AMS",
            "icao": "EHAM",
            "terminal": "3",
            "gate": null,
            "baggage": null,
            "delay": null,
            "scheduled": "2020-06-07T13:35:00+00:00",
            "estimated": "2020-06-07T13:35:00+00:00",
            "actual": null,
            "estimated_runway": "2020-06-07T13:35:00+00:00",
            "actual_runway": null
        },
        "airline": {
```

```
            "name": "KLM",
            "iata": "KL",
            "icao": "KLM"
        },
        "flight": {
            "number": "1958",
            "iata": "KL1958",
            "icao": "KLM1958",
            "codeshared": null
        },
        "aircraft": {
            "registration": "PH-EZS",
            "iata": "E190",
            "icao": "E190",
            "icao24": "484CC3"
        },
        "live": {
            "updated": "2020-06-07T09:43:59+00:00",
            "latitude": 52.25,
            "longitude": 4.63,
            "altitude": 0,
            "direction": 351.56,
            "speed_horizontal": 4.644,
            "speed_vertical": 0,
            "is_ground": false
        }
    },
    "depCity": {
        "airportName": "Zürich Airport",
        "latitude": 47.458218,
        "longitude": 8.555475
    },
    "arrCity": {
        "airportName": "Amsterdam Airport Schiphol",
        "latitude": 52.31054,
        "longitude": 4.7682743
    }
}
```

### 12.1.2. getArrivalTimeTable *response*


```
{
    "arrivalInfo": [
        {
            "flight_date": "2020-06-06",
            "flight_status": "active",
            "departure": {
                "airport": "Ataturk Airport",
                "timezone": "Europe/Istanbul",
                "iata": "IST",
                "icao": "LTBA",
                "terminal": null,
                "gate": null,
                "delay": null,
                "scheduled": "2020-06-06T04:10:00+00:00",
                "estimated": "2020-06-06T04:10:00+00:00",
                "actual": null,
                "estimated_runway": "2020-06-06T03:56:00+00:00",
                "actual_runway": "2020-06-06T03:56:00+00:00"
            },
            "arrival": {
                "airport": "El Prat De Llobregat",
                "timezone": "Europe/Madrid",
                "iata": "BCN",
                "icao": "LEBL",
                "terminal": "1",
                "gate": null,
                "baggage": null,
                "delay": null,
                "scheduled": "2020-06-06T06:00:00+00:00",
                "estimated": "2020-06-06T06:00:00+00:00",
                "actual": null,
                "estimated_runway": "2020-06-06T05:53:00+00:00",
                "actual_runway": null
            },
            "airline": {
                "name": "Turkish Airlines",
```

```
                "iata": "TK",
                "icao": "THY"
            },
            "flight": {
                "number": "6351",
                "iata": "TK6351",
                "icao": "THY6351",
                "codeshared": null
            },
            "aircraft": null,
            "live": null
        }
    ]
}
```


### 12.1.3. getAirlineInfo *response*


```
{
    "airlineData": {
          "fleet_average_age": "6.7",
          "callsign": "VUELING",
          "hub_code": "BCN",
          "iata_code": "VY",
          "icao_code": "VLG",
          "country_iso2": "ES",
          "date_founded": "2004",
          "iata_prefix_accounting": "30",
          "airline_name": "Vueling",
          "country_name": "Spain",
          "fleet_size": "105",
          "status": "active",
          "type": "scheduled"
      }
  }
```

### 12.1.4. getAirportInfo *response*


```
{
    "airportInfo": {
        "id": 583,
        "iata": "BCN",
        "icao": "LEBL",
        "name": "Barcelona-El Prat Airport",
        "location": "Barcelona, Catalonia, Spain",
        "street_number": "",
        "street": "",
        "city": "El Prat de Llobregat",
        "county": "Barcelona",
        "state": "Catalonia",
        "country_iso": "ES",
        "country": "Spain",
        "postal_code": "8820",
        "phone": "+34 902 40 47 04",
        "latitude": 41.297443,
        "longitude": 2.0832942,
        "uct": 120,
"website": "http://www.aena-aeropuertos.es/csee/Satellite/Aeropuerto-Barcelona"
    },
    "airportDepartures": [
        {
            "flight_date": "2020-06-07",
            "flight_status": "active",
            "departure": {
                "airport": "El Prat De Llobregat",
                "timezone": "Europe/Madrid",
                "iata": "BCN",
                "icao": "LEBL",
                "terminal": "2",
                "gate": null,
                "delay": null,
                "scheduled": "2020-06-07T09:50:00+00:00",
                "estimated": "2020-06-07T09:50:00+00:00",
                "actual": null,
```

```
            "estimated_runway": "2020-06-07T10:20:00+00:00",
            "actual_runway": "2020-06-07T10:20:00+00:00"
        },
        "arrival": {
            "airport": "Düsseldorf International Airport",
            "timezone": "Europe/Berlin",
            "iata": "DUS",
            "icao": "EDDL",
            "terminal": "2",
            "gate": "B92",
            "baggage": null,
            "delay": 1,
            "scheduled": "2020-06-07T12:15:00+00:00",
            "estimated": "2020-06-07T12:15:00+00:00",
            "actual": null,
            "estimated_runway": "2020-06-07T12:04:00+00:00",
            "actual_runway": null
        },
        "airline": {
            "name": "Lufthansa",
            "iata": "LH",
            "icao": "DLH"
        },
        "flight": {
            "number": "5193",
            "iata": "LH5193",
            "icao": "DLH5193",
            "codeshared": {
                "airline_name": "eurowings",
                "airline_iata": "ew",
                "airline_icao": "ewg",
                "flight_number": "9445",
                "flight_iata": "ew9445",
                "flight_icao": "ewg9445"
            }
        },
        "aircraft": null,
        "live": null
    },
],
```

```
"airportArrivals": [
    {
        "flight_date": "2020-06-06",
        "flight_status": "active",
        "departure": {
            "airport": "Ataturk Airport",
            "timezone": "Europe/Istanbul",
            "iata": "IST",
            "icao": "LTBA",
            "terminal": null,
            "gate": null,
            "delay": null,
            "scheduled": "2020-06-06T04:10:00+00:00",
            "estimated": "2020-06-06T04:10:00+00:00",
            "actual": null,
            "estimated_runway": "2020-06-06T03:56:00+00:00",
            "actual_runway": "2020-06-06T03:56:00+00:00"
        },
        "arrival": {
            "airport": "El Prat De Llobregat",
            "timezone": "Europe/Madrid",
            "iata": "BCN",
            "icao": "LEBL",
            "terminal": "1",
            "gate": null,
            "baggage": null,
            "delay": null,
            "scheduled": "2020-06-06T06:00:00+00:00",
            "estimated": "2020-06-06T06:00:00+00:00",
            "actual": null,
            "estimated_runway": "2020-06-06T05:53:00+00:00",
            "actual_runway": null
        },
        "airline": {
            "name": "Turkish Airlines",
            "iata": "TK",
            "icao": "THY"
        },
        "flight": {
            "number": "6351",
```

```
                "iata": "TK6351",
                "icao": "THY6351",
                "codeshared": null
            },
            "aircraft": null,
            "live": null
        }
    ]
}
```


### 12.1.5. getCityData *response*


```
{
    "cityName": "New York City",
    "cityPopulation": 8175133,
    "cityGeoname_id": 5128581,
    "countryName": "United States",
    "countryIso3": "USA",
    "countryPopulation": 310232863,
    "countryCurrency": "USD",
    "countryGeoname_id": 6252001,
    "scoreData": {
        "scoreInfo": [
            {
                "color": "#f3c32c",
                "name": "Housing",
                "score_out_of_10": 1
            },
            {
                "color": "#f3d630",
                "name": "Cost of Living",
                "score_out_of_10": 2.342
            }
        ],
        "summary": " The New York metropolitan area is one of the most populated cities in the world. New York City itself is an international center for numerous industries including finance, theater, television, film, arts, and technology."
```

```
    },
    "images": [
"https://images.pexels.com/photos/466685/pexels-photo-466685.jpeg?auto=
CUT",
"https://images.pexels.com/photos/290386/pexels-photo-290386.jpeg?auto=
CUT",
"https://images.pexels.com/photos/313782/pexels-photo-313782.jpeg?auto=
CUT",
"https://images.pexels.com/photos/1486222/pexels-photo-1486222.jpeg?aut
o=CUT",
"https://images.pexels.com/photos/378570/pexels-photo-378570.jpeg?auto=
CUT",
"https://images.pexels.com/photos/1525612/pexels-photo-1525612.jpeg?aut
o=CUT"
    ]
}
```


NOTE: “CUT” should be replaced by:
compress&cs=tinysrgb&fit=crop&h=1200&w=800

### 12.1.6. getKiwiFlightTickets *response*


```
[
    {
        "airline": "VY",
        "flightId": "VY1420",
        "from": "BCN",
        "to": "BIO",
        "from_city": "Barcelona",
        "to_city": "Bilbao",
        "country_from": "ES",
        "country_to": "ES",
        "depTime": "2020-07-28T07:00:00.000Z",
        "arrTime": "2020-07-28T08:15:00.000Z",
        "date": "07/07/2020",
        "duration": "1h 15m",
        "distance": 467.32,
```

```
        "price": 35,
        "availability": 4,
        "bookingToken": "AyiACQOiOy6ht-ClNNhM_7hf1ONCcvDCkk5zIMfa40",
        "deep_link":
"https://www.kiwi.com/deep?from=BCN&to=BIO&departure"
    },
    {
        "airline": "VY",
        "flightId": "VY1420",
        "from": "BCN",
        "to": "BIO",
        "from_city": "Barcelona",
        "to_city": "Bilbao",
        "country_from": "ES",
        "country_to": "ES",
        "depTime": "2020-07-24T07:00:00.000Z",
        "arrTime": "2020-07-24T08:15:00.000Z",
        "date": "07/07/2020",
        "duration": "1h 15m",
        "distance": 467.32,
        "price": 35,
        "availability": 1,
        "bookingToken": "A1yXd6kqKBGCnEetIRrT2gHJpHwOc2MJ",
        "deep_link":
"https://www.kiwi.com/deep?from=BCN&to=BIO&departure"
    }
]
```

## 12.2. Ejemplos de Código

### 12.2.1. Ejemplo de Código *Flutter template*

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
        visualDensity: VisualDensity.adaptivePlatformDensity,
      ),
      home: MyHomePage(title: 'Flutter Demo Home Page'),
    );
  }
}

class MyHomePage extends StatefulWidget {
  MyHomePage({Key key, this.title}) : super(key: key);

  final String title;

  @override
  _MyHomePageState createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
  int _counter = 0;

  void _incrementCounter() {
    setState(() {
```

```
      _counter++;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: <Widget>[
            Text(
              'You have pushed the button this many times:',
            ),
            Text(
              '$_counter',
              style: Theme.of(context).textTheme.headline4,
            ),],),
      ),
      floatingActionButton: FloatingActionButton(
        onPressed: _incrementCounter,
        tooltip: 'Increment',
        child: Icon(Icons.add),
      ),);
  }
}
```

Aunque pueda parecer extenso, no lo es, puesto que con este trozo de código ya disponemos de una aplicación que gestiona el estado de un contador, así como también incluye el diseño de la página mostrada en la figura 17.

El código que conforma la figura 17, existen 2 clases. Una para crear la aplicación MyApp y MyHomePage dónde se define la estructura, la lógica y los elementos visuales que aparecen, la lógica sería la función _incrementCounter(), el estado estaría almacenado en la variable _counter y el accionador de la función sería el onPressed del widget FloatingActionButton.

### 12.2.2. Ejemplo de CF onRequest

```
exports.getAirlineInfo = functions.https.onRequest(async (req,
resp)=>{

    let airlineSearch:string = req.query.airline;

    const optionAirline = {
        method: 'GET',
      url:
      `http://api.aviationstack.com/v1/airlines?access_key=${acces
      s_key}`,
        headers: {
        'content-type': 'application/json'
      }
    };
    try{
  await request(optionAirline, function(error:any, notShadow:any,
body:any){
          const airlinesDoc = Converted.toAirlines(body);
          console.log(airlinesDoc);
          const airlinesList = airlinesDoc.data;
          let i = 0;
          let trobat = 0;
          while(i < airlinesList.length && trobat == 0){
            let anal = airlinesList[i].airline_name;
            console.log(anal + ' ' + i);
           if(anal.toLowerCase() == airlineSearch.toLowerCase()){
              console.log(airlinesList[i].toString());
              trobat = 1;
            }else{
              i++;
            }}
          if(trobat === 1){
            return resp.send({airlineData: airlinesList[i]});
          }else{
            return resp.send({airlineData: 'NO match found'});
          }
```

```
        })
      }catch(e){
        console.log(e.toString());
      }
    })
```

En el código anterior existen funciones como por ejemplo `.toAirlines`, que nos permiten procesar un objeto JSON y convertirlo a un objeto manipulable para construir nuestra respuesta.

