



**ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA**

**Trabajo fin de grado**

**Ingeniería del software**



Sport Clubs' Management

**Aplicación móvil para la gestión de escuelas deportivas**

**Realizado por**

**Antonio Manuel Montaña Aguilera**

**Pablo Vázquez Zambrano**

**Dirigido por**

**Carlos Guillermo Müller Cejas**

**Departamento**

**Lenguajes y sistemas informáticos**

**Sevilla, septiembre de 2020**

---

## Resumen

---

**ManageMySport** (en adelante *MMSport*) es una aplicación móvil para *iOS* y *Android* que permite la gestión de escuelas deportivas de cualquier ámbito. Concretamente, se facilitará la comunicación de los directores y entrenadores con los alumnos de dichas escuelas facilitando la división de los alumnos en distintos grupos, con chats privados entre usuarios de la misma escuela, calendarios de eventos, mensajes *broadcast* a un grupo de usuarios, etc...

La aplicación se ha desarrollado en el lenguaje de programación *Dart*, usando el *framework Flutter*, ambos desarrollados por *Google* para desarrollar tanto aplicaciones móviles como *webs*. Este *framework* tiene una serie de ventajas respecto a sus versiones nativas *Android* e *iOS*, sobre todo porque permite crear aplicaciones para ambos sistemas operativos basadas en un mismo código.

En este documento, el objetivo es documentar el desarrollo de la aplicación con la finalidad de resolver posibles cuestiones sobre la aplicación, mostrando las funciones de la aplicación en profundidad, las tecnologías y herramientas que nos han ayudado en el desarrollo, además de, sobre todo, hacer la función de memoria del *TFG* para el que ha sido desarrollada la aplicación.

### Palabras clave

*Flutter*, *MMSport*, *Dart*, *Android*, *iOS*, deporte, *móvil*, *app móvil*, *smartphone*, escuelas deportivas, *Framework*, aplicación nativa, *TFG*, gestión.

---

## Abstract

---

**ManageMySport** (hereafter *MMSport*) is a mobile application available for *iOS* and *Android* that allows managing sport schools of any kind. Specifically, it will permit communication between directors and trainers with their students, making easier dividing those students into groups, private chat rooms between users in the same sport school, event calendars, *broadcast* messages to a group of users, etc..

This application was developed using the programming language *Dart*, besides a *framework* called *Flutter*, both made by *Google* in order to develop either mobile applications or even websites. This *framework* has some advantages regarding its native versions for *Android* and *iOS*, especially because it allows creating applications for both operative systems using just one same code.

In this *wiki*, our objective is to document the application's development for the purpose of resolving possible questions about the application, showing the functionalities of the application in depth, technologies and tools that we used in the development, besides, of course, achieving its function of final degree (*TFG*) memory which this application was made for.

## Keywords

*Flutter*, *MMSport*, *Dart*, *Android*, *iOS*, sport, mobile, mobile application, *smartphone*, sport schools, *Framework*, native application, *TFG*, *final degree*, management.

---

## Agradecimientos

---

**Antonio:** A mis padres y mis tíos, por haberme apoyado siempre en el camino que hoy acaba y haberme hecho la persona que soy hoy día. A mi pareja, Rosa, por haberme apoyado y aconsejado en todo momento sobre este proyecto y por haberme soportado en los momentos duros durante mi etapa académica y personal.

**Pablo:** A mi familia por el gran apoyo y el hecho de alentarme constantemente a mejorar la app, especialmente a mi primo por toda la ayuda proporcionada en general, desde la elección de las herramientas hasta la resolución de todo tipo de dudas respecto a las mismas.

También queremos agradecer a Carlos Müller por el esfuerzo que ha hecho ejerciendo de tutor durante el desarrollo del proyecto, una gran persona siempre dispuesta a ayudar y facilitar al máximo posible la ejecución de este TFG a pesar de las dificultades para ello por la pandemia del *Covid-19*.

Por último, agradecer también a Rubén Pena, director de la escuela de tenis *ITTSport* por habernos dado ideas para el proyecto y *feedback* para mejorar cada vez más, tanto en el ámbito tenístico, en el personal y en el proyecto aquí expuesto.

---

# Índice

---

Resumen .....	I
Palabras clave .....	I
Abstract .....	II
Keywords .....	II
Agradecimientos .....	III
Índice .....	IV
1.    Introducción .....	1
1.1.    ¿Porqué MMSport? .....	1
1.2.    Problema tecnológico .....	2
1.2.1.    El fracaso de la versión nativa de Android e iOS .....	3
1.2.2.    Decidiendo nueva tecnología .....	3
1.3.    Objetivo del proyecto .....	5
1.4.    Conocimientos adquiridos .....	5
1.4.1. <i>Android</i> .....	5
1.4.2. <i>Firebase</i> .....	6
1.4.3. <i>Dart y Flutter</i> .....	6
1.4.4. <i>Codemagic</i> .....	7
2.    Tecnología y herramientas empleadas .....	8
2.1.    Tecnología .....	8
2.2.    Herramientas .....	13
2.2.1.    Herramientas de gestión .....	13
2.2.2.    Herramientas de documentación .....	14
2.2.3.    Herramientas de control de versiones .....	14
2.2.4.    Herramientas de desarrollo .....	14
2.2.5.    Herramientas de pruebas .....	15
3.    Aplicación desarrollada .....	16
3.1.    Glosario de términos .....	16
3.2.    Visión general del sistema .....	16
3.2.1.    Participantes en el proyecto .....	16
3.2.2.    Organizaciones involucradas .....	17
3.2.3.    Objetivos generales del sistema .....	17
3.3.    Requisitos del sistema .....	17
3.3.1.    Actores .....	17

3.3.2. Requisitos de información .....	18
3.3.3. Requisitos funcionales .....	19
3.3.4. Reglas de negocio .....	25
3.4. Análisis de costes .....	25
3.4.1. Costes directos .....	25
3.4.2. Costes indirectos .....	26
3.4.3. Reservas.....	27
3.4.4. Presupuesto total .....	27
3.5. Planificación y metodología de desarrollo .....	27
3.6. Análisis del sistema.....	30
3.6.1. Modelo del sistema .....	30
3.6.2. Arquitectura del sistema.....	34
3.7. Implementación del sistema.....	37
3.7.1. Entorno de desarrollo.....	37
Hardware.....	37
Software.....	38
3.7.2. Implementación de los distintos módulos del sistema .....	38
¿Cómo se suben documentos a <i>Cloud Firestore</i> ? .....	38
¿Cómo se consulta a la base de datos para obtener un documento? .....	39
¿Cómo se suben fotografías a <i>Firebase Storage</i> ? .....	40
RF-01: Registro en el sistema .....	41
RF-02: Inicio de sesión en el sistema .....	43
RF-03: Cierre de sesión en el sistema.....	44
RF-04: Creación de escuela en el sistema .....	45
RF-05: Listado de escuelas de una cuenta de usuario .....	48
RF-06: Listado de escuelas disponibles en el sistema.....	49
RF-07: Inscripción en una escuela .....	51
RF-08: Editar perfil social .....	52
RF-09: Listado de perfiles sociales inscritos en una escuela .....	53
RF-10: Menú principal .....	54
RF-11: Caché.....	59
RF-12: Ventana de <i>chats</i> .....	60
RF-13: <i>Chat Room</i> .....	63
RF-14: Aceptar y rechazar perfiles sociales .....	65
RF-15: Aceptar y rechazar escuelas.....	67
RF-16: Información sobre el grupo .....	67
RF-17: Crear grupo .....	70

RF-18: Editar grupo.....	77
RF-19: Asignar y eliminar alumnos/as de un grupo .....	80
RF-20: Eliminar grupo .....	80
RF-21: Información del calendario de eventos .....	80
RF-22: Crear eventos .....	81
RF-23: Editar eventos.....	84
RF-24: Eliminar eventos .....	85
RF-25: Eliminar perfiles sociales de una escuela .....	86
RF-26: Banear/desbanear usuarios del sistema .....	88
RF-27: Eliminar escuelas del sistema.....	88
RF-28: Confirmar correo electrónico.....	89
RF-29: Restablecer contraseña.....	89
3.8. Pruebas del sistema.....	90
3.8.1. Tests unitarios.....	90
3.8.2. Tests de aceptación.....	91
3.9. Instalación del sistema.....	110
3.9.1. Desarrolladores.....	110
3.10. Manual de usuario .....	110
4. Conclusiones.....	118
4.1. Problemas encontrados y soluciones aportadas .....	118
4.2. Cumplimiento de los objetivos.....	118
4.3. Futuras posibles implementaciones .....	119
4.4. Bibliografía.....	120

---

# 1. Introducción

---

## 1.1. ¿Porqué MMSport?

Actualmente, las escuelas deportivas conforman un sector muy poco digitalizado, en el que todas las gestiones se realizan a través de burocracia soporífera, llamadas de teléfono, suponiendo un gasto de tiempo enorme para los dirigentes de dichas escuelas, entregas de matrículas en persona, etc.



















Sin embargo, las escuelas deportivas son una parte muy activa en nuestras vidas, sobre todo en la de los jóvenes, participando en todo tipo de deportes como fútbol, baloncesto, balonmano, tenis, pádel, etc., suponiendo una gran carga económica en la sociedad actual.

Además, aparte del gasto económico que supone para las familias que sus miembros puedan ser partícipes de estas actividades deportivas en las mencionadas escuelas, la gestión de dicha participación supone un gasto de tiempo bastante destacable, con ejemplos como llamadas a los entrenadores para comprobar si un día se puede realizar un entrenamiento por cuestiones climatológicas, abonos mensuales, matriculaciones en las escuelas, cuestiones competitivas, ya sea por torneos, ligas, etc.

Con lo mencionado anteriormente, podemos encontrar un gran potencial de negocio en este ámbito, ya que se puede proporcionar una gran mejora en el tiempo invertido en las cuestiones de gestión expuestas. De ahí nace la idea de ManageMySport, una aplicación que permitirá a las escuelas deportivas una mejoría bastante notable en su gestión, tanto para entrenadores como para los miembros, ofreciendo servicios tales como mensajería instantánea, calendarios de eventos, notificaciones, etc.

Si hacemos un análisis sobre nuestros posibles competidores, encontramos lo que se muestra en la siguiente tabla:



	Novanet	GPASport	Playoff	TeamApp	MMSport
Centrado en escuelas deportivas					
Calendario de eventos					
Mensajería instantánea					
División en grupos					
Aplicación móvil					

En este análisis encontramos notables diferencias con casi todos nuestros competidores, que se centran más en equipos de fútbol, por ejemplo, o gestión de gimnasios y clubes de ocio para gestionar instalaciones deportivas. Sí que encontramos una competencia mucho más fuerte en **TeamApp**, siendo una aplicación móvil muy completa para gestionar equipos deportivos de todo tipo, que además se puede aplicar a la gestión de escuelas deportivas como nuestro caso, pero sin una división de grupos como la nuestra. También partimos con una ventaja notable para nuestro entorno, ya que TeamApp solamente está disponible en inglés, mientras que la nuestra está en español, por lo que la competencia aquí baja.

También podemos encontrar páginas web que sugieren el uso de varias tecnologías distintas, como es el caso de [esta página web](#), que sugiere nada menos que 41 tecnologías distintas. Nosotros, en **MMSport**, ofrecemos la unión de varias de estas tecnologías en una sola, facilitando enormemente las tareas de gestión de las escuelas deportivas.

## 1.2. Problema tecnológico

Nuestro objetivo final es desarrollar una aplicación capaz de resolver el problema planteado en el apartado anterior, y que además esté disponible para el mayor número de usuarios posible. Por tanto, debe estar disponible tanto para dispositivos con sistema operativo Android como para los que usen iOS. Por este motivo, debemos encontrar tecnologías que nos ayuden a cumplir dicho objetivo de la manera más eficaz posible, de las cuáles hablamos en los siguientes apartados.

### 1.2.1. El fracaso de la versión nativa de Android e iOS

La idea original del proyecto era desarrollar la aplicación de forma nativa tanto para *Android* como para *iOS*, empleando sus correspondientes formas de desarrollar de forma nativa. En el caso de *Android*, utilizando los *SDK* de *Android* correspondientes, con lenguaje de programación *Java*, y la arquitectura de *Android*, es decir, sus *activities*, *adapters*, etc. En el caso de *iOS*, nuestra idea original era utilizar máquinas virtuales para desarrollar en los entornos preparados para dicho desarrollo, siendo esto *XCode* como *IDE*, por ejemplo, *Swift* como lenguaje de programación, etc.

Durante bastante tiempo desarrollamos la aplicación en *Android* de forma nativa tal y como se especifica en el anterior párrafo. Sin embargo, mientras terminábamos la aplicación *Android*, nos planteamos el problema del desarrollo para el sistema operativo de *Apple*, y con investigación sobre la solución, nos encontramos con los siguientes problemas:

- Para poder publicar aplicaciones en la *App Store* de *Apple*, dichas aplicaciones deben estar desarrolladas en un entorno propio de *Apple*, es decir, ordenadores *Mac*, algo de lo que no disponemos y de lo que tampoco podemos disponer por temas económicos. De lo contrario, dichas aplicaciones son rechazadas automáticamente por la propia *App Store*.
- Las máquinas virtuales del sistema operativo *MacOS*, llamadas *Hackintosh*, no funcionan tan bien como otras máquinas virtuales de cualquier otro sistema operativo, además de ser sumamente complicadas de configurar y de no ser compatibles con bastante cantidad de hardware.
- La virtualización de imágenes de dicho sistema operativo mediante el uso de, por ejemplo, *Virtual Box* reduce drásticamente el rendimiento del sistema operativo, algo que hace inviable el desarrollo.

Por lo tanto, tuvimos que investigar otras posibilidades para poder desarrollar nuestra aplicación en *iOS*.

### 1.2.2. Decidiendo nueva tecnología

Existen muchas herramientas para desarrollar aplicaciones móviles nativas para los distintos sistemas operativos, pero nos fijamos especialmente en tres herramientas: *React Native*, *Ionic* y *Flutter*. Cada herramienta tiene sus ventajas e inconvenientes, como todas las herramientas dentro del ámbito software. A continuación, en la siguiente tabla, exponemos las diferencias entre dichas herramientas:

	React Native	Ionic	Flutter
Lenguaje de programación	JavaScript/React	HTML/CSS/JavaScript	Dart
Año de lanzamiento	Marzo 2015	2013	Mayo 2017
Desarrollado por	Facebook y comunidad	Drifty Co.	Google y comunidad

<b>Front-end</b>	Componentes propios	HTML, CSS y diseños propios	Widgets internos con Material Design
<b>Dificultad de aprendizaje</b>	Alta por TypeScript y sus componentes propios	Fácil, ya aprendido	Fácil, buena documentación y muy intuitivo
<b>Rendimiento</b>	Bueno, experiencia casi nativa	Buena interacción con el usuario	Muy bueno, optimizada la interacción y los gráficos

Como podemos ver, los tres tienen sus propias características. Respecto al lenguaje de programación, el más fácil a priori es *Ionic*, ya que son tecnologías que ya conocemos y son muy fáciles de manejar. *React Native* es más complejo al utilizar *TypeScript* también. Sin embargo, *Dart* es un lenguaje completamente nuevo, aunque es muy fácil de aprender.

Hablando sobre su comunidad, *Flutter* es la que menos comunidad tiene al ser relativamente reciente, aunque tiene un crecimiento bastante elevado. Además, al tener un *front-end* tan novedoso basado en *widgets*, tiene una documentación muy extensa, bastante mejor que las otras dos.

Sobre el rendimiento, la que mejor rendimiento tiene es *Flutter*, ya que es una tecnología reciente, aprendiendo de los errores de sus competidores y optimizando a un nivel muy alto sus aplicaciones. Esto se debe a que compila las aplicaciones de forma nativa para cada sistema operativo, y tanto *React Native* como *Ionic* usan "puentes" como intermediarios entre el *framework* y su sistema operativo de destino. Aunque para la tecnología web no tiene tan buen rendimiento, en las aplicaciones móviles tal como se ha mencionado es la que mejor rendimiento tiene y para nosotros es algo a tener muy en cuenta.

Tras haber analizado las posibilidades, además, buscamos herramientas que pudieran facilitarnos el desarrollo, encontrando *Codemagic*, una aplicación web que implementa integración continua en proyectos desarrollados en *Flutter* de forma muy sencilla, sin necesidad de configuración interna del proyecto a diferencia de otras como *Travis*.

Por tanto, nuestra elección final será *Flutter*, por su rendimiento tanto en *Android* como en *iOS*, además de su novedosa forma de desarrollar el *front-end* con los *widgets*. Como añadido, tras nuestra elección, decidimos detener el desarrollo de la aplicación nativa de *Android*, ya que consideramos que nuestro objetivo de aprender a desarrollar aplicaciones en el sistema operativo de forma nativa se ha cumplido, y no queremos desarrollar una aplicación nativa aparte de la de *Flutter* porque cuando finalizásemos el desarrollo en el *framework* ya tendríamos una aplicación en el sistema operativo de *Google*. Esta decisión también está respaldada por la diferencia en la formación, ya que para comenzar el desarrollo en *Android* nos formamos durante 40 horas, y aun así nos costó empezar, mientras que con *Flutter* con una formación de 2 horas empezamos a desarrollar la aplicación. Esto será más detallado en el apartado de [conocimientos adquiridos](#).

### 1.3. Objetivo del proyecto

El objetivo de este proyecto como Trabajo de Fin de Grado consiste en desarrollar una aplicación móvil que funcione tanto en *Android* como en *iOS* usando el *framework Flutter*. Para ello, se establecen objetivos más específicos que habrá que alcanzar para dar por finalizado el *TFG*:

- Iniciarnos en el ámbito de la movilidad, algo que motiva a los integrantes del equipo como posible futuro profesional.
- Aprender a desarrollar aplicaciones en *Flutter*. Además, nos llevamos el desarrollo de la aplicación de *Android*, aunque no se haya finalizado, pero conservamos conocimientos muy valiosos.
- Mejorar nuestros conocimientos actuales en las herramientas usadas que se especifican más adelante.
- Obtener una aplicación móvil real para comercializarla una vez se encuentre en un estado publicable tras la realización del trabajo de fin de grado.
- Relacionado con el objetivo anterior, llegar al máximo número de escuelas deportivas posibles para que usen nuestra aplicación en el futuro y obtener beneficios gracias a la aplicación.

Hay que añadir que *MMSport* es una aplicación aplicable a prácticamente todas las escuelas deportivas que se pueden encontrar. Sin embargo, este objetivo no era así originalmente, pues el caso de estudio original era el de una escuela de tenis, con el fin de desarrollar una aplicación exclusiva para esa escuela, en la cual entrenamos los dos integrantes del equipo. A raíz de esa idea y durante el desarrollo de la misma, nos dimos cuenta de la posibilidad de generalizar el proyecto y, con la ayuda de nuestro entrenador, al que agradecemos al inicio de la memoria, pudimos generalizar la idea y desarrollar lo que es *MMSport* con su actuación como *Product Owner* dando *feedback* sobre el proyecto en todo momento y aportando nuevas ideas, además de ser el primero en probar la aplicación y ponerla en práctica en su escuela deportiva.

