**COLEGIO SALESIANO SAN JOSÉ**

**CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA**

****

**PROYECTO FIN DE CICLO**

iParty

Helena Juliá Sánchez

**Febrero, 2020**

**COLEGIO SALESIANO SAN JOSÉ**

**CFGS DESARROLLO DE APLICACIONES MULTIPLATAFORMA**

**PROYECTO FIN DE CICLO**

iParty

**Autor**: Helena Juliá Sánchez

**Tutor**: María Isabel Martínez García

**Mes, Año**

# Índice

[Índice 1](#_Toc35259685)

[Lista de símbolos 3](#_Toc35259686)

[1. Introducción 4](#_Toc35259687)

[1.1. Objetivos del proyecto 5](#_Toc35259688)

[1.2. Enfoque 5](#_Toc35259689)

[1.2.1. Límites 5](#_Toc35259690)

[1.2.2. Detalles de los objetivos 5](#_Toc35259691)

[2. Antecedentes 6](#_Toc35259692)

[2.1. Aplicaciones para conocer a gente 6](#_Toc35259693)

[2.1.1. Mobifriends 6](#_Toc35259694)

[2.1.2. FriendZone 7](#_Toc35259695)

[2.1.3. MeetMe 8](#_Toc35259696)

[2.2. Buscar mesa 8](#_Toc35259697)

[2.3. Lenguaje de programación: Dart 10](#_Toc35259698)

[2.4. Framework: Flutter 11](#_Toc35259699)

[3. Objetivos del proyecto 15](#_Toc35259700)

[*3.1.* Objetivos generales 15](#_Toc35259701)

[*3.2.* Objetivos específicos 15](#_Toc35259702)

[4. Hipótesis del proyecto 16](#_Toc35259703)

[5. Desarrollo real del proyecto 17](#_Toc35259704)

[5.1. Prototipado 17](#_Toc35259705)

[5.2. Elección de la paleta de colores 19](#_Toc35259706)

[5.3. Instalación del IDE y de FLutter 23](#_Toc35259707)

[5.4. Estructura del código 24](#_Toc35259708)

[5.5. Main () 24](#_Toc35259709)

[5.5.1. Información básica 25](#_Toc35259710)

[5.5.2. Diseño del tema base 25](#_Toc35259711)

[5.5.3. Árbol de rutas 28](#_Toc35259712)

[5.5.4. Redirección: página principal o página de Login/Registro 28](#_Toc35259713)

[5.6. Página de Intro 34](#_Toc35259714)

[5.6.1. Stateless o Stateful widget 35](#_Toc35259715)

[5.6.2. Scaffold widget 35](#_Toc35259716)

[5.6.3. IntroSlider 35](#_Toc35259717)

[5.6.4. Slide 38](#_Toc35259718)

[5.7. Página DE LOGIN/REGISTRO 40](#_Toc35259719)

[5.7.1. Operativa de carga de la página de introducción 40](#_Toc35259720)

[5.7.2. Diseño de los modales de inicio de sesión y registro 46](#_Toc35259721)

[6. Conclusiones y propuestas 46](#_Toc35259722)

[7. Bibliografía 47](#_Toc35259723)

[8. Planos 51](#_Toc35259724)

[9. Anexos 51](#_Toc35259725)

# Lista de símbolos

# Introducción

El disfrute del tiempo libre se ha convertido en un lujo en el siglo XXI. Y no solo a la hora de disfrutarlo por lo que es, entre responsabilidades y quehaceres. Esto es especialmente cierto cuando esos planes requieren de otras personas. Y más aún cuando el ocio del que pretendes disfrutar es minoritario. Incluso en ciudades grandes, en las que la lógica dice que hay cientos de personas que comparten tus gustos, el anonimato y las distancias pueden resultar un impedimento para realizar estos planes.

Es el caso específico de los juegos de rol y de mesa: las editoriales venden, pero muchas personas se encuentran con dificultades para encontrar nuevos grupos con los que realizar su afición. Las opciones más populares para solventarlo pasan por los foros de páginas web anticuadas, como puede ser Comunidad Umbría, o redes sociales enfocadas a otros usos como Facebook o Twitter. Pero no existe ninguna herramienta específica, ágil y enfocada a los usos tecnológicos de esta época.

Precisamente con la idea en mente de crear una aplicación capaz de estar a la altura de las necesidades tecnológicas de los usuarios, se elige el lenguaje de Dart y su framework de Flutter para el desarrollo del proyecto. Flutter permite desarrollar de forma nativa aplicaciones de iOS y Android al mismo tiempo. Se pueden resumir algunas de sus características principales en:

* Compilación rápida y muy personalizable
* Compilación en caliente, que permite visualizar los cambios a medida que se va desarrollando el código
* Desarrollo de la interfaz programática
* Desarrollo de aplicaciones ágiles y eficientes

Para optimizar el rendimiento de la aplicación se emplearán bases de datos no relacionales, utilizando los sistemas proporcionados por Firebase.

## Objetivos del proyecto

El objetivo del proyecto es la creación de una aplicación móvil en la cuál los usuarios se podrán registrar, crear partidas y apuntarse a aquellas que estén abiertas y, en definitiva, conocer a gente con la que compartan gustos. Para ello se utilizará tecnología puntera en el desarrollo de estas: el framework de Google Flutter basado en el lenguaje Dart.

A pesar de que Flutter permite el desarrollo de aplicaciones nativas para iOS y Android y su objetivo pasa porque estas aplicaciones se vean naturales dentro del diseño único y característico de ambos dispositivos, la imposibilidad material de compilar para iOS implica que en este proyecto no se va a lograr este fin. La aplicación funcionará tanto para Android como para iOS, pero su aspecto será más parecido a los estándares de Android que los de iOS.

## Enfoque

El enfoque del proyecto es el siguiente:

### Límites

A pesar de que tanto el lenguaje de Dart como el framework de Flutter pertenecen a Google y Google ha puesto un gran empeño en documentar y popularizarlos, son dos tecnologías relativamente jóvenes y con pocos adeptos. Existen pocos recursos en la red, exceptuando los oficiales (que en ocasiones resultan parcos a la hora de dar ejemplos de uso), que podrían dificultar el desarrollo del proyecto.

### Detalles de los objetivos

Los objetivos del proyecto son los siguientes:

* Aprender y familiarizarme con el lenguaje Dart y el framework de Flutter.
* El desarrollo de una app móvil que permita a los distintos usuarios comunicarse y conocerse en base a unos gustos comunes.

# Antecedentes

## Aplicaciones para conocer a gente

Las aplicaciones para conectar a gente no son muy populares y existen pocos ejemplos funcionales cuyo único objetivo no sea otro que el de ligar. Aun así, existen algunos ejemplos de desarrollo que pueden de ejemplo de qué prácticas y funciones implementar y qué prácticas y funciones que evitar.

Estas aplicaciones tienen algunos elementos comunes, que se verán a continuación y que pueden resumirse en:

* Listado de usuarios
* Chat
* Perfil de usuario

### Mobifriends

Se trata de una aplicación que busca conectar a gente, ya sea con intereses románticos o para buscar amistad. Según inicias sesión, muestra un listado de usuarios que se encuentran en tu zona.

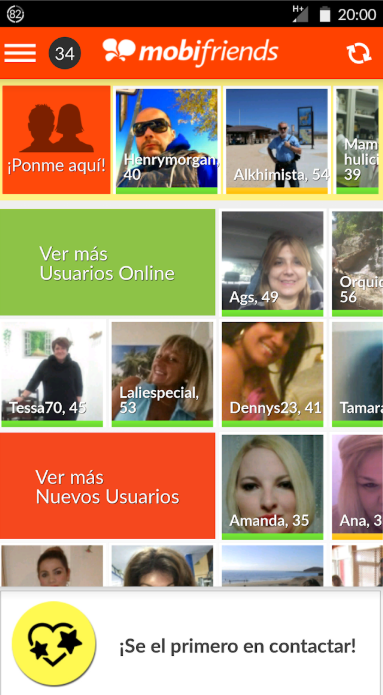
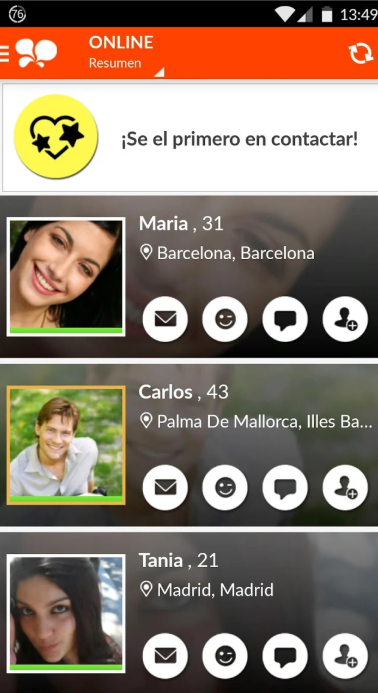
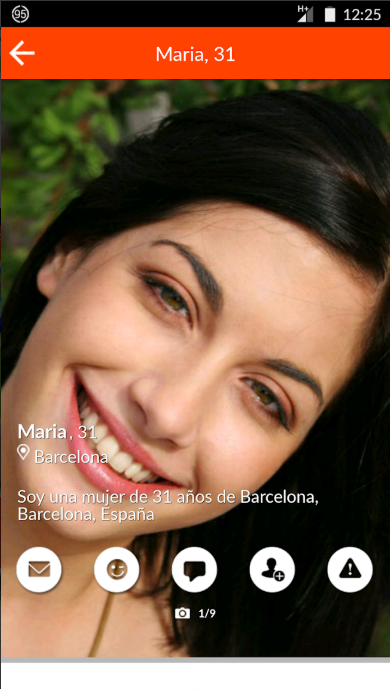
   

Ilustración 1: Imágenes promocionales de la app Mobifriends (Mobifriends, 2019)

La aplicación cuenta con varias páginas que permiten buscar entre usuarios aplicando diversos filtros (como el de localización), ver su perfil y contactar con ellos. También dispone de un apartado de personas favoritas que permite acceder fácilmente a aquellos usuarios que hayamos marcado como tal.

Se aprecian las siguientes características, tal y como se resume en la Tabla 1.

Tabla 1: Pros y contras de Mobifriends

|  |  |
| --- | --- |
| PROS | CONTRAS |
| * Filtros entre usuarios * Contacto de usuarios * Usuarios favoritos | * Interfaz anticuada y fea * Orientada especialmente a las interacciones románticas * Es una página de contactos, no un portal para buscar grupos de amigos |

### FriendZone

FriendZone es una app cuyo enfoque principal es unir gente a partir de sus intereses, por lo que se parece mucho más al objetivo de desarrollo de iParty.

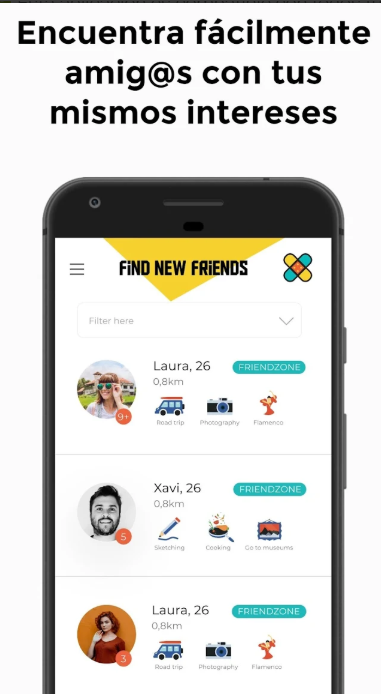
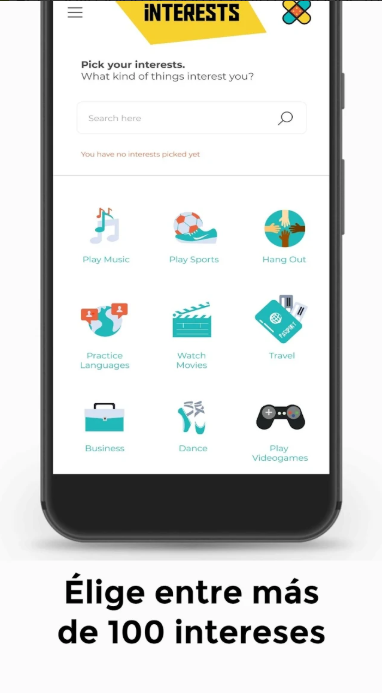
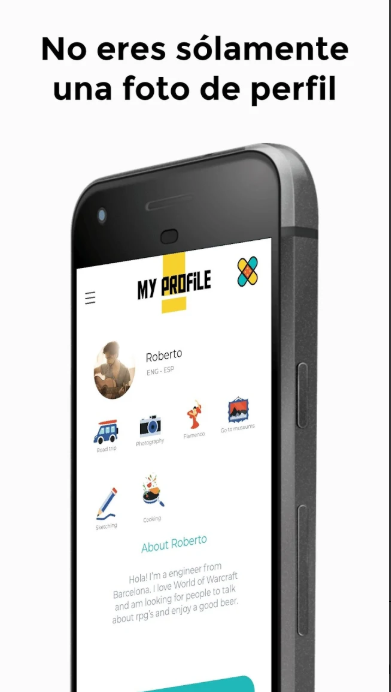
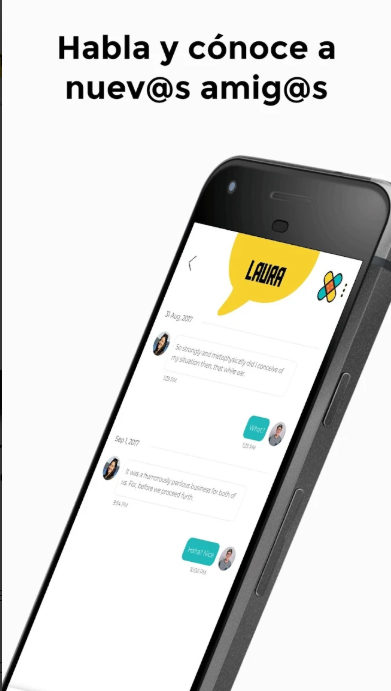
   

Ilustración 2: Imágenes promocionales de FriendZone (FriendZone, 2020)

El punto de interés en el desarrollo de la aplicación radica en los gustos de cada uno, poniendo especial interés en las aficiones de cada uno. La aplicación permite filtrar a la gente en grupos y subgrupos (por ejemplo, películas > terror) y distancia y dispone de un algoritmo para encontrar a posibles amigos.