Ahora podemos ver que la CF devuelve de la lista de aerolíneas que procesa, el elemento [i]. El objeto JSON respuesta de esta función ya ha sido mostrado en el apartado de APIs bajo el nombre de getAirlineInfo().

### 12.2.3. Ejemplo de CF onCall

```
exports.sendDeviceToken = functions.https.onCall(async (data, context)
=> {
    const uid = context.auth?.uid;
    return db.doc(`users/${uid}`).update({
        tokenId: data.tokenId
    }).then(() => {
        console.log('Done');
        return true;
    }).catch(console.error);
})
```

Para que este código realmente sea ejecutado se debe establecer acceso de administrador.

En esta CF se puede apreciar cómo somos capaces de extraer el uid (user identifier) del parámetro data, que va intrínseco en la request. Esta función nos permite conocer a qué dispositivo debemos enviar las notificaciones, puesto que el tokenId es único para cada usuario junto con su dispositivo.

### 12.2.4. Ejemplo de CF Automatic Trigger

```
exports.decideToDeleteTopicSubscription =
functions.firestore.document('subscriptions/{subscription}').onUpdate((
change, context) => {

    const subscription = context.params.subscription;
    const newValue = change.after.data();
    const oldValue = change.before.data();
    const oldToken = oldValue?.fcmTokens;
    const newToken = newValue?.fcmTokens;
    const oldUid = oldValue?.subscribersUid;
    const newUid = newValue?.subscribersUid;

     if (oldToken.length == 1 && oldUid.length == 1 && newToken.length
== 0 && newUid.length == 0) {
        return db.doc(`subscriptions/${subscription}`).delete()
.then(() => {
            return db.doc(`topics/${subscription}`).delete()
.then(() => {
                console.log('Topic & subscription were deleted');
            })
        })
    } else {
            console.log(oldToken.length, oldUid.length, newToken.length,
newUid.length)
        return;
    }})
```

Existen distintos tipos de *triggers*. Principalment *.onCreate()*, *.onUpdate()*, *.onWrite() y .onDelete()*.

En esta CF al tratarse de onUpdate() podemos analizar los cambios que se producen antes y después de la modificación pudiendo comprobar cuál ha sido el cambio producido en dicho documento y actuar en consecuencia.

Hay que ir con sumo cuidado con este tipo de funciones pues al no disponer de un accionador directo, podríamos generar un bucle dentro de nuestras CF de forma que pese a disponer de un número elevado de CF de forma gratuïta el

hecho de crear un bucle podría llegar a generar una facturación desorbitada. Algo muy desagradable y a tener muy presente en el desarrollo de este proyecto.

### 12.2.5. Ejemplo de Código *ChangeNotifier*

A continuación se muestra un ejemplo del primer ChangeNotifier que implementamos en el proyecto. Su funcionalidad es gestionar la navegación principal de la App.

```
class NavegacionModel with ChangeNotifier {
  int _paginaActual = 0;
  PageController _pageController = new PageController();

  int get paginaActual => this._paginaActual;
  set paginaActual(int valor) {
    this._paginaActual = valor;
    _pageController.animateToPage(valor,
        duration: Duration(milliseconds: 450),
        curve: Curves.fastLinearToSlowEaseIn);
    notifyListeners();
  }
  PageController get pageController => this._pageController;
}
```

Como podemos ver la classe NavegacionModel extiende ChangeNotifier lo que implica que incluye las características y hereda sus propiedades.

### 12.2.6. Ejemplo de Código *ChangeNotifierProvider*

```
return MultiProvider(

      providers: [
        ChangeNotifierProvider(create: (_) => HomeService()),
        ChangeNotifierProvider(create: (_) => SearchProvider()),
        ChangeNotifierProvider(create: (_) => ProfileServices()),
        ChangeNotifierProvider(create: (_) => TopicServices()),
        Provider<ImagePickerService>(
          create: (_) => ImagePickerService(),
        ),
      ]);
```

### 12.2.7. Ejemplo de Código *Consumer*

```
class _CurvedNavegacion extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    final navegacionModel = Provider.of<NavegacionModel>(context);

    return CurvedNavigationBar(
      color: Colors.blue.withOpacity(0.8),
      backgroundColor: Colors.grey.withOpacity(0.4),
      items: <Widget>[
        Icon(Icons.flight, size: 25, color: Colors.white),
        Icon(Icons.search, size: 25, color: Colors.white),
        Icon(Icons.notifications, size: 25, color: Colors.white),
        Icon(Icons.person_outline, size: 25, color: Colors.white)
      ],
      index: navegacionModel.paginaActual,
      onTap: (i) => navegacionModel.paginaActual = i,
    );}}
```

Si atendemos al ejemplo mostrado previamente del *ChangeNotifier*, se puede observar cómo hacemos la llamada al modelo de la clase NavegacionModel y saber en qué página se encuentra el usuario y cúal está solicitando para poder mostrarla.

### 12.2.8. Ejemplo de Código setState

```
CarouselSlider(
                items: imageSliders,
                options: CarouselOptions(
                    autoPlay: true,
                    enlargeCenterPage: true,
                    aspectRatio: 24 / 9,
                    onPageChanged: (index, reason) {
                      setState(() {
                        _current = index;
                      });
                    }),
```

```
        );
```

### 12.2.9. Ejemplo de Código Provider

```
import 'package:flights/src/models/dbuser_model.dart';
import 'package:flights/src/services/auth_service.dart';
import 'package:flutter/material.dart';

class ProfileServices with ChangeNotifier {
  int _bodySelected = 0;
  DbUser _dbUser;
  User _userUid;

  String _userName;
  String _nickName;
  String _frase;
  String _userWeb;
  String _phoneNumber;

  int _menuSelected = 0;

  int get bodySelected => this._bodySelected;

  set bodySelected(int valor) {
    this._bodySelected = valor;
    notifyListeners();
  }
  DbUser get dbUser => this._dbUser;

  set dbUser(DbUser valor) {
    this._dbUser = valor;
    notifyListeners();
  }
  int get menuSelected => this._menuSelected;

  set menuSelected(int valor) {
    this._menuSelected = valor;
    notifyListeners();
  }

  User get userUid => this._userUid;
```

```
set userUid(User valor) {
  this._userUid = valor;
  notifyListeners();
}

String get userName => this._userName;

set userName(String valor) {
  this._userName = valor;

  notifyListeners();
}

String get nickName => this._nickName;

set nickName(String valor) {
  this._nickName = valor;

  notifyListeners();
}

String get frase => this._frase;

set frase(String valor) {
  this._frase = valor;
  notifyListeners();
}

String get userWeb => this._userWeb;

set userWeb(String valor) {
  this._userWeb = valor;
  notifyListeners();
}
String get phoneNumber => this._phoneNumber;
set phoneNumber(String valor) {
  this._phoneNumber = valor;
  notifyListeners();
}
```

```
}
```

### 12.2.10. Ejemplo de Código *SubscriptionsRegister*

```
return db.doc(`users/${uid}`).update({
  subscriptionsRegister:admin.firestore.FieldValue.arrayUnion({
                    orig: req.query.orig.toUpperCase(),
                    dest: req.query.dest.toUpperCase(),
                    price: parseFloat(req.query.price),
                    dayInici: parseInt(req.query.dayInici),
                    monthInici: parseInt(req.query.monthInici),
                    yearInici: parseInt(req.query.yearInici),
                    dayFinal: parseInt(req.query.dayFinal),
                    monthFinal: parseInt(req.query.monthFinal),
                    yearFinal: parseInt(req.query.yearFinal)
            })})
```

### 12.2.11. Ejemplo de Código *onMessage*

```
 onMessage: (info) async {
      dynamic argumento = 'no-data';
      if (Platform.isAndroid) {
        argumento = info['notification']['body'] ?? 'no-data';
      } else {
        argumento = info['aps']['alert']['body'] ?? 'no-data-iOS';
      }
      final didRequestSeeNotification = await PlatformAlertDialog(
        title: 'New Notification', cancelActionText: 'Cancel',
        content: argumento, defaultActionText: 'See',
      ).show(context);
      if (didRequestSeeNotification == true) {
        _mensajesStreamController.sink.add(argumento);
      }
    }
```

### 12.2.12. Ejemplo de Código *onResume*

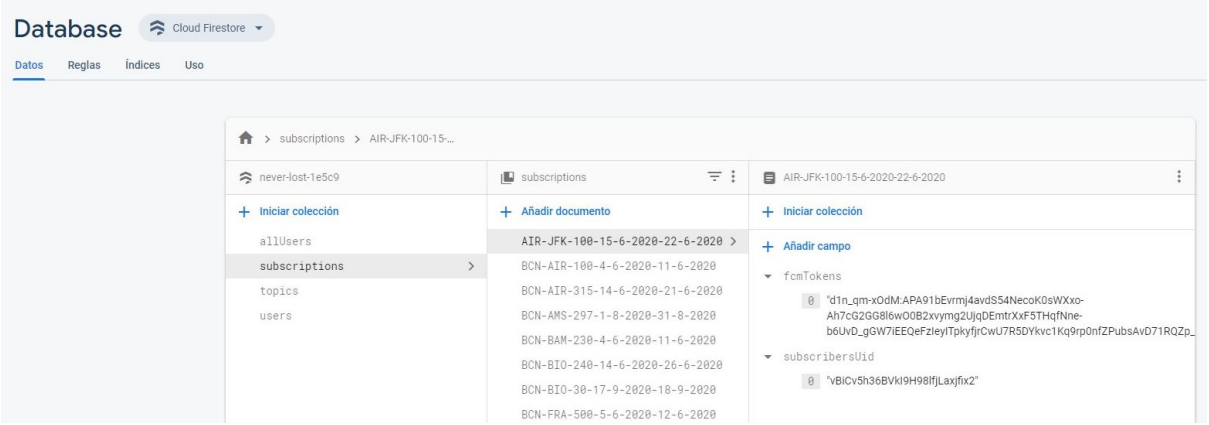
```
onResume: (info) async {
      String argumento = 'no-data';
      if (Platform.isAndroid) {
        argumento = info['notification']['body'] ?? 'no-data';
      } else {
        argumento = info['aps']['alert']['body'] ?? 'no-data-iOS';
      }
   _mensajesStreamController.sink.add(argumento);
}
```

### 12.2.13. Ejemplo de Código *onLaunch*

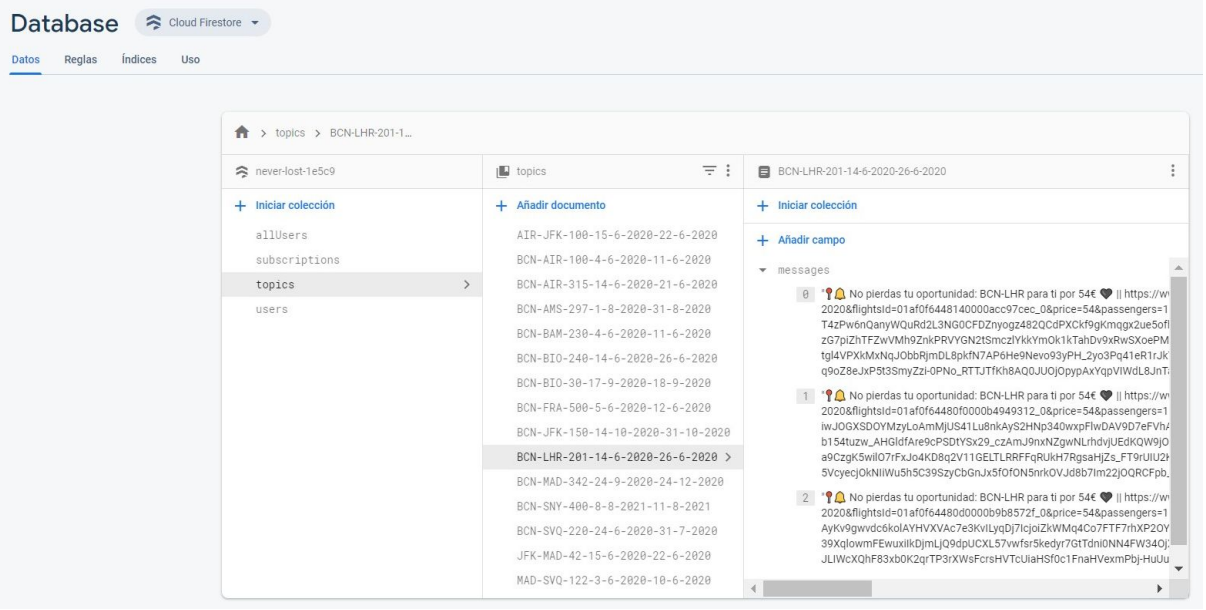
```
onLaunch: (info) async {
      _mensajesStreamController.sink.add(info['data']);
    }
```