### 1.4. Conocimientos adquiridos

Antes y durante la realización del proyecto, tuvimos que adquirir una serie de conocimientos necesarios para el correcto desarrollo del proyecto. Estos conocimientos los hemos obtenido de diversas plataformas en Internet, las cuales se enumeran en los siguientes párrafos:

#### 1.4.1. *Android*

- **Udacity:** Para iniciarnos en *Android* encontramos muy interesante [este curso](#), el cual tiene una duración de 60 horas aproximada. Sin embargo, nosotros nos formamos durante 40 horas. Esta cantidad de horas se debe a que, primero, *Android* de forma nativa tiene una arquitectura muy compleja de entender inicialmente, la cual sigue la arquitectura MVC (*Model-View-Controller*), pero de una forma bastante compleja de entender para principiantes como nosotros. En segundo lugar, omitimos algunas secciones del curso ya que no nos ofrecían

nada que nos interesara porque ya poseíamos esos conocimientos al haberlos obtenido durante nuestra formación en el Grado.

- **Documentación oficial:** Por suerte para nosotros, la [documentación oficial de Android](#) está muy bien estructurada y explica con claridad muchos conceptos de *Android*. Aquí se pueden encontrar guías de inicio, ejemplos de funcionalidades, buenas prácticas, etc.
- **Internet en general:** En general, hemos consultado bastantes páginas web distintas para dudas que teníamos sobre el desarrollo en *Android*. Sobre todo, como todos, destacamos [Stack Overflow](#) y [YouTube](#). En especial, para la funcionalidad de los chats queremos destacar [este tutorial](#) para esta funcionalidad, el cuál sirvió de inspiración para desarrollar esto.

### 1.4.2. *Firebase*

- **Documentación:** En este caso no hicimos ningún curso, simplemente investigamos en la documentación de *Firebase* lo que necesitábamos para las diferentes interacciones con la base de datos. En el caso de *Android*, se puede consultar [aquí](#) todas las dudas sobre *Firebase*, destacando en nuestro caso:
  - **Cloud Firestore:** **Cloud Firestore** ofrece una base de datos NoSQL alojada en la nube, especialmente optimizada para aplicaciones móviles y tecnologías web.
  - **Authentication:** Es un servicio que ofrece a los desarrolladores un sistema de identificación y registro de usuarios que facilita enormemente la implementación de los sistemas de gestión de usuarios.
  - **Storage:** Esto es un servicio que facilita el almacenamiento de archivos multimedia.

La información proporcionada sobre estos servicios introducidos será ampliada en las próximas secciones.

- **Stack overflow:** En [Stack Overflow](#) hemos resuelto muchísimas dudas sobre las interacciones con la base de datos de *Firebase*, como por ejemplo [esta duda](#) en la que intentamos resolver un problema sobre hacer múltiples consultas en un mismo método, o [esta otra](#) en la que intentamos implementar el inicio de sesión en nuestro sistema.
- **Caso de Flutter:** Para el desarrollo en *Flutter* la documentación es distinta. La configuración inicial de la aplicación se encuentra [aquí](#), aunque el resto de la documentación no está en la web oficial, sino que se encuentra alojada en [esta web](#). Aquí dividen las funcionalidades y los paquetes, encontrándolos en los apartados [Cloud Firestore](#), [Authentication](#) y [Storage](#).

### 1.4.3. *Dart y Flutter*

- **Udemy:** Para formarnos en *Flutter* usamos [este curso](#), el cual ofrece una introducción al *framework*, explicando su estructura, una base del lenguaje que emplea, *Dart*, y tutoriales de cómo usar los *widgets* y funcionalidades más utilizadas en los proyectos desarrollados con esta tecnología. En este curso únicamente invertimos unas 2 horas aproximadamente, ya que consideramos que tanto *Flutter* como *Dart* son muy fáciles de aprender y de utilizar, con una arquitectura muy sencilla.

- **Documentación:** La [documentación](#) de *Flutter* es, simplemente, excelente. Tiene múltiples ejemplos de sus funcionalidades, de sus *widgets*, como navegar entre distintas pantallas, etc.
- **Stack Overflow:** La clásica mención, ciertas dudas resueltas gracias a esta maravillosa página, como [esta](#) en la que teníamos un problema con los textos más largos que la pantalla, o [esta otra](#) en la que no sabíamos como implementar un menú en la esquina superior derecha de la ventana principal.

#### 1.4.4. Codemagic

- **Documentación:** Esta herramienta oficial para ofrecer servicios de integración continua con *Flutter* nos ha sido muy útil. Su documentación está [aquí](#).

---

## 2. Tecnología y herramientas empleadas

---

### 2.1. Tecnología

En el desarrollo de nuestra aplicación usamos distintas tecnologías que nos han ayudado a alcanzar nuestro objetivo. Al igual que todas las aplicaciones, tenemos una división clara entre la parte visual e interacción del usuario, el *front-end*, y la base de datos, nuestro *back-end*.

Nuestro *front-end* está implementado usando el *framework Flutter*, que usa el lenguaje de programación *Dart*. Por la parte del *back-end* hemos utilizado *Firebase*. Procedemos a introducir las tecnologías:



**Dart**, según la información sacada de [Wikipedia](#) es un lenguaje de programación de código abierto desarrollado por *Google*. Su presentación data del 10 de octubre de 2011, aunque su primera versión estable no llegaría hasta el 14 de noviembre de 2013, con su versión 1.0. *Dart* fue lanzado originalmente con la idea de ofrecer una alternativa más moderna a *Javascript* para el desarrollo de aplicaciones web. Sin embargo, hoy en día podemos usar *Dart* para crear aplicaciones web, de escritorio, servidores o, como en nuestro caso, aplicaciones móviles.

A continuación, vamos a enumerar las características más importantes que hemos encontrado en nuestra experiencia que definen a *Dart*:

- Es un lenguaje de [Programación Orientada a Objetos](#), soportando características de este tipo de lenguajes tales como interfaces, clases, herencia de tipos, etc.
- Al igual que ocurre con *Javascript*, no es necesario (aunque sí es muy recomendable para la mayor comprensión del código) especificar el tipo de variable u objeto que estamos utilizando, ya que *Dart* es capaz de averiguar el tipo que estamos utilizando. También existe un tipo genérico, *dynamic*.
- A diferencia de otros lenguajes, en *Dart* no existen las palabras reservadas *public*, *private* o *protected*. Para especificar que una función o clase es privada, se debe incluir en la definición de dicha función o clase el carácter `_`. Estará más claro tras ver el ejemplo:



```

void listarEscuelas(){
    print('Este método es público');
}

void _listarPerfiles(){
    print('Este método es privado');
}

```

- Al igual que casi todos los lenguajes de programación orientados a objetos, *Dart* dispone de todas las operaciones aritméticas como la suma y la resta y las operaciones lógicas tales como && y ||, además de ofrecer la posibilidad de ejecutar condicionales con if, else, bucles con for o foreach, etc.
- Algo que usamos mucho en nuestro proyecto son las operaciones asíncronas. En *Dart* tenemos los tipos Stream y Future, lo que permite que los objetos de este tipo devuelvan algún valor en el momento en el que la operación termina (por ejemplo, consulta a nuestra base de datos) sin necesidad de que la operación termine, lo que permite el diseño y la implementación de una experiencia de usuario mucho más notable para el usuario. Para ello, también tenemos las palabras reservadas async y await, facilitando la inserción de operaciones asíncronas. async convierte una función en asíncrona, mientras que la palabra await indica que la ejecución de dicha función asíncrona debe esperar a obtener el resultado de la definición que la acompaña. Aquí vemos un ejemplo:

```

Future<List<SportSchool>> _listarEscuelas() async {
    // Esta función devolverá una lista de escuelas deportivas
    // al terminar su ejecución.
    var data = await jsonDecode(http.get('urlRandom'));
    // Con la palabra await, esta función no continuará hasta que
    // la consulta a la url termine y la variable tenga los datos.
    return data;
}

```

Como podemos ver, *Dart* es un lenguaje de programación muy sencillo, además de parecerse mucho a *Javascript*, lo que nos ha facilitado mucho el aprendizaje tal y como mencionamos en el apartado de [conocimientos adquiridos](#). Además, su capacidad para trabajar con las operaciones asíncronas nos ha hecho muy sencillo las operaciones con la base de datos y con la caché del propio móvil para guardar información de interés que nos resulte útil.

Podemos acceder a la página oficial de *Dart* [aquí](#).





**Flutter**, con la información obtenida de [Wikipedia](#) es un SDK de código abierto desarrollado por *Google* orientado al desarrollo de aplicaciones móviles. Su primera versión estable data del 4 de diciembre de 2018, por lo que es un *framework* muy reciente, bastante modernizado. Lo más destacado de este *framework* es que permite desarrollar aplicaciones móviles y web desde una sola base de código, por lo que podemos desarrollar nuestra aplicación *Android* e *iOS* sin usar otra tecnología diferente. Además, usa el lenguaje de programación descrito anteriormente, *Dart*.

Aparte de lo mencionado sobre su compilación nativa tanto para *Android* como para *iOS*, *Flutter* tiene como objetivo ofrecer un **desarrollo rápido** de aplicaciones, ofreciendo un gran control sobre la *UI*, alcanzando un rendimiento similar a una aplicación puramente nativa. Por lo tanto, hay que destacar dos características fundamentales que lo hacen una opción increíble a la hora de desarrollar aplicaciones como la nuestra:

- Las funciones de **hot reload** y **hot restart** son, sin lugar a duda, un gran avance en el desarrollo de las aplicaciones móviles. Estas funciones, tal y como sus nombres indican, realizan cambios en la aplicación según los cambios que haga el desarrollador en el código, de forma que, una vez la aplicación está compilada, *Flutter* reconstruye la aplicación en base a lo que hay compilado, permitiendo ver en aproximadamente dos segundos toda la aplicación compilada de nuevo, con sus cambios correspondientes. Esto agiliza muchísimo el desarrollo, permitiendo probar varias opciones de diseño en cuestión de segundos, a diferencia de las compilaciones tradicionales que ralentizan muchísimo el flujo de trabajo.
- Lo que hace algo único a *Flutter* es que la programación de la aplicación se basa en el uso de *widgets*. Estos *widgets* son una gran cantidad de componentes basados en [Material Design](#) que vienen ya definidos en el propio *framework*, de forma que el programador enlace unos *widgets* dentro de otros para dar forma a las distintas pantallas de la aplicación, y finalmente a la aplicación en sí. Esto se puede ver en el siguiente ejemplo, donde construiremos una pequeña vista con un botón de adorno para mostrar cómo funciona el árbol de *widgets*:

```
// Todas las vistas comienzan con este Widget, para especificar que la
// vista va a emplear elementos de Material Design.
return Material(
  // El Scaffold es la plantilla para casi todas las pantallas,
  // ya que permite añadir un AppBar en la
  // vista que queramos, además del cuerpo de la pantalla.
  child: Scaffold(
    // El Scaffold tiene como hijos la AppBar de la pantalla (en
    // este caso vacía) y el cuerpo, un
    // contenedor en el que pondremos nuestro botón
    appBar: AppBar(),
    body: Container(
      // El contenedor tiene propiedades como el padding, el
      // ancho, y el alto de dicho contenedor.
      // Tendrá como hijo el botón.
      height: 50,
      width: 400,
      padding: EdgeInsets.all(10.0),
      child: RaisedButton(
        // El botón también tendrá características y un hijo.
        // En este caso será el texto que aparece
        // dentro del botón, y le cambiaremos el color. Además
        // por ser un RaisedButton, tendrá
        // cierta elevación respecto a la pantalla.
        color: Colors.deepPurple,
        child: Text("Press here!",
          // Como propiedad del texto, le añadimos
          // un estilo para cambiar el color y el
          // tamaño de la fuente.
          style:
            TextStyle(fontSize: 20,
              color: Colors.greenAccent)),
        // Al pulsar en el botón se ejecutará la función
        // onPressed
        onPressed: () {
          // Cualquier cosa que queramos hacer al
          // pulsar el botón
        }
      )),
  );
```

Como podemos observar, cada *widget* tiene sus propiedades como si se trataran de parámetros en cualquier otro lenguaje. De esta forma lo hace muy intuitivo y sencillo de aprender y de programar, anidando *widgets* para dar lugar a pantallas bastante atractivas para el usuario contando con la ayuda de los diseños de *Material Design*. Sin embargo, al contener tanta anidación, es posible que dé lugar a un código demasiado extenso anidando *widgets* uno detrás de otro, por lo que conviene estructurar muy bien los *widgets*, posiblemente dividiendo funcionalidades en el código de una misma pantalla, para facilitar el mantenimiento y la comprensión del mismo.

Además, existen multitud de paquetes y librerías que hemos utilizado en el desarrollo del proyecto que introduciremos cuando sea conveniente analizando la aplicación desarrollada.

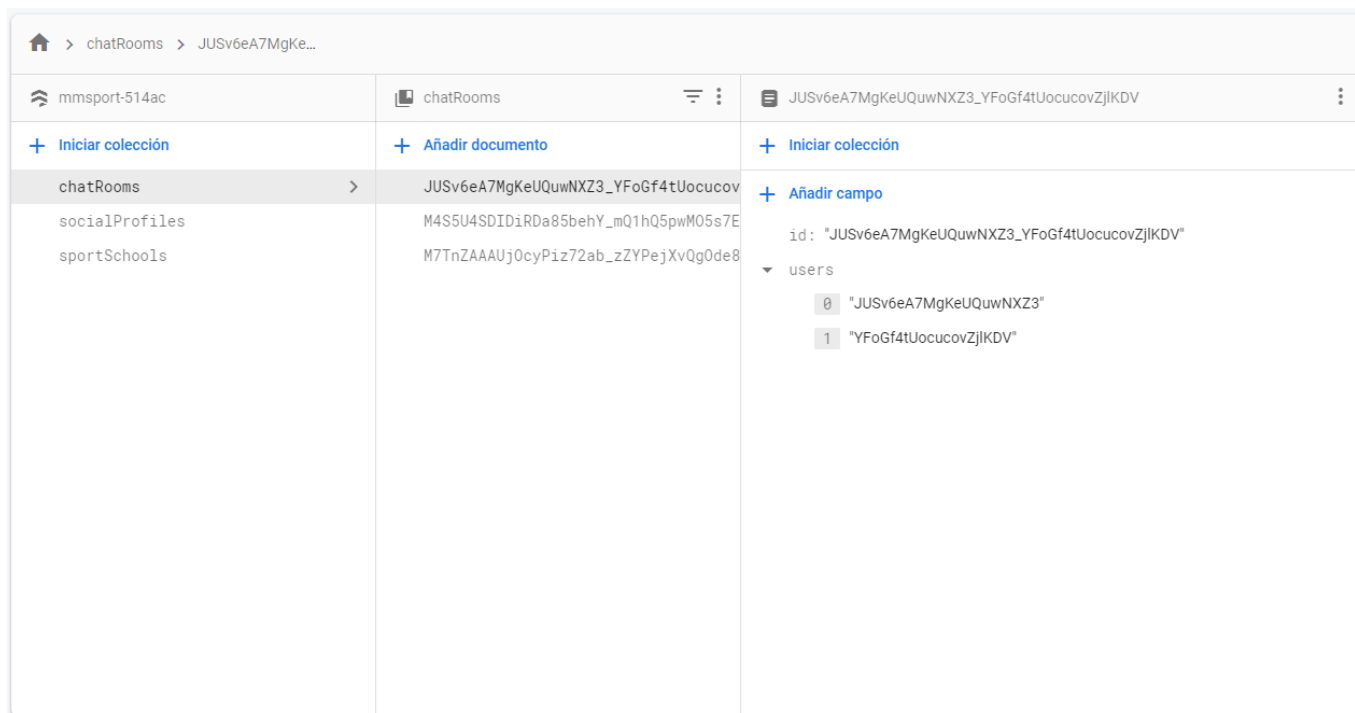
Podemos acceder a la página oficial de *Flutter* [aquí](#).



**Firestore**, según la información obtenida de [Wikipedia](#) es una plataforma digital para el desarrollo de aplicaciones web y móviles desarrollada por *Google* en 2014, que ofrece diversas funciones con el objetivo de mejorar el rendimiento de la aplicación ofreciendo medios para aumentar la usabilidad y la seguridad para los usuarios de la aplicación. Además, ofrece estadísticas de uso de la aplicación que pueden ser de gran utilidad tanto a desarrolladores como a clientes potenciales.

*Firestore* ofrece una gran cantidad de servicios y funcionalidades diferentes, de los cuáles utilizamos en el proyecto:

- **Cloud Firestore:** Ofrece una base de datos NoSQL alojada en la nube, muy optimizada para las aplicaciones móviles y webs. El hecho de ser NoSQL nos ha hecho ver otro tipo de base de datos, algo que prácticamente no habíamos hecho hasta ahora, y nos ha facilitado mucho la inserción y obtención de datos. A continuación, podemos ver un ejemplo de cómo se estructura la base de datos NoSQL:



- **Authentication:** Es un servicio que ofrece a los desarrolladores un sistema de identificación y registro de usuarios, facilitando enormemente la implementación de dicha funcionalidad en la aplicación, además de ofrecer la seguridad conveniente.
- **Storage:** Este servicio pone a disposición de los usuarios la posibilidad de almacenar contenidos que no se almacenarían de manera trivial en la base de

datos NoSQL, de forma que permite al usuario cargar y descargar archivos de vídeo, imágenes o cualquier otro tipo de contenido.

Como cabía de esperar, *Flutter* y *Firebase* están muy optimizados para funcionar entre ellos, ya que han sido desarrolladas por la misma compañía y son bastante recientes, algo que aumenta el rendimiento y la compatibilidad junto con la facilidad de desarrollo.

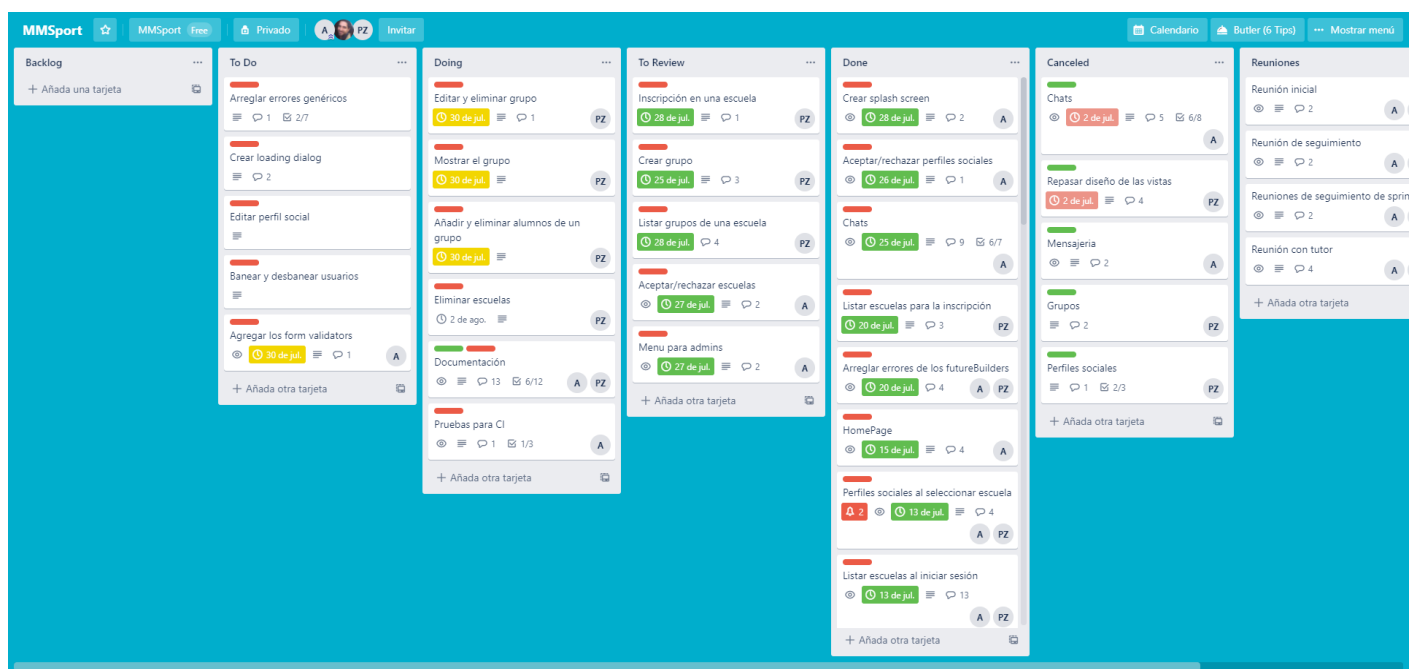
Podemos acceder a la página oficial de *Firebase* [aquí](#).