Se aprecian las siguientes características, resumidas en la Tabla 2.

Tabla 2: Pros y contras de FriendZone

|  |  |
| --- | --- |
| PROS | CONTRAS |
| * Interfaz bonita y moderna | * Demasiado genérica |

### MeetMe

MeetMe se vende como una aplicación para buscar amigos con los que chatear.

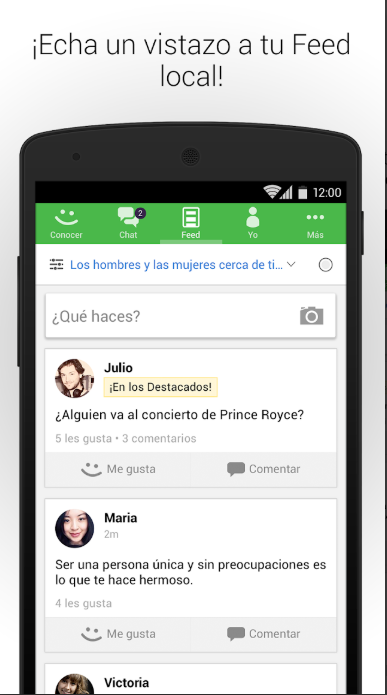
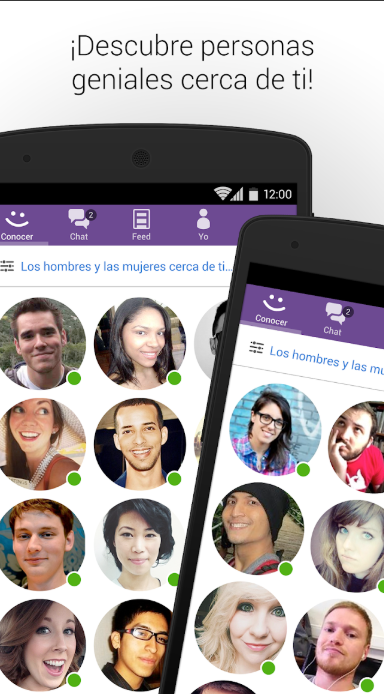
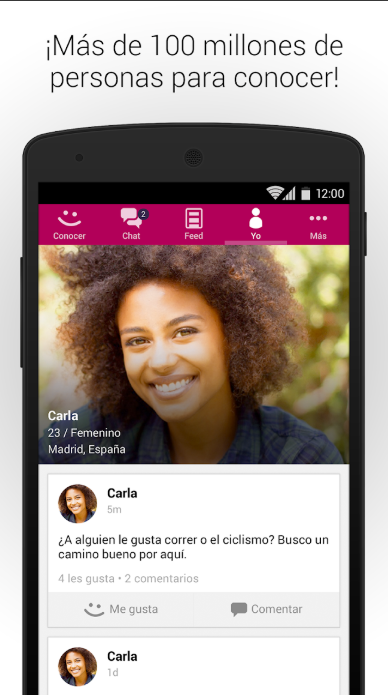
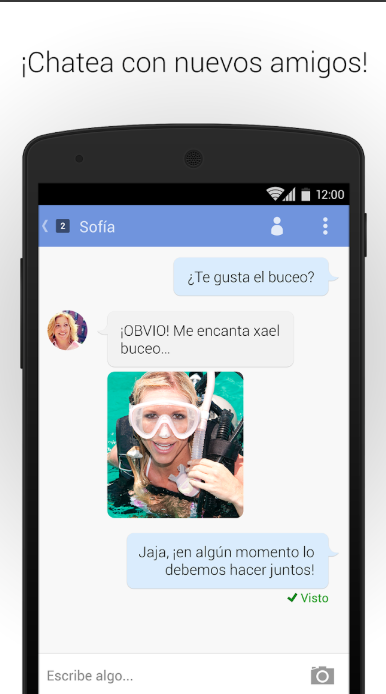
  

Ilustración 3: Imágenes promocionales de MeetMe (MeetMe.com, 2020)

Además del chat, dispone de un *feed* que recuerda al muro de Facebook. Se pueden ver sus características principales en la Tabla 3.

Tabla 3: Pros y contras de MeetMe

|  |  |
| --- | --- |
| PROS | CONTRAS |
| * Navegación intuitiva | * Sin objetivo claro * Más red social que punto de encuentro * No da información clara sobre sus usuarios * Sin filtros claros de búsqueda |

## Buscar mesa

Si las opciones a la hora de conocer a gente a través de aplicaciones son limitadas, estas empeoran cuando tu objetivo es conocer a gente para jugar a rol y a juegos de mesa. La única aplicación de las analizadas anteriormente (FriendZone) que dispone de filtros sobre aficiones no tiene los adecuados para nuestro objetivo y se centran más en conocer a gente que en realizar una actividad.

Es por eso por lo que normalmente se usan otros canales que no siempre resultan óptimos para la búsqueda. Un buen ejemplo es la Ilustración 4, en la que podemos ver un llamamiento de una mesa que busca jugadores:



Ilustración 4: Ejemplo de búsqueda de jugadores en Twitter (En5MinutosRol, 2020)

En5MinutosRol hace así un llamamiento en Twitter utilizando como únicos medios de difusión:

* Su cuenta
* La cuenta de una editorial de juegos de rol

Este tipo de comunicación, por llamamiento, puede resultar óptima si consigue popularizarse a través de los retweets. Sin embargo, también puede acabar en el olvido sin conseguir ninguna clase de respuesta. Aunque haya jugadores que quieran jugar y una partida organizada con tal fin.

Otra opción popularizada a la hora de buscar grupos de rol es acudir a foros especializados, en los que los usuarios pueden publicar mensajes como el de la Ilustración 5:

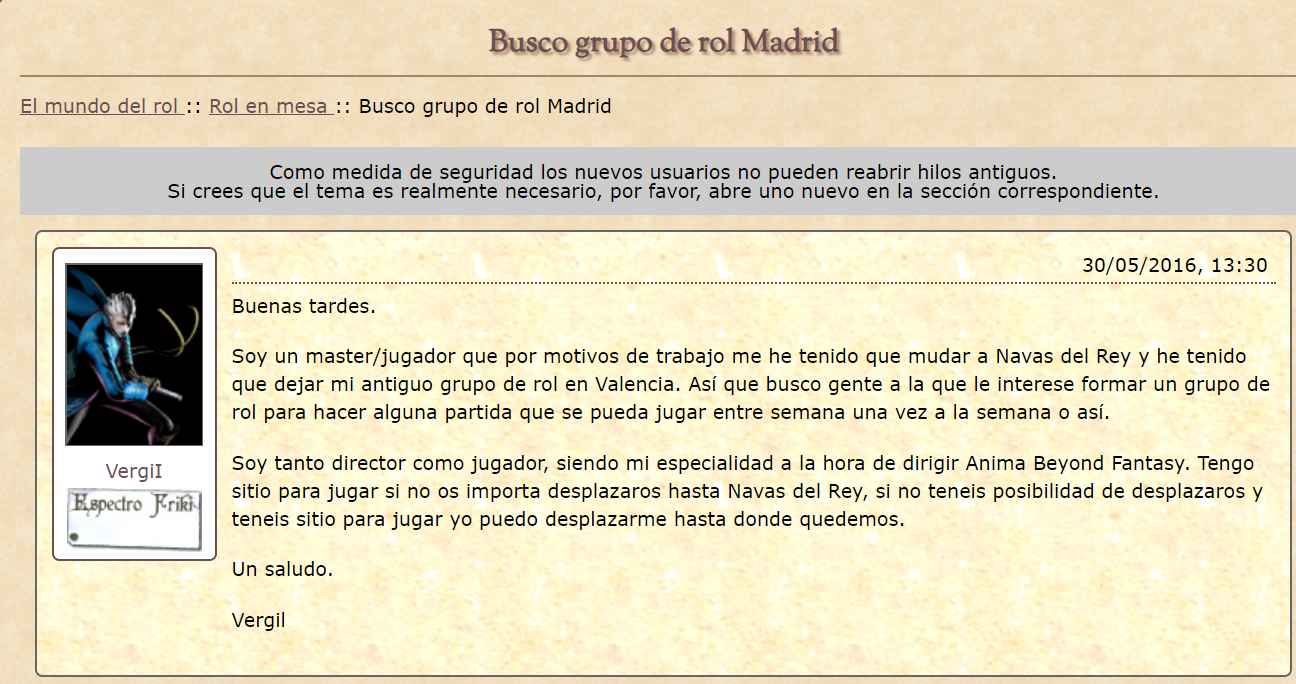


Ilustración 5: Mensaje buscando grupo de rol en Madrid (VergiI, 2016)

De nuevo, este tipo de opciones no resultan óptimas debido a que no es su objetivo fundamental. Es un foro, en el que los mensajes pueden quedar fácilmente tapados por otros mensajes más populares.

## Lenguaje de programación: Dart

Dart es desarrollado por Google. Se trata de un lenguaje dinámico e inspirado en JavaScript. Además, está influido por Java y C# (Oppong, 2019). Se desarrolló en 2011 y, por aquella época, su característica principal se basaba en que estaba orientado a objetos y basado en el uso de clases (algo de lo que JavaScript, su principal competencia, carecía). Sin embargo, y aunque era el objetivo de Google (fue bautizado como el *JavaScript Killer[[1]](#footnote-1)* en la red), no fue capaz de superarlo y en 2013, con la aparición de TypeScript, fue superado.

Aun así, Dart sigue desarrollándose. Y, todo en cuenta, tiene ciertos puntos fuertes que caben destacar (Swehli, 2018):

* Tiene una sintaxis sencilla y basada en C, por lo que resulta muy similar a la que se pueden encontrar en lenguajes como C++, Java y JavaScript (Bolton, 2019)
* Es un lenguaje rápido, eficiente[[2]](#footnote-2) y *libre de sorpresas* (Dart, s.f.)
* Dispone de muchísimas librerías para facilitar su desarrollo
* Es multidisciplinar: está enfocado al desarrollo móvil y web, de aplicaciones de escritorio y de servidores
* Es capaz de compilar de manera nativa en JavaScript
* Está desarrollado por Google

A pesar de su impopularidad, desde Google se sigue desarrollando en Dart. Como ejemplos de desarrollo web están Google AdWords y Google AdSense (AngularDart, s.f.), que el framework de AngularDart. Fuchsia, otro proyecto de Google, también está desarrollado en Dart. Se trata de un sistema operativo *open-source* que ha sido introducido en la gama de productos de Google Home Hub (Bolton, 2019).

## Framework: Flutter

Flutter es un framework de Dart desarrollado por Google en 2017. Con él se pueden desarrollar aplicaciones para cualquier dispositivo móvil, ya sea iOS o Android, con una interfaz unificada y sin errores y manteniendo la velocidad y el rendimiento de las aplicaciones nativas (Bolton, 2019). Y, lo más importante, manteniendo un único código fuente (Thomas, 2019). Además, se pretende que estas aplicaciones no solo funcionen en cualquier entorno, sino que resulte natural en las diferentes plataformas (Flutter, s.f.).