## 12.3. Firestore

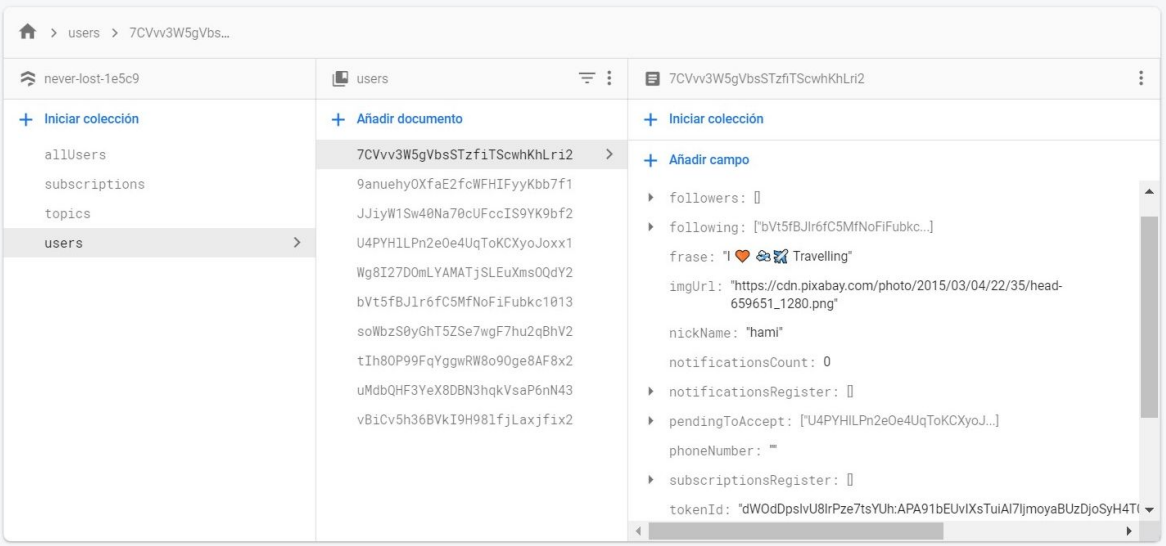
### 12.3.1. Subscriptions



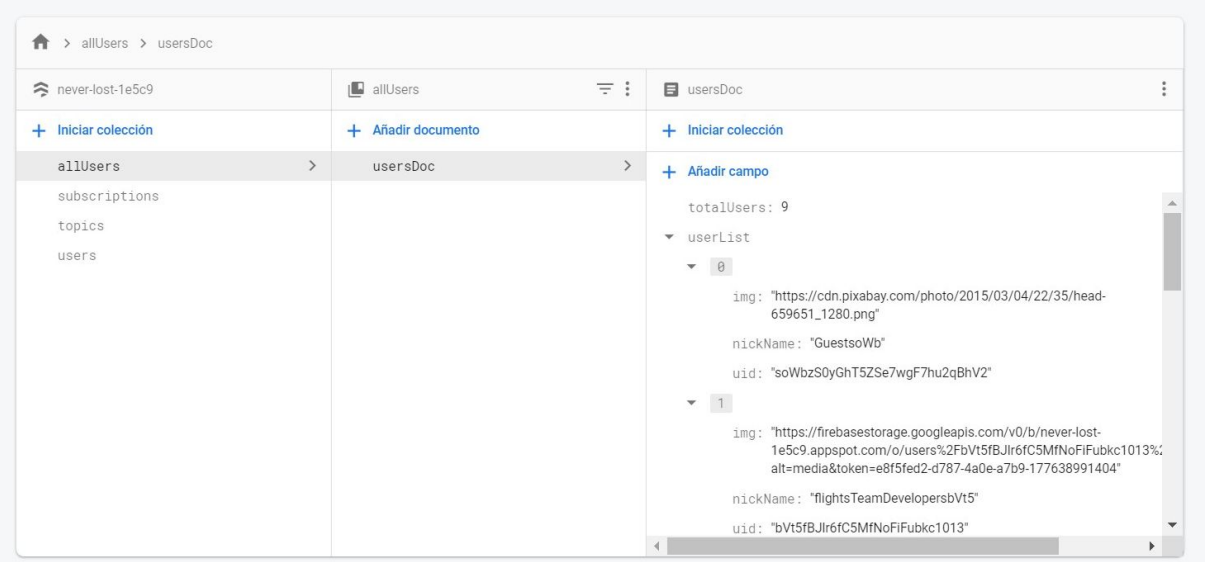

### 12.3.2. Topics




### 12.3.3. Users



### 12.3.4. allUsers




### 12.3.5. Ejemplo Subscriptions Document

```
{
    "fcmTokens": [
        "cWF3sWkgnUo:APA91bH0-8",
        "QKw3NcyJnUo:BsdwaWeQb-",
        "lshvijfih-CETMcm@3rtxe"
    ],
    "subscribersUid": [
        "JJiyW1Sw40Nl70cUFccIS9YK9bf2",
        "vBiCv5h36BVkI0H98lfjLaxjfix2",
        "U4PYHlLPn2eOe4UqToKCXpoJoxx1"
    ]}
```

### 12.3.6. Ejemplo Topics Document

```
{
    "messages": [
        "📍🔔 No pierdas tu oportunidad: BCN-LHR para ti por 54€ 🖤 || https://www.kiwi.com/deep?from=BCN&to=LHR",
        "📍🔔 No pierdas tu oportunidad: BCN-LHR para ti por 49€ 🖤 || https://www.kiwi.com/deep?from=BCN&to=LHR" ],    "msg_count": 2 }
```

### 12.3.7. Ejemplo Users Document


```
{
    "backgroundColor": "Colors.blue",
    "followers": [
        "Wg8I27DOmLYAMATLSLEuXmsOQdY2",
        "U4PjHlLPn2eOe4UqToKCXyoJoxx1"
    ],
    "following": [
        "I27DOmLYAMU4PjHlL90dasmJDDII"
    ],
    "frase": "I ❤ exploring 🌎",
    "imgUrl": "https://cdn.pixabay.com/photohead-659651_1280.png",
    "nickName": "Real Testers 👨🏻💻👨🏽💻",
    "notificationsCount": 2,
    "notificationsRegister": [
        {
            "deep_link": " https://www.kiwi.com/deep?from=BCN&to=LHR",
            "msg": "📍🔔 No pierdas: BCN-LHR para ti por 54€ 🖤 "
        },
        {
            "deep_link": " https://www.kiwi.com/deep?from=BCN&to=LHR",
            "msg": "📍🔔 No pierdas: BCN-LHR para ti por 49€ 🖤 "
        }
    ],
    "pendingToAccept": [
        "bVt5fBJlr6ZC5MfNoFiFubkc1013"
    ],
    "phoneNumber": "+34 643219044",
    "subscriptionsRegister": [
        {
            "dayFinal": 26,
            "dayInici": 14,
            "dest": "LHR",
            "monthFinal": 6,
            "monthInici": 6,
            "orig": "BCN",
            "price": 201
        },
        {
```

```
            "dayFinal": 12,
            "dayInici": 1,
            "dest": "MAD",
            "monthFinal": 10,
            "monthInici": 8,
            "orig": "BCN",
            "price": 35
        }
    ],
    "tokenId": "cwU4pNsXQsU:APA91bH6NXLyLWt",
    "uid": "JJi8I27DOmLYAMATjSEuXmsQdY2",
    "userEmail": "developer_testing@mail.com",
    "userName": "UFly Team",
    "userWeb": "www.uflyNow.com"
}
```


### 12.3.8. Ejemplo allUsers Document


```
{
    "totalUsers": 3,
    "userList": [
        {
            "img": "https://cdn.pixabay.com/photohead-659651_1280.png",
            "nickName": "GuestsoWb",
            "uid": "soWbzS0yGhT5ZSe7wgF7hu4qBhV2"
        },
        {
            "img": "https://cdn.pixabay.com/photohead-659651_1280.png",
            "nickName": "flightsTeamDevelopers",
            "uid": "bVt5fBJlr6fC5MfNoFiFubkc10163"
        },
        {
            "img": "https://cdn.pixabay.com/photohead-659651_1280.png",
            "nickName": "Testing Nick Names",
            "uid": "tIh8OP99FqYggwRz8o9Oge8AF8x2"
        }
    ]
}
```

## 12.4. Cloud Functions

### 12.4.1. Modelo User

```
export class User {
    img: string;
    nickName: string;
    uid: string;

    constructor(img: string, nickName: string, uid: string) {
        this.img = img;
        this.nickName = nickName;
        this.uid = uid;

    }}
```

### 12.4.2. Index.ts

#### 12.4.2.1. Imports

```
import * as functions from 'firebase-functions';
import * as admin from 'firebase-admin';

const  now = require("performance-now")
const request = require("request");
const {Storage} = require('@google-cloud/storage');
const storage = new Storage();

//myImports
import {AviationStack2} from './AviationStack2';
import {Airport} from './Airport';
import {ArrivalTimeTable} from './ArrivalTimeTable';
import {Airlines} from './Airlines';
import {Score, ScoreData} from './ScoreData';
import {TripAdvisorPollResp} from './TripAdvisorPollResp';
import {User} from './UserList';
```

```
import {KiwiResponse} from './kiwiResponse';
import {KiwiTicket} from './kiwiTickets';
//end myImports
export class Convert {
    public static toAviationStack2(json: string): AviationStack2 {
        return JSON.parse(json);
    }}
export class Converted {
    public static toAirlines(json: string): Airlines {
        return JSON.parse(json);
    }}
export class Convertion {
    public static toArrivalTimeTable(json: string): ArrivalTimeTable {
        return JSON.parse(json);
    }}
export class Converting {
    public static toAirport(json: string): Airport {
        return JSON.parse(json);
    }}
export class Conversion {
 public static toScore(json: string): Score[] {
     return JSON.parse(json);
 }
 public static scoreToJson(value: Score[]): string {
       return JSON.stringify(value);
   }}
export class ConvertTransf {
    public static toTripAdvisorPollResp(json: string):
TripAdvisorPollResp {
        return JSON.parse(json);
    }}
export class Convertir {
    public static toKiwiResponse(json: string): KiwiResponse {
        return JSON.parse(json);
    }}

admin.initializeApp();
const db = admin.firestore();
const fcm = admin.messaging();
const access_key = 'Our private API Key';
```

#### 12.4.2.2. onUserWelcome

```
// AUTOTRIGGER: Cuando un usuario se registra por primera vez
exports.onUserWelcome = functions.auth.user().onCreate(async (user) =>
{
  let userName4Display:string;
  let userName:string;
  let userUID = user.uid;
  if(user.displayName !== null){
    if(user.displayName !== undefined){
    userName = user.displayName;
    userName4Display = userName;
    }else{
      userName4Display = 'Guest';
    }}else{
      userName4Display = 'Guest';
    }
    userUID.split('');
    userName = `${userName4Display.replace(/
/g,"")}${userUID[0]}${userUID[1]}${userUID[2]}${userUID[3]}`;
  return db.doc(`users/${user.uid}`).create({
    uid: userUID,
    userEmail: user.email,
    userName: user.displayName == null? 'Guest': user.displayName,
    nickName: `${userName4Display.replace(/
/g,"")}${userUID[0]}${userUID[1]}${userUID[2]}${userUID[3]}`,
    imgUrl: user.photoURL == null?
'https://cdn.pixabay.com/photo/2015/03/04/22/35/head-659651_1280.png':
user.photoURL,
    frase: "I 🧡 🛫✈ Travelling",
    backgroundColor: "Colors.blue",
    tokenId: "12345678909876543mytokenidpendentdeagadar",
    following: [],
    followers: [],
    pendingToAccept:[],
    userWeb: "",
    phoneNumber: "",
    notificationsCount: 0,
    notificationsRegister: [],
```

```
    subscriptionsRegister: []
  }).then(()=>{
    if(user.email !== null){
    return db.doc('allUsers/usersDoc').set({
      totalUsers : admin.firestore.FieldValue.increment(+1),
      userList : admin.firestore.FieldValue.arrayUnion({
        uid: user.uid,
          img: user.photoURL == null?
'https://cdn.pixabay.com/photo/2015/03/04/22/35/head-659651_1280.png':
user.photoURL,
        nickName: userName
      })
    }, {merge:true});
  }else{
    console.log('Anonimous users like u dont fit in our
UserSystemList');
    return;
  }
  }).catch(err=>{
      console.log(err)
    }) })
```