## 2.2. Herramientas

### 2.2.1. Herramientas de gestión

Para la gestión y organización de nuestro proyecto hemos usado dos herramientas que nos han ayudado muchísimo:

- **Trello** es una herramienta que permite una sencilla e intuitiva gestión y organización de proyectos. Como somos dos integrantes, nos ha servido para dividir las tareas pendientes y asignarlas, además de saber el progreso en todo momento. Esto es un ejemplo de nuestro tablero:



- **Plus for Trello** es una extensión de *Trello* que permite una fácil gestión del tiempo en el proyecto, asignando a cada tarea creada estimaciones, tiempo invertido y, algo importante, genera automáticamente gráficas y tablas con el tiempo invertido por cada usuario. Este es un ejemplo de nuestro tablero:

Report	Spent By User		Spent By Board		Chart				
under			over			Smooth format			
User	2020-W35	2020-W34	2020-W33	2020-W32	2020-W31	2020-W30	2020-W29	2020-W28	2020-W27
antmonagu	7	35	22	2.5	26	20.5	19	21	58.25
pabvazzam	4	34.5	21	9	25	18.5	11	18.5	64.25
	11	69.5	43	11.5	51	39	30	39.5	122.5

Podemos acceder a la página oficial de *Trello* [aquí](#) y a la de *Plus for Trello* [aquí](#).

### 2.2.2. Herramientas de documentación

La memoria del proyecto ha sido escrita en la **wiki de GitHub**, por el simple hecho de que aporta una gran facilidad para documentar al escribir en *Markdown*, facilitando mucho el formateo de la memoria. Una vez escrita, hemos usado [esta](#) herramienta de código abierto llamada **GitHub-wikito-Converter** que permite transformar la *wiki* de un repositorio en un archivo de tipo *PDF* y hemos retocado la memoria para que quede perfecta para su entrega.

### 2.2.3. Herramientas de control de versiones









Tal y como se puede intuir por el apartado anterior, la herramienta para el control de versiones ha sido **GitHub**, ya que hemos usado esta herramienta desde hace unos años y estamos bastante familiarizados con ella. *GitHub* es una plataforma de alojamiento de proyectos y de control de versiones usando la tecnología de *Git*, además de ofrecer sus propias funciones como *wikis* en cada proyecto, control de acceso a los repositorios de los proyectos, etc.

Podemos acceder a la página oficial de *GitHub* [aquí](#). En nuestro caso no vamos a poner el acceso a nuestro repositorio porque tenemos la intención de comercializarlo, y, por tanto, el repositorio está privado.

### 2.2.4. Herramientas de desarrollo

Para el desarrollo de la aplicación hemos usado dos herramientas:

- **Android Studio** es un *IDE* oficial para el desarrollo de aplicaciones *Android*. Es también el que usábamos en el desarrollo fallido de la versión nativa de *Android*. Para el desarrollo de la aplicación en *Flutter* instalamos el *plugin* de *Flutter* y *Dart* tal y como indica [esta guía](#).
- **Codemagic** es una herramienta *web* de gestión de integración continua oficial y desarrollada específicamente para proyectos desarrollados con *Flutter*. Todo lo necesario está explicado [aquí](#).

#57		develop 59edb64 develop	building a minute ago · 58s
#56		develop 36cdfa7 develop	finished 6 hours ago · 1m 50s
#55		develop c68572d develop	finished 20 hours ago · 1m 48s
#54		develop a4fbed9 develop	finished a day ago · 5m 13s
#17		release iOS 1.0.17 3bbaf01 master	 finished 3 days ago · 1h 18m 7s
#16		release iOS 1.0.0 3bbaf01 master	 failed 3 days ago · 1h 8m 36s

Este es un ejemplo de nuestro *Codemagic* donde se pueden ver distintas ramas de integración continua. En la de *develop*, comprobamos la calidad del código y ejecutamos las pruebas unitarias. La rama *release iOS* es una rama que se encarga de coger la rama *master* de *GitHub* y publicar directamente la compilación realizada en la *App Store*, algo muy interesante para nosotros ya que nos evita compilar en ordenadores *Mac*.

### 2.2.5. Herramientas de pruebas

*Flutter* ofrece una serie de paquetes que facilitan muchísimo el desarrollo de pruebas para la aplicación. Estos paquetes son los siguientes:

- **test** ofrece herramientas para realizar *tests* unitarios en métodos y funciones. [Aquí](#) podemos ver una introducción.
- Existen también dos paquetes de pruebas llamados [flutter test](#) y [flutter driver](#) que comprueban el correcto funcionamiento de los *widgets* y la integración de los mismos entre ellos. Sin embargo, consideramos que estas pruebas son más efectivas como pruebas de aceptación ya que así comprobamos tanto el funcionamiento de los *widgets* como su correcto diseño.

---

## 3. Aplicación desarrollada

---

### 3.1. Glosario de términos

- **Escuela deportiva:** Cualquier escuela que se registre en el sistema y exista en la realidad. También nos referiremos a este término con **escuelas**.
- **Director:** Máximo mandatario en una escuela deportiva.
- **Usuario:** Cualquier persona que haga uso de la aplicación.
- **Perfil social:** Cualquier persona registrada en el sistema como usuario puede tener perfiles sociales en una escuela. Esto hará referencia a una persona real que está inscrita en dicha escuela.
- **Grupo:** Conjunto de personas reales, en nuestro ámbito perfiles sociales, que realizan entrenamientos colectivos.
- **Chat:** Mensajería instantánea entre perfiles sociales de una misma escuela en la aplicación.

### 3.2. Visión general del sistema

#### 3.2.1. Participantes en el proyecto

El equipo de desarrollo está formado por dos integrantes:

	Datos
Nombre	Antonio Manuel Montaña Aguilera
DNI	49130324R
Email	antoniomma97@gmail.com

	Datos
Nombre	Pablo Vázquez Zambrano
DNI	49136060X
Email	pabvazzam@gmail.com

Además, contamos con la tutorización del profesor Carlos Guillermo Müller Cejás.

### 3.2.2. Organizaciones involucradas

No contamos con ninguna organización involucrada en el desarrollo de este proyecto.

### 3.2.3. Objetivos generales del sistema

Los objetivos principales del sistema son los siguientes:

- Permitir a cualquier usuario registrarse al sistema como director de una escuela, creándola, o como futuro alumno/entrenador.
- Permitir a los usuarios inscribirse en las escuelas como alumnos o entrenadores.
- Ofrecer un servicio de *chats* a los perfiles sociales dentro de una misma escuela.
- Ofrecer al director de una escuela control sobre los inscritos de su escuela, sus grupos, y sus eventos.
- Permitir a los administradores aceptar o rechazar escuelas inscritas.
- Ofrecer a los miembros de un grupo información sobre su calendario de eventos.

## 3.3. Requisitos del sistema

### 3.3.1. Actores

<b>ACT-01</b>	<b>Director</b>
<b>Descripción</b>	Representa a un perfil social de un usuario que se registra como dueño de una escuela

<b>ACT-02</b>	<b>Entrenador</b>
<b>Descripción</b>	Representa a un perfil social inscrito a una escuela como entrenador

<b>ACT-03</b>	<b>Alumno</b>
<b>Descripción</b>	Representa a un perfil social inscrito a una escuela como alumno



### 3.3.2. Requisitos de información

<b>RI-01</b>	<b>Información sobre los usuarios</b>
<b>Descripción</b>	El sistema debe guardar la siguiente información sobre las cuentas de usuario: el correo electrónico, la ID del usuario (generada automáticamente por Firebase) y la contraseña del usuario (asegurada y cifrada por Firebase automáticamente).

<b>RI-02</b>	<b>Información sobre los perfiles sociales</b>
<b>Descripción</b>	El sistema debe guardar la siguiente información sobre los perfiles sociales: la ID (generada por Firebase), nombre, primer apellido, segundo apellido si procede, el rol del perfil social (Director, Entrenador o Alumno), el estado del perfil (aceptado o pendiente), la escuela a la que pertenece, el grupo al que está asignado, la imagen de perfil y la cuenta de usuario a la que pertenece.

<b>RI-03</b>	<b>Información sobre las escuelas deportivas</b>
<b>Descripción</b>	El sistema debe guardar la siguiente información sobre las escuelas deportivas: la ID (generada por Firebase), nombre de la escuela, su dirección, su provincia, el pueblo/ciudad en el que se localiza, el estado en el que se encuentra (aceptada o pendiente), y el logo de la escuela.

<b>RI-04</b>	<b>Información sobre los <i>chats</i></b>
<b>Descripción</b>	El sistema debe guardar la siguiente información sobre los <i>chats</i> : los perfiles sociales que participan en el <i>chat</i> , la ID, que será la ID del primer perfil social y la ID del segundo, separadas por una " " (ejemplo, 1234_5678_) y los mensajes que se envían en ese <i>chat</i> .

<b>RI-05</b>	<b>Información sobre los mensajes</b>
<b>Descripción</b>	El sistema debe guardar la siguiente información sobre los mensajes: el contenido del mensaje, la fecha de envío y la ID del perfil social que ha enviado el mensaje.

<b>RI-06</b>	<b>Información sobre los grupos</b>
<b>Descripción</b>	El sistema debe guardar la siguiente información sobre los grupos: la ID del grupo, el nombre, el horario en el que se realizan los entrenamientos y la ID del entrenador responsable del grupo.

<b>RI-07</b>	<b>Información sobre los calendarios de eventos</b>
<b>Descripción</b>	El sistema debe guardar la siguiente información sobre los eventos: el nombre o descripción del evento, la fecha en la que ocurrirá el evento y los grupos que se verán involucrados en dicho evento.

### 3.3.3. Requisitos funcionales

<b>RF-01</b>	<b>Registro en el sistema</b>
<b>Descripción</b>	<p>El sistema debe permitir a los usuarios genéricos crear una cuenta en el sistema. Además, en ese registro se debe diferenciar si se registra un director de escuela o no. Para ello:</p> <ul style="list-style-type: none"> <li>• Deben poder acceder al formulario de registro mediante un botón en la pantalla principal de la aplicación, creando el usuario en la base de datos una vez el formulario esté relleno correctamente.</li> <li>• En caso de elegir que el usuario que se registra es director de una escuela, se le redirige al formulario de creación de escuela.</li> <li>• En caso contrario, se le redirige a la pantalla principal.</li> </ul>

<b>RF-02</b>	<b>Inicio de sesión en el sistema</b>
<b>Descripción</b>	El sistema debe permitir a los usuarios genéricos iniciar sesión con sus cuentas previamente registradas tal y como se especifica en el requisito <i>RF-01</i> . Para ello, se habilitará en la ventana de inicio de la aplicación un formulario de inicio de sesión.

<b>RF-03</b>	<b>Cierre de sesión en el sistema</b>
<b>Descripción</b>	El sistema debe permitir a los usuarios genéricos cerrar sesión con sus cuentas. Para ello, se habilitará un botón de cierre de sesión en la ventana de selección de escuelas, selección de perfiles y menú principal, redirigiendo a la ventana de inicio de la aplicación.

<b>RF-04</b>	<b>Creación de escuela en el sistema</b>
<b>Descripción</b>	El sistema debe permitir a los usuarios, en el formulario de registro, indicar si quieren crear una escuela en el sistema o no, tal y como se especifica en el requisito <i>RF-01</i> . En caso de que seleccionen la opción de crear una escuela, se les ofrecerá un formulario de creación de escuela y creación de perfil social con el rol de director, con toda la información necesaria. Al finalizar el formulario se creará tanto la escuela como el perfil social del director en estado <i>pendiente</i> .

<b>RF-05</b>	<b>Listado de escuelas de una cuenta de usuario</b>
<b>Descripción</b>	El sistema debe mostrar a los usuarios que inicien sesión en el sistema las escuelas que contienen perfiles sociales asociados a esa cuenta de usuario. Debe mostrar dicha escuela con el número de perfiles sociales que están inscritos en esa escuela. Se debe permitir seleccionar cada escuela y redirigir a la ventana de selección de perfiles sociales inscritos en dicha escuela.

<b>RF-06</b>	<b>Listado de escuelas disponibles en el sistema</b>
<b>Descripción</b>	El sistema debe ofrecer a los usuarios que inicien sesión la posibilidad de mostrar el listado completo de las escuelas disponibles en el sistema para inscribirse en dichas escuelas. Este listado incluirá el <i>logo</i> de la escuela y su nombre, permitiendo además filtrar las escuelas por su nombre. Pulsando sobre una de ellas se abrirá el formulario de inscripción en dicha escuela.

<b>RF-07</b>	<b>Inscripción en una escuela</b>
<b>Descripción</b>	El sistema debe ofrecer a los usuarios la posibilidad de inscribirse en una escuela. Para ello, se debe seleccionar una escuela tal y como indica el requisito <i>RF-06</i> , mostrando un formulario de inscripción en el que se debe elegir la imagen de perfil, el nombre y los apellidos, y si el perfil social tendrá el rol de alumno o de entrenador. Tras esto, se creará el perfil social como pendiente, a la espera de que el director de la escuela lo acepte.

<b>RF-08</b>	<b>Editar perfil social</b>
<b>Descripción</b>	El sistema debe ofrecer a los usuarios la posibilidad de editar la información de su perfil social, siendo ésta el nombre, sus apellidos y cambiar su imagen de perfil.

<b>RF-09</b>	<b>Listado de perfiles sociales inscritos a una escuela</b>
<b>Descripción</b>	El sistema debe mostrar los perfiles sociales inscritos a una escuela seleccionada previamente tal y como refleja el requisito <i>RF-05</i> . Se debe permitir seleccionar un perfil social y redirigir a la ventana donde se encuentra el menú principal. Se debe habilitar un botón para volver a la selección de escuelas si el usuario así lo desea.

<b>RF-10</b>	<b>Menú principal</b>
<b>Descripción</b>	El sistema debe mostrar un menú principal a los usuarios de la aplicación que hayan iniciado sesión y seleccionado tanto escuela como perfil social. Este menú principal contendrá distinta información útil dependiendo del rol del perfil social seleccionado que servirá como acceso directo a las distintas funcionalidades de la aplicación. Además, debe incluir un menú que permita cambiar de escuela, de perfil social o cerrar sesión si el usuario así lo desea.

<b>RF-11</b>	<b>Caché</b>
<b>Descripción</b>	<p>El sistema debe almacenar en memoria caché el usuario que ha iniciado sesión, la escuela y el perfil seleccionados para agilizar el rendimiento del sistema. Por lo tanto:</p> <ul style="list-style-type: none"> <li>• Si no hay nada guardado, la aplicación se iniciará en la ventana de inicio de sesión.</li> <li>• Si tiene el usuario que ha iniciado sesión, el sistema iniciará en la ventana de selección de escuela.</li> <li>• Si tiene el usuario y la escuela seleccionada, el sistema iniciará en la ventana de selección de perfiles sociales.</li> <li>• Si el sistema tiene toda la información anterior, iniciará en la ventana del menú principal.</li> </ul>

<b>RF-12</b>	<b>Ventana de <i>chats</i></b>
<b>Descripción</b>	<p>El sistema debe mostrar al usuario una ventana con los <i>chats</i> correspondientes divididos en dos pestañas:</p> <ul style="list-style-type: none"> <li>• En una pestaña se mostrarán los <i>chats</i> que el perfil social tiene abiertos con el último mensaje que se ha mandado en ese <i>chat</i>. Se entiende por abierto los <i>chats</i> con algún perfil social en el que ya se han mandado mensajes.</li> <li>• En otra pestaña se mostrarán los perfiles sociales con los que se puede abrir un <i>chat</i>, que serán los perfiles sociales inscritos en la misma escuela que el perfil social seleccionado.</li> </ul>

<b>RF-13</b>	<b><i>Chat room</i></b>
<b>Descripción</b>	Una <i>chat room</i> es, tal y como su traducción literalmente indica, una "habitación para <i>chatear</i> ". Por lo tanto, el sistema debe permitir a los perfiles sociales chatear entre sí. Para ello, existirá una ventana de <i>chat</i> en la que participarán los dos perfiles sociales implicados en dicho <i>chat</i> , seleccionado previamente tal y como indica el requisito <i>RF-08</i> , y en el que se mostrarán los mensajes que se han enviado entre sí dichos participantes, diferenciando claramente los

	enviados por un perfil y por otro, además de permitir escribir y enviar un mensaje nuevo al otro perfil social.
--	---

<b>RF-14</b>	<b>Aceptar y rechazar perfiles sociales</b>
<b>Descripción</b>	<p>El sistema debe ofrecer a los directores de las escuelas una funcionalidad que permita aceptar y rechazar perfiles sociales con estado <i>pendiente</i> en sus respectivas escuelas. Para ello, se le ofrecerá una ventana con un listado de perfiles sociales con ese estado, mostrando la imagen del perfil, nombre, apellidos y el rol que tendrá dentro de la escuela, además de, por supuesto, botones para aceptar y rechazar:</p> <ul style="list-style-type: none"> <li>• En caso de aceptar, el estado de dicho perfil social pasará a <i>aceptado</i> y podrá acceder a las funcionalidades de la aplicación en dicha escuela.</li> <li>• En caso de rechazar, el perfil social será eliminado del sistema.</li> </ul>

<b>RF-15</b>	<b>Aceptar y rechazar escuelas</b>
<b>Descripción</b>	<p>El sistema debe ofrecer a los administradores del sistema una funcionalidad que permita aceptar y rechazar escuelas con estado <i>pendiente</i>, además de a sus directores. Para ello, se le ofrecerá una ventana con un listado de escuelas con ese estado, mostrando el <i>logo</i> de la escuela, nombre, dirección, ciudad y provincia, además de la información del perfil social del director y, por supuesto, botones para aceptar y rechazar:</p> <ul style="list-style-type: none"> <li>• En caso de aceptar, el estado de la escuela pasará a <i>aceptado</i>, al igual que el perfil social del director, y podrá acceder a las funcionalidades de la aplicación en dicha escuela.</li> <li>• En caso de rechazar, el perfil social del director y la escuela serán eliminados del sistema.</li> </ul>

<b>RF-16</b>	<b>Información sobre el grupo</b>
<b>Descripción</b>	<p>El sistema debe ofrecer a los perfiles sociales información sobre el grupo al que están asignados, si es que tienen algún grupo asignado, en el caso de los alumnos. Para los entrenadores y directores, se les debe ofrecer un listado de los grupos a los que entrenan y los que existen en la escuela respectivamente, pudiendo seleccionar uno de ellos para acceder a su información, que consistirá en el nombre del grupo, el entrenador encargado de dicho grupo, el horario en el que entrena dicho grupo y un listado de los alumnos que integran ese grupo.</p>

<b>RF-17</b>	<b>Crear grupo</b>
<b>Descripción</b>	El sistema debe ofrecer a los directores la posibilidad de crear grupos en su escuela. Para ello, se ofrecerá un formulario en el que deben asignar un entrenador al grupo, darle un nombre, un horario y los alumnos que integran dicho grupo.