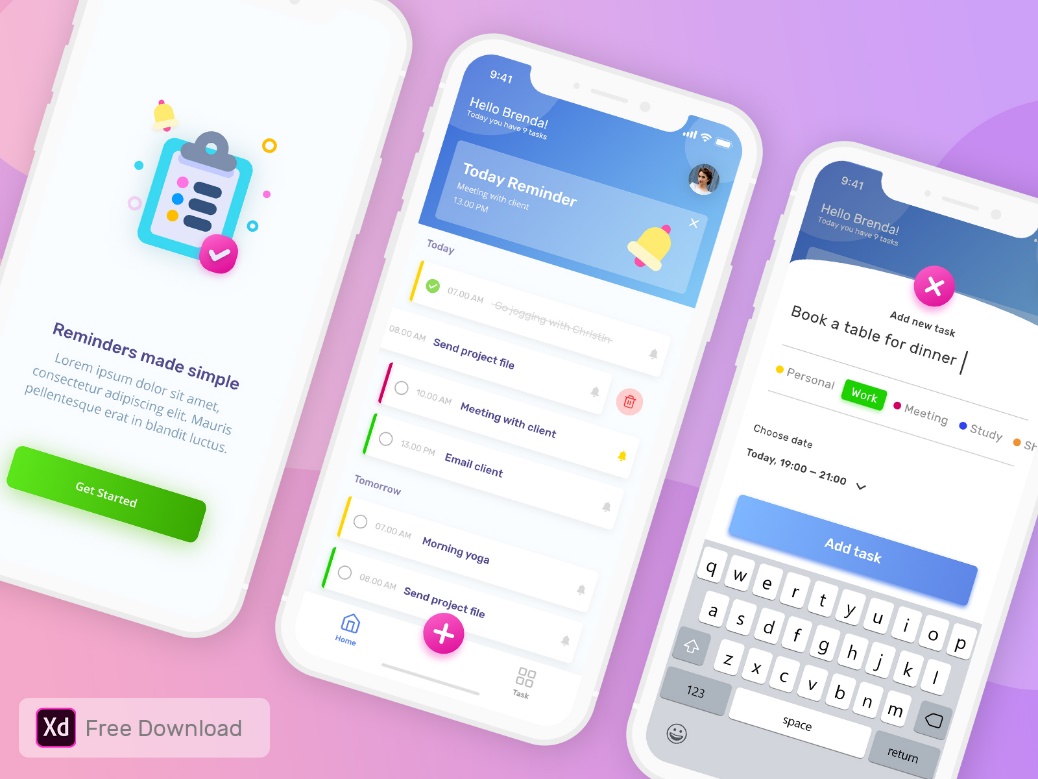


Ilustración 6: To do list es un ejemplo de aplicación que se puede desarrollar en Flutter (LiveLikeCounter, 2019)

A pesar de que no se trata de una tecnología popularizada en el desarrollo de aplicaciones, Google sí que la utiliza para desarrollar. Gracias a esto, el equipo de Flutter no para de actualizar los componentes para asegurarse de que estos están al día y mantiene una documentación extensiva y que recoge cada uno de sus elementos (Concise Software, 2019).

El núcleo de Flutter se trata de los widgets: prácticamente todos los elementos que nos encontramos en el desarrollo de aplicaciones lo van a ser. Estemos hablando de columnas y filas o imágenes y texto, se tratan del mismo elemento adaptado a las necesidades específicas de cada momento. Así se crean las pantallas y componentes en Flutter: anidando widgets en lo que se llama *widget tree*. Un ejemplo del diseño de una visualización se puede ver en la Ilustración 7:

Ilustración 7: Ejemplo de widget tree en Flutter

El esquema anterior es una posible interpretación de la siguiente pantalla, pudiendo conseguirse el mismo resultado utilizando otra combinación de widgets:

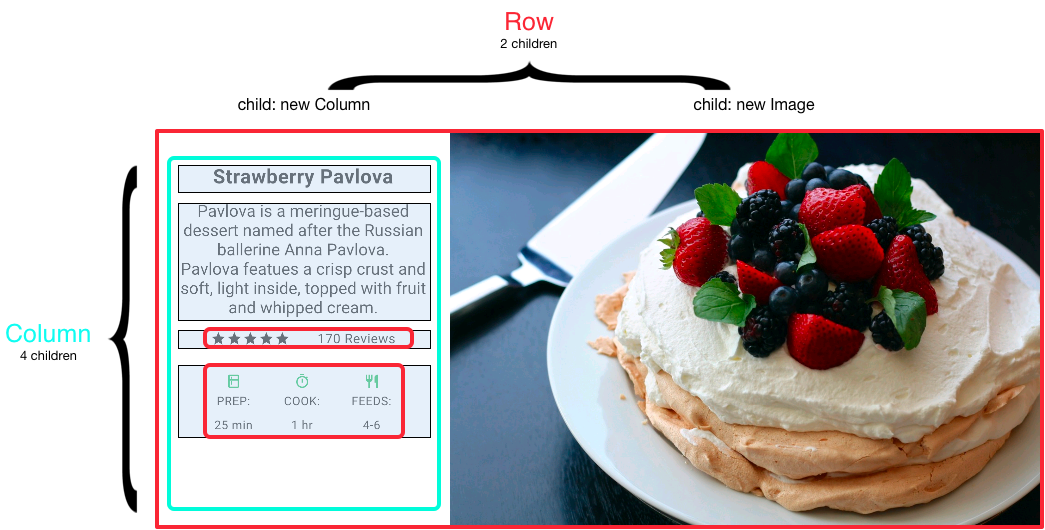


Ilustración 8: Diseño de una pantalla en Flutter (Layouts in Flutter, s.f.)

Algunas de las características de Flutter son (Bolton, 2019):

* Compilación Ahead-of-Time (AOT), lo que permite que la aplicación sea capaz de lanzarse de manera rápida
* Compilación Just-in-Time (JIT), que es la que permite la *hot reload*, es decir, la visualización de cambios de la aplicación en tiempo real
* Muestra entre 60 y 120 imágenes por segundo, si el dispositivo lo permite

Para conseguir un aspecto natural para iOS y para Android, Flutter utiliza dos librerías: cupertino y material. A continuación, se muestra esta dualidad adaptativa a partir de la misma pantalla de una aplicación de ejemplo orientadas a cada una de estas plataformas:

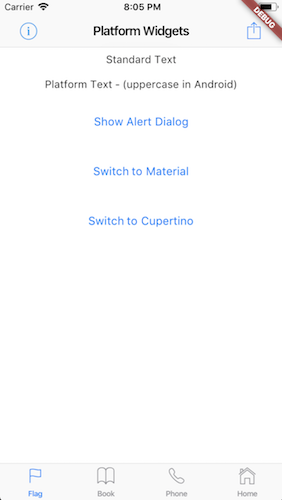
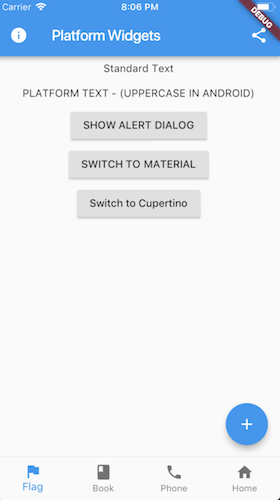
 

Ilustración 9: Comparativa entre la librería de cupertino (para iOS) y material (para Android)

# Objetivos del proyecto

Los objetivos que se plantean en el desarrollo del siguiente TFC son:

## Objetivos generales

A nivel general, se plantea como objetivo la realización de una aplicación móvil para conectar a gente que comparta aficiones de juegos de mesa y rol.

## Objetivos específicos

Los objetivos específicos del TFC son los siguientes:

1. Permitir el registro y acceso de usuarios.
2. Crear mesas de juego que busquen jugadores, pudiendo agregar comentarios y formas de contacto.
3. Realizar búsquedas de mesas de juego o contactos empleando de filtros.
4. Aprender las tecnologías de Flutter y Dart, así como familiarizarme con el proveedor de servicios de Firebase y su forma de interactuar con aplicaciones móviles.
5. Aunar los conocimientos adquirido a lo largo de toda la formación de DAM, empleando para ello las siguientes competencias: planificación de proyectos, desarrollo de bases de datos no relacionales, procesos, programación orientada a objetos y arquitectura y análisis del software.

# Hipótesis del proyecto

# Desarrollo real del proyecto

## Prototipado

Se desarrolla un prototipado de iParty a partir de la herramienta Quick Proto, que permite diseñar y enlazar de manera intuitiva diferente pantallas (O'Brien, 2019). Se pretende tener una sensación general de las diferentes secciones que va a contener iParty, así como la interacción que tendrá con sus diferentes componentes.

El diseño básico de la aplicación tendrá una pantalla de inicio como presentación. También dispondrá de una página de inicio de sesión y otra de registro.

Imagen que contiene texto

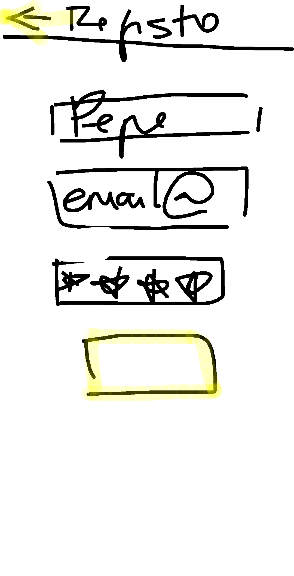
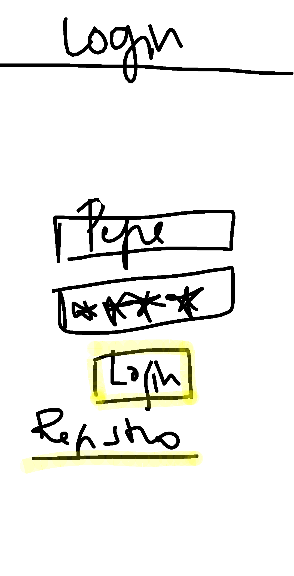
Descripción generada automáticamente

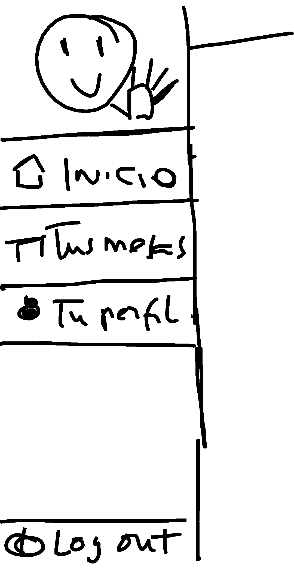
Ilustración 10: iParty, diseño de inicio

El inicio de la aplicación será una página de resumen (compuesto por una imagen, un título y otros datos que puedan ser de interés) de las mesas que buscan jugadores. En ella podrán aplicarse distintos filtros para reducir o ampliar la búsqueda. También se podrán crear mesas desde esta pantalla, pulsando sobre un botón con un icono de más que se encontrará tanto en la cabecera como en el apartado inferior de la pantalla.

Desde esta pantalla, además, se podrá acceder al drawer de la aplicación. Desde él se podrá navegar por los apartados más importantes de la misma.

Por último, en la Ilustración 11 se observa el diseño provisional de una publicación de mesa. En él se podrá ver toda la información relativa a la misma, estado actual, resumen y botón de solicitar plaza.

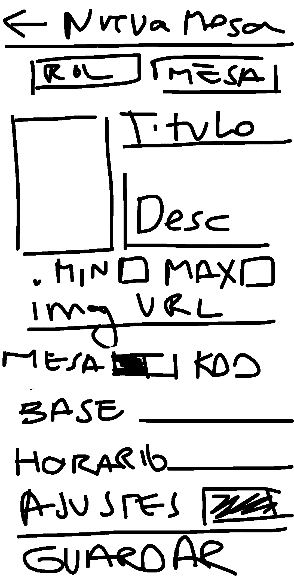
Imagen que contiene texto

Descripción generada automáticamenteImagen que contiene texto

Descripción generada automáticamente

Ilustración 11: iParty diseño página de inicio, drawer y publicación

Si se pulsa en el icono de añadir mesa, este te llevará a una nueva pantalla en la que podrá añadirse la información necesaria para crearla. Esta nueva mesa aparecerá en el apartado de *Tus mesas* (accesible desde el *drawer*). Se podrá filtrar a partir de unos botones en la parte baja de la pantalla (según si es una mesa creada para una única ocasión o si se trata de una actividad recurrente).

 Imagen que contiene texto

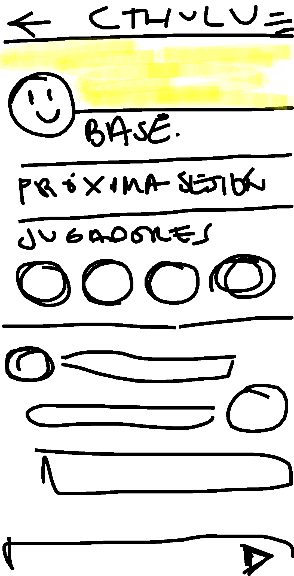
Descripción generada automáticamenteImagen que contiene texto

Descripción generada automáticamente

Ilustración 12: iParty diseño de nueva mesa y tus mesas

En la Ilustración 12 se puede observar la previsualización de una mesa en la que el usuario activo ha sido aceptado. En ella se verá la misma información de la que se disponía en la pantalla de solicitud, las demás personas aceptadas en dicha partida y un pequeño chat.

Por último, el usuario será capaz de editar su perfil y visualizar los perfiles de otros usuarios con los que compartan mesa:

 Imagen que contiene texto

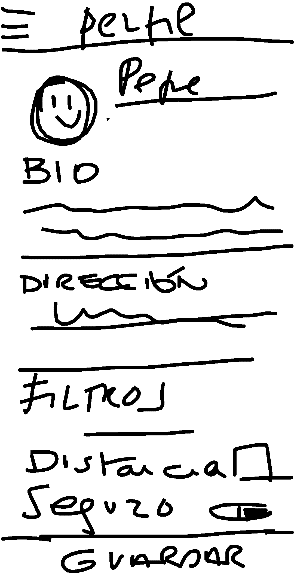
Descripción generada automáticamente 

Ilustración 13: iParty diseño de pantalla departida aceptada y perfil

## Elección de la paleta de colores

Para elegir los colores de la aplicación, se ha elegido proceder analizando los colores básicos de una imagen y, a partir de los más significativos, desarrollar una paleta de colores a través de las herramientas que ofrece Google.



Ilustración 14: Análisis de colores de imagen (Color palette generator, s.f.)

De los colores básicos que extrae la web, *Light Blue* será utilizado como color primario dentro de iParty y Crimson tomará el papel de color de error. Se utiliza la web de Paletton para encontrar el color secundario de la app, consiguiendo el siguiente resultado:

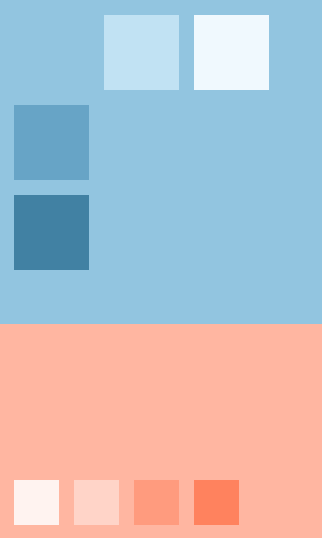


Ilustración 15: Color primario y secundario (Paletton, s.f.)