#### 12.4.2.3. onUserGoodbye

```
// AUTOTRIGGER: Cuando un usuario es eliminado
exports.onUserGoodbye = functions.auth.user().onDelete((user,
context)=>{
  const uid = user.uid;
  let followers:[];
  let following:[];
  let subscriptions:[];
  let tokenId:string;
  let imgUrl:string;
  let subscripcio:string;

  return db.doc(`users/${uid}`).get().then((userDoc)=>{
    const data = userDoc.data();
    followers = data?.followers;
    following = data?.following;
```

```
  subscriptions = data?.subscriptionsRegister;
  tokenId = data?.tokenId;
  imgUrl = data?.imgUrl;
}).then(()=>{
  return db.doc(`users/${user.uid}`).delete().then(()=>{
    return db.doc('allUsers/usersDoc').get().then((list)=>{
      let img;
      let nickName;
      let found = false;
      const userList = list.data()?.userList;
      for (let index = 0; index < userList.length; index++) {
          if(userList[index].uid === uid){
             img = userList[index].img;
             nickName = userList[index].nickName;
            found = true;
          }}
      if(found == true){
        console.log(`DELETING USER: ${uid}`);
        return db.doc('allUsers/usersDoc').update({
          totalUsers: admin.firestore.FieldValue.increment(-1),
          userList: admin.firestore.FieldValue.arrayRemove({
            "img":img,
            "nickName":nickName,
            "uid":uid,
          }) })
      }else{
        return;
      }
    }).then(()=>{
      followers.forEach((follower)=>{
        return db.doc(`users/${follower}`).update({
          following: admin.firestore.FieldValue.arrayRemove(uid)
        })
      })
      following.forEach((followed)=>{
        return db.doc(`users/${followed}`).update({
          followers: admin.firestore.FieldValue.arrayRemove(uid)
        })
      })
      }).then(()=>{
```

```
          console.log(subscriptions);
          if(subscriptions !== undefined){
          subscriptions.forEach((subscripcioElement:any)=>{
           subscripcio =
`${subscripcioElement.orig}-${subscripcioElement.dest}-${subscripcioEle
ment.price}-${subscripcioElement.dayInici}-${subscripcioElement.monthIn
ici}-${subscripcioElement.yearInici}-${subscripcioElement.dayFinal}-${s
ubscripcioElement.monthFinal}-${subscripcioElement.yearFinal}`;
           console.log(subscripcio);
            return db.doc(`subscriptions/${subscripcio}`).update({
              fcmTokens:
admin.firestore.FieldValue.arrayRemove(tokenId),
              subscribersUid:
admin.firestore.FieldValue.arrayRemove(uid)
            })
          })
        }else{
            return;
          }
        }).then(()=>{
          if(imgUrl !==
'https://cdn.pixabay.com/photo/2015/03/04/22/35/head-659651_1280.png'){
          const path = `users/${uid}/avatar.png`;

          const bucket = storage.bucket('Project-root');
          const file = bucket.file(path)
          return file.delete().then(()=>{
            console.log('Everything was fine 🤭🙈');
            return;
          })
        }else{
          return;
        }})
      })
  }).catch(console.error);
})
```

#### 12.4.2.4. onAvatarChange

```
// AUTOTRIGGER: Cuando un usuario cambia su Imágen de Perfil
exports.onAvatarChanged = functions.storage.object().onFinalize((event)
=>{
  const object = event;
  const filePath = object.name;
  let userUid = filePath?.split('/')[1];
  console.log(`userUID 1st parse:   ${userUid}`);
  userUid = userUid?.split('/')[0];
  console.log(`userUID 2nd parse:   ${userUid}`);

  return db.doc(`users/${userUid}`).get().then((doc)=>{
    const docData = doc.data();
    const imgURL = docData?.imgUrl;
    console.log(`this is the new imgURL --> ${imgURL}`);
    return db.doc('allUsers/usersDoc').get().then((userDoc)=>{
      const usersData = userDoc.data();
      const userList = usersData?.userList;
      let userNickName:string='';
      let oldImg;
      for (let index = 0; index < userList.length; index++) {
        if(userList[index].uid === userUid){
           userNickName = userList[index].nickName;
           oldImg = userList[index].img;
        }}
      return db.doc('allUsers/usersDoc').update({
        userList : admin.firestore.FieldValue.arrayRemove({
          "img":oldImg,
          "nickName": userNickName,
          "uid":userUid,
        })}).then(()=>{
        console.log('USER IS GOING TO BE SAVED WITH NEW IMGURL ((:')
        return db.doc('allUsers/usersDoc').update({
          userList: admin.firestore.FieldValue.arrayUnion({
            "img":imgURL,
            "nickName": userNickName,
            "uid":userUid,
          })})})})}).catch(console.error) })
```

#### 12.4.2.5. searchUser

```
//ONCALL: Cuando se busca un usuario

exports.searchUser = functions.https.onCall(async (data, context) => {

  const userToFind = data.userToFind;
  const uid = context.auth?.uid;

  return db.doc('allUsers/usersDoc').get().then((allUsersDocu)=>{

    const allUsers = allUsersDocu.data()?.userList;
    let usersMatching: User[] = new Array();

    for (let index = 0; index < allUsers.length; index++) {

if(allUsers[index].nickName.toString().toUpperCase().includes(userToFin
d.toUpperCase())){
        if(allUsers[index].uid == uid){
          console.log('Users profile, no need to display')
        }else{
        usersMatching.push({img:allUsers[index].img,
nickName:allUsers[index].nickName ,uid: allUsers[index].uid});
        console.log(`${allUsers[index].nickName} inserted to the
ARRAY`);
      }}
      console.log(`Evaluating ${allUsers[index].nickName}`)
    }
    return usersMatching;

  })
  .catch(console.error);
})
```

#### 12.4.2.6. getUserProfileData

```
//ONCALL: Cuando se ha encontrado un usuario y se quiere obtener su
información
exports.getUserProfileData = functions.https.onCall(async (data,
context) => {
  const reqUser = data.uid;
  const getUid = data.getUid;

  return db.doc(`users/${getUid}`).get().then((userInfo)=>{
    const userData = userInfo.data();
    for (let index = 0; index < userData?.followers.length; index++) {
      if(userData?.followers[index] == reqUser){
        return {
          nickName: userData?.nickName,
          imgUrl: userData?.imgUrl,
          backgroundColor: userData?.backgroundColor,
          frase: userData?.frase,
          followers: userData?.followers,
          following: userData?.following,
          friendOrNot: "true"
        };
      }}
    for (let index = 0; index < userData?.pendingToAccept.length;
index++) {
      if(userData?.pendingToAccept[index] == reqUser){
        return {
          nickName: userData?.nickName,
          imgUrl: userData?.imgUrl,
          backgroundColor: userData?.backgroundColor,
          frase: userData?.frase,
          followers: userData?.followers,
          following: userData?.following,
          friendOrNot: "pendent"
        };
      }}
      return {
        nickName: userData?.nickName,
        imgUrl: userData?.imgUrl,
```

```
        backgroundColor: userData?.backgroundColor,
        frase: userData?.frase,
        followers: userData?.followers,
        following: userData?.following,
        friendOrNot: "false"  };})})
```

#### 12.4.2.7. getUserInfo

```
//ONCALL: Cuando se quiere cargar la información del usuario

exports.getUserInfo = functions.https.onCall(async (data,
context:any)=>{

  const reqUser = data.uid.toString();
  const userId = context.auth.uid.toString();
  //NEED TO STORE THE TOKEN!
  //security lvl.1 checking
  if(reqUser === userId){
    try{
      return db.doc(`users/${userId}`).get()
      .then((doc)=>{
        console.log(doc.data());
        return doc.data();
      });

    }catch(e){
      console.log(e);
      return 'Error in the CLOUD FUNC :()'
    }
    //userTryingToFake a req UID
}else{
  return 'GLMF';
  }
})
```

#### 12.4.2.8. updateUserInfo

```
//ONCALL: Cuando un usuario desea actualizar su información de la BD
exports.updateUserInfo = functions.https.onCall(async (data,
context:any)=> {
  const userId = context.auth.uid.toString();
  const reqUser = data.uid.toString(); //ON REQUEST: req.query.uid;
  const nickName = data.nickName;      //ON REQUEST:
req.query.nickName;
  const userName = data.userName;//req.query.userName;
  const frase = data.frase;//req.query.frase;
  const phoneNumber = data.phoneNumber;//req.query.phoneNumber;
  const userWeb = data.userWeb;//req.query.userWeb;
  let notFound = true; //si el trobo aleshores --> FALSE
  console.log(userId, reqUser, nickName, userWeb, userName,
phoneNumber, frase);

  if(userId === reqUser){
    return new Promise((resolve, reject)=>{
      return db.doc('allUsers/usersDoc').get()
     .then((usersDoc) => {
       const usersList =  usersDoc.data()?.userList;
       for (let i = 0; i < usersList.length; i++) {
           if(usersList[i].nickName === nickName && usersList[i].uid
!== reqUser){
             console.log(`${usersList[i].nickName} == ${nickName} (?)`)
             notFound = false;
           }
            console.log(i);
       }
       if(notFound == true){
         console.log('not Found == TRUE');
         return notFound;
       }else{
         return notFound;
       }
     }).then(async(found)=>{
       console.log(found);
```

```
        if(found === true){
          console.log('going toChangeList');
          const p1 = await changeFromList(reqUser, nickName);
          console.log('going toChangeUser');
          const p2 = await changeFromUser(reqUser, nickName);
          return Promise.all([p1,p2]).then(async()=>{
            console.log(userWeb, userName, phoneNumber, frase);
            await db.doc(`users/${reqUser}`).update({
              userWeb: userWeb.toString(),
              userName: userName.toString(),
              phoneNumber: phoneNumber.toString(),
              frase: frase.toString()
            })
      .then(()=>{
              console.log('TOT A ANAT PERFECTE');
              resolve(true);
              });
          })
        }else{
          console.log('EUREKA arribo aqui (:');
          resolve(false);
        }});
    }).then((val)=>{
      console.log(val);
      return val;
    })
}else{
  return 'GLMF';
}})

  async function changeFromList(uid:string, newNickName:string){
    let img:any;
    let oldNickName:string;
    return db.doc('allUsers/usersDoc').get().then((list)=>{

      const mylist = list.data()?.userList;
      for (let index = 0; index < mylist.length; index++) {
        if(mylist[index].uid == uid){
          img = mylist[index].img;
          oldNickName = mylist[index].nickName;        }}
```

```
      return db.doc('allUsers/usersDoc').update({
        userList : admin.firestore.FieldValue.arrayRemove({
          "img":img,
          "nickName": oldNickName,
          "uid":uid,
        })
      }).then(()=>{
        return db.doc('allUsers/usersDoc').update({
          userList: admin.firestore.FieldValue.arrayUnion({
            "img":img,
            "nickName": newNickName == null? oldNickName : newNickName,
            "uid":uid,
          })
        })})
    });
  }
  async function changeFromUser(uid:string, newNickName:string){
    if(newNickName !== null){
    return db.doc(`users/${uid}`).update({
      nickName: newNickName
    });
  }else{
      return;
    }}
```

#### 12.4.2.9. sendDeviceToken

```
//ONCALL: Establecer el Token Id del dispositivo en la BD

exports.sendDeviceToken = functions.https.onCall(async (data, context)
=> {
  const uid = context.auth?.uid;
  return db.doc(`users/${uid}`).update({
    tokenId: data.tokenId
  }).then(()=>{
    console.log('Done');
    return true;
  }).catch(console.error);
})
```