<b>RF-18</b>	<b>Editar grupo</b>
<b>Descripción</b>	El sistema debe ofrecer a los directores un formulario para poder editar la información de sus grupos, siendo ésta el nombre del grupo, el entrenador que está asignado a dicho grupo y el horario. Para ello, se habilitará en la ventana de información del grupo un botón que cambie el modo de "lectura" a "escritura", pudiendo editar toda la información mencionada anteriormente y guardarla al finalizar.

<b>RF-19</b>	<b>Asignar y eliminar alumnos a/de un grupo</b>
<b>Descripción</b>	El sistema debe ofrecer a los directores la posibilidad de asignar y eliminar alumnos a/de un grupo después de haberlo creado. Para ello, se habilitará en la ventana de información del grupo un botón que cambie el modo de "lectura" a "escritura", pudiendo agregar nuevos alumnos o eliminar los actuales de la lista, guardando al finalizar.

<b>RF-20</b>	<b>Eliminar grupo</b>
<b>Descripción</b>	El sistema debe ofrecer a los directores un botón en la ventana de información del grupo, mientras se edita el grupo, para eliminar dicho grupo.

<b>RF-21</b>	<b>Información del calendario de eventos</b>
<b>Descripción</b>	El sistema debe ofrecer a los perfiles sociales una vista del calendario de eventos de la escuela, en el que se ofrece información sobre los eventos que conciernen a la escuela en cuestión.

<b>RF-22</b>	<b>Crear eventos</b>
<b>Descripción</b>	El sistema debe ofrecer a los directores y entrenadores de la escuela la posibilidad de crear eventos en la escuela a la que pertenecen. Para ello, se les ofrece un formulario en el que deben añadir la información, siendo ésta el nombre del evento, el horario y el día del evento, y una descripción.

<b>RF-23</b>	<b>Editar eventos</b>
<b>Descripción</b>	El sistema debe ofrecer a los directores y entrenadores de la escuela la posibilidad de editar los eventos previamente creados por cualquier director o entrenador de la escuela. Se permitirá editar toda la información del evento.

<b>RF-24</b>	<b>Eliminar eventos</b>
<b>Descripción</b>	El sistema debe permitir a los directores y entrenadores de una escuela eliminar eventos previamente creados, sea cual sea el creador del evento.

<b>RF-25</b>	<b>Eliminar perfiles sociales de una escuela</b>
<b>Descripción</b>	El sistema debe permitir a los directores de una escuela eliminar perfiles sociales de su escuela.

<b>RF-26</b>	<b>Banear/desbanear usuarios del sistema</b>
<b>Descripción</b>	El sistema debe ofrecer a los administradores una funcionalidad que permita banear y desbanear usuarios del sistema.

<b>RF-27</b>	<b>Eliminar escuelas del sistema</b>
<b>Descripción</b>	El sistema debe ofrecer a los administradores la posibilidad de eliminar escuelas que sean previamente aceptadas del sistema.

<b>RF-28</b>	<b>Confirmar correo electrónico</b>
<b>Descripción</b>	El sistema debe verificar que el correo electrónico con el que se ha registrado el usuario es válido, por lo que debe enviar un correo de confirmación a la dirección proporcionada para verificarla. Mientras tanto, el usuario no podrá iniciar sesión y si lo intenta, se le avisará para que confirme el correo electrónico.

<b>RF-29</b>	<b>Restablecer contraseña</b>
<b>Descripción</b>	El sistema debe permitir a los usuarios restablecer la contraseña con la que se han registrado. Para ello, se ofrecerá un botón en la pantalla de inicio de sesión con el que, proporcionando el correo electrónico de la cuenta de usuario a restablecer, se enviará un correo electrónico con el que se podrá recuperar la contraseña.

### 3.3.4. Reglas de negocio

<b>RN-01</b>	<b>Usuario anónimo</b>
<b>Descripción</b>	Un usuario anónimo no podrá hacer uso de ninguna funcionalidad de la aplicación como usuario anónimo. Para ello, deberá iniciar sesión con una cuenta ya existente.

<b>RN-02</b>	<b>Registro de usuario</b>
<b>Descripción</b>	<p>El formulario de registro de un nuevo usuario en la aplicación debe cumplir las siguientes restricciones:</p> <ul style="list-style-type: none"><li>• Ningún campo debe estar vacío.</li><li>• Debe registrar un correo electrónico válido.</li><li>• La contraseña debe tener una longitud mínima de 6 caracteres.</li></ul>

<b>RN-03</b>	<b>Creación de perfil social</b>
<b>Descripción</b>	<p>El formulario de creación de un nuevo perfil social en la aplicación debe cumplir las siguientes restricciones:</p> <ul style="list-style-type: none"><li>• Ningún campo debe estar vacío.</li><li>• Si es un director o un entrenador, la imagen de perfil es obligatoria.</li></ul>

<b>RN-04</b>	<b>Directores en las escuelas</b>
<b>Descripción</b>	En una escuela, sólo puede existir un perfil social creado cuyo rol sea el de director, no se permite ningún otro.

<b>RN-05</b>	<b>Verificar creación de escuela</b>
<b>Descripción</b>	Al registrar el usuario, se le pregunta a dicho usuario si es director de una escuela. En caso afirmativo, no se permitirá a dicho usuario acceder a las funcionalidades de la aplicación hasta que no cree dicha escuela en el sistema.

## 3.4. Análisis de costes

### 3.4.1. Costes directos

El presupuesto del proyecto se basa en los costes estimados de él, tanto los costes directos como los costes indirectos.

En la siguiente tabla se encuentra un desglose de los costes del desarrollo del proyecto:



	Salario anual bruto	Cotización Seguridad Social	Coste para la empresa	Salario bruto a la hora	Salario total
<b>Desarrollador</b>	22.000€	6.578€	28.578€	15.87€	4.763€

El salario de un desarrollador se calcula partiendo del salario bruto anual que cobraría en una empresa con contrato indefinido con aproximadamente un año de experiencia en el sector. A ese salario bruto anual hay que sumarle los gastos de cotización en la seguridad social, suponiendo este total el coste para la empresa anualmente.

A partir de este coste anual, calculamos, con las 1800 horas anuales descritas en el Convenio Colectivo estatal de empresas de consultoría y estudios de mercado y opinión pública, el salario bruto de un desarrollador a la hora.

Dado que el proyecto consumirá un total de 300 horas por cada desarrollador, hay que multiplicar el salario bruto a la hora del desarrollador por las horas que serán invertidas en el desarrollo del proyecto. Por lo tanto, dado que el equipo está compuesto por dos desarrolladores, el coste directo total será de **9.526€**.

### 3.4.2. Costes indirectos

Los desarrolladores del proyecto dispondrán de los siguientes equipos *Acer Nitro 5 AN515-52* durante el desarrollo:

- *8GB RAM*
- *Intel Core i5-8750H @2.20GHz*
- *Nvidia GeForce GTX 1050 4GB*
- *Windows 10, 64 Bits*

Estos equipos tienen un coste de 900€. Asumiendo que cada año se trabajan 1800 horas, se amortiza un 25% del ordenador. El proyecto consumirá 300 horas de desarrollo, por lo que se amortizará a su vez el 16,67% de ese porcentaje de amortización anual, dando lugar a una amortización del 4,16%. Por lo tanto, el precio de amortización para cada PC es de 37,44€.

Por lo tanto, al disponer de dos equipos idénticos, el gasto de amortización total será de **74,88€**.

Además, a los costes indirectos hay que sumarle los costes de infraestructura en la realización del proyecto. Suponiendo que aproximadamente el gasto indirecto de un desarrollador ronda los 150€ en un mes, hay que sumar el total de **600€** a los costes indirectos.

### 3.4.3. Reservas

Es posible que en nuestro proyecto surjan fallos inesperados de software o cualquier tipo de evento inesperado. Por lo tanto, hay que añadir a nuestro presupuesto una cantidad que nos sirva para poder subsanar dichas circunstancias sin problemas mayores, añadiendo 20 horas de salario al presupuesto para cubrir estas necesidades.

	Coste unitario (20 horas)	Total
Costes directos	317,4€	634,48€
Costes indirectos		20€
TOTAL		674,48€

### 3.4.4. Presupuesto total

Una vez desglosado el presupuesto del proyecto, calculamos el total. Para ello, sumaremos los costes directos e indirectos y el presupuesto reservado en caso de que haya cualquier tipo de retraso en la entrega del proyecto. Finalmente, se le aplicará el IVA al total.

	Cantidad
Costes directos	9.526€
Costes indirectos	674,88€
Reservas	674,48€
Total SIN IVA	10.875,36€
IVA aplicado	2.283,82€
TOTAL	13.159,18€

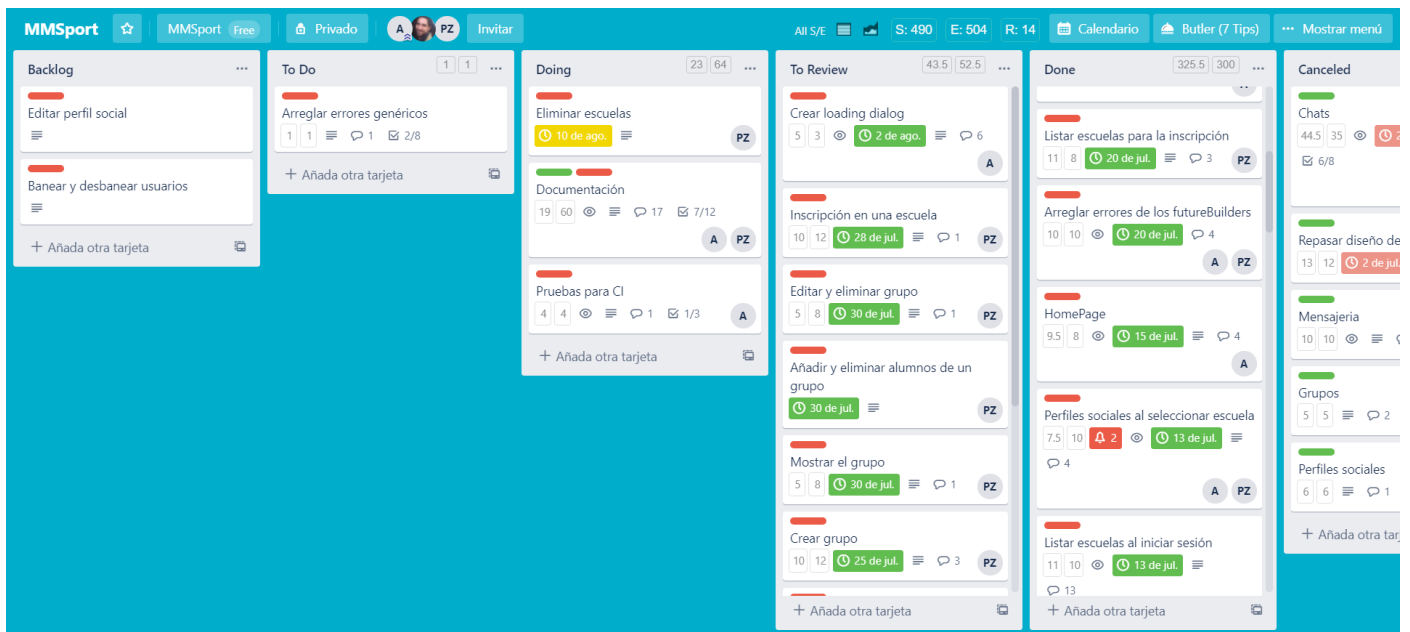
## 3.5. Planificación y metodología de desarrollo

La planificación del proyecto se ha tenido que hacer en 380 horas, debido al previo desarrollo de la versión nativa de *Android*. Esto no supone ningún cambio, ya que las 220 horas previas iban a ser igualmente consumidas en caso de que la versión nativa de *iOS* se hubiera llevado a cabo, por lo que la planificación sería la misma para las dos versiones y, tal y como se especificó antes, no supone ningún cambio en la planificación. Por lo tanto, la planificación aproximada será la siguiente (hay que añadir las 30 horas de estudio de los requisitos del proyecto realizada previamente en *Android*):

Título de la tarea	Estimación en horas individuales	Estimación en horas totales
<b>Inicio del proyecto</b>	5	10
- Estudio sobre el proyecto	3	6
- Estudio de las tecnologías	2	4
<b>Requisitos del sistema</b>	15	30
<b>Configuración del entorno de desarrollo</b>	3	6
<b>Implementación del sistema</b>	140	280
<b>Pruebas del sistema</b>	10	20
- Pruebas automáticas	5	10
- Pruebas de aceptación	5	10
<b>Generar documentación y memoria</b>	25	50
<b>Presentación</b>	7	14
- Realizar la presentación	4	8
- Práctica de la presentación	3	6
<b>TOTAL</b>		410

Tal y como se especificó antes, la planificación del desarrollo con el *framework Flutter* realmente es de 380 horas, pero se ha añadido a la planificación anterior las 30 horas de estudio de requisitos que se realizó en *Android*, de forma que esta planificación quede más completa.

El reparto de las tareas se ha realizado una vez iniciada la implementación del sistema, ya que todo el trabajo previo lo realizamos tanto Pablo como Antonio a la vez, ya que el estudio del proyecto y el estudio de las tecnologías nuevas es algo necesario para los dos, además del estudio y la elicitación de los requisitos del sistema, necesario para iniciar la planificación y el desarrollo del sistema. Al principio del desarrollo se decidió repartir las horas de desarrollo de forma equitativa entre los dos tal y como se especifica en la tabla de planificación del proyecto. Este reparto se especificará mejor cuando describamos los módulos desarrollados por cada uno. En la siguiente imagen se puede observar el tablero de *Trello* que hemos usado para el reparto de las tareas:



En el tablero se pueden apreciar distintas listas de tareas:

- **Backlog:** En el *Backlog* se incluyen las tareas que se deben hacer en un futuro, es decir, todas las tareas pendientes por hacer en el proyecto.
- **To Do:** Esta lista incluye las tareas que se han planificado para hacerlas en un futuro cercano, estimando su duración y asignadas a algún miembro. También pueden incluirse tareas genéricas como el arreglo de errores que se hayan encontrado.
- **Doing:** Se incluyen las tareas que se encuentran actualmente en desarrollo por alguno de los miembros del equipo.
- **To Review:** Son las tareas que han finalizado su desarrollo y falta revisar su correcto desarrollo para darla como finalizada.
- **Done:** Son tareas que se han revisado tras su finalización y se dan por completadas.
- **Cancelled:** Esta lista incluye tareas que han sido canceladas por algún motivo. También se incluyen tareas que estaban en progreso cuando aún realizábamos el desarrollo de la versión nativa de *Android*.
- **Reuniones:** Aquí se imputan las horas de reunión del equipo o con el tutor.

Adicionalmente, se pueden observar dos tipos de etiquetas en nuestro tablero. Una etiqueta de color rojo, que representa a las tareas que tienen que ver con el desarrollo de *Flutter*, y otra etiqueta de color verde, que representa a las tareas que se dieron en el desarrollo de la versión nativa de *Android*. Esto se ha hecho para no perder el seguimiento del tiempo invertido hasta el momento del cambio, de forma que, al final

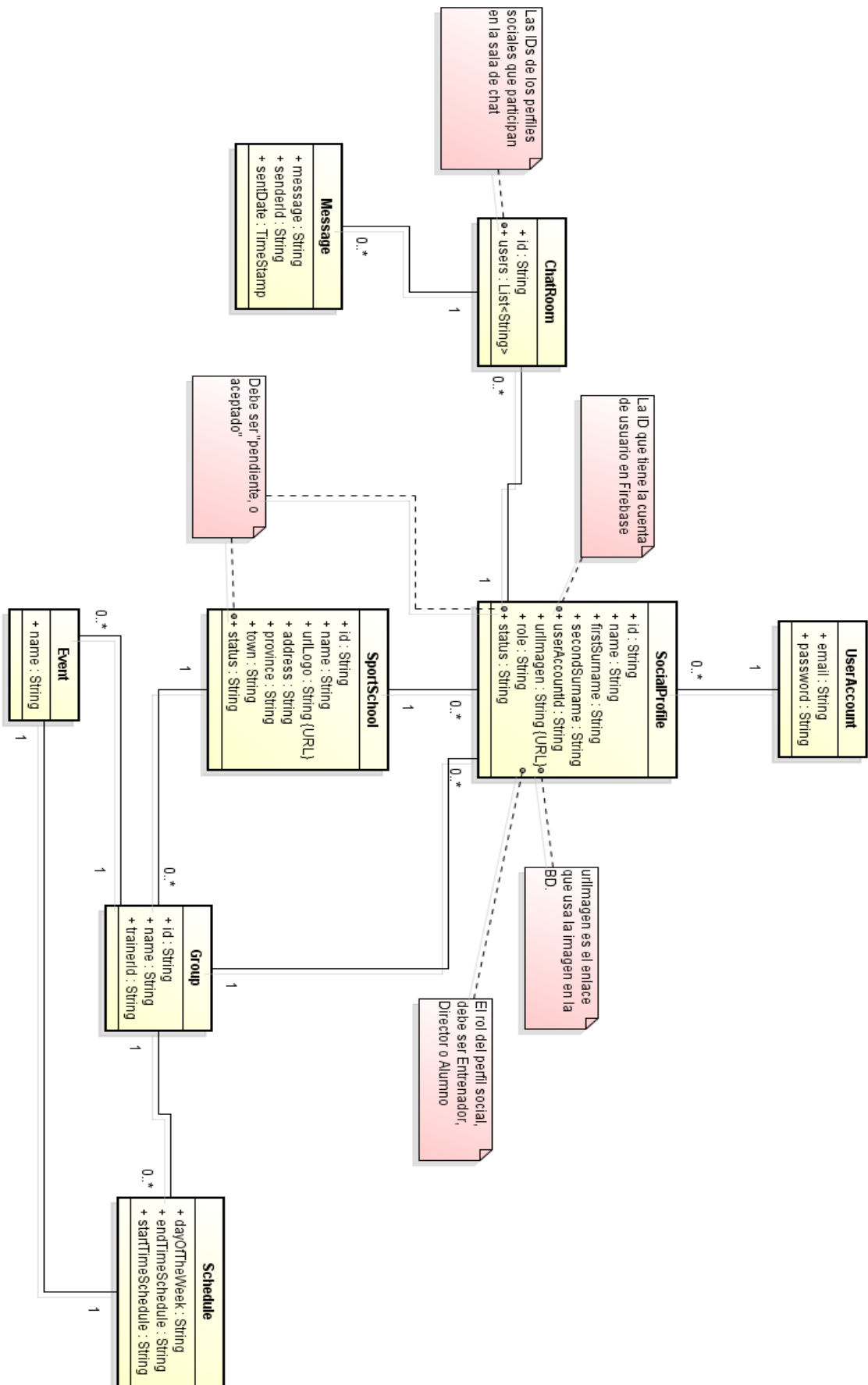
del proyecto, el número de horas fueran 600 horas y no solo las 410 mencionadas anteriormente.