Por último, se utiliza la herramienta de Color Tool de Google para ver todas las tonalidades que podemos utilizar en el proyecto, así como otros datos de accesibilidad que deberían ser tenidos en cuenta para hacer que la aplicación lo más *user-friendly* posible.

Esquema básico de colores:

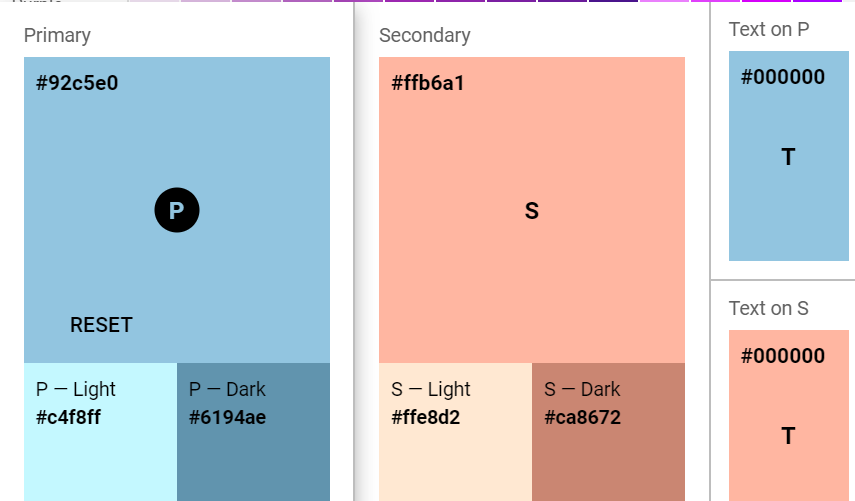
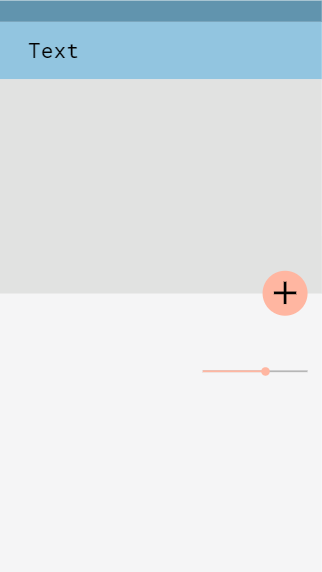
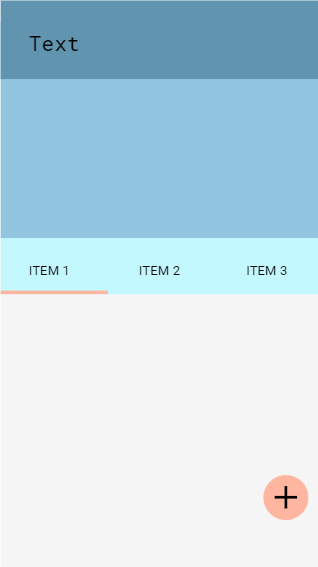
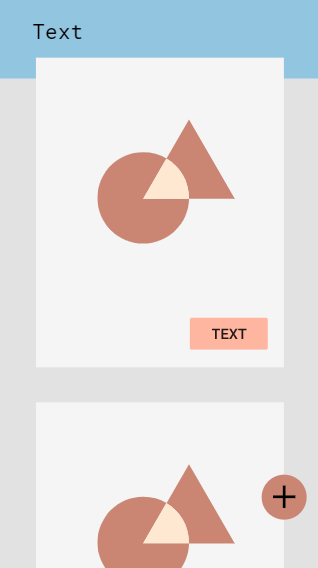


Ilustración 16: Paleta final de colores (Color Tool, s.f.)

Diseño pregenerado de la apariencia que tendrá la aplicación utilizando dicho diseño:

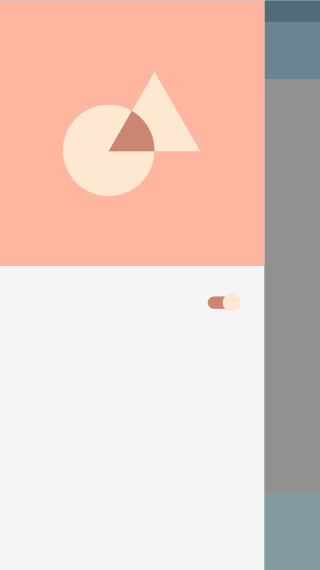
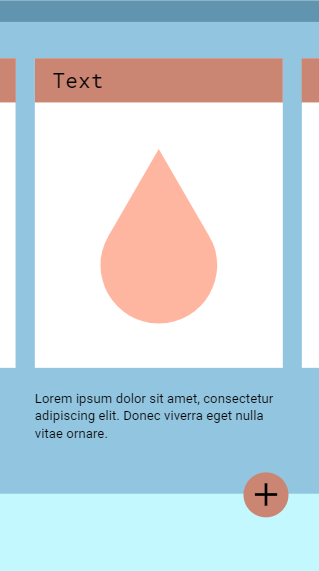
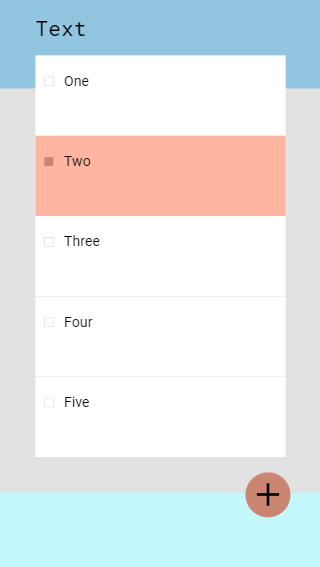
  

Ilustración 17: Previsualización del esquema de colores de iParty (Color Tool, s.f.)

Datos de accesibilidad de la aplicación teniendo en cuenta los colores primario y secundario y sus tonalidades:

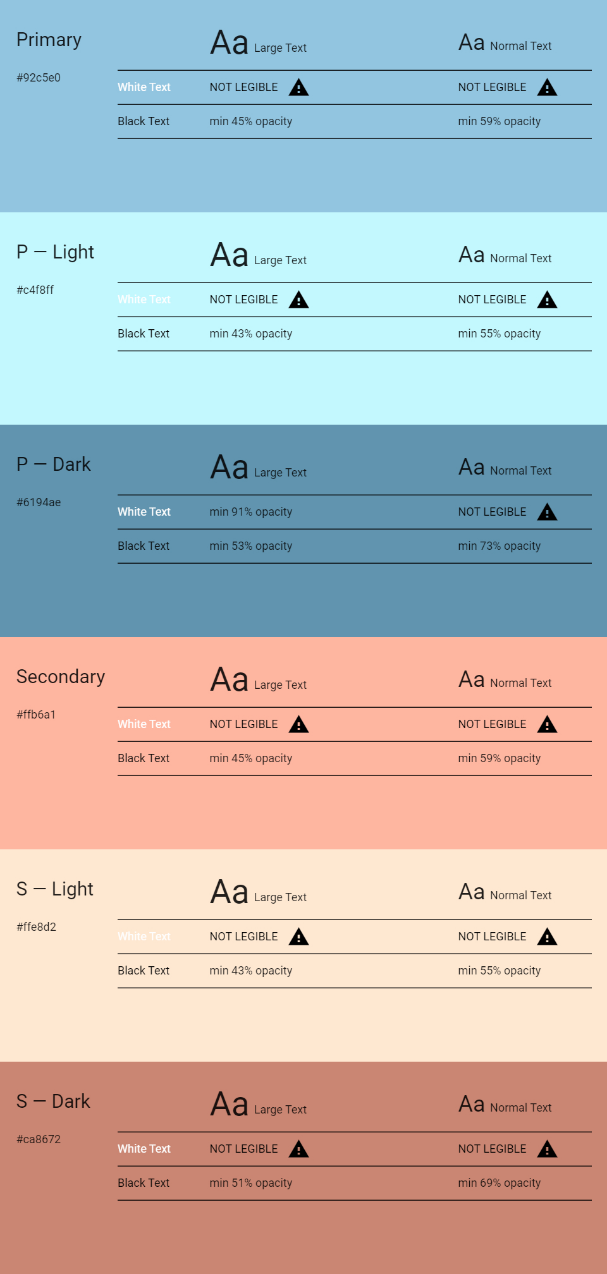


Ilustración 18: Datos de accesibilidad del color primario (Color palette generator, s.f.)

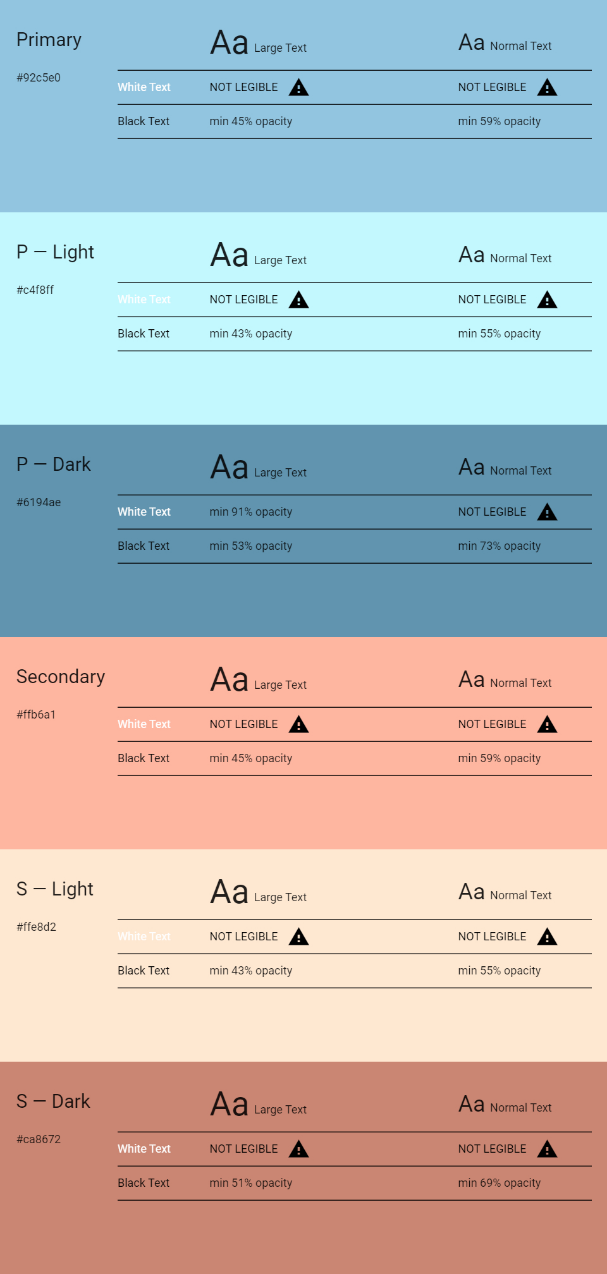


Ilustración 19: Datos de accesibilidad del color primario (Color palette generator, s.f.)

## Instalación del IDE y de FLutter

El IDE escogido para el desarrollo del proyecto es Visual Studio Code. Flutter puede desarrollarse en otros entornos orientados a este fin, como Android Studio o IntelliJ. Se escoge VSC debido a que se trata de un editor ligero y con una alta personalización a través de paquetes. Precisamente, de esa instalación caben destacar los siguientes paquetes (Cambi, 2019):

* Flutter y Dart: Google ha desarrollado estos paquetes para integrar tanto el framework como el lenguaje a VSC
* Bracket Pair Colorizer 2, debido a que Flutter es un framework que trabaja en casacada, facilitará el visualizar los fragmentos de código
* Pubspec Assist, para instalar fácilmente librerías

Por otro lado, Flutter necesita de Java 8 o anterior[[3]](#footnote-3) y de tener instalado la SDK[[4]](#footnote-4) de Android para poder desarrollar. Además, hace falta descargar su SDK y descomprimilar en la ruta deseada del ordenador (Windows install, s.f.). Una vez hecho esto, hace falta iniciar el archivo flutter\_console.bat. Desde allí hay que ejecutar flutter doctor, la herramienta que permite comprobar el estado de instalación y configuración de Flutter en el sistema.

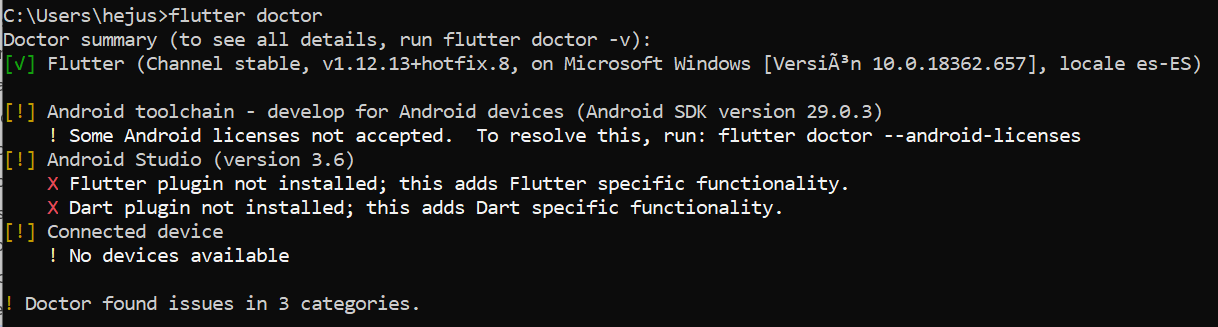


Ilustración 20: Primer análisis de flutter doctor

El único de los requisitos anteriores que debe de ser tenido en cuenta es el de aceptar las licencias de Android, debido a que el IDE que se va a utilizar para desarrollar es VSC. Para ello simplemente haya que ejecutar el comando flutter doctor --android-licenses y aceptarlas.