#### 12.4.2.10. askForFriendship

```
//ONCALL: Cuando un usuario solicita una petición de amistad a otro

exports.askForFriendship = functions.https.onCall(async (data, context)
=> {
  const uid = data.uid;
  const suposedFriend =  data.futureFriendUid;
//req.query.futureFriendUid;
  return db.doc(`users/${suposedFriend}`).update({
    pendingToAccept: admin.firestore.FieldValue.arrayUnion(uid)
  }).then(()=>{
    console.log('Petition submitted');
    return true;
  }).catch(console.error);
})
```

#### 12.4.2.11. cancelAskForFriendship

```
//ONCALL: Cuando un usuario desea anular una amistad

exports.cancelAskForFriendship = functions.https.onCall(async (data,
context) => {

  const uid =  data.uid;       //req.query.uid;
  const cancelFriend =  data.cancelFriend;    //req.query.cancelFriend;

  return db.doc(`users/${cancelFriend}`).update({
    pendingToAccept: admin.firestore.FieldValue.arrayRemove(uid)
  }).then(()=>{
    console.log(`${cancelFriend} will no be able to accept your petiton
for a friendship`);
    return true;
  }).catch(console.error);
})
```

#### 12.4.2.12. acceptFriend

```
//ONCALL: Cuando se acepta una solicitud de seguimiento
exports.acceptFriend = functions.https.onCall(async (data, context) =>
{
  const uid = data.uid;                          //req.query.uid;
  const friendAccepted = data.newFriend;

  return db.doc(`users/${uid}`).update({
    pendingToAccept:
admin.firestore.FieldValue.arrayRemove(friendAccepted),
    followers: admin.firestore.FieldValue.arrayUnion(friendAccepted)
  }).then(()=>{
    return db.doc(`users/${friendAccepted}`).update({
      following: admin.firestore.FieldValue.arrayUnion(uid)
    }).then(()=>{
      console.log(`Now you allowed ${friendAccepted} to be your friend,
you are: ${uid}`);
      return true;
    })
  }).catch(console.error);
})
```

#### 12.4.2.13. declineFriend

```
//ONCALL: Cuando se quiere denegar una petición de seguimiento

export const declineFriend = functions.https.onCall(async (data,
context) => {
  const uid = data.uid;           //req.query.uid;
  const noFriend =  data.noFriend;       // req.query.noFriend;

  return db.doc(`users/${uid}`).update({
    pendingToAccept: admin.firestore.FieldValue.arrayRemove(noFriend)
  }).then(()=>{
    console.log(`${noFriend} is like shit! I dont want it as
friend...`);
    return true;
  }).catch(console.error);})
```

#### 12.4.2.14. deleteFriend

```
//ONCALL: Cuando un usuario quiere dejar de seguir a otro (Unfollow)

export const deleteFriend = functions.https.onCall(async (data,
context) => {

  const uid =  data.uid;                    //req.query.uid;
  const noMoreFriend = data.noMoreFriend; //req.query.noMoreFriend;

  return db.doc(`users/${uid}`).update({
    following: admin.firestore.FieldValue.arrayRemove(noMoreFriend),
  }).then(()=>{
    console.log(`Ya no te sigo ${noMoreFriend}`);
    return db.doc(`users/${noMoreFriend}`).update({
      followers: admin.firestore.FieldValue.arrayRemove(uid)
    }).then(()=>{
      console.log('-1 AMIGO :((');
      return true;
    })
  }).catch(console.error);
})
```

#### 12.4.2.15. getUsersPendingToAccept

```
//ONCALL: Conseguir la lista de usuarios pendientes de acceptar

export const getUsersPendingToAccept = functions.https.onCall(async
(data, context) => {

  const uid =  data.uid;     //req.query.uid;
  let pdtList:[];
  let finalArray:any[] = new Array;

  return db.doc(`users/${uid}`).get().then((doc)=>{
    const docData = doc.data();
     pdtList = docData?.pendingToAccept;
  }).then(()=>{
    return db.doc('allUsers/usersDoc').get().then((usersDoc)=>{
      const usersData = usersDoc.data();
```

```
      for (let index = 0; index < pdtList.length; index++) {
        usersData?.userList.forEach((user:any)=>{
          if(pdtList[index] == user.uid){
            finalArray.push({img: user.img, nickName: user.nickName,
uid: user.uid, frase: ''});
          // }else{
          //   console.log(user);
          }
        })
      }
    }).then(()=>{
      console.log(finalArray);
      return finalArray;
    })
  }).catch(console.error);})
```

#### 12.4.2.16. getFlightInfo

```
//ONREQUEST: Cuando se busca un vuelo en función de su ID
exports.getFlightInfo = functions.https.onRequest(async (req, resp)=>{
  let respostaFuncio:any;
  const flightId = req.query.flightId;
  // let vols:number = 0;
  let depCity:any;
  let arrCity:any;
  const optionAviationStack = {
    method: 'GET',
    url:
`http://api.aviationstack.com/v1/flights?access_key=${access_key}&fligh
t_iata=${flightId}`,
  };

  try{
     await request(optionAviationStack, async
function(error:any,algo:any, body:any){
       console.log(body);
      const flightAS:AviationStack2 = Convert.toAviationStack2(body);
      console.log(flightAS);
```

```
        respostaFuncio = flightAS.data[0];
        depCity = respostaFuncio.departure.iata;
        arrCity = respostaFuncio.arrival.iata;
        depCity = await getCityName(depCity);
        arrCity = await getCityName(arrCity);

      await Promise.all([depCity, arrCity]).then(()=>{
        return resp.send({flightInfo:respostaFuncio , depCity: depCity,
arrCity: arrCity});
      });
});

}catch(e){
  console.log(e.toString());
}
})

  async function getCityName(iataAirport:string){
   let cityName = iataAirport;
   const optionCity = {
     method: 'GET',
     url: 'https://airport-info.p.rapidapi.com/airport',
     qs: {iata: iataAirport},
     headers: {
       'x-rapidapi-host': 'airport-info.p.rapidapi.com',
       'x-rapidapi-key': 'Rapid_API_KEY'
     }
     };

   return new Promise(function(resolve, reject){
     request.get(optionCity, cityName,
function(error:any,something:any, body:any){
       if(error){
         console.log(error.toString());
         reject(error);
       }else{
         console.log(body);
         const airportInfo:Airport = Converting.toAirport(body);
         const cityInfo = {
           airportName: airportInfo.name,
```

```
            latitude: airportInfo.latitude,
            longitude: airportInfo.longitude,
          };

          resolve(cityInfo);
        }
      })
    }) }
```

#### 12.4.2.17. getArrivalTimeTabel

```
//ONREQUEST: Cuando se solicitan las llegadas (ARRIVALS)

exports.getArrivalTimeTable = functions.https.onRequest(async (req,
resp)=>{

  let respostaFuncio:any;
  const airport = req.query.airport;
  const limit = 50;
  const options = {
    method: 'GET',
    url:
`http://api.aviationstack.com/v1/flights?access_key=${access_key}&limit
=${limit}&arr_iata=${airport}&flight_status=active`,   }
  try{
    await request(options, function(error:any,response:any,body:any){
      const airportAS = Convertion.toArrivalTimeTable(body);
      respostaFuncio = airportAS.data;
      return resp.send({arrivalInfo:respostaFuncio});
    })
  }catch(e){
    console.log(e.toString());
  }
})
```

#### 12.4.2.18. getDepartureTimeTable

```
//ONREQUEST: Cuando se solicitan las salidas (DEPARTURES)

exports.getDepartureTimeTable = functions.https.onRequest(async (req,
resp)=>{

  let respostaFuncio:any;
  const airport = req.query.airport;
  const limit = 50;
  const options = {
    method: 'GET',
    url:
`http://api.aviationstack.com/v1/flights?access_key=${access_key}&limit
=${limit}&dep_iata=${airport}&flight_status=active`,
  }
  try{
    await request(options, function(error:any,response:any,body:any){
      const airportAS = Convertion.toArrivalTimeTable(body);
      respostaFuncio = airportAS.data;
      return resp.send({departureInfo:respostaFuncio});
    })
  }catch(e){
    console.log(e.toString());
  }
})
```

#### 12.4.2.19. getAirlineInfo

```
//ONREQUEST: Cuando se solicitan información de una aerolínea

exports.getAirlineInfo = functions.https.onRequest(async (req, resp)=>{

  let airlineSearch:string = req.query.airline;

  const optionAirline = {
      method: 'GET',
      url:
`http://api.aviationstack.com/v1/airlines?access_key=${access_key}`,
      headers: {
      'content-type': 'application/json' }  };

  try{
```

```
    await request(optionAirline, function(error:any, notShadow:any,
body:any){
        const airlinesDoc = Converted.toAirlines(body);
        console.log(airlinesDoc);
        const airlinesList = airlinesDoc.data;
        let i = 0;
        let trobat = 0;
        while(i < airlinesList.length && trobat == 0){
          let anal = airlinesList[i].airline_name;
          console.log(anal + ' ' + i);
          if(anal.toLowerCase() == airlineSearch.toLowerCase()){
            console.log(airlinesList[i].toString());
            trobat = 1;
          }else{
            i++;
          }
        }
        if(trobat === 1){
          return resp.send({airlineData: airlinesList[i]});
        }else{
          return resp.send({airlineData: 'NO match found'});
        }})
  }catch(e){
    console.log(e.toString());
  }
})
```

#### 12.4.2.20. getAirportInfo

```
//ONREQUEST: Cuando se solicita información del aeropuerto (info
airport + ARRIVALS + DEPARTURES)
exports.getAirportInfo = functions.https.onRequest(async (req, resp)=>{

  const airport = req.query.airport;
  const optionCity = {
    method: 'GET',
    url: 'https://airport-info.p.rapidapi.com/airport',
    qs: {iata: airport},
    headers: {
```

```
      'x-rapidapi-host': 'airport-info.p.rapidapi.com',
      'x-rapidapi-key': 'Rapid_API_KEY'
    }
  };

  await request(optionCity, async function(error:any,something:any,
body:any){
    const airportInfo:Airport = Converting.toAirport(body);
    const myInfo = airportInfo;
    const arrivals = await getAirportArrivals(airport);
    const departures = await getAirportDepartures(airport);

    await Promise.all([arrivals, departures]).then(()=>{
      resp.send({airportInfo: myInfo, airportDepartures: departures,
airportArrivals: arrivals});

    })
  })
})

  async function getAirportDepartures(airport:string){

    const limit = 20;
    const optionAirport = {
        method: 'GET',
        url:
`http://api.aviationstack.com/v1/flights?access_key=${access_key}&limit
=${limit}&dep_iata=${airport}&flight_status=active`,       }

    return new Promise(function(resolve, reject){
      request.get(optionAirport, airport,
function(error:any,something:any, body:any){
        if(error){
          console.log(error.toString());
          reject(error);
        }else{
          const airportInfo:ArrivalTimeTable =
Convertion.toArrivalTimeTable(body);
          let departures = airportInfo.data;
          console.log('DEPARTURES');
```

```
          console.log(departures);
          if(departures === undefined){
            console.log('I got departures === undefined');
            return;
          }else{
          departures.sort(function (a,b){
            let h1 = new Date(a.departure.estimated).getTime();
            let h2 = new Date(b.departure.estimated).getTime();
            if(h1 > h2){
              return 1;}
            if(h1 < h2){
            return -1;}
            return 0;
          });

          resolve(departures);
        }}
      })})}


  async function getAirportArrivals(airport:string){

    const limit = 20;
    const optionAirport = {
        method: 'GET',
        url:
`http://api.aviationstack.com/v1/flights?access_key=${access_key}&limit
=${limit}&arr_iata=${airport}&flight_status=active`,
      }

    return new Promise(function(resolve, reject){
      request.get(optionAirport, airport,
function(error:any,something:any, body:any){
        if(error){
          console.log(error.toString());
          reject(error);
        }else{
          const airportInfo:ArrivalTimeTable =
Convertion.toArrivalTimeTable(body);
          const arrivals = airportInfo.data;
```

```
          console.log('ARRIVALS');
          console.log(arrivals);

          if(arrivals === undefined){
            console.log('I got arrivals === undefined');
            return;
          }else{
          arrivals.sort(function (a,b){
            let h1 = new Date(a.arrival.estimated).getTime();
            let h2 = new Date(b.arrival.estimated).getTime();
            if(h1 > h2){
              return 1;}
            if(h1 < h2){
            return -1;}
            return 0;
          });

          resolve(arrivals);
        }}
      })
    })

  }
```