Semanalmente, antes de comenzar a trabajar, tenemos una reunión muy breve para decidir qué tareas se van a realizar esa semana, es decir, qué tareas pasan de la lista *Backlog* a la lista *To Do* para su desarrollo en esa semana. El flujo de trabajo, una vez decidida la tarea que se va a realizar, pasaría dicha tarea a la lista *Doing* para comenzar su desarrollo. Una vez finalizado el desarrollo, pasará a la lista *To Review*, pendiente de revisar que el desarrollo se ha finalizado correctamente, pasando dicha tarea a *Done*.

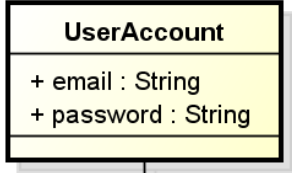
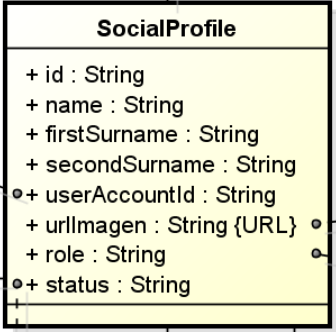
## **3.6. Análisis del sistema**

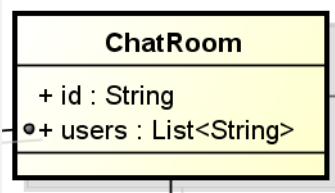
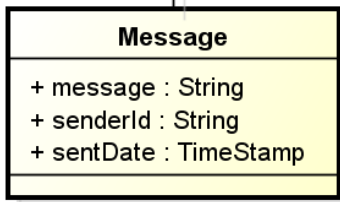
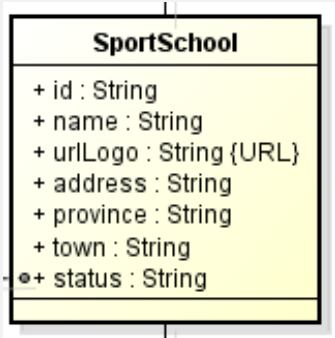
### **3.6.1. Modelo del sistema**

A continuación, mostramos el modelo de datos o de entidades del sistema. Un modelo de dominio no tiene sentido al no haber una base de datos relacional, por lo que esto es una idea del modelo:

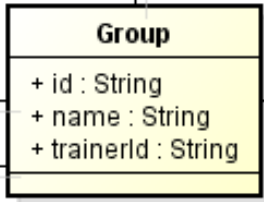
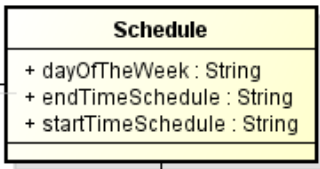
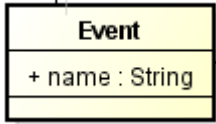


Y aquí detallamos las distintas clases del modelo conceptual del sistema:

Entidad	Descripción
 <pre> classDiagram     class UserAccount {         +email : String         +password : String     } </pre>	<p>Esta clase <i>UserAccount</i> representa la entidad de una cuenta de usuario. Esta entidad se refleja en el sistema mediante <i>Firebase Auth</i> en nuestra base de datos. Las propiedades son:</p> <ul style="list-style-type: none"> <li>• <b>email</b>: Cadena que representa el correo electrónico con el que se registra el usuario en el sistema.</li> <li>• <b>password</b>: Cadena que representa la contraseña del usuario en el sistema. Este campo se cifra automáticamente gracias a <i>Firebase Auth</i>, lo cual lo hace muy seguro ya que, además de su cifrado, <i>Firebase</i> no muestra el valor de la cadena en la base de datos.</li> </ul>
 <pre> classDiagram     class SocialProfile {         +id : String         +name : String         +firstSurname : String         +secondSurname : String         +userAccountId : String         +urlImagen : String {URL}         +role : String         +status : String     } </pre>	<p>Esta clase <i>SocialProfile</i> representa la entidad de un perfil social en el sistema. Es una entidad clave en el sistema, ya que será la representación física de una persona en el sistema. Sus propiedades son:</p> <ul style="list-style-type: none"> <li>• <b>id</b>: Identificador único en el sistema de la entidad, generado automáticamente por <i>Firebase</i>.</li> <li>• <b>name</b>: Cadena que representa el nombre del perfil social, es decir, nombre de la persona física.</li> <li>• <b>firstSurname</b>: Cadena que representa el primer apellido del perfil social, es decir, primer apellido de la persona física.</li> <li>• <b>secondSurname</b>: Cadena que representa el segundo apellido del perfil social, es decir, segundo apellido de la persona física. Este campo es opcional, ya que algunas personas no tienen segundo apellido.</li> <li>• <b>userAccountId</b>: Identificador único del usuario al que pertenece el perfil social, obtenido de <i>Firebase</i>.</li> <li>• <b>urlImagen</b>: URL que representa la imagen del perfil social, obtenida mediante <i>Firebase Storage</i> una vez que la imagen se ha subido al sistema.</li> <li>• <b>role</b>: Cadena que representa el rol del perfil social dentro de la escuela, que puede ser <i>Director</i>, <i>Trainer</i> (entrenador), o <i>Student</i> (alumno).</li> <li>• <b>status</b>: Cadena que representa el estado del perfil social, que puede ser <i>pending</i> (pendiente) o <i>accepted</i> (aceptado). Este estado cambiará</li> </ul>

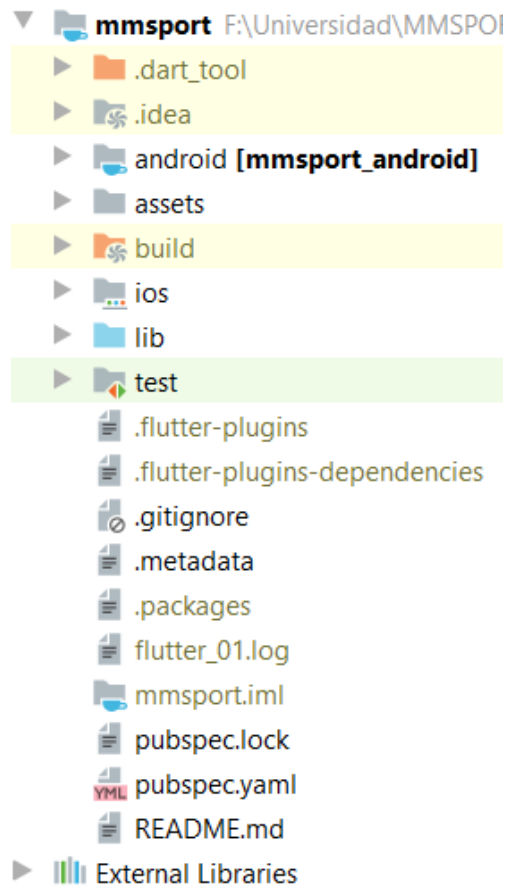
	<p>al ser aceptado por el director, iniciado por defecto en <i>pending</i>.</p>
 <pre> classDiagram     class ChatRoom {         +id : String         +users : List&lt;String&gt;     } </pre>	<p>Esta clase <i>ChatRoom</i> representa la entidad de una sala de <i>chat</i>. En esta clase encontramos las propiedades:</p> <ul style="list-style-type: none"> <li>• <b>id</b>: Un identificador único en el sistema de la entidad, generado automáticamente por <i>Firebase</i>.</li> <li>• <b>users</b>: Colección de cadenas que representas los identificadores únicos de los perfiles sociales que participan en la sala de <i>chats</i>.</li> </ul>
 <pre> classDiagram     class Message {         +message : String         +senderId : String         +sentDate : Timestamp     } </pre>	<p>Esta clase <i>Message</i> representa una entidad de mensaje en una sala de <i>chat</i>. Obviamente, un conjunto de estos mensajes representa una colección de mensajes dentro de la sala de <i>chat</i>. En esta clase encontramos:</p> <ul style="list-style-type: none"> <li>• <b>message</b>: Cadena que representa el mensaje enviado por alguno de los perfiles sociales.</li> <li>• <b>senderId</b>: Identificador único que representa la <i>id</i> del perfil social que ha enviado el mensaje. Esta <i>id</i> debe participar en la sala de <i>chat</i>.</li> <li>• <b>sentDate</b>: Fecha del envío del mensaje, que nos servirá para ordenar los mensajes enviados en la sala de <i>chat</i>.</li> </ul>
 <pre> classDiagram     class SportSchool {         +id : String         +name : String         +urlLogo : String {URL}         +address : String         +province : String         +town : String         +status : String     } </pre>	<p>Esta clase <i>SportSchool</i> representa la entidad de una escuela deportiva en el sistema. Es otra entidad clave en el sistema ya que sin esta entidad no existirían los perfiles sociales, por ejemplo. Sus propiedades son:</p> <ul style="list-style-type: none"> <li>• <b>id</b>: Un identificador único generado automáticamente por <i>Firebase</i>.</li> <li>• <b>name</b>: Cadena que representa el nombre de la escuela deportiva.</li> <li>• <b>urlLogo</b>: Cadena que representa la <i>URL</i> del logo de la escuela deportiva, proporcionada por <i>Firebase Storage</i> una vez que se ha subido a la base de datos.</li> <li>• <b>address</b>: Cadena que representa la dirección física de la escuela deportiva, es decir, la ubicación de la escuela.</li> <li>• <b>province</b>: Cadena que representa la provincia en la que se encuentra la escuela deportiva.</li> <li>• <b>town</b>: Cadena que representa la localidad/pueblo/ciudad en la que se encuentra la escuela deportiva.</li> <li>• <b>status</b>: Cadena que representa el estado de la escuela deportiva en el sistema, siendo éste <i>pending</i> (pendiente) o <i>accepted</i> (aceptado). Este estado cambia cuando el administrador</li> </ul>



	<p>acepte la escuela, comenzando con el estado <i>pendiente</i> por defecto.</p>
 <pre> classDiagram     class Group {         +id : String         +name : String         +trainerId : String     } </pre>	<p>Esta clase <i>Group</i> representa la entidad de un grupo en la escuela deportiva, en el cuál participan un grupo determinado de alumnos y un entrenador asignado que realizan entrenamientos juntos en la realidad. Sus propiedades son:</p> <ul style="list-style-type: none"> <li>• <b>id</b>: Identificador único generado automáticamente por <i>Firebase</i>.</li> <li>• <b>name</b>: Cadena que representa el nombre del grupo.</li> <li>• <b>trainerId</b>: Identificador único del perfil social del entrenador asignado a este grupo.</li> </ul>
 <pre> classDiagram     class Schedule {         +dayOfTheWeek : String         +endTimeSchedule : String         +startTimeSchedule : String     } </pre>	<p>Esta clase <i>Schedule</i> representa la entidad de un horario de un grupo. El grupo puede tener varios horarios. Las propiedades son:</p> <ul style="list-style-type: none"> <li>• <b>dayOfTheWeek</b>: Cadena que representa el día de la semana de la instancia de este horario.</li> <li>• <b>endTimeSchedule</b>: Cadena que representa la hora en la que finaliza el entrenamiento.</li> <li>• <b>startTimeSchedule</b>: Cadena que representa la hora a la que comienza el entrenamiento.</li> </ul>
 <pre> classDiagram     class Event {         +name : String     } </pre>	<p>Esta clase <i>Event</i> representa la entidad de un evento en la escuela deportiva. Estos eventos están asociados a un grupo, y tienen un horario asignado. Su única propiedad es <b>name</b>, cadena que representa el nombre o la descripción del evento.</p>

### 3.6.2. Arquitectura del sistema

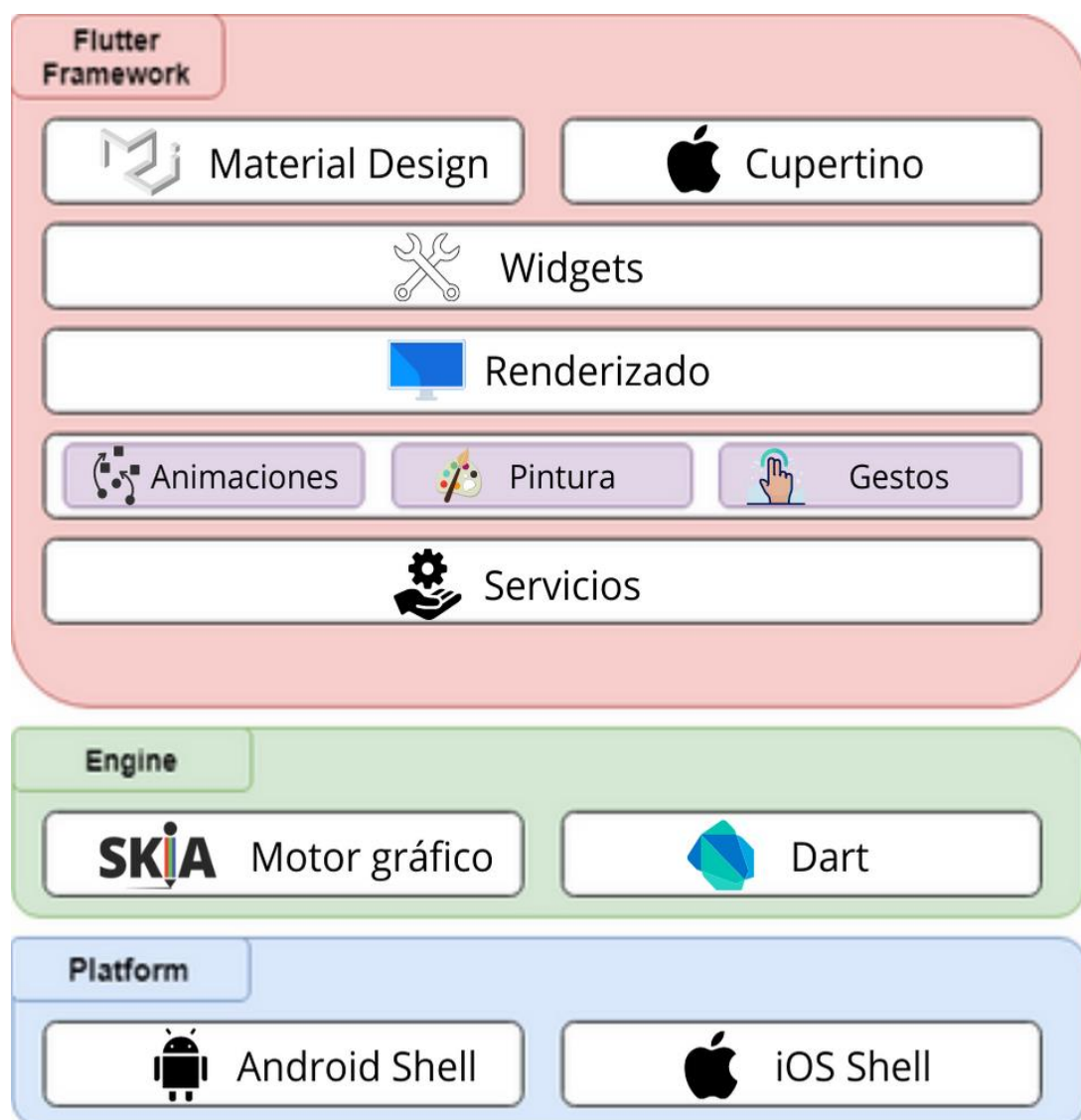
La estructura de un proyecto *Flutter* es algo básico de entender para poder desarrollar un proyecto en el *framework*. Para ello, presentamos una imagen y una explicación en la que se detalla la estructura de nuestro proyecto como ejemplo para facilitar su comprensión:



- Hay que destacar los archivos `.dart_tool`, `.idea`, `.flutter-plugins`, `.flutter-plugins-dependencies`, `.metadata`, `.packages`, los archivos `log` y el archivo `.iml`. Estos archivos son simplemente archivos de configuración y de metadatos del *framework*, *plugins*, etc. Estos archivos no nos interesan, ya que se modifican automáticamente con nuestro desarrollo.
- El paquete **android** contiene un proyecto específico para *Android* que incluye la configuración necesaria para ejecutar nuestro proyecto *Flutter*.
- El paquete **assets** contiene los archivos multimedia que nos pueden hacer falta en nuestro desarrollo, tales como imágenes de algún ámbito, el logotipo de nuestra aplicación, etc.
- El paquete **build** contiene compilaciones de las dependencias de nuestra aplicación. Es algo de lo que no nos tenemos que preocupar, ya que al añadir las dependencias al proyecto se compilan automáticamente.
- El paquete **ios** contiene un proyecto específico para *iOS* que incluye la configuración necesaria para ejecutar nuestro proyecto *Flutter*.
- El paquete **lib** contiene todos los archivos `.dart` de nuestro proyecto. Es el paquete en el que pasaremos más tiempo de desarrollo durante la realización de nuestro proyecto.

- El paquete **test** contiene los archivos con las pruebas que se realizan a los distintos módulos de nuestro proyecto.
- Los archivos **pubspec.yaml** y **pubspec.lock** son los archivos que registran las dependencias que añadimos a nuestro proyecto con sus respectivas versiones. Para ello, se añaden las dependencias con sus versiones al archivo **pubspec.yaml** y se ejecutan mediante el comando `flutter pub get` en la consola, de forma que se obtienen las dependencias y se compilan, añadiendo un registro en el archivo **pubspec.lock**. Además, en el archivo **pubspec.yaml** también se especifican las localizaciones de distintos archivos multimedia dentro del proyecto.

A continuación, mostramos un diagrama en el que se resume la arquitectura general del *framework Flutter*, con su posterior explicación:



Este es un diagrama de capas que resume la arquitectura general del funcionamiento de *Flutter*. Para este diagrama nos hemos basado en [este enlace](#), pero este diagrama ha sido realizado por nosotros, con los contenedores (los que tienen la

letra en inglés) hechos en [Draw.io](#) y el resto realizados con la ayuda de la licencia universitaria en [Canva](#).

En la capa más baja encontramos la capa de plataforma, la cual ofrece un *Shell* optimizado para la compilación del código *Dart* con una máquina virtual en cada una de las plataformas, tanto para *Android* como para *iOS*. Estos *Shell* dan acceso a las plataformas de manera nativa mediante sus respectivas *API*.