## Estructura del código

Se van a tener en cuenta las convicciones más generalizadas a la hora de desarrollar en Flutter a la hora de organizar el código:

* Todos los recursos no programáticos se encontrarán dentro de la carpeta raíz *assets*. Dicha carpeta incluirá otras con nombres identificativos, como *fonts* o *images*
* El código base se encuentra en la carpeta *lib*. Se dividirá entre *screens*, *widgets* y *models*.

## Main ()

El archivo main.dart contiene la información básica de ejecución de la aplicación: es desde dónde se indican la página principal, el árbol de navegación y los estilos. Es, además, la encargada de definir el contexto de la aplicación.

Esto lo consigue gracias a la función reservada void main() que indica el punto de inicio de ejecución. Este punto de inicio debe contener la información de la que se ha hablado anteriormente y se consigue generando un widget llamado MaterialApp:

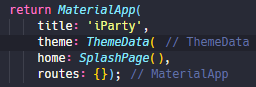
* La información básica
* Los temas que utilizará
* El árbol de rutas

A continuación, se desgranará el proceso de definición de dicho widget.

### Información básica

Se trata de la información necesaria para que la aplicación funcione. Entre ellas, se incluye el título, los temas, la pantalla principal y las rutas para la Navegación.

Código 1: Definición de MaterialApp



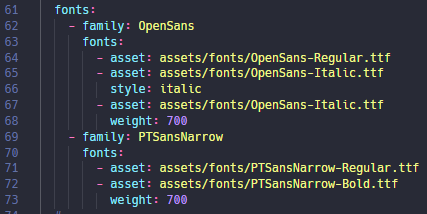
### Diseño del tema base

El primer paso en el diseño de una aplicación consiste en establecer la paleta de colores que utilizará, al igual que los estilos de texto y de los demás componentes. Para lograr este fin, se han utilizado dos herramientas:

* La inclusión en el archivo pubspec.yaml de las dependencias de las fuentes
* La definición de una clase ThemeData dentro del widget que define nuestra aplicación

El archivo pubspec.yaml dispone de toda la información que puede necesitar la aplicación para construirse. Deben definirse en ella los *assets* (como fuentes e imágenes) que vayan a utilizarse, así como las librerías y demás paquetes que quieran importarse. Utiliza su propia notación, en la que es muy importante respetar los espacios.

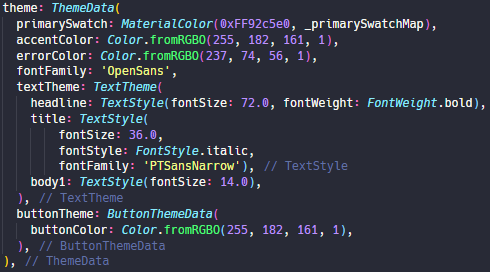
Código 2: Ejemplo de definición de las fuentes dentro del archivo pubspec.yaml



Como se puede observar en el fragmento anterior, para definir una fuente debe de hacerse dentro del apartado *fonts*. Dentro, es importante darle un nombre a través de la palabra reservada *family*. Para indicar la localización de las fuentes se utiliza *fonts* de nuevo. Cada una de las versiones de la fuente que se quiera incluir debe de tener una dirección relativa dentro del proyecto (palabra clave: *asset*). A continuación, si tiene alguna característica adicional, debe de indicarse. Si se trata de una letra itálica debe de indicarse la palabra clave *style*. En cuanto al grosor de las letras se establece con *weight* y números entre 100 y 900, en grupos de 100.

Una vez incluidas las fuentes dentro del proyecto, se deben de indicar dentro de nuestro widget MaterialApp con la propiedad ThemeData. Esta, por su parte, es la clase que configura la apariencia estética de la aplicación. Se puede leer más sobre este en la documentación de Google (ThemeData class, s.f.). En el caso de iParty, esta es la porción correspondiente:

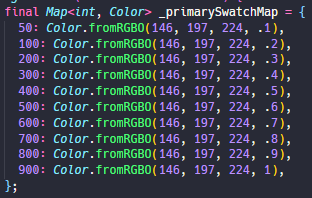
Código 3: Definición del estilo de iParty



|  |
| --- |
| En la definición de estilos, podemos identificar dos tipos de elementos. Aquellos genéricos que se aplicarán de manera automática a todos los elementos de la aplicación y aquellos específicos que solo aparecerán cuando se utilice el widget definido o cuando se indique. Si no se definen reglas, MaterialApp dispone de una clase básica (un ejemplo de dicho diseño básico se puede encontrar en la página 14) que se ocupará de dicha definición. Elementos de estilo genéricos Aquí se engloban los colores y el estilo de la fuente. Cabría destacar:   * primarySwatch con el que se define la paleta de colores primarios * accentColor, que es el color secundario * errorColor, define el color de errores * fontFamily, donde se hará referencia a una de las fuentes que hemos instalado en la aplicación anteriormente |

Es importante destacar que primarySwatch, la propiedad que define el color principal de la aplicación, necesita construirse a partir de un objeto MaterialColor, para lo que es necesario mapear las opacidades del color (Kumar, 2018):

Código 4:Mapeado de colores[[5]](#footnote-5)



Se puede leer más sobre la definición de las fuentes en la documentación oficial (Use a custom font, s.f.).

#### Estilos específicos

Se trata de definiciones de estilos que se aplicarán de manera automática cuando se utilicen dentro de la aplicación. En el Código 3 se puede observar que se han definido diferentes apartados:

* Diseño de texto (textTheme), en la que se ha definido la cabecera, títulos y un estilo de texto genérico
* Botones (buttonColor)

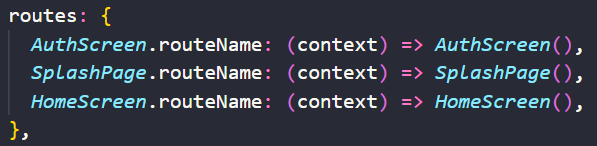
### Árbol de rutas

El árbol de rutas es una herramienta para facilitar la navegación dentro de la aplicación. Al tener toda la información unificada en un único lugar, centralizado, se reducen los errores y facilita la refactorización en caso de necesitarla.

El árbol de rutas se incluye dentro del widget MaterialApp y está compuesto por un *string* identificativo y un widget.

Es importante destacar que se ha estandarizado que el nombre de las rutas de navegación sea iniciado por una barra lateral / seguido por un nombre identificativo.

Código 5: Definición de rutas dentro de MaterialApp



### Redirección: página principal o página de Login/Registro

Esta operativa se realizará utilizando el servicio de autentificación de Firebase.

#### Inclusión de Firebase en el proyecto

En primer lugar se debe de acceder a la consola de Firebase (<https://console.firebase.google.com/>) y crear un nuevo proyecto:

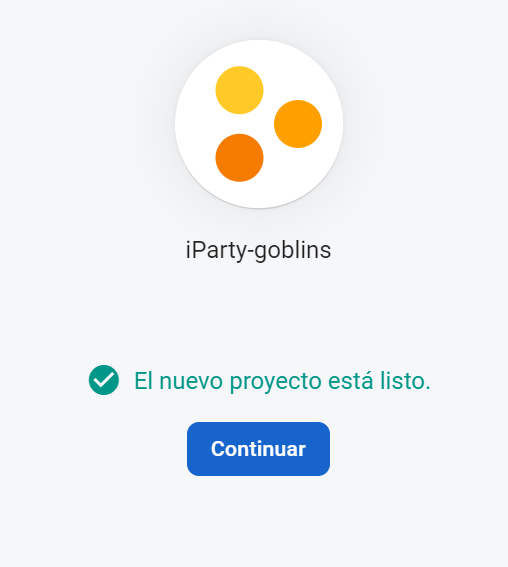


Ilustración 21: Pantalla de confirmación de creación de nuevo proyecto

Una vez creado el nuevo proyecto, se debe de configurar el tipo de aplicación que queremos crear. Aunque se pueden crear varios tipos de aplicación, para configurar la versión de iOS de la aplicación se necesita acceder a los metadatos a través del editor de MAC Xcode (Agrega Firebase a tu app de Flutter, s.f.). Por lo tanto, ese paso no podrá realizarse.



Ilustración 22: Dentro de la consola de Firebase, elección del tipo de aplicación

Por supuesto, se puede configurar el apartado de Android. Para ello, entramos en la carpeta y buscamos el AndroidManifiest para buscar el nombre del paquete

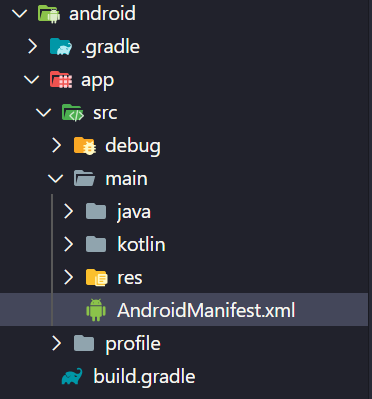


Ilustración 23: AndroidManifiest dentro del árbol de archivos de Flutter

Allí, en la primera línea, se encuentra el nombre de la aplicación:



Ilustración 24: Información de la aplicación en el manifiest de Android

Una vez conseguida esta información, debe de rellenarse los datos en la ventana de Firebase:

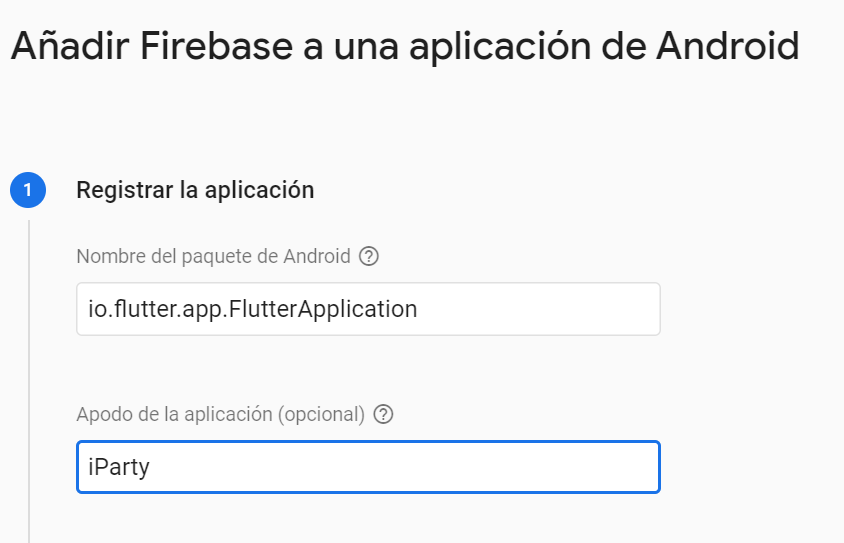


Ilustración 25: Inclusión de la app en Firebase

También hay que actualizar el build.gradle añadiendo la siguiente línea:

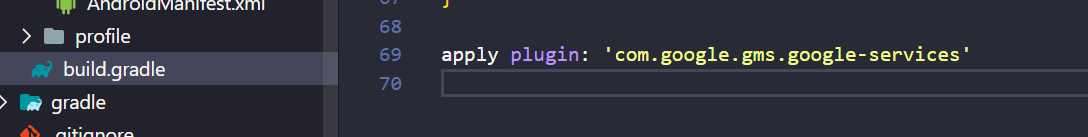


Ilustración 26:Actualización del build.gradle dentro de la jerarquía de Android

Y actualizar las dependencias del proyecto con la última versión de *Google Services*:

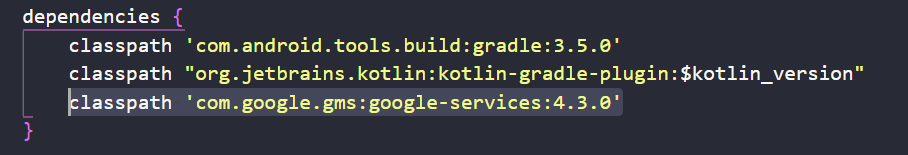


Ilustración 27: Inclusión de Google Services en las dependencias del proyecto

Por supuesto, hace falta incluir las librerías necesarias dentro del archivo pubspec.yaml:

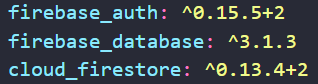


Ilustración 28: Librerías de Firebase

Con esto, la configuración del proyecto para incluir Firebase ha terminado.

#### Configuración de Firebase para hacer login

Se habilita dentro de la consola de Firebase el inicio de sesión a partir del correo electrónico y contraseña.