#### 12.4.2.21. getFirebaseAirport

```
exports.getFirebaseAirport = functions.https.onRequest(async
(req,resp)=>{

  const airport = req.query.airport;
  const airportDoc = req.query.firebase;
    try{
      const airDoc =  await
db.collection('AirData').doc(`${airportDoc}`).get();
        if(airDoc.exists){
          const json = airDoc.data();
          if(json!==undefined){
            Object.keys(json).forEach(async(ele)=>{
              if(ele.toLowerCase() === airport.toLowerCase() &&
json!==undefined){
```

```
                resp.send({airportData:json[ele]})
              } })
          }
        }else{
          console.log('Doc does not exists');
        }
    }catch(e){
      console.log(e.toString());
      resp.send(e.toString());
    }

});
```

#### 12.4.2.22. getCityData

```
//ONREQUEST: Cuando se solicita información de una ciudad

exports.getCityData = functions.https.onRequest(async (req,resp)=>{
  let t0 = now();
  const cityName = req.query.city;
  const optionCity = {
      method: 'GET',
      url:
`https://api.teleport.org/api/cities/?search=${cityName}&embed=city:sea
rch-results/city:item/city:country&embed=city_search-results[0]/city:it
em`,
      headers: {
        "templated": true
      }};

    try{
      await request(optionCity,  async
function(error:any,something:any, body:any){
       const jsonDoc = JSON.parse(body);
       let t1 = now()
       console.log('Time to get Teleport Data: '+ (t0-t1).toFixed(3));
       const vari = jsonDoc._embedded["city:search-results"];
       const vari2 = vari[0]._embedded["city:item"];
       const scoreUrl = vari2._links["city:urban_area"].href;
```

```
        const scoreData = await getCityScores(scoreUrl);
        const images = await getCityImages(cityName);

      await Promise.all([scoreData, images]).then(()=>{
        let t2 = now()
        console.log('Time to get send resp: ' + (t2-t0).toFixed(3));
        resp.send({
          cityName: vari2.name,
          cityPopulation: vari2.population,
          cityGeoname_id: vari2.geoname_id,
          countryName: vari2._embedded["city:country"].name,
          countryIso3: vari2._embedded["city:country"].iso_alpha3,
          countryPopulation:
vari2._embedded["city:country"].population,
          countryCurrency:
vari2._embedded["city:country"].currency_code,
          countryGeoname_id:
vari2._embedded["city:country"].geoname_id,
          scoreData: scoreData,
          images: images
          })
        })
      })
  }catch(e){
    console.log(e.toString());
  }
})

  async function getCityScores(url:string):Promise<ScoreData>{
    let t3 = now();
    const cityUrl = {
      method: 'GET',
      url: url + 'scores',     }

    return new Promise(function(resolve, reject){
      request.get(cityUrl, function(error:any,something:any, body:any){
        if(error){
          console.log(error.toString());
          reject(error);
        }else{
```

```
            const data = JSON.parse(body);
            let summ = data.summary.replace(/<\/?[^>]+(>|$)/g,'');
            summ = summ.replace(/[\n\r]/g,'');
            summ = summ.replace('    ','');
            const jsonCat = Conversion.scoreToJson(data.categories);
            const cat = Conversion.toScore(jsonCat);
            const resolving = {
              scoreInfo : cat,
              summary   : summ
            }
            let t4 = now()
            console.log('Time to resolve City Scores: '+
(t4-t3).toFixed(3));
            resolve(resolving);
          }
        })
      })
    }

    async function getCityImages(cityName:string){
      let t5 = now()
      const imgNum = 7;
      const imgUrl = {
          method: 'GET',
          url:
`https://api.pexels.com/v1/search?query=${cityName}&per_page=${imgNum}&
page=1`,
          headers: {
            'Authorization': 'Authoritzation_String',
          }
        }

      return new Promise(function(resolve, reject){
        request.get(imgUrl, function(error:any,something:any, body:any){
          if(error){
            console.log(error.toString());
            reject(error);
          }else{
            const data = JSON.parse(body);
            let picsArr:string[] = [];
```

```
          for (let i = 0; i < data.photos.length; i++) {
            if(i == 6 || data.photos[i] == null || data.photos[i] ==
undefined){
              break;
            }
            picsArr.push(data.photos[i].src.portrait);
          }
          let t6 = now()
          console.log('Time to resolve Pexel Images: '+
(t6-t5).toFixed(3));
          resolve(picsArr);
        }
      })
    })
  }
```

#### 12.4.2.23. setNotification

```
//ONREQUEST: Se activa cuando un usuario desea guardar un tópico,
analiza si existe o si debe crear uno nuevo, pues no existe en nuestra
BD

exports.setNotification = functions.https.onRequest(async (req,
resp)=>{

  const uid = req.query.uid;

  const newTopic =
`${req.query.orig.toUpperCase()}-${req.query.dest.toUpperCase()}-${pars
eFloat(req.query.price)}-${req.query.dayInici}-${req.query.monthInici}-
${req.query.yearInici}-${req.query.dayFinal}-${req.query.monthFinal}-${
req.query.yearFinal}`;

  // if(uid == reqUid){

  const userData = await db.doc(`users/${uid}`).get();

  Promise.all([userData]).then(()=>{
    const tokenId = userData.data()?.tokenId;
```

```
    return db.doc(`topics/${newTopic}`).get().then((doc)=>{
      if (!doc.exists) {
       console.log('No topics, so lets CREATE IT!');
       return db.doc(`topics/${newTopic}`).create({
         messages: [],
         msg_count: 0
       }).then(()=>{
         return db.doc(`subscriptions/${newTopic}`).create({
           subscribersUid: [uid],
           fcmTokens: [tokenId]
         }).then(()=>{
           return db.doc(`users/${uid}`).update({
             subscriptionsRegister:
admin.firestore.FieldValue.arrayUnion({
               orig: req.query.orig.toUpperCase(),
               dest: req.query.dest.toUpperCase(),
               price: parseFloat(req.query.price),
               dayInici: pfarseInt(req.query.dayInici),
               monthInici: parseInt(req.query.monthInici),
               yearInici: parseInt(req.query.yearInici),
               dayFinal: parseInt(req.query.dayFinal),
               monthFinal: parseInt(req.query.monthFinal),
               yearFinal: parseInt(req.query.yearFinal)
             })
           }).then(()=>{
              resp.send('TOPIC CREATED, USER SUBSCRIBE IT & user has
recorded (:');
           })
         })
       })

     } else {
       console.log('We found the document, so lets SUBSCRIBE to
TOPIC');
       return db.doc(`subscriptions/${newTopic}`).update({
         subscribersUid: admin.firestore.FieldValue.arrayUnion(uid),
         fcmTokens: admin.firestore.FieldValue.arrayUnion(tokenId)
       }).then(()=>{
         return db.doc(`users/${uid}`).update({
```

```
            subscriptionsRegister:
admin.firestore.FieldValue.arrayUnion({
              orig: req.query.orig.toUpperCase(),
              dest: req.query.dest.toUpperCase(),
              price: parseFloat(req.query.price),
              dayInici: req.query.dayInici,
              monthInici: req.query.monthInici,
              yearInici: req.query.yearInici,
              dayFinal: req.query.dayFinal,
              monthFinal: req.query.monthFinal,
              yearFinal: req.query.yearFinal
            })
          }).then(()=>{
            resp.send('USER WAS SUBSCRIBED, TOPIC WAS ALREADY CREATED
(:')
          })
        })

      }
     });

  }).catch(console.error);
})
```

#### 12.4.2.24. deleteNotification

```
//ONREQUEST: Se encarga de eliminar un tópico, la suscripción y el
registro (user) correspondiente

exports.deleteNotification = functions.https.onRequest(async (req,
resp) => {
  const uid = req.query.uid;
  const topic =
`${req.query.org.toUpperCase()}-${req.query.dest.toUpperCase()}-${parse
Float(req.query.price)}-${req.query.dayInici}-${req.query.monthInici}-$
{req.query.yearInici}-${req.query.dayFinal}-${req.query.monthFinal}-${r
eq.query.yearFinal}`
  let userToken:string;
```

```
  let elementToDelete = {
    dayFinal: parseInt(req.query.dayFinal),
    dayInici: parseInt(req.query.dayInici),
    dest: req.query.dest.toUpperCase(),
    monthFinal: parseInt(req.query.monthFinal),
    monthInici: parseInt(req.query.monthInici),
    orig: req.query.org.toUpperCase(),
    price: parseFloat(req.query.price),
    yearFinal: parseInt(req.query.yearFinal),
    yearInici: parseInt(req.query.yearInici),
  }
  console.log(elementToDelete);
  return db.doc(`users/${uid}`).get().then((user)=>{
    const userData = user.data();
    userToken = userData?.tokenId;
    console.log(userToken);
  }).then(()=>{
    return db.doc(`subscriptions/${topic}`).update({
      fcmTokens: admin.firestore.FieldValue.arrayRemove(userToken),
      subscribersUid: admin.firestore.FieldValue.arrayRemove(uid)
    }).then(()=>{
      return db.doc(`subscriptions/${topic}`).get().then((topicData)=>{
        const topicInfo = topicData.data();
        if(topicInfo?.fcmTokens.length === 0 &&
topicInfo.subscribersUid.length === 0){
          return db.doc(`topics/${topic}`).delete().then(()=>{
            return db.doc(`subscriptions/${topic}`).delete().then(()=>{
              return db.doc(`users/${uid}`).get().then((userData)=>{
                const userInfo = userData.data();
                const myTopics = userInfo?.subscriptionsRegister;
                console.log('MYTOPICS:', myTopics)
                for (let index = 0; index < myTopics.length; index++) {
                  console.log(myTopics[index].dayFinal,
parseInt(req.query.dayFinal), myTopics[index].dayInici,
parseInt(req.query.dayInici) , myTopics[index].dest,
req.query.dest.toUpperCase(),
myTopics[index].monthFinal,parseInt(req.query.monthFinal),
myTopics[index].monthInici,parseInt(req.query.monthInici),
myTopics[index].orig,req.query.org.toUpperCase(),parseFloat(myTopics[in
dex].price),parseFloat(req.query.price),
```

```
myTopics[index].yearFinal,parseInt(req.query.yearFinal),
myTopics[index].yearInici,parseInt(req.query.yearInici))
                  if(myTopics[index].dayFinal ==
parseInt(req.query.dayFinal) && myTopics[index].dayInici==
parseInt(req.query.dayInici) && myTopics[index].dest ==
req.query.dest.toUpperCase() &&
myTopics[index].monthFinal==parseInt(req.query.monthFinal) &&
myTopics[index].monthInici==parseInt(req.query.monthInici) &&
myTopics[index].orig==req.query.org.toUpperCase() &&
parseFloat(myTopics[index].price) == parseFloat(req.query.price) &&
myTopics[index].yearFinal==parseInt(req.query.yearFinal) &&
myTopics[index].yearInici==parseInt(req.query.yearInici)){
                    console.log('IM IN THE RIGHT ITERATION (:)');
                    return db.doc(`users/${uid}`).update({
                      subscriptionsRegister:
admin.firestore.FieldValue.arrayRemove(myTopics[index])
                    }).then(()=>{
                      resp.send('User subscription was deleted ;)');
                    })
                  }
                }
                return;
              })})})
        }else{
           console.log('Still having subscriptions');
           return db.doc(`users/${uid}`).get().then((userData)=>{
             const userInfo = userData.data();
             const myTopics = userInfo?.subscriptionsRegister;
             console.log('MYTOPICS:', myTopics)
             for (let index = 0; index < myTopics.length; index++) {
               console.log(myTopics[index].dayFinal,
parseInt(req.query.dayFinal), myTopics[index].dayInici,
parseInt(req.query.dayInici) , myTopics[index].dest,
req.query.dest.toUpperCase(),
myTopics[index].monthFinal,parseInt(req.query.monthFinal),
myTopics[index].monthInici,parseInt(req.query.monthInici),
myTopics[index].orig,req.query.org.toUpperCase(),parseFloat(myTopics[in
dex].price),parseFloat(req.query.price),
myTopics[index].yearFinal,parseInt(req.query.yearFinal),
myTopics[index].yearInici,parseInt(req.query.yearInici))
```

```
                if(myTopics[index].dayFinal ==
parseInt(req.query.dayFinal) && myTopics[index].dayInici==
parseInt(req.query.dayInici) && myTopics[index].dest ==
req.query.dest.toUpperCase() &&
myTopics[index].monthFinal==parseInt(req.query.monthFinal) &&
myTopics[index].monthInici==parseInt(req.query.monthInici) &&
myTopics[index].orig==req.query.org.toUpperCase() &&
parseFloat(myTopics[index].price) == parseFloat(req.query.price) &&
myTopics[index].yearFinal==parseInt(req.query.yearFinal) &&
myTopics[index].yearInici==parseInt(req.query.yearInici)){
                  console.log('IM IN THE RIGHT ITERATION (:)');
                  return db.doc(`users/${uid}`).update({
                    subscriptionsRegister:
admin.firestore.FieldValue.arrayRemove(myTopics[index])
                  }).then(()=>{
                    resp.send('User subscription was deleted ;)');
                  })
                }else{console.log('sorry')}
              }
              return;
            })}})
      })
  }).catch(console.error)
})
```