En la capa intermedia encontramos la capa de motor o *engine*. En dicha capa encontramos el motor de *Flutter*, que consiste en un *runtime* ofrecido por *Skia*. *Skia*, según la [Wikipedia](#), es una librería de gráficos de código abierto desarrollada en C++ originalmente por *Skia Inc.* y adquirida más tarde por *Google* que ofrece todo lo necesario para el funcionamiento de nuestra aplicación, tales como librerías de animaciones, gráficos, herramientas de compilación, etc. Además, encontramos el *runtime* de *Dart*.

Por último y no menos importante, encontramos el propio *framework* de *Flutter*. Es la parte más importante para los desarrolladores de *Flutter* como es lógico, ya que contiene todo lo necesario para el desarrollador y su interacción con el *framework*:

- **Animaciones, pinturas y gestos:** El *framework* de *Flutter* ofrece una serie de herramientas sobre las animaciones, la pintura de píxeles y los gestos para que el desarrollador tenga un control absoluto sobre el funcionamiento de la aplicación y lo que desea mostrar al usuario en cada *widget* construido.
- **Renderizado:** *Flutter* ofrece librerías de renderizado para que el desarrollador solo se preocupe por la construcción de los *widgets* y renderizar las vistas tal y como el programador desea.
- **Widgets:** Tal y como se explica en el apartado de *Tecnología*, los *widgets* son las herramientas que darán forma a la aplicación, predefinidos por el propio *Flutter* con la ayuda de la siguiente capa, en la que se encuentran las librerías *Material Design* y *Cupertino*.
- **Material Design y Cupertino:** Son las librerías que implementa *Flutter* para dar forma a la interfaz de usuario, siendo *Material Design* ofrecido por *Google* y *Cupertino* por *Apple*, siendo ésta una adaptación de *Material Design*. Estas librerías hacen que la aplicación se vea de la misma forma tanto en un dispositivo *Android* como en uno *iOS*.

## 3.7. Implementación del sistema

### 3.7.1. Entorno de desarrollo

#### Hardware

Para el desarrollo de la aplicación se han utilizado los siguientes dispositivos:

- 2 ordenadores portátiles Acer Nitro 5 AN515-52 con Intel Core i5-8750H @ 2.20GHz y 8GB de RAM.

- Smartphone Xiaomi Mi A1.
- Smartphone Redmi Note 9 Pro.

## Software

Se han utilizado las siguientes versiones de software para el desarrollo del proyecto:



- Flutter SDK v1.17.5.
- Intérprete Dart 2.8.4.
- Android Studio v4.0.

### 3.7.2. Implementación de los distintos módulos del sistema

En esta sección vamos a ofrecer una descripción detallada de los distintos módulos para facilitar la comprensión del código y del sistema. Para ello, vamos a analizar cómo se han cumplimentado los distintos requisitos funcionales del sistema y, además, vamos a ofrecer un ejemplo de distintas operaciones con la base de datos *Cloud Firestore* y la subida de archivos a *Firebase Storage*.

#### ¿Cómo se suben documentos a *Cloud Firestore*?

Un documento en nuestra base de datos *Cloud Firestore* (recordamos, *NoSQL*) no es más que un documento en formato *JSON* que representa un objeto de nuestro modelo en la base de datos. Por ejemplo, aquí podemos ver una representación de un objeto de tipo *SportSchool*:

 sportSchools	 j5z7ZWDb3C7PI5xG0clU
<a href="#">+ Agregar documento</a>	<a href="#">+ Iniciar colección</a>
j5z7ZWDb3C7PI5xG0clU >	<div data-bbox="639 1413 671 1447">+ Agregar campo</div> <pre data-bbox="683 1473 1538 1877">address: "Av. Portugal, s/n" id: "j5z7ZWDb3C7PI5xG0clU" name: "ITTSport" province: "Sevilla" status: "PENDING" town: "Alcalá" urlLogo: "https://firebasestorage.googleapis.com/v0/b/mmsport-514ac.appspot.com/o/Rub%C3%A9nPenaRold%C3%A1n1yObEB7AJhNalt=media&amp;token=2d4ceba2-0f35-4342-b4ed-7369576787e1"</pre>

Esto representa un *mapa clave/valor* que simboliza cada atributo de nuestro objeto con su valor correspondiente. Para insertar estos documentos a la base de datos, debemos escribir un objeto *JSON* por nosotros mismos para que *Cloud Firestore* lo entienda y lo inserte. Aquí podemos ver un ejemplo de la creación de escuelas deportivas en el sistema:

```
SportSchool sportSchool = new SportSchool(nameSportSchool, addressSportSchool, townSportSchool, provinceSportSchool, "PENDING", urlLogo);
final databaseReference = Firestore.instance;
DocumentReference ref = await databaseReference.collection("sportSchools").add({
    "name": sportSchool.name,
    "address": sportSchool.address,
    "town": sportSchool.town,
    "province": sportSchool.province,
    "status": sportSchool.status,
    "urlLogo": sportSchool.urlLogo
});
```

En este ejemplo, creamos un objeto de tipo *SportSchool* al que le pasamos en su constructor los valores obtenidos del formulario de creación de escuelas del que hablaremos más tarde. Lo importante es lo que viene después de la creación del objeto. Instanciamos la referencia a la base de datos de *Firebase* en primer lugar. Después, creamos un objeto de tipo *DocumentReference* para guardar la información del documento que vamos a crear. Por último, llamamos a la instancia de la base de datos, le especificamos que queremos que inserte el documento en la colección *sportSchools* y posteriormente, con el método *add*, creamos el objeto *JSON*. Tras esto, nuestro objeto estará creado en la base de datos tal y como hemos visto en el primer ejemplo.

Una vez agregado, para facilitarnos la tarea para tratar con este objeto, le agregamos la *ID* generada automáticamente por *Firebase* para el documento, tal y como se aclara aquí:

```
await databaseReference
    .collection("sportSchools")
    .document(ref.documentID)
    .setData({"id": ref.documentID}, merge: true);
```

En este fragmento, le pedimos al objeto de tipo *DocumentReference* creado anteriormente la *ID* generada automáticamente para, posteriormente, llamar de nuevo a la base de datos, concretamente a la colección *sportSchools* que acabamos de modificar, y le pedimos el documento que acabamos de crear. A este documento le agregamos otro objeto *JSON*, especificando que lo debe mezclar con lo que ya existe en el documento mediante la opción *merge: true*, con la *ID* del documento.

Con esto, ya tenemos agregado el documento a la base de datos.

## ¿Cómo se consulta a la base de datos para obtener un documento?

Las consultas a la base de datos *Cloud Firestore* son bastante sencillas y parecidas a lo ya visto en el ejemplo anterior.

Aquí mostramos un ejemplo de una consulta que explicaremos después:

```

await Firestore.instance
    .collection("socialProfiles")
    .where('role', isEqualTo: 'TRAINER')
    .where('role', isEqualTo: 'DIRECTOR')
    .where('sportSchoolId', isEqualTo: sportSchool.id)
    .getDocuments()
    .then((value) {
        value.documents.forEach((element) async {
            SocialProfile trainer = SocialProfile.socialProfileFromMap(element.data);
            trainer.id = element.documentID;
            trainers.add(trainer);
        });
    });

```

Esta consulta tiene como finalidad obtener una lista de perfiles sociales que sean candidatos para ser asignados como entrenadores a un grupo. Para ello, en primer lugar llama a la instancia de la base de datos para pedirle primero la colección de perfiles sociales *socialProfiles*. Después, le añade la condición de que el atributo *role* de los perfiles sociales que se quieren obtener debe ser o entrenador o director, además de pertenecer a la escuela sobre la que queremos crear un grupo. Finalmente, se obtienen los documentos y se opera con ellos en un bucle *forEach*, irrelevante para la explicación.

Como podemos ver, operar con la base de datos es muy sencillo e intuitivo, algo que hemos valorado mucho en la elección de la tecnología.

## ¿Cómo se suben fotografías a *Firebase Storage*?

*Firebase Storage* nos lo pone muy sencillo a la hora de operar con archivos multimedia en la aplicación. Para ello, mostramos un ejemplo:

```

String fileName = nameSportSchool;
StorageReference storageReference = FirebaseStorage.instance.ref().child(fileName + getRandomString(12));
StorageUploadTask uploadTask = storageReference.putFile(imageSchool);
StorageTaskSnapshot taskSnapshot = await uploadTask.onComplete;
var url = await taskSnapshot.ref.getDownloadURL();
urlLogo = url;

```


En el ejemplo se pretende subir la fotografía que el director de la escuela quiere subir como *logotipo* al crear dicha escuela en el sistema. Para ello, primero seleccionamos como nombre del archivo el nombre de la escuela. Luego, instanciamos *Firebase Storage* para operar con la base de datos, y le especificamos que vamos a subir el archivo con dicho nombre, además de agregarle una cadena aleatoria para evitar la repetición de nombres de archivo en caso de que se añadan dos escuelas con el mismo nombre a la aplicación. Tras esto, declaramos un objeto de tipo *StorageUploadTask*, que sirve para operar con la tarea de subida de archivos, y con la instancia creada anteriormente, guardamos el archivo (*imageSchool* contiene la URL local de la imagen seleccionada) en la base de datos. Con el objeto *StorageTaskSnapshot*, podemos obtener la información del objeto guardado una vez completada la subida del archivo. Con esta información obtendremos la URL del archivo desde la base de datos, lo que nos servirá para guardar esta información en el objeto *SportSchool* que queremos insertar en la base de datos con el ejemplo proporcionado, para consultarlo más tarde en distintas partes de la aplicación.

Es algo muy sencillo de hacer, y lo mejor es que nos generará una *URL* para después, en lugar de consultar la base de datos dos veces, una para el objeto y otra para la imagen, sólo la consultamos para el objeto, ya que la imagen está alojada en *internet* mediante dicha *URL*.

### **RF-01: Registro en el sistema**

Este requisito se ha desarrollado conjuntamente entre los dos integrantes del grupo, ya que fue de las primeras vistas que hicimos en el desarrollo, y, con el fin de aprender mejor y más rápido el funcionamiento de *Flutter*, lo decidimos así.

Aquí hemos implementado un sistema clásico de registro, pidiendo la información básica que se especifica para la cuenta de usuario en el requisito *RI-01*. La pantalla de registro ha quedado así:



**MMSPORT**  
Sport Clubs' Management

Correo electrónico

Contraseña

Confirmar contraseña

¿Tienes una escuela?

☐ Sí

☐ No

**REGISTRARSE**



Hemos añadido las comprobaciones del correo electrónico (que el campo no esté vacío y que la dirección insertada cumpla con una expresión regular de comprobación de correos electrónicos) y las contraseñas (que su longitud no sea inferior a 6 caracteres y que los dos campos de contraseñas coincidan). Por último, se añade una pregunta al usuario para saber si su deseo es crear una escuela o no, es decir, si es director de una escuela y por lo tanto su intención es crear una escuela en el sistema con un perfil de director.

Además, se han incluido comprobaciones que *Firebase* realiza a los datos introducidos tal y como se observa aquí:

```
catch (e) {
    String message;
    if (e.code == "ERROR_INVALID_EMAIL") {
        message = "El email no es válido";
    } else if (e.code == "ERROR_EMAIL_ALREADY_IN_USE") {
        message = "El email ya está en uso";
    } else if (e.code == "ERROR_WEAK_PASSWORD") {
        message = "La contraseña es demasiado débil";
    } else {
        message = "Ha ocurrido un error";
    }
}
```

En estas líneas se comprueba si la excepción se produce por una dirección de correo electrónico inválida (doble comprobación, ya que también se comprueba en el cliente), si el *email* está ya en uso, si la seguridad de la contraseña es demasiado baja o si ha ocurrido algún error inesperado.

Una vez que el formulario se ha cumplimentado y enviado, el usuario se creará usando los útiles proporcionados por *Firebase Auth*:

```
final FirebaseUser newUser = (await _auth.createUserWithEmailAndPassword(
    email: email,
    password: password,
)).user;
```

Donde *email* es la cadena que representa la dirección de correo electrónico introducida, *password* es la cadena que representa la contraseña que ha insertado el usuario, *\_auth* es la variable que instancia a *Firebase Auth* y *createUserWithEmailAndPassword* es el método proporcionado por la librería de *Firebase* para crear usuarios en la aplicación. Internamente, *Firebase Auth* se encargará de crear el usuario y cifrar y ocultar la contraseña en la base de datos.

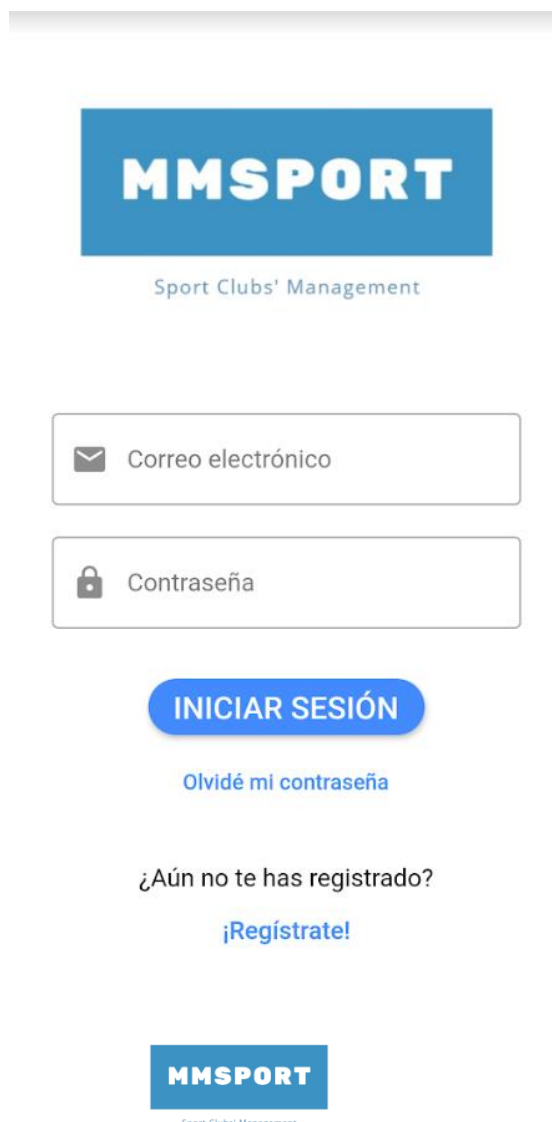
Al finalizar el proceso, el usuario queda registrado en la base de datos y se comprueba si ha decidido crear una escuela o no, redirigiéndole a la pantalla para crear una escuela en caso afirmativo o a la pantalla de inicio de sesión en caso contrario.

<div>          Buscar por dirección de correo electrónico, número de teléfono o UID de usuario       </div> <div> <a href="#">Agregar usuario</a>   </div>				
Identificador	Proveedores	Creado	Accediste a tu cuenta	UID de usuario ↑
pabvazzam@gmail.com		21 jul. 2020	13 ago. 2020	5SIQ56D8UIVQ3jteovFQnhDt7xm1
tonio@tonio.com		14 ago. 2020	14 ago. 2020	kNc8xD0BJvT7GxBN0lph2nTEiZU2
<div>Filas por página: 50 ▾</div> <div>1 a 2 de 2 &lt; &gt;</div>				

## RF-02: Inicio de sesión en el sistema

Este es otro requisito que se ha desarrollado de forma conjunta por los dos integrantes, ya que al igual que en el anterior, estábamos en proceso de aprendizaje y decidimos realizarlo entre los dos.

Para el inicio de sesión hemos desarrollado un formulario clásico en el que el usuario debe ingresar un correo electrónico y una contraseña que coincida con una cuenta de usuario registrada en el sistema. La pantalla ha quedado tal que así:



The login screen features the MMSPORT logo at the top, followed by the text 'Sport Clubs' Management'. Below this, there are two input fields: one for 'Correo electrónico' (Email) with an envelope icon, and another for 'Contraseña' (Password) with a lock icon. A blue 'INICIAR SESIÓN' button is positioned below the password field. A link 'Olvidé mi contraseña' is located below the login button. At the bottom, there is a link '¿Aún no te has registrado?' followed by a blue '¡Regístrate!' button. The MMSPORT logo and 'Sport Clubs' Management' text are repeated at the very bottom of the page.

Hemos añadido la comprobación de la dirección de correo electrónico (que el campo no esté vacío y que la cadena insertada cumpla con una expresión regular que comprueba la validez del correo electrónico insertado) y que el campo de la contraseña no esté vacío.

Además, se han incluido las comprobaciones de *Firebase* siguientes:

```
catch (e) {  
    String message;  
    if (e.code == "ERROR_USER_NOT_FOUND") {  
        message = "El usuario no existe";  
    } else if (e.code == "ERROR_WRONG_PASSWORD") {  
        message = "La contraseña no es correcta";  
    } else {  
        message = "Ha ocurrido un error";  
    }  
}
```

En estas líneas se comprueba si la excepción lanzada se ha producido porque el usuario no existe o si, por el contrario, el usuario existe pero la contraseña es errónea. También se comprueba si la excepción se produce por una causa desconocida.

Una vez que el formulario se ha rellenado y se han validado las anteriores comprobaciones, el usuario envía los datos a *Firebase* mediante *Firebase Auth*:

```
final FirebaseUser user = (await _auth.signInWithEmailAndPassword(  
    email: email, password: password)).user;
```

Donde *email* representa la dirección de correo electrónico introducida y *password* es la cadena que representa la contraseña introducida. Además, *\_auth* es la instancia de *Firebase Auth* y *signInWithEmailAndPassword* es el método proporcionado por la librería de *Firebase* para iniciar sesión mediante correo electrónico en la aplicación.

Al finalizar el proceso, el usuario inicia sesión en la aplicación y la *ID* de su usuario queda registrada en la caché de la aplicación para su uso más tarde.

### **RF-03: Cierre de sesión en el sistema**

Este requisito también se implementó al principio conjuntamente, ya que es algo que viene dado por *Firebase* y era muy sencillo de implementar.

Para este requisito hemos implementado un botón en las vistas que se indican en el mismo, es decir, en la ventana de selección de escuelas, en la de selección de perfiles y en el menú principal. El que se ve aquí en la esquina superior derecha es un ejemplo:



Al pulsar en dicho botón, se ejecuta la siguiente función:

```
void _logout() async {
  deleteLoggedInUserId();
  await FirebaseAuth.instance.signOut();
  logout(context);
}
```

En dicha función, primero se elimina la información del usuario que ha iniciado sesión de la caché de la aplicación. Posteriormente, se instancia a *Firebase Auth* para avisar a la base de datos de que el usuario ha salido de la aplicación para liberar memoria, y por último, el método *logout* es una llamada al navegador entre vistas de la aplicación. Este navegador viene dado por el propio *Flutter* y permite crear rutas entre las distintas pantallas de la aplicación. Por lo tanto, este método lo que hace es redirigir al usuario a la ventana de inicio de la aplicación, es decir, la de inicio de sesión.


#### **RF-04: Creación de escuela en el sistema**

Este es otro requisito que hicimos en conjunto al ser de los primeros requisitos que hicimos. Además, este era de los primeros en los que había que hacer inserciones

a la base de datos y queríamos saber cómo se hacían de forma correcta para poder implementar después bien el resto de las funcionalidades por separado.