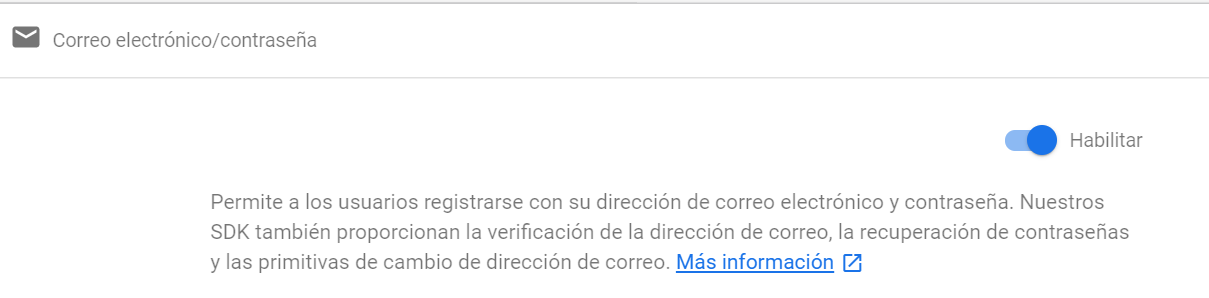


Ilustración 29: Firebase correo electrónico/contraseña

#### Operativa usuario registrado

La operativa de si existe un usuario *logueado* o no se realizará en este documento de arranque, ya que así nos permitirá gestionar la página de inicio en cada momento. Para ello, vamos a utilizar un *provider* (cuya librería incluiremos en el proyecto) (dash-overflow.net, 2020). Un *provider* permite el manejo de información de la base de datos y, en el caso de *Firebase* (que se trata de un base de datos a tiempo real), el manejo de los cambios y actualizaciones de esta. Para realizar este apartado, se han seguido las indicaciones de Aaron Saunders (Simple Firebase Login Flow in Flutter, Now Firebase, 2019).

Con el siguiente esquema se puede ejemplificar el uso del *provider* dentro de la aplicación. En él se ve como es el *provider* el que ejecuta el inicio de sesión y el que lo distribuye al resto de la aplicación.

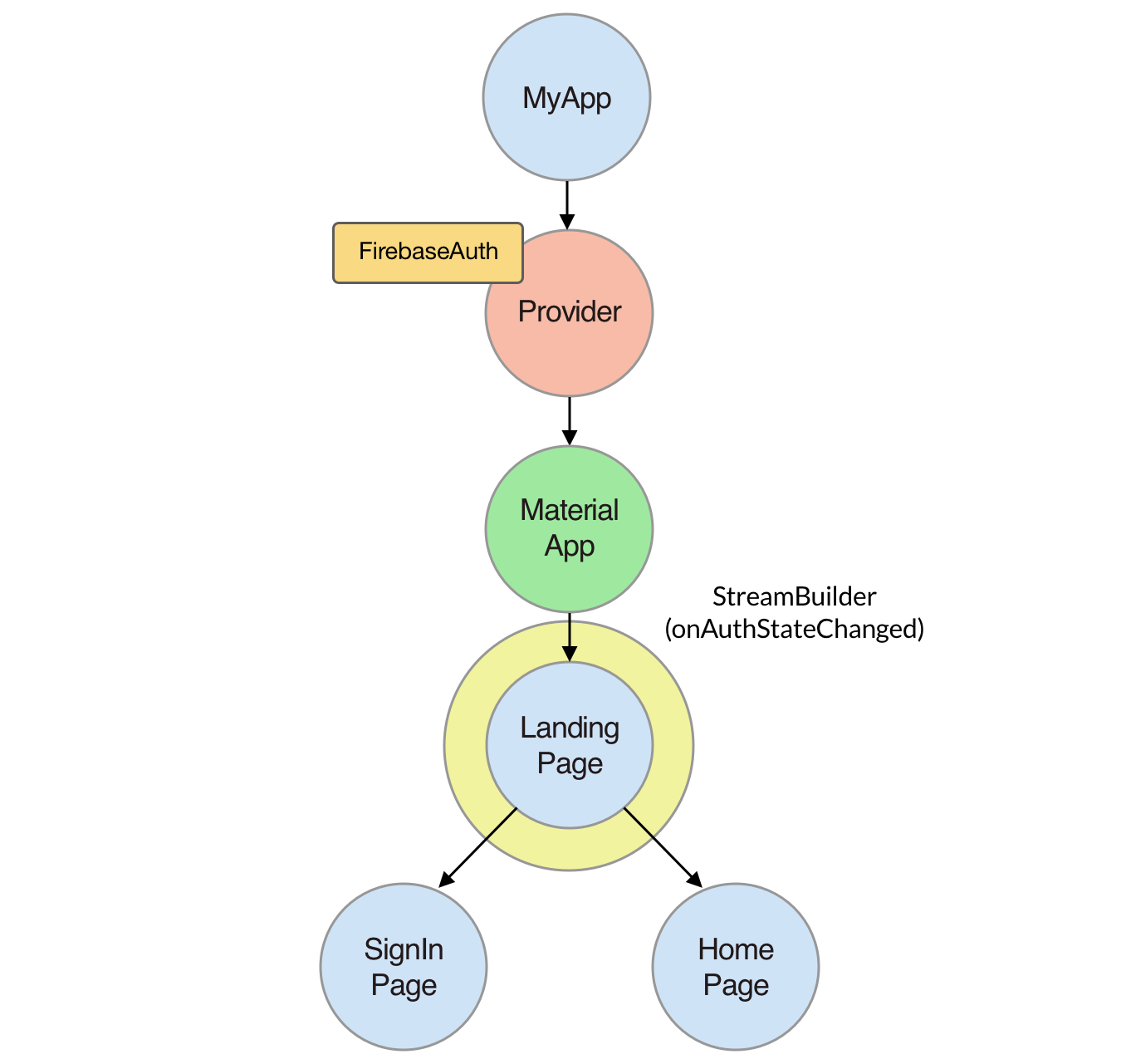


Ilustración 30: Esquema representativo del inicio de sesión utilizando Provider y Firebase (Bizzotto, 2019)

En primer lugar, es necesario construir el *provider*. En él, la operativa que se va a seguir es la misma que tienen los métodos nativos de *Firebase*, solo que además se incluirán las notificaciones de que se han actualizado los datos en inicio de sesión y cerrar sesión. Por lo tanto, lo único que dispondremos será:

* Un *get* que devuelva al usuario activo
* Una función que cierre sesión
* Y una función de inicie sesión

Esta clase debe de extender de la librería *provider* con ChangeNotifier.

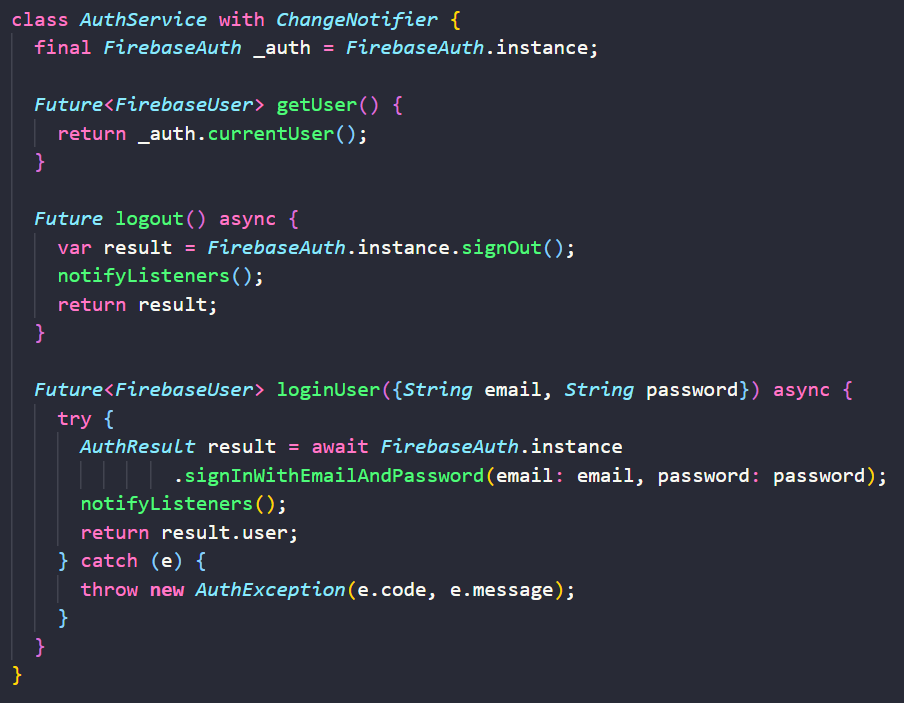


Ilustración 31: Clase de inicio de sesión

Si se vuelve a mirar la Ilustración 30 se puede observar como el *provider* se encuentra en las capas primeras del ejemplo. Esto se debe a que uno de los requisitos de su uso es que el *provider* siempre se tiene que incluir en la capa posterior o padre del widget que lo vaya a utilizar.

En este caso, se va a incluir dentro del método reservado main(), para que MaterialApp() pueda utilizarlo:



Ilustración 32: Método main() con provider

Y, después, en el home de MyApp() se debe de incluir la operativa que comprobará si se está o no *logueado* y qué ocurre en cada uno de los casos.

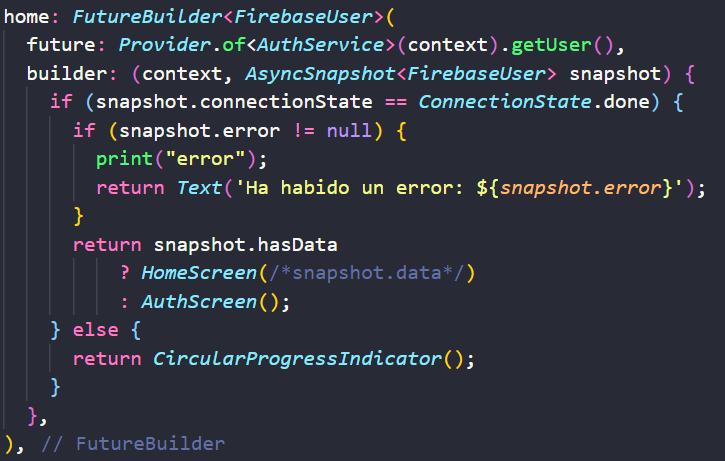


Ilustración 33: Operativa de inicio de sesión

Como puede verse en la imagen anterior, se crea un *future* para comprobar si existe un usuario *logueado*. De existirlo, lanza como *home* a *HomeScreen()*. De no, a AuthScreen(). En caso de que la conexión esté tardando, se mostrará un círculo que no para de girar en el centro de la pantalla:

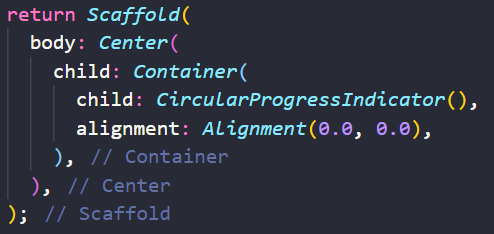


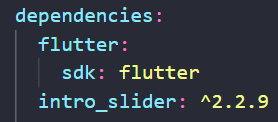
Ilustración 34: Modificación de CircularProgressIndicator

## Página de Intro

La creación de la página de *intro* dependerá de si es o no la primera vez que inicias la aplicación. De ser así, se cargará un *slider* que explicará brevemente cuál es su función y cómo se utiliza. Si ya se ha accedido anteriormente, esta página no se mostrará.

Para que la página de *intro* sea iterable y muestre varias pantallas, así como una navegación en el apartado inferior, se ve a utilizar una librería llamada *Flutter Intro Slider* (duytq94, 2020). El primer paso para instalar una librería es añadirlo a las dependencias del proyecto (que se encuentran, como las fuentes, en pubspec.yaml.

Código 6: Añadir una librería a las dependencias del proyecto



Una vez instalada hace falta importarla dentro del documento en el que se vaya a utilizar. En este caso, splash-screen.dart. El widget importado consta de dos partes que se van a utilizar:

* IntroSlider
* Slide

El IntroSlider representa el elemento de pantalla completa que tendrá una serie de propiedades para definirlo y, como hijos, una lista de sliders.

Pero antes de entrar a eso, hace falta señalar dos elementos que se utilizan a la hora de crear una pantalla en Flutter. En primer lugar, la clase que hereda nuestra pantalla. En segundo lugar, la clase Scaffold:

### Stateless o Stateful widget

A la hora de crear cualquier widget dentro de la aplicación, Flutter necesita que herede de una de las dos clases anteriores. La diferencia entre ambos radica en que el Stateless widget no es capaz de actualizarse. Es decir, que sus propiedades son inmutables. Es decir, una vez cargado, no es capaz de cambiar la información o disposición que nos muestra. Un Stateful widget, por su parte, es capaz de volver a cargar las partes de la aplicación que lo puedan necesitar.

En este caso concreto, al ser una pantalla informativa, será inmutable. Es decir, se construirá con un StatelessWidget.

### Scaffold widget

Esta clase implementa una disposición básica basada en el estilo de material design. Entre otras cosas, es la que provee de la API para mostrar el menú lateral o los botones flotantes (Scaffold class, s.f.).