#### 12.4.2.25. deliverTopicMessage

```
//AUTOTRIGGER: Se encarga de analizar si se ha añadido un nuevo MSG en
el tópico, de ser así procede al envío de las notificaciones

exports.deliverTopicMessage =
functions.firestore.document('topics/{topicName}').onUpdate((change,
context)=>{
  const topic = context.params.topicName;
  const newValue = change.after.data();
  const oldValue = change.before.data();
  const message = newValue?.messages;
  const oldCounter = oldValue?.msg_count;
  const newCounter = newValue?.msg_count;
```

```
  let uidList:[];
  if(newCounter > oldCounter){
  const messageRecived = message.pop();
  const arrMsg = messageRecived.split('||');
  const messageToSend = arrMsg[0];
  const link = arrMsg[1];
  console.log(arrMsg);

  const payload : admin.messaging.MessagingPayload ={
    notification: {
      title: 'Your Trip',
      body: messageToSend,
      icon:
'https://cdn.clipart.email/9d52b5a36143d2fd830cce1f67bf82bf_universal-h
oliday-travel-route-world-travel-vacation-travel-png-_650-651.jpeg',
      clickAction: 'FLUTTER_NOTIFICATION_CLICK'
    }
  }

  return db.doc(`subscriptions/${topic}`).get().then((data)=>{
    const docData = data.data();
    uidList  = docData?.subscribersUid;
    const tokenList = docData?.fcmTokens;
    for (let index = 0; index < tokenList.length; index++) {
      console.log(tokenList[index]);
    }
    console.log(payload);
    return fcm.sendToDevice(tokenList,payload);
  }).then(()=>{
    console.log('EY DONT WORRY 2MUCH')
    uidList.forEach((uid)=>{
      return db.doc(`users/${uid}`).update({
        notificationsCount: admin.firestore.FieldValue.increment(+1),
        notificationsRegister: admin.firestore.FieldValue.arrayUnion({
          msg: messageToSend,
          deep_link: link
        })
      })
    })
  })
```

```
}else{
    console.log('A message was deleted so no msg is send through
FCMessaging (:');
  return;
}
})
```

#### 12.4.2.26. decideToDeleteTopicSubscribtion

```
//AUTOTRIGER: Evalua si es necesario o no eliminar un topico en función
de sus subscripciones (creada para el soporte técnico de los
desarrolladores)

exports.decideToDeleteTopicSubscription =
functions.firestore.document('subscriptions/{subscription}').onUpdate((
change, context)=>{

  const subscription = context.params.subscription;
  const newValue = change.after.data();
  const oldValue = change.before.data();
  const oldToken = oldValue?.fcmTokens;
  const newToken = newValue?.fcmTokens;
  const oldUid = oldValue?.subscribersUid;
  const newUid = newValue?.subscribersUid;

  if( .length == 1 && oldUid.length == 1 && newToken.length == 0 &&
newUid.length == 0 ){
    return db.doc(`subscriptions/${subscription}`).delete().then(()=>{
      return db.doc(`topics/${subscription}`).delete().then(()=>{
        console.log('Topic & subscription were delteted');
      })
    })
  }else{
    console.log(oldToken.length, oldUid.length, newToken.length,
newUid.length)
    return;
  }})
```

#### 12.4.2.27. getKiwiFlightTickets

```
//ONREQUEST: Permite encontrar un vuelo en función de los parámetros de búsqueda

exports.getKiwiFlightTickets = functions.https.onRequest(async (req, resp) => {

  const origen = req.query.orig.toUpperCase();
  const destino = req.query.dest.toUpperCase();
  const date =`${req.query.day}/${req.query.month}/${req.query.year}`; //'15/08/2020';
  console.log(date);
  const currency = 'EUR';
  const flight_type = 'oneway'; // 'round' --> with nights_in_dst of return date is given
  const direct_flights = 1; // 1 = YES, 0 = NO
  const adults = 1;
  const children = 0;
  const infants = 0;

  const kiwiOption = {
      method: 'GET',
      url: `https://api.skypicker.com/flights?fly_from=${origen}&fly_to=${destino}&date_from=${date}&partner=picky&curr=${currency}&direct_flights=${direct_flights}&flight_type=${flight_type}&adults=${adults}&children=${children}&infants=${infants}`,
      headers: {
      'content-type': 'application/json'
    }};

  let kTicket:KiwiTicket;
  let kTickets:KiwiTicket[] = new Array();
  let lowPrice:number[];

    try{
      await request(kiwiOption, async function(error:any, something:any, body:any){
          const kiwiResponse = Convertir.toKiwiResponse(body);
```

```
            const myData = kiwiResponse.data;

            if(myData !== undefined){
            for (let index = 0; index < myData.length; index++) {
              if(myData[index].availability?.seats !== null){
              let flightId
=`${myData[index].airlines[0]}${myData[index].route[0].flight_no}`
              kTicket = new KiwiTicket(myData[index].airlines[0],
flightId, myData[index].cityCodeFrom, myData[index].cityCodeTo,
myData[index].cityFrom, myData[index].cityTo,
myData[index].countryFrom.code, myData[index].countryTo.code, new
Date(myData[index].dTime * 1000), new Date(myData[index].aTime * 1000),
date, myData[index].fly_duration, myData[index].distance,
myData[index].price, myData[index].availability.seats,
myData[index].booking_token, myData[index].deep_link);
              kTickets.push({airline: kTicket.airline, flightId:
kTicket.flightId, from: kTicket.from, to: kTicket.to, from_city:
kTicket.from_city, to_city: kTicket.to_city, country_from:
kTicket.country_from, country_to: kTicket.country_to, depTime:
kTicket.depTime, arrTime: kTicket.arrTime, date: kTicket.date,
duration: kTicket.duration, distance: kTicket.distance, price:
kTicket.price, availability: kTicket.availability, bookingToken:
kTicket.bookingToken, deep_link: kTicket.deep_link});
              }}

              lowPrice = getLowestPrice(kTickets);
              console.log(lowPrice);
              await publishMessage(origen, destino, lowPrice[0],
req.query.day, req.query.month, req.query.year,
kTickets[lowPrice[1]].deep_link).then(()=>{
                console.log(kTickets)
                resp.send(kTickets);
              })
          }else{
            console.log('my data is equal to UNDEFINED');
            return;
          }
})
}catch(e){
  console.log(e.toString());
```

```
}})

    function getLowestPrice(ticketInfo:KiwiTicket[]){

      let array:number[] = [];
      let position:number = 0;
      let response:number[]= [];
      console.log(ticketInfo);
      for (let index = 0; index < ticketInfo.length; index++) {
          array.push(ticketInfo[index].price);
        }

      let min = array[0];

      for(let i = 0; i < array.length; i++){
        if(array[i] < min){
          min = array[i];
          position = i;
          console.log(position);
        }
      }
      response.push(min,position);
      console.log(response);
      return response;
    }

    async function publishMessage(orig:string, dest:string,
price:number, day:number, month:number, year:number, url:string){
//day:number, month:number, year:number
      //date te format == ( DD/MM/YYYY )
      console.log(url)
      console.log('url');

      await db.collection('topics').get().then((docs)=>{
        if(!docs.empty){
          docs.forEach((doc)=>{
            let topic = doc.id;
            console.log(topic);
            let arrTopic = topic.split('-');
             let or = arrTopic[0];
```

```
            let des = arrTopic[1];
            let preu = parseFloat(arrTopic[2]);
            const start = new Date(parseInt(arrTopic[5]),
parseInt(arrTopic[4]), parseInt(arrTopic[3]));
            const startEpoch = start.getTime()/1000.0;
            const end = new Date(parseInt(arrTopic[8]),
parseInt(arrTopic[7]), parseInt(arrTopic[6]));
            const endEpoch = end.getTime()/1000.0;
            const anal = new Date(year, month, day);
            const analEpoch = anal.getTime()/1000.0;
            // console.log(or, des, preu, url);
            if(or === orig && des === dest && preu >= price &&
analEpoch >= startEpoch && analEpoch <= endEpoch){
              console.log(topic,or, orig,des, dest, preu)
              return db.doc(`topics/${topic}`).update({
                messages: admin.firestore.FieldValue.arrayUnion(`📍🔔
No pierdas tu oportunidad: ${orig}-${dest} para ti por ${price}€ 🖤 ||
${url}`),
                msg_count: admin.firestore.FieldValue.increment(+1)
              }).then(()=>{
                console.log('messageDelivered!');
                return;
              })
            }
            return;
          })
        }else{
          return;
        }})
      }
```

### 12.4.3. searchUser JSON *response*

```
      {
            "img": "https://cdn.pixabay.com/photohead-659651_1280.png",
            "nickName": "GuestsoWb",
            "uid": "soWbzS0yGhT5ZSe7wgF7hu4qBhV2"
        }
```