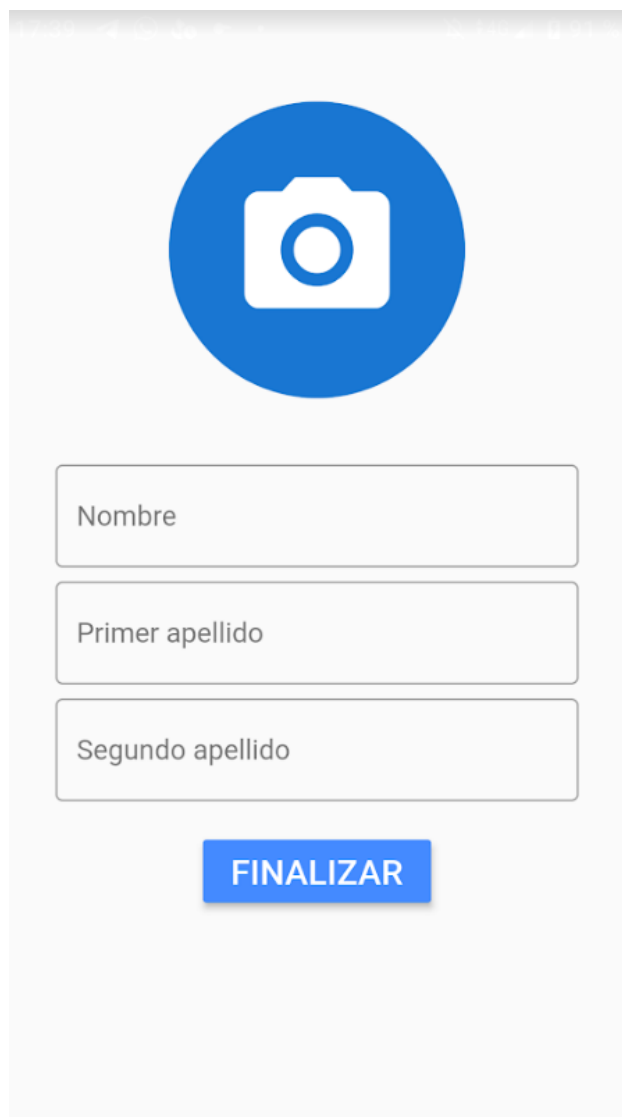
Cabe destacar que, para acceder a esta funcionalidad, tal y como se especifica en el requisito *RF-01*, el usuario debe haber marcado en el formulario de registro que es dueño de una escuela y por lo tanto desea agregarla al sistema. Entonces, se le redirige a este formulario.

Para la creación de escuelas hemos desarrollado un formulario muy interesante que está dividido en dos páginas distintas. El primero incluye la información básica de la escuela, que consiste en la inclusión de una fotografía como *logotipo* de la escuela, el nombre, la dirección física de la escuela, el municipio y la provincia en la que se encuentra. Se puede ver el formulario aquí:



En este formulario se comprueba que la fotografía haya sido añadida y que los campos de texto no estén vacíos. Una vez rellenados los campos y detectar que el usuario pulsa el botón "Siguiente", se comprobará que el formulario se ha rellenado

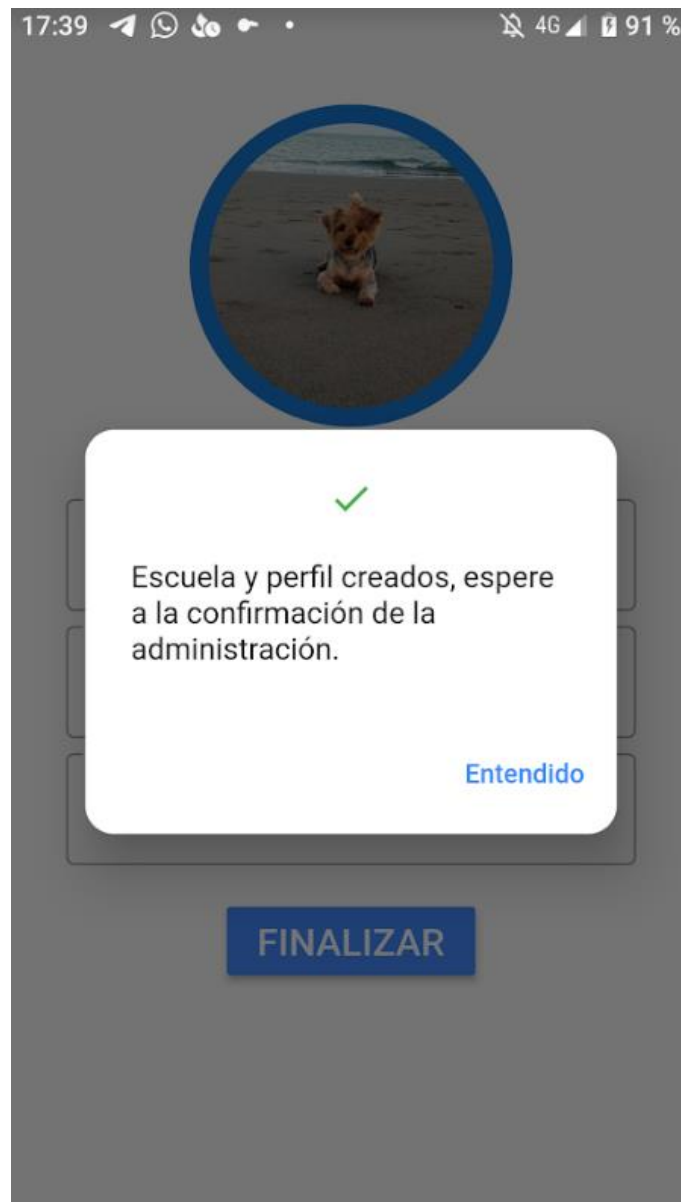
correctamente. En caso negativo, se le mostrarán al usuario los errores encontrados, y en caso contrario, se pasará a la siguiente página del formulario, que incluye la información del perfil social del director, tal y como se puede apreciar aquí:

The image shows a mobile application interface for adding a profile picture. At the top, there is a blue circular button with a white camera icon. Below this, there are three text input fields: 'Nombre', 'Primer apellido', and 'Segundo apellido'. At the bottom of the form is a blue button with the text 'FINALIZAR' in white capital letters. The background of the form is light gray.

En este segundo formulario se comprueba que la fotografía de imagen de perfil haya sido añadida (en este caso y en el de los entrenadores es obligatoria la imagen de perfil), y que el nombre y el primer apellido han sido agregados. El segundo apellido es opcional al existir personas que no disponen de un segundo apellido.

Una vez que el formulario es enviado por el usuario y validado por el sistema, se procede a la inserción de los datos en la base de datos *Firebase* mediante *Cloud Firestore* y las imágenes mediante *Firebase Storage* tal y como se explica en los ejemplos al principio de la sección.

Cuando se ha insertado en la base de datos toda la información, la escuela y el perfil social estarán en estado *pendiente*, a la espera de que un administrador acepte o rechace la petición, redirigiendo al usuario a la ventana principal de la aplicación:



### **RF-05: Listado de escuelas de una cuenta de usuario**

Esta fue la última funcionalidad que implementamos los dos juntos durante nuestro desarrollo, ya que es la primera vez que implementábamos consultas a la base de datos *Cloud Firestore* y queríamos asegurarnos de hacerlo bien antes de separar las tareas.

En este requisito hemos implementado una ventana que ya se ha mostrado en la implementación del requisito *RF-03*, pero vamos a volver a mostrarla para que quede aún más claro, en la que mostramos una lista de escuelas que tienen un perfil social asociado a la cuenta de usuario que acaba de iniciar sesión, ya que esta vista se muestra justo después del inicio de sesión del usuario. Para ello, consultamos a la base de datos los perfiles sociales que tiene dicho usuario, consultamos las escuelas en las que están inscritos dichos perfiles sociales y lo metemos en un mapa clave/valor, cuya clave es la escuela y el valor es el número de perfiles que tiene el usuario en dicha escuela:



Tal y como se puede ver, hemos implementado una lista de escuelas que se puede recorrer desplazando de izquierda a derecha. Para cada escuela mostramos el *logotipo* de dicha escuela, el nombre de la escuela y el número de perfiles sociales verificados (es decir, aceptados por el director de la escuela) que existen en la escuela para este usuario. El resto de los botones serán explicados en los próximos requisitos.

Como añadido, hay que destacar que el "punto" que se ve abajo de la escuela mostrada en la imagen es una especie de índice en el que se puede saber en qué posición de la lista se encuentra el usuario mientras se desplaza por ella. Esto ha sido implementado mediante una librería externa llamada *smooth\_page\_indicator*, que se puede encontrar [aquí](#). Esta librería ofrece varias posibilidades para indicar el elemento de la lista en el que se encuentra el usuario, con distintas animaciones, indicadores, etc.

#### **RF-06: Listado de escuelas disponibles en el sistema**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

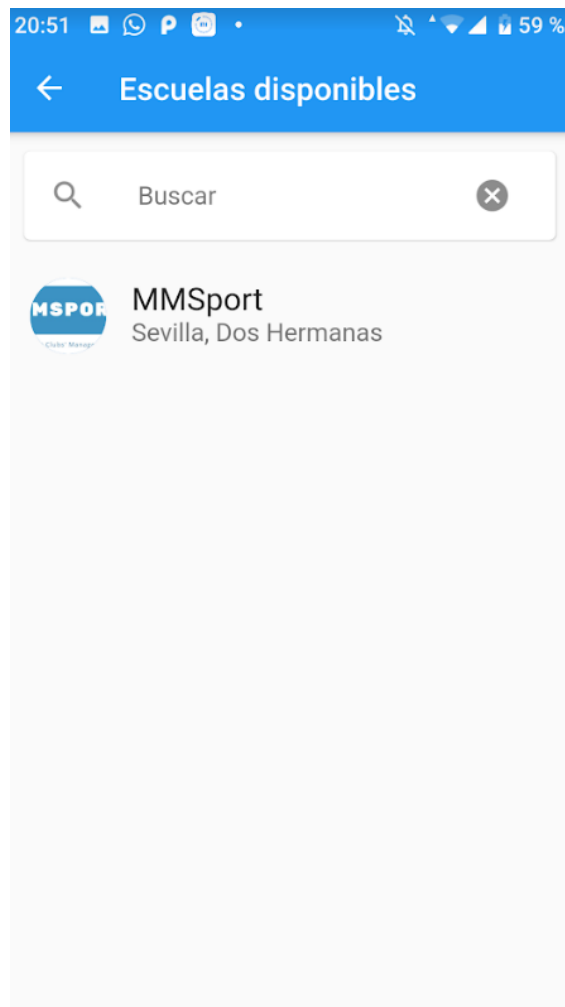
Para cumplimentar este requisito hemos incluido un listado de todas las escuelas que existen en el sistema, con un filtro para poder buscar la deseada por el usuario, con la finalidad de poder inscribirse en una escuela como alumno o como entrenador.



Para acceder a esta vista, el usuario debe pulsar el botón con el símbolo "+" que se observa en la imagen del listado de escuelas del usuario:



En el listado, se muestra, para cada escuela, el nombre de la escuela, el municipio y la provincia de la escuela, para poder diferenciarlas en caso de que haya varias cuyos nombres sean idénticos en varios puntos de España. En caso de pulsar en una de las escuelas, el usuario será redirigido a la ventana de inscripción a la escuela. (En la imagen se observa una, de momento, ya que es la única escuela en la aplicación)

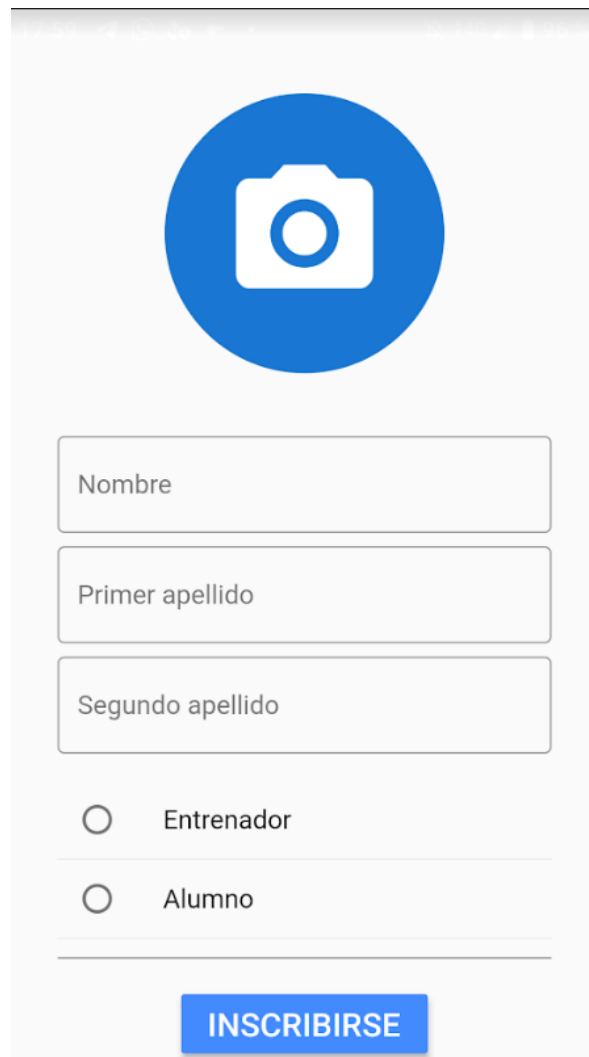


Hay que destacar que el filtrado de esta lista se realiza con la ayuda de un paquete llamado *search\_app\_bar* que se puede encontrar [aquí](#). Este paquete ofrece una barra de búsqueda animada para implementarla en la *AppBar* de nuestra aplicación de gran utilidad, ofreciendo una gran experiencia de usuario además de rapidez a la hora de filtrar las listas.

### **RF-07: Inscripción en una escuela**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito se ha desarrollado un formulario muy parecido al de creación de director explicado anteriormente, con la única diferencia de que se ha añadido una pregunta al usuario para que diga si quiere inscribir un perfil social que tenga el rol de alumno o de entrenador. La pantalla ha quedado tal que así:

El formulario de inscripción social está diseñado para ser intuitivo y fácil de usar. Comienza con un icono de cámara dentro de un círculo azul, indicando que se debe subir una foto de perfil. A continuación, hay tres campos de texto para el nombre, el primer apellido y el segundo apellido. Debajo de estos campos, hay dos opciones de selección con botones de radio: 'Entrenador' y 'Alumno'. Al final del formulario, hay un botón azul con el texto 'INSCRIBIRSE' en blanco.

Para este formulario se ha añadido la comprobación de que los campos no estén vacíos, a excepción del segundo apellido por la razón de que no todas las personas tienen segundo apellido. En este caso, la imagen de perfil no será obligatoria para los alumnos, es decir, si el perfil social a inscribir es un entrenador, la imagen de perfil será obligatoria.

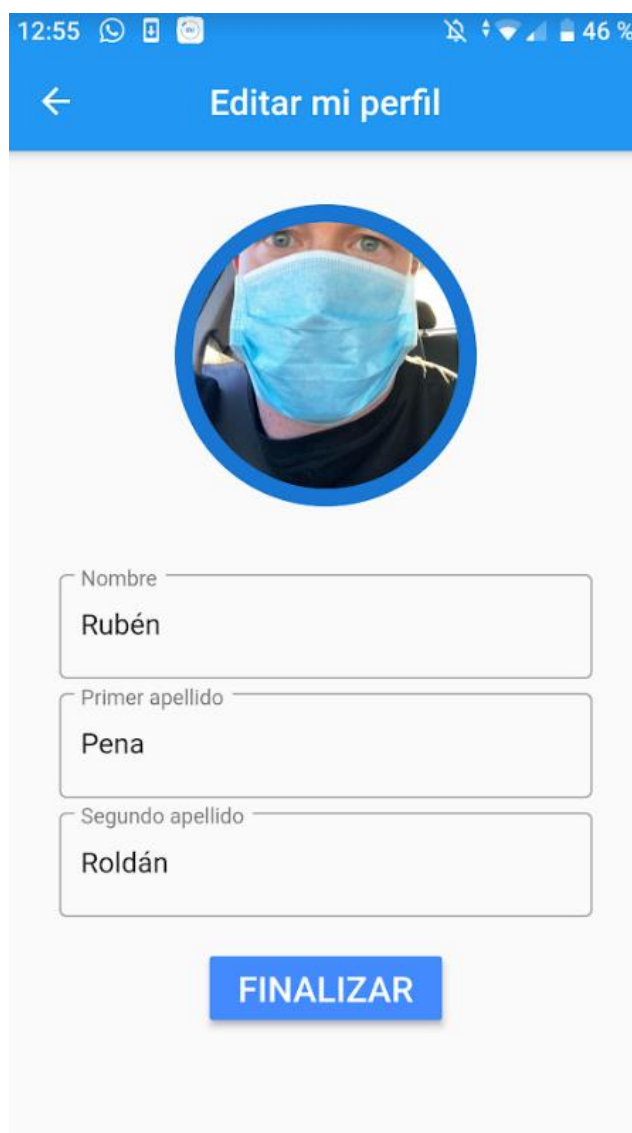
Una vez que el formulario se ha enviado y validado, el perfil social se guardará en la base de datos con estado *pendiente*, esperando a que el director de la escuela lo acepte como alumno o como entrenador en la escuela. El guardado en la base de datos se produce al igual que se explicó al principio de la sección. Tras esto, se redirige al usuario a la vista de la lista de las escuelas del usuario.

Como añadido especial, en la lista de *Mis escuelas* del requisito *RF-05*, se ha incluido un botón que dice "+ Añadir perfil social". Si pulsamos en el, nos llevará directamente al formulario de creación de perfil social para dicha escuela, sin necesidad de buscarla en la lista.

### **RF-08: Editar perfil social**

Este requisito ha sido desarrollado en su totalidad por Antonio Montaña.

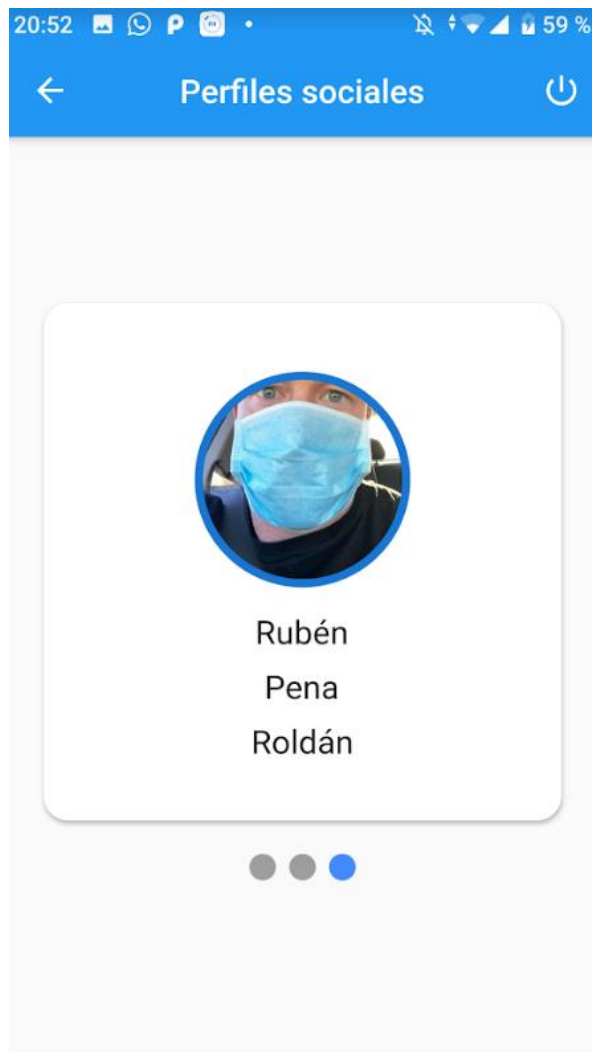
Para cumplimentar este requisito hemos implementado una ventana idéntica a la vista en la creación de director del requisito *RF-04*. Para ello, cargamos los datos del perfil social dado en cada campo, es decir, la imagen de perfil actual, el nombre y los apellidos. La ventana ha quedado tal que así:



#### ***RF-09: Listado de perfiles sociales inscritos en una escuela***

Este requisito fue desarrollado principalmente por Antonio Montaña, aunque luego fue retocado respecto a la parte del diseño por Pablo Vázquez.