(//Todo: ejemplo con/sin Scaffold)

### IntroSlider

El widget IntroSlider tiene una serie de propiedades prestablecidas que pueden configurar la apariencia de nuestra página. Sin ellas, tendría la apariencia básica. Esto es:

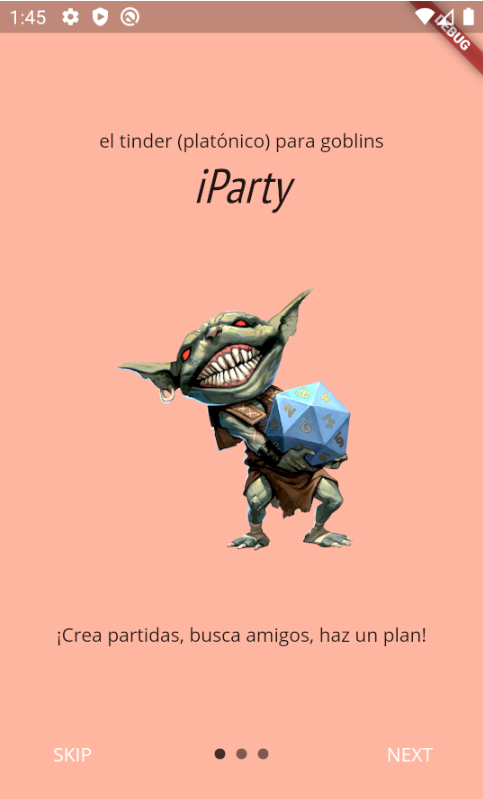
 

Ilustración 35: IntroSlider sin personalizar

El texto está en blanco, cosa que no nos recomendaba el software de elección de plantillas de Google para una óptima accesibilidad. Además, los textos están en inglés y los botones de *SKIP* y *DONE* no tienen ninguna funcionalidad. Esto se soluciona con el siguiente código:

Código 7: Creación del IntroSlider



Utilizamos el paquete de iconos que viene por defecto con Material Designs para hacer que en lugar de texto (*NEXT*) haya unas flechas indicando la dirección. Además, traducimos el texto y se determina que la función que hará el botón *SKIP* será la misma que la del *DONE*: cerrar la pantalla de introducción. No hace falta definir el estilo del texto que hay en la aplicación, ya que al utilizar el widget *Text* coge inmediatamente el que ya habíamos predefinido. El resultado es el siguiente:



Ilustración 36: Página de Intro modificada

La operativa de si se verá la página de *intro* o no se encuentra en la página de main-screen.dart.

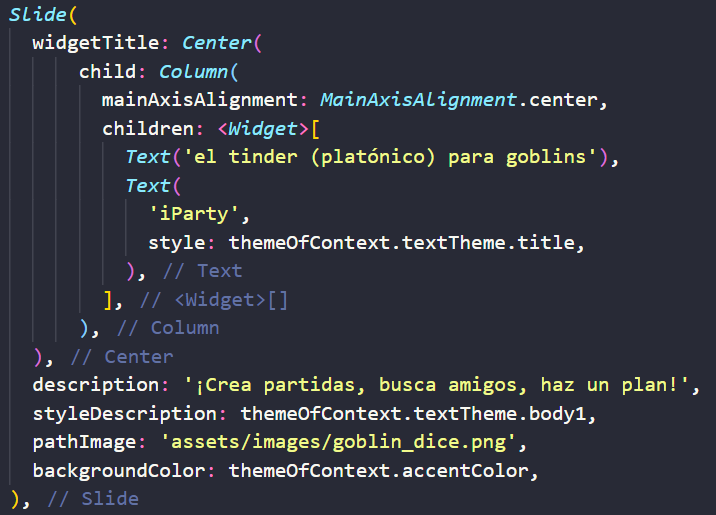
### Slide

Para mantener el código lo más limpio posible, se crea una función privada llamado \_genSlides que contiene una lista de *Slides*. Es precisamente esto el tipo de elemento que se espera IntroSlider en su propiedad slides.

Esta función tiene como parámetro el tema de la aplicación, para poder acceder a sus propiedades cuando las necesitemos. La primera propiedad que necesita es un título o un widget que tome su lugar. Como se quiere utilizar los estilos predefinidos y no los de aplicación, se escoge utilizar este último. Así que se utiliza un widget contenedor *Center* para indicar que el contenido estará centrado y cuyo hijo será *Column*. *Column* permite anidar widgets hijos uno debajo de otro. Dos widgets de *Text* completan el código, uno de ellos definido como título.

Además, Slide tiene como propiedades description (con su propiedad de estilo correspondiente), pathImage y backgroundColor. Con todo esto construimos la imagen que puede verse en la Ilustración 35, a la izquierda.

Código 8: Construcción del Slide



## Página DE LOGIN/REGISTRO

Es la pantalla que se carga cuando no hay ningún usuario registrado.

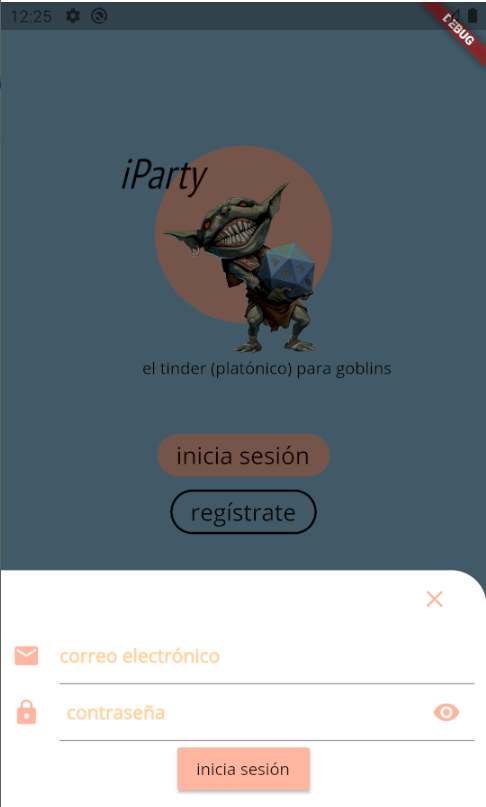
 

Ilustración 37: Resultado final de la página de Inicio de sesión/registro

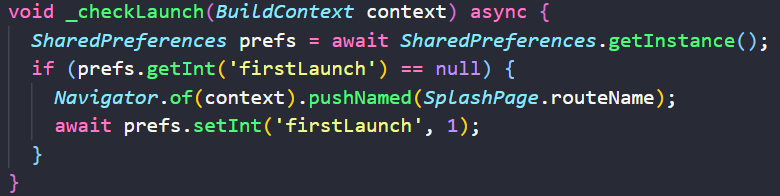
### Operativa de carga de la página de introducción

Para comprobar si es la primera vez que se ha abierto una aplicación, se utilizará SharedPreferences. Se trata también de una librería, que hay que definir en dependencias y llamar desde la propia página (Shared preferences plugin, 2020).

Entonces, con una operación condicional se comprobará si existe o no dicho elemento dentro de SharedPreferences. De no existir, es la primera vez que se abre la aplicación y debe de realizar dos cosas:

* Iniciar la página de introducción, utilizando para ello el árbol de navegación creado
* Escribir dentro de SharedPreferences para que cuando compruebe si se ha escrito o no, no devuelva nulo

Código 9: Operativa de la carga de la página de introducción



Este método se ejecuta desde el *builder* de la ventana. Como necesita que la ventana ya esté construida antes de llamarlo, se va a meter dentro de un método que solo ejecuta el código dentro cuando esta condición se cumple. Sin embargo, como se ejecuta cada vez que se reconstruyen los diferentes widgets dentro de la página, se utiliza una variable booleana para que únicamente se ejecute una vez.

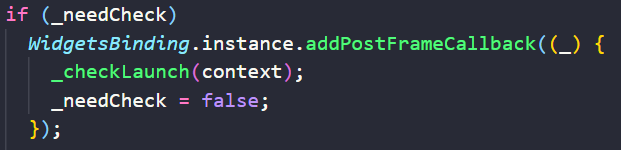


Ilustración 38: Ejecuta únicamente una vez la comprobación de la página de inicio de sesión

El esquema de la página será el siguiente:

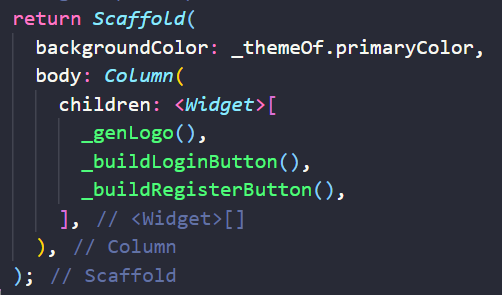


Ilustración 39: Esquema de la página de inicio de sesión

Existirán tres elementos clave: un logo, compuesto por una forma, texto y una imagen, y dos botones que abrirán unas ventanas modales desde la parte inferir de la página que nos permitirán desarrollar la operativa.

#### Logo

El posicionamiento del logo se consigue gracias al uso del widget Stack() y el de sus hijos Positioned(), que permiten: apilar widgets unos encima de otros y colocarlos en la posición deseada de la pantalla.



Ilustración 40: Estructura para la generación del logo

Cabe destacar la programación del círculo, ya que es un elemento BoxDecoration() con una forma circular y al que se le han dado propiedades de color y dimensión.

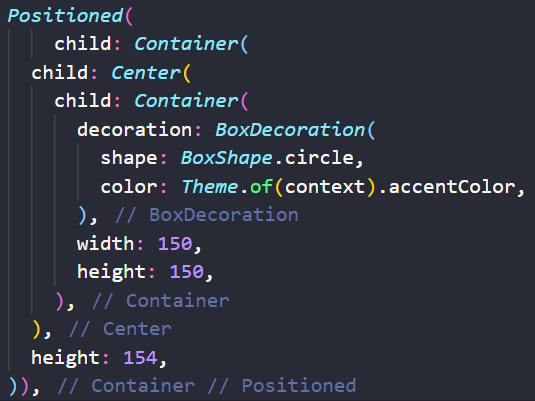


Ilustración 41: Diseño del círculo del logo

#### Ventana modal

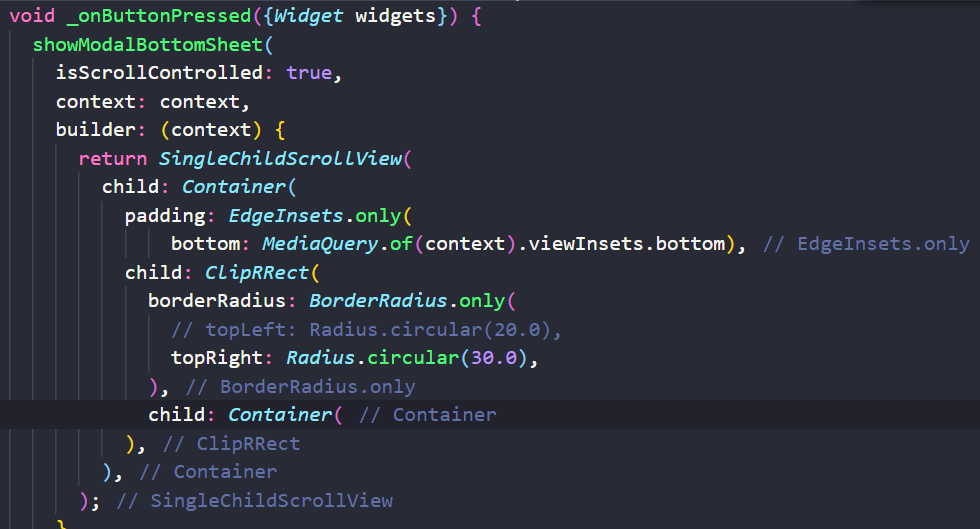


Ilustración 42: Ventana modal inferior

Para mostrar la ventana modal, se utiliza el método showModalBottomSheet(). Se le da la propiedad isScrollControlled para que se pueda descartar arrastrándola hacia abajo como se puede observar en la siguiente imagen:

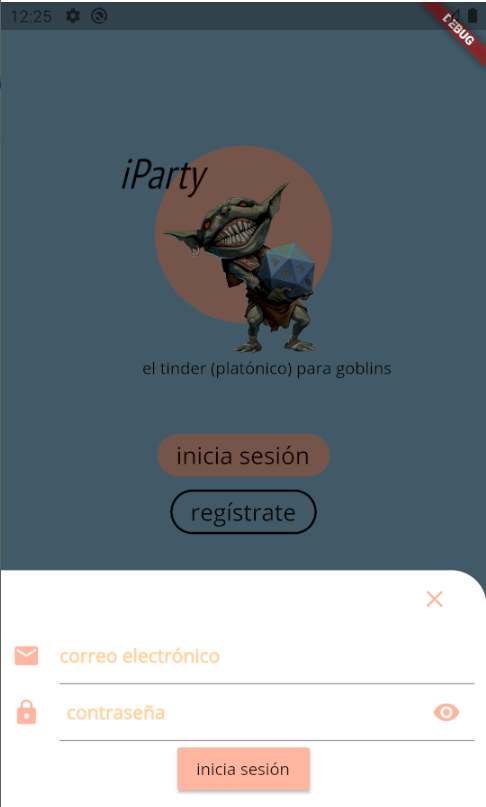
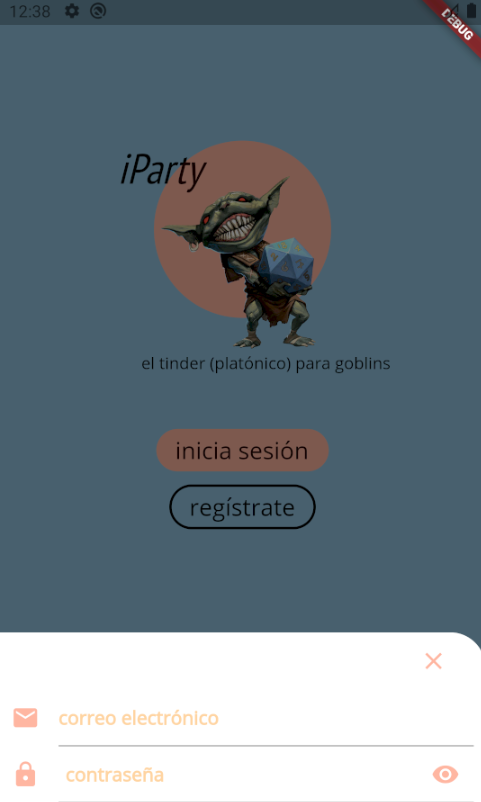
 

Ilustración 43: La ventana modal se puede descartar con un gesto hacia abajo

Lo que le da la forma redondeada es el uso de ClipRRect(). Este widget permite dar a cualquier otro widget una forma determinada. En este caso, simplemente se ha redondeado una de las esquinas.