## 12.5. Modelos Dart

### 12.5.1. Modelo dbUser

```
class DbUser {
    String uid;
    String userEmail;
    String userName;
    String nickName;
    String imgUrl;
    String frase;
    String backgroundColor;
    String tokenId;
    List<String> following;
    List<String> followers;
    List<String> pendingToAccept;
    String userWeb;
    String phoneNumber;
    int notificationsCount;
    List<NotificationsRegister> notificationsRegister;
    List<SubscriptionsRegister> subscriptionsRegister;

    DbUser({
        this.uid,
        this.userEmail,
        this.userName,
        this.nickName,
        this.imgUrl,
        this.frase,
        this.backgroundColor,
        this.tokenId,
        this.following,
        this.followers,
        this.pendingToAccept,
        this.userWeb,
        this.phoneNumber,
        this.notificationsCount,
        this.notificationsRegister,
        this.subscriptionsRegister,
```

```
    });

    DbUser copyWith({
        String uid,
        String userEmail,
        String userName,
        String nickName,
        String imgUrl,
        String frase,
        String backgroundColor,
        String tokenId,
        List<String> following,
        List<String> followers,
        List<String> pendingToAccept,
        String userWeb,
        String phoneNumber,
        int notificationsCount,
        List<NotificationsRegister> notificationsRegister,
        List<SubscriptionsRegister> subscriptionsRegister,
    }) =>
        DbUser(
            uid: uid ?? this.uid,
            userEmail: userEmail ?? this.userEmail,
            userName: userName ?? this.userName,
            nickName: nickName ?? this.nickName,
            imgUrl: imgUrl ?? this.imgUrl,
            frase: frase ?? this.frase,
            backgroundColor: backgroundColor ?? this.backgroundColor,
            tokenId: tokenId ?? this.tokenId,
            following: following ?? this.following,
            followers: followers ?? this.followers,
            pendingToAccept: pendingToAccept ?? this.pendingToAccept,
            userWeb: userWeb ?? this.userWeb,
            phoneNumber: phoneNumber ?? this.phoneNumber,
            notificationsCount: notificationsCount ??
this.notificationsCount,
            notificationsRegister: notificationsRegister ??
this.notificationsRegister,
            subscriptionsRegister: subscriptionsRegister ??
this.subscriptionsRegister,
```

```
        );

    factory DbUser.fromJson(Map<String, dynamic> json) => DbUser(
        uid: json["uid"] == null ? null : json["uid"],
        userEmail: json["userEmail"] == null ? null :
json["userEmail"],
        userName: json["userName"] == null ? null : json["userName"],
        nickName: json["nickName"] == null ? null : json["nickName"],
        imgUrl: json["imgUrl"] == null ? null : json["imgUrl"],
        frase: json["frase"] == null ? null : json["frase"],
        backgroundColor: json["backgroundColor"] == null ? null :
json["backgroundColor"],
        tokenId: json["tokenId"] == null ? null : json["tokenId"],
        following: json["following"] == null ? null :
List<String>.from(json["following"].map((x) => x)),
        followers: json["followers"] == null ? null :
List<String>.from(json["followers"].map((x) => x)),
        pendingToAccept: json["pendingToAccept"] == null ? null :
List<String>.from(json["pendingToAccept"].map((x) => x)),
        userWeb: json["userWeb"] == null ? null : json["userWeb"],
        phoneNumber: json["phoneNumber"] == null ? null :
json["phoneNumber"],
        notificationsCount: json["notificationsCount"] == null ? null :
json["notificationsCount"],
        notificationsRegister: json["notificationsRegister"] == null ?
null :
List<NotificationsRegister>.from(json["notificationsRegister"].map((x)
=> NotificationsRegister.fromJson(x))),
        subscriptionsRegister: json["subscriptionsRegister"] == null ?
null :
List<SubscriptionsRegister>.from(json["subscriptionsRegister"].map((x)
=> SubscriptionsRegister.fromJson(x))),
    );

    Map<String, dynamic> toJson() => {
        "uid": uid == null ? null : uid,
        "userEmail": userEmail == null ? null : userEmail,
        "userName": userName == null ? null : userName,
        "nickName": nickName == null ? null : nickName,
        "imgUrl": imgUrl == null ? null : imgUrl,
```

```
        "frase": frase == null ? null : frase,
        "backgroundColor": backgroundColor == null ? null :
backgroundColor,
        "tokenId": tokenId == null ? null : tokenId,
        "following": following == null ? null :
List<dynamic>.from(following.map((x) => x)),
        "followers": followers == null ? null :
List<dynamic>.from(followers.map((x) => x)),
        "pendingToAccept": pendingToAccept == null ? null :
List<dynamic>.from(pendingToAccept.map((x) => x)),
        "userWeb": userWeb == null ? null : userWeb,
        "phoneNumber": phoneNumber == null ? null : phoneNumber,
        "notificationsCount": notificationsCount == null ? null :
notificationsCount,
        "notificationsRegister": notificationsRegister == null ? null :
List<dynamic>.from(notificationsRegister.map((x) => x.toJson())),
        "subscriptionsRegister": subscriptionsRegister == null ? null :
List<dynamic>.from(subscriptionsRegister.map((x) => x.toJson())),
    };
}

class NotificationsRegister {
    String msg;
    String deep_link;

    NotificationsRegister({
        this.msg,
        this.deep_link,
    });

    NotificationsRegister copyWith({
        String msg,
        String deep_link,
    }) =>
        NotificationsRegister(
            msg: msg ?? this.msg,
            deep_link: deep_link ?? this.deep_link,
        );
```

```
    factory NotificationsRegister.fromJson(Map<String, dynamic> json)
=> NotificationsRegister(
        msg: json["msg"] == null ? null : json["msg"],
        deep_link: json["deep_link"] == null ? null :
json["deep_link"],
    );

    Map<String, dynamic> toJson() => {
        "msg": msg == null ? null : msg,
        "deep_link": deep_link == null ? null : deep_link,
    };
}

class SubscriptionsRegister {
    String orig;
    String dest;
    int price;
    int dayInici;
    int monthInici;
    int yearInici;
    int dayFinal;
    int monthFinal;
    int yearFinal;

    SubscriptionsRegister({
        this.orig,
        this.dest,
        this.price,
        this.dayInici,
        this.monthInici,
        this.yearInici,
        this.dayFinal,
        this.monthFinal,
        this.yearFinal,
    });

    SubscriptionsRegister copyWith({
        String orig,
        String dest,
        int price,
```

```
        int dayInici,
        int monthInici,
        int yearInici,
        int dayFinal,
        int monthFinal,
        int yearFinal,
    }) =>
        SubscriptionsRegister(
            orig: orig ?? this.orig,
            dest: dest ?? this.dest,
            price: price ?? this.price,
            dayInici: dayInici ?? this.dayInici,
            monthInici: monthInici ?? this.monthInici,
            yearInici: yearInici ?? this.yearInici,
            dayFinal: dayFinal ?? this.dayFinal,
            monthFinal: monthFinal ?? this.monthFinal,
            yearFinal: yearFinal ?? this.yearFinal,
        );

    factory SubscriptionsRegister.fromJson(Map<String, dynamic> json)
=> SubscriptionsRegister(
        orig: json["orig"] == null ? null : json["orig"],
        dest: json["dest"] == null ? null : json["dest"],
        price: json["price"] == null ? null : json["price"],
        dayInici: json["dayInici"] == null ? null : json["dayInici"],
        monthInici: json["monthInici"] == null ? null :
json["monthInici"],
        yearInici: json["yearInici"] == null ? null :
json["yearInici"],
        dayFinal: json["dayFinal"] == null ? null : json["dayFinal"],
        monthFinal: json["monthFinal"] == null ? null :
json["monthFinal"],
        yearFinal: json["yearFinal"] == null ? null :
json["yearFinal"],
    );

    Map<String, dynamic> toJson() => {
        "orig": orig == null ? null : orig,
        "dest": dest == null ? null : dest,
        "price": price == null ? null : price,
```

```
        "dayInici": dayInici == null ? null : dayInici,
        "monthInici": monthInici == null ? null : monthInici,
        "yearInici": yearInici == null ? null : yearInici,
        "dayFinal": dayFinal == null ? null : dayFinal,
        "monthFinal": monthFinal == null ? null : monthFinal,
        "yearFinal": yearFinal == null ? null : yearFinal,
    };
}}
```

### 12.5.2. Modelo GoSoon

```
import 'package:sqflite/sqflite.dart';

class GoSoon {
  int id;
  String name;
  String img;
  GoSoon({this.id, this.name, this.img});

  Map<String, dynamic> toMap() {
    return {"name": name, "img": img};
  }

  GoSoon.fromMap(Map<String, dynamic> map) {
    name = map['name'];
    img = map['img'];
    id = map["id"];
  }}
```

### 12.5.3. Pendent Bloc

```
import 'package:bloc/bloc.dart';
import 'package:equatable/equatable.dart';

class PendentEvent extends Equatable {
  @override
  //Definimos una clase evento (tipo) --> PendentEvent
  List<Object> get props => [];}

//A - Evento que solicita la lista Solicitudes Pendientes
class FetchPendent extends PendentEvent {
```

```
  final _uid;
  FetchPendent(this._uid);
  List<Object> get props => [_uid];}

//B - Evento que obliga a solicitar de nuevo la lista
class ReFetchPendent extends PendentEvent {
  final _uid;
  ReFetchPendent(this._uid);
  List<Object> get props => [_uid];}

class PendentState extends Equatable {
  @override
  // Definimos una clase estado (tipo) --> PendentEvent
  List<Object> get props => [];}



//A continuación se listan los distintos estados

//1 - La lista esta siendo cargada
class PendentListIsLoading extends PendentState {}

//2 - La lista ha sido cargada, lista para ser usada
class PendentListLoaded extends PendentState {
  final _pendentList;
  PendentListLoaded(this._pendentList);

  List<UsersData> get getPdtList => _pendentList;
  @override
  List<Object> get props => [_pendentList];}

//3 - La lista no ha podido ser cargada
class PendentListNotLoaded extends PendentState {}

//Aquí aparece la lógica
class PendentBloc extends Bloc<PendentEvent, PendentState> {
  final uid;
  PendentBloc({this.uid});
  @override
  // Definimos una clase BLOC que permite generar la relación entre
```

```
// eventos (A-B) y los diferentes estados (1-3)
PendentState get initialState => PendentListIsLoading();

@override
Stream<PendentState> mapEventToState(event) async* {
  // TODO: implement mapEventToState
  if (event is FetchPendent) {
    try {
      List<UsersData> pendentList =
          await DataBase(uid: event._uid).getUsersPendingToAccept();
      //NOTA: getUsersPendingToAccept es una CF que devuelve
      //      la lista de solicitudes de seguimiento
      yield PendentListLoaded(pendentList);
    } catch (_) {
      yield PendentListNotLoaded();
    }

  } else if (event is ReFetchPendent) {
    try {
      List<UsersData> pendentList =
          await DataBase(uid: event._uid).getUsersPendingToAccept();
      yield PendentListLoaded(pendentList);
    } catch (_) {
      yield PendentListNotLoaded();
    }
  }}}
```

## 12.6. Scripts

### 12.6.1. fireAirport.ts

```
import {Airport} from './Airport.js';

import {Airports} from './Airports.js'
const request = require('request')
const sortBy = require('lodash.sortby');
import * as _ from "lodash";
const cliProgress = require('cli-progress');
const fs = require('fs');
import {AirportInfo} from './AirportInfo.js';
import {Airporty} from './Airporty.js';


// Converts JSON strings to your object
export class Convert {
    public static toAirporty(json: string): Airporty {
        return JSON.parse(json);
    }}

let array = [];
main();

async function main(){
 const done = await getMyDataForMyList()
 Promise.all([done]).then(()=>{
   console.log(array);
 })
}

async function getMyDataForMyList(){
    let airCount = 0;
    const docuNum = 5;
    console.log('N3D Files processing: (AirportList)')
    await
db.collection('AirData').doc(`airports${docuNum}`).get().then((function
(doc) {
        if (doc.exists) {
```

```
            let json = doc.data();
            Object.keys(json).forEach(async (ele)=>{
              // let country =new Airport(json[ele]).country_name;
              let iata = new Airport(json[ele]).iata_code;
               let name =  await printIata(iata);
               console.log(name);
               await Promise.all([name]).then(()=>{
                 array = array.concat({[ele]:{
                   cityName: name,
                   iata: iata,
                   firebase: `airports${docuNum}`
                 }});
                 airCount++;
                 console.log(airCount);
                 if(airCount == 865){
                   console.log(array)
                    let data = JSON.stringify(array);
                    fs.writeFileSync(`airports${docuNum}`,data);
                 }
               })
            })
        } else {
            console.log("No such document!");
        }
    }));}

async function printIata(iata:string){
  const optionCity = {
    method: 'GET',
    url: 'https://airport-info.p.rapidapi.com/airport',
    qs: {iata: iata},
    headers: {
      'x-rapidapi-host': 'airport-info.p.rapidapi.com',
      'x-rapidapi-key': Rapid_API_KEY
    }
  };

   return new Promise(function(resolve, reject){
     let cityInfo:string;
     try{
```

```
        request.get(optionCity, function(error:any,something:any,
body:any){

        if(error){
          console.log(error.toString());
          reject(error);
        }else{
            if(body.startsWith('<',0) !== true){
          const airportInfo:Airporty = Convert.toAirporty(body);
         if(airportInfo.location !== undefined){
           const cityArr = airportInfo.location.split(",");
           cityInfo = cityArr[0];
          // console.log(cityInfo[0]);
        }else if(airportInfo.city!== ""){
           cityInfo = airportInfo.city;
        }else if(airportInfo.county!==""){
           cityInfo = airportInfo.county;
        }else{
           cityInfo = "N/A";
        }
        }else{
            cityInfo = "N/A";
        }

          resolve(cityInfo);
        }
      })

    }catch(e){
      console.log(e.toString())
    }
    })
}
```

### 12.6.2. Aviation.ts

```
var request = require('request');

import {AviationStack} from './aviationStack';
const access_key = A private ACCES_KEY;

var d = new Date();
var curr_date = d.getDate();
var curr_month = d.getMonth() + 1; //Months are zero based
var curr_year = d.getFullYear();
const date = (curr_year + '-' + curr_month + '-' + curr_date);
const flightId = 'VY1156'

 console.log(date);

class Convert {
    public static toAviationStack(json: string): AviationStack {
        return JSON.parse(json);
    }
}

var optionAviationStack = {
  method: 'GET',
  url:
`http://api.aviationstack.com/v1/flights?access_key=${access_key}
        &flight_date=${date}&flight_iata=${flightId}`,
  headers: {
  }
}

getFlightInfo();

function getFlightInfo(){
  request(optionAviationStack, function(err,response,body){
    if(err) throw Error(err);
    const data = body;
    console.log(body)
  })}
```