Para que el efecto sea de transparencia, hace falta modificar el estilo básico de estos elementos dentro del MaterialApp():



Ilustración 44: Modificación del estilo de las ventanas modales inferiores

Además, es importante señalar el uso de un padding relativo para que, al utilizar el teclado este no nos tape las cajas de texto. Esto se consigue con MediaQuery.of(context) que permite acceder, entre otras cosas, al tamaño disponible de la pantalla como se puede observar en las siguientes imágenes:

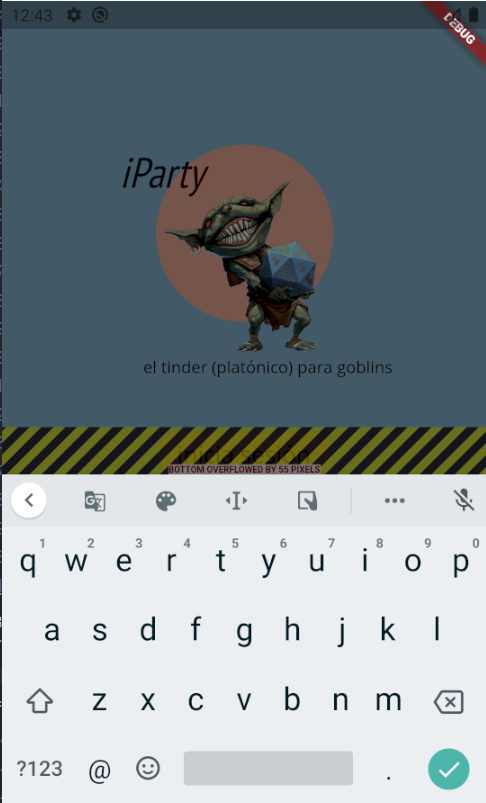
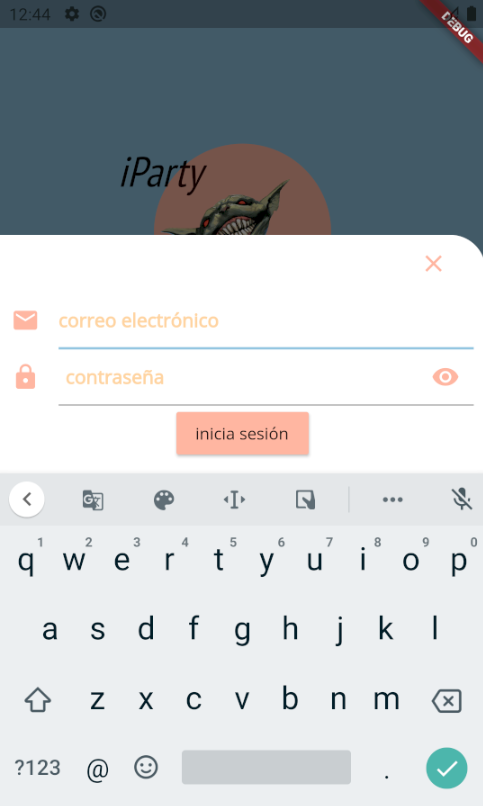
 

Ilustración 45: Diferencia entre carencia y uso de MediaQuery.of(context)

Dentro del Container(), tenemos:

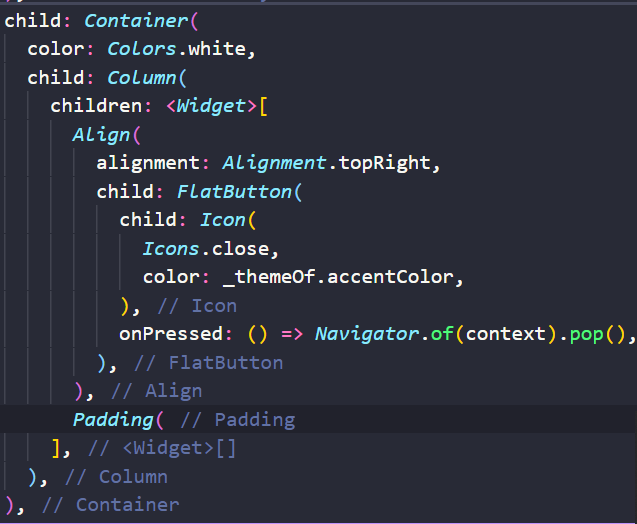


Ilustración 46: Diseño de los widgets del modal

El container ocupará todo el espacio disponible y dará el color blanco a nuestro modal. Además, se coloca un botón con forma de *x* en la esquina superior derecha que permitirá cerrar fácilmente el modal con el evento onPressed definido.

Por lo demás, se incluye el widget que se pedía por parámetro dentro del widget de Padding(), lo que permitirá reutilizar la ventana tanto para el modal de inicio de sesión como para el de registro.

#### Inicio de sesión

### Diseño de los modales de inicio de sesión y registro

<https://stackoverflow.com/questions/52663445/flutter-show-bottomsheet-transparency>

<https://github.com/flutter/flutter/issues/18564>

<https://stackoverflow.com/questions/52414629/how-to-update-state-of-a-modalbottomsheet-in-flutter>

# Conclusiones y propuestas

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

# Bibliografía

*Agrega Firebase a tu app de Flutter*. (s.f.). Obtenido de Firebase: https://firebase.google.com/docs/flutter/setup

*AngularDart*. (s.f.). Obtenido de Google Open Source: https://opensource.google/projects/angulardart

Bizzotto, A. (10 de junio de 2019). *Flutter: Global Access vs Scoped Access with Provider*. Obtenido de Code With Andrea: https://codewithandrea.com/articles/2019-06-10-global-access-vs-scoped-access/

Bolton, D. (27 de marzo de 2019). *The Fall and Rise of Dart, Google’s ‘JavaScript Killer’*. Obtenido de Dice: https://insights.dice.com/2019/03/27/fall-rise-dart-google-javascript-killer/

Cambi, D. (8 de agosto de 2019). *Must-have VS Code extensions for working with Flutter*. Obtenido de Medium: https://medium.com/flutter-community/must-have-vs-code-extensions-for-working-with-flutter-e31a421b9c68

*Color palette generator*. (s.f.). Obtenido de Canva: https://www.canva.com/colors/color-palette-generator/

*Color Tool*. (s.f.). Obtenido de Material Design: https://material.io/resources/color/#!/?view.left=1&view.right=0&primary.color=92c5e0&secondary.color=FFB6A1

Concise Software. (26 de agosto de 2019). *What is Flutter? Here is everything you should know*. Obtenido de Medium: https://medium.com/@concisesoftware/what-is-flutter-here-is-everything-you-should-know-faed3836253f

*Dart*. (s.f.). Obtenido de Google Open Source: https://opensource.google/projects/dart

*Dart vs JavaScript*. (5 de marzo de 2019). Obtenido de codemagic: https://blog.codemagic.io/dart-vs-javascript/

dash-overflow.net. (12 de febrero de 2020). *provider 4.0.4*. Obtenido de pub.dev: https://pub.dev/packages/provider

duytq94. (15 de febrero de 2020). *FLUTTER INTRO SLIDER*. Obtenido de pub.dev: https://pub.dev/packages/intro\_slider

En5MinutosRol. (10 de marzo de 2020). *En 5 Minutos Rol*. Obtenido de Twitter: https://twitter.com/En5MinutosRol/status/1237373767661338625?s=09

Firebase. (s.f.). *Firebase*. Obtenido de Firebase: https://firebase.google.com/?hl=es

*Flutter*. (s.f.). Obtenido de Google Open Source: https://opensource.google/projects/flutter

Flutter. (s.f.). *Flutter*. Obtenido de Flutter: https://flutter.dev/

flutter.dev. (10 de marzo de 2020). *Shared preferences plugin*. Obtenido de pub.dev: https://pub.dev/packages/shared\_preferences

FriendZone, A. T. (4 de marzo de 2020). *FriendZone*. Obtenido de PlayStore: https://play.google.com/store/apps/details?id=com.app.friendzone

inVerita. (10 de marzo de 2020). *Flutter vs Native vs React-Native: Examining performance*. Obtenido de Medium: https://medium.com/swlh/flutter-vs-native-vs-react-native-examining-performance-31338f081980

Kumar, M. (16 de diciembre de 2018). *Turn any color to Material Color for flutter*. Obtenido de Medium: https://medium.com/py-bits/turn-any-color-to-material-color-for-flutter-d8e8e037a837

*Layouts in Flutter*. (s.f.). Obtenido de Flutter: https://flutter.dev/docs/development/ui/layout

LiveLikeCounter. (10 de diciembre de 2019). *Flutter-Todolist*. Obtenido de GitHub: https://github.com/LiveLikeCounter/Flutter-Todolist

lookfirst. (30 de agosto de 2019). *flutter doctor with java 12 #39549*. Obtenido de GitHub Flutter: https://github.com/flutter/flutter/issues/39549

MeetMe.com. (24 de febrero de 2020). *MeetMe*. Obtenido de PlayStore: https://play.google.com/store/apps/details?id=com.myyearbook.m

Mobifriends. (16 de diciembre de 2019). *Mobifriends* . Obtenido de PlayStore: https://play.google.com/store/apps/details?id=com.mobifriends.app

O'Brien, D. (3 de enero de 2019). *Quick Proto*. Obtenido de Google Play: https://play.google.com/store/apps/details?id=net.dobrien.quickproto

Oppong, J. (11 de marzo de 2019). *Why Dart?* Obtenido de Creative Bracket: https://creativebracket.com/why-dart/

*Paletton*. (s.f.). Obtenido de Paletton: http://paletton.com/#uid=33t0u0kbLQl1T+-6UWRgeHhkgwh

Ramirez, V. (13 de febrero de 2017). *Qué es Firebase de Google: 7 motivos para usar esta plataforma en tu app*. Obtenido de Marketing 4 ecommerce: https://marketing4ecommerce.net/

Saunders, A. K. (7 de junio de 2019). *Simple Firebase Login Flow in Flutter, Now Firebase*. Obtenido de dev.to: https://dev.to/aaronksaunders/simple-firebase-login-flow-in-flutter-now-firebase-23nk

*Scaffold class*. (s.f.). Obtenido de Flutter: https://api.flutter.dev/flutter/material/Scaffold-class.html

Swehli, M. (12 de marzo de 2018). *Why Dart is the Language to Learn of 2018*. Obtenido de Medium: https://medium.com/@mswehli/why-dart-is-the-language-to-learn-of-2018-e5fa12adb6c1

*ThemeData class*. (s.f.). Obtenido de Flutter: https://api.flutter.dev/flutter/material/ThemeData-class.html

Thomas, G. (12 de diciembre de 2019). *What is Flutter and Why You Should Learn it in 2020*. Obtenido de freeCodeCamp: https://www.freecodecamp.org/news/what-is-flutter-and-why-you-should-learn-it-in-2020/

*Use a custom font*. (s.f.). Obtenido de Flutter: https://flutter.dev/docs/cookbook/design/fonts

Vega, C. (5 de marzo de 2019). *¿Por qué Flutter usa Dart?* Obtenido de Medium: https://medium.com/

VergiI. (30 de marzo de 2016). *Busco grupo de rol Madrid*. Obtenido de Comunidad Umbria: https://www.comunidadumbria.com/comunidad/foros/tema/41863

waldestrand. (5 de julio de 2008). *Fases del Proceso de Desarrollo del Software*. Obtenido de Sistemas VD: https://sistemasvd.wordpress.com/

*Windows install*. (s.f.). Obtenido de Flutter: https://flutter.dev/docs/get-started/install/windows

# Planos

# Anexos

1. Tal y como señala Bolton en su artículo para Dice (2019). [↑](#footnote-ref-1)
2. En pruebas de rendimiento, llega a duplicar la velocidad de procesamiento de JavaScript (Dart vs JavaScript, 2019). Cuando se habla de desarrollo móvil, mantiene un rendimiento muy similar al de Swift/Objetive-C y Java/Kotlin en pruebas de rendimiento Gauss-Legendre (rendimiento de memoria). En pruebas con el algoritmo Borwein (rendimiento de CPU), Flutter se posiciona entre 2 (Android) y 5 (iOS) veces más lento (inVerita, 2020). [↑](#footnote-ref-2)
3. Esto se debe a un problema de compatibilidad entre la SDK de Android y versiones posteriores de Java (lookfirst, 2019). [↑](#footnote-ref-3)
4. En la última actualización de Android Studio (3.6.1) se ha renombrado el SDK (a *Android SDK Platform-Tools*) de Android. Debido a este hecho, hace falta instalar *Android SDK Tools* (26.1.1) para que Flutter sea capaz de reconocerla. [↑](#footnote-ref-4)
5. Dart no utiliza palabras clave para definir atributos o funciones públicas o privadas. En su lugar utiliza el carácter \_ al inicio del nombre para indicar que se trata de una propiedad privada. [↑](#footnote-ref-5)