Para cumplimentar este requisito hemos implementado una ventana en la que mostramos una lista de los perfiles sociales relacionados con la escuela que ha seleccionado previamente el usuario, es decir, los perfiles sociales cuyo atributo *sportSchoolId* es el mismo que la *ID* de la escuela que ha seleccionado anteriormente el usuario tal y como se refleja en el requisito *RF-05*. Para ello, consultamos a la base de datos los perfiles sociales tal y como se acaba de especificar y se meten en una lista que servirá para rellenar la lista de *cards* que se muestran en la pantalla. La pantalla ha quedado tal que así:



Tal y como se puede ver, hemos implementado la lista desplazable de izquierda a derecha, con los perfiles sociales insertados en una *card*. En dicha *card* se muestra la imagen de perfil del perfil social, y el nombre y los apellidos del perfil. Pulsando en uno de los perfiles sociales, dicho perfil quedará como seleccionado, representando al usuario en todas las funcionalidades de la aplicación y redirigiendo al usuario a la pantalla del menú principal.

(Esto aún no se ha implementado, pero se va a hacer y se va a dejar aquí ya puesto) Como añadido, hay que destacar que el "punto" que se ve debajo de cada *card* indica la posición en la que se encuentra el usuario respecto a la lista. Esto ha sido implementado gracias al paquete *smooth\_page\_indicator*, que se puede encontrar [aquí](#). Esta librería ofrece varias posibilidades para esta paginación, las cuáles se pueden encontrar en la página anterior mencionada.

### **RF-10: Menú principal**

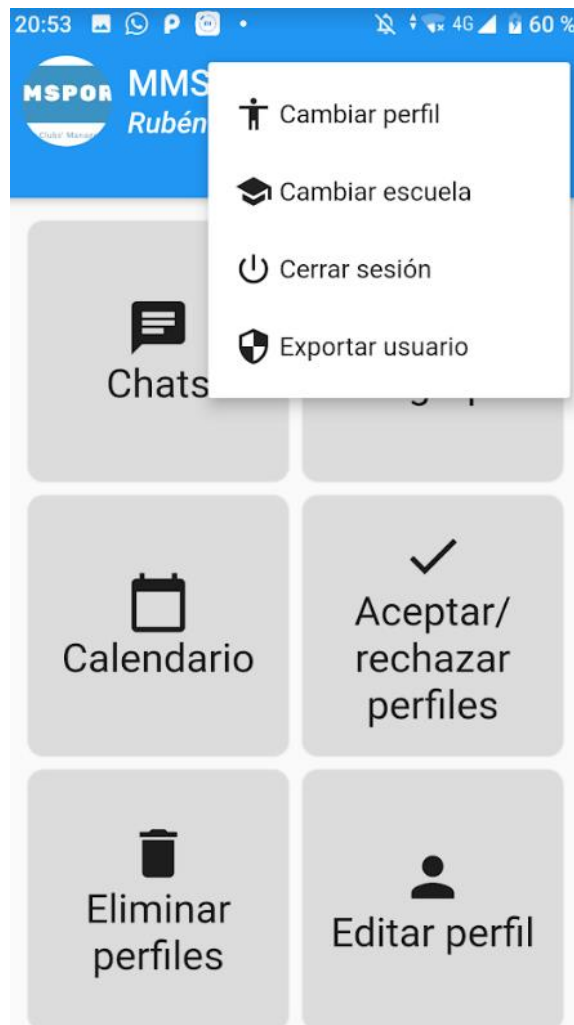
Este requisito ha sido desarrollado en su totalidad por Antonio Montaña.

El menú principal ha sido implementado de una forma muy generalizada, de forma que, cuando se añada en el futuro más funcionalidad, sea muy sencillo implementar nuevos botones. Para esta vista se ha desarrollado una *AppBar* personalizada que incluye el *logotipo* de la escuela seleccionada, el nombre de la misma

y el nombre y apellidos del perfil social seleccionado. La pantalla ha quedado tal que así:



En este caso se muestra el menú principal que se encontrará un director de escuela con sus respectivas opciones. También se ha incluido un menú de estilo *PopUp* en la esquina superior derecha que incluye las opciones de cambiar de escuela, cambiar de perfil social o cerrar sesión, además de la posibilidad de exportar la información del usuario como cumplimentación de la *GDPR*:



Para añadir los botones del menú principal se ha añadido en el cuerpo del *Scaffold* la llamada a un método general que comprueba el rol del perfil social seleccionado y muestra las opciones para dicho rol:

body: `menuGrid(context, snapshots.data[1].role)`

Siendo `snapshots.data[1]` el perfil social seleccionado después de haberlo obtenido de la caché de la aplicación. Este método ha sido implementado en un archivo aparte llamado *menu\_helper* que hará lo siguiente:

```
if (role == "STUDENT") {
  return [
    _menuItem("Chats", Icons.chat, "STUDENT", 1, context),
    _menuItem("Mi grupo", Icons.group, "STUDENT", 2, context),
    _menuItem("Calendario", Icons.calendar_today, "STUDENT", 3, context),
    _menuItem("Editar perfil", Icons.person, "STUDENT", 4, context)
  ];
} else if (role == "TRAINER") {
  return [
    _menuItem("Chats", Icons.chat, "TRAINER", 1, context),
    _menuItem("Mis grupos", Icons.group, "TRAINER", 2, context),
  ];
}
```

```

        _menuItem("Calendario", Icons.calendar_today, "TRAINER", 3, context),
        _menuItem("Editar perfil", Icons.person, "TRAINER", 4, context),
    ],
    } else if(role == "DIRECTOR"){
    return [
        _menuItem("Chats", Icons.chat, "DIRECTOR", 1, context),
        _menuItem("Mis grupos", Icons.group, "DIRECTOR", 2, context),
        _menuItem("Aceptar/rechazar perfiles", Icons.check, "DIRECTOR", 3, context),
        _menuItem("Crear grupo", Icons.group_add, "DIRECTOR", 4, context),
        _menuItem("Editar perfil", Icons.person, "DIRECTOR", 5, context),
    ];
    }
}

```

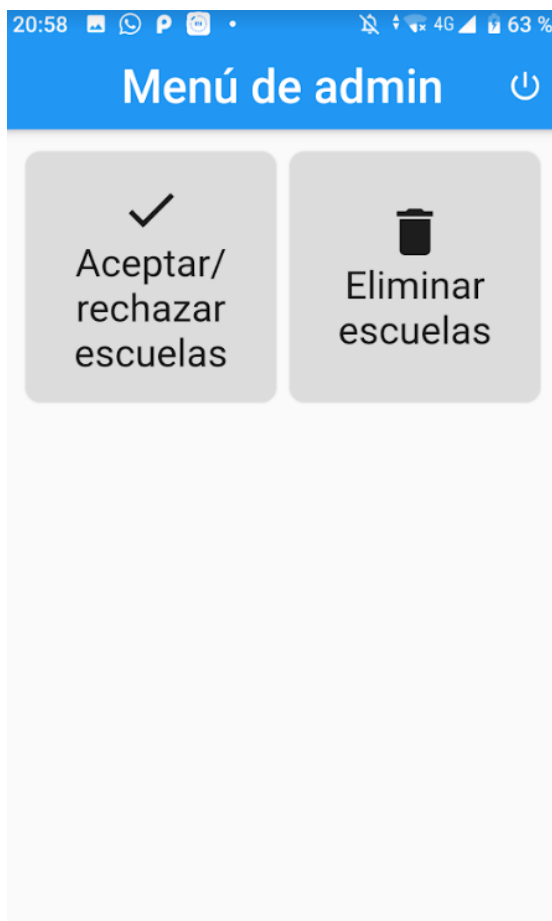
Como se puede comprobar, dependiendo del rol del perfil social, se creará un *array* de *widgets*, con cada llamada al *widget* indicando el título del botón, el icono que se mostrará, el rol del perfil que nos servirá para dirigir luego según el rol a una vista u otra y el número indicado es el lugar que ocupará en el menú principal.

Es un desarrollo que se ha hecho de forma muy general como se puede observar, de forma que si necesitamos añadir otra funcionalidad, con añadir una línea a uno de los *arrays* y la navegación en otro método aparte (otra línea) se crea el botón en el menú de forma muy rápida. Aquí mostramos el menú de los alumnos como prueba:





Hay que destacar que el menú de un administrador funciona de distinta forma al de los demás, ya que, para este caso, no hay que elegir escuela ni perfil social, simplemente al iniciar sesión con una cuenta de administrador se le redirigirá a la vista del menú de administradores, el cual ha quedado así:



### **RF-11: Caché**

Este requisito ha sido implementado por Antonio Montaña en su totalidad.

La memoria caché es algo que agiliza muchísimo el rendimiento de todas las aplicaciones, por lo que esta aplicación no podía ser menos. Se ha implementado una caché para guardar útiles usados en muchos puntos de la aplicación. Esto se ha implementado con la ayuda de un paquete llamado *shared\_preferences*, que se puede encontrar [aquí](#). Este paquete implementa una especie de mapa clave/valor en el que la clave será una cadena con la que se identifica el objeto guardado, y el valor es el propio objeto, del tipo que sea, que se recuperará al llamar a la clave deseada. Además, es muy útil ya que esta caché se mantiene incluso al cerrar la aplicación, de forma que al abrir la aplicación se puede iniciar desde un punto distinto a la ventana de inicio de sesión en caso de que se haya cerrado la aplicación sin cerrar dicha sesión, por ejemplo.

Esto es:

- Si no hay nada guardado en la caché, se inicia la aplicación en la ventana de inicio de sesión.
- Si la caché tiene la *ID* del usuario guardada, se iniciará desde la ventana de selección de escuelas.
- Si tiene el objeto de la escuela guardado, se iniciará desde la ventana de selección de perfiles sociales.

- Si tiene el objeto de perfil social y todo lo anterior, se inicia desde el menú principal.

Aquí mostramos un ejemplo de obtención de datos de la caché:

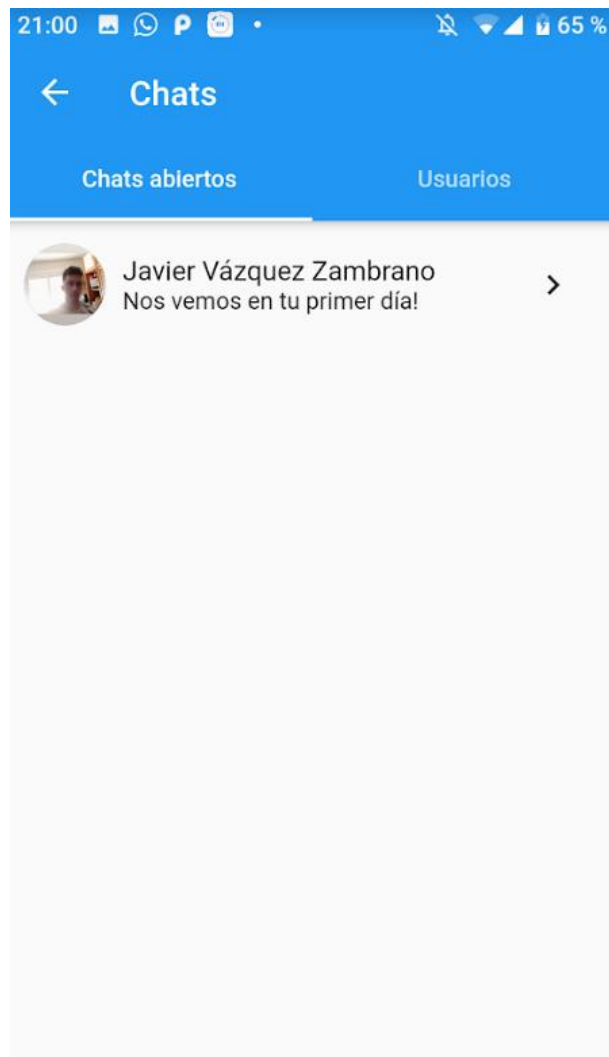
```
Future<SportSchool> _loadSportSchool() async {
  SharedPreferences preferences = await SharedPreferences.getInstance();
  Map aux = await jsonDecode(preferences.get("chosenSportSchool"));
  chosenSportSchool = SportSchool.fromMap(aux);
  return chosenSportSchool;
}
```

Primero, se instancia el objeto *SharedPreferences*, que tiene guardada la clave *chosenSportSchool*, que representa la escuela seleccionada por el usuario. Este objeto viene dado en forma de *JSON*, por lo que lo convertimos a objeto *SportSchool* y lo devolvemos, obteniendo la escuela sin problemas. Con esto, obtenemos la escuela que ha sido seleccionada desde la caché, ahorrando muchísimas consultas a la base de datos que ralentizarían el uso del sistema.

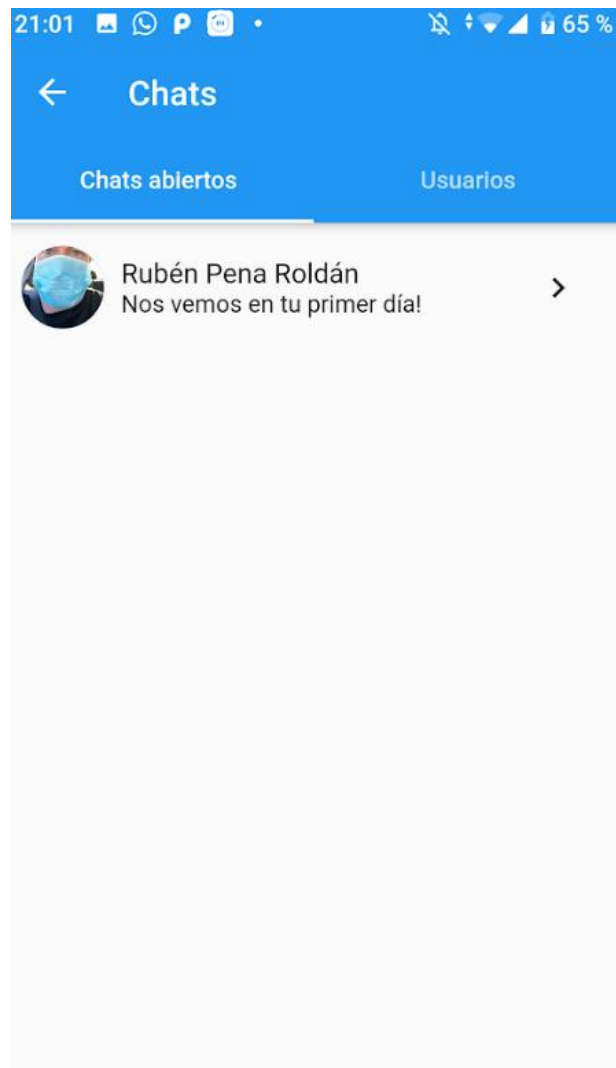
## RF-12: Ventana de chats

Este requisito ha sido implementado por Antonio Montaña en su totalidad.

Para implementar este requisito se ha usado un *widget* llamado *TabBarView*, que consiste en una *AppBar* con un título y un número dado de "pestañas" en las que se pueden implementar distintos cuerpos como si se tratasen de *Scaffold*. Para ello, se ha implementado en una pestaña los *chats* que se encuentren abiertos, es decir, los *chats* en los que el perfil social seleccionado participe, o mejor dicho, los objetos de tipo *ChatRoom* en cuya lista de usuarios se encuentre la *ID* del perfil social junto con el último mensaje enviado en esa sala de *chat*. En la otra pestaña, se encuentran los distintos perfiles sociales de la escuela, todos con los que el perfil social puede abrir una sala de *chat*. La pantalla ha quedado tal que así:

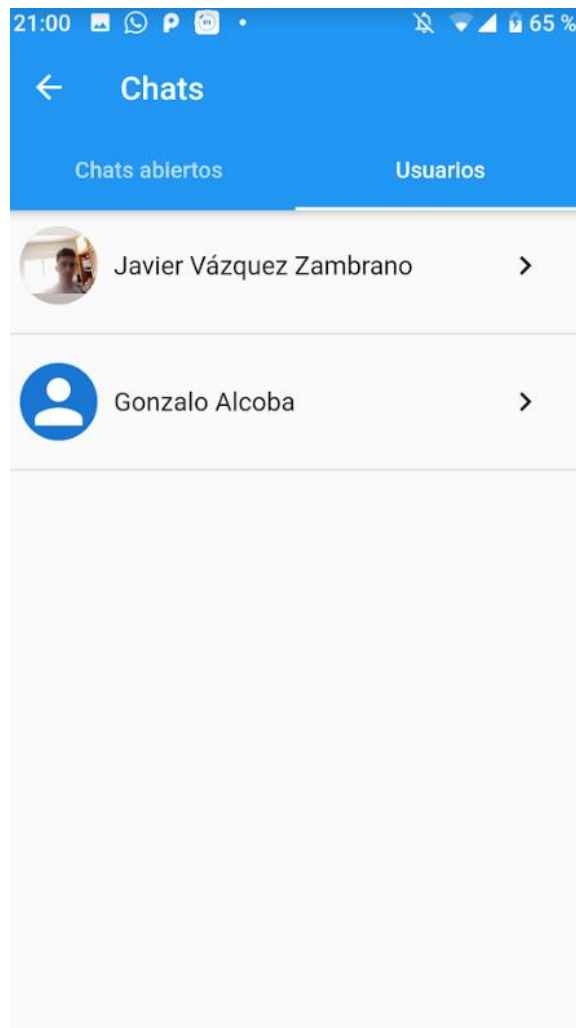


Esta foto se ha hecho desde el punto de vista del director de la escuela de ejemplo, hablando con un alumno. La siguiente se hace desde el punto de vista del alumno:



Como se puede observar, se trata de una lista de perfiles sociales con la imagen de perfil, el nombre y apellidos de el perfil social y el último mensaje enviado en dicha sala de *chat*. Si se pulsa en uno de los *chats* abiertos, se redirigirá al usuario a la sala de *chat* correspondiente.

En la otra pestaña, se incluye la lista de perfiles sociales pero sin incluir el último mensaje, de forma que si se pulsa en cada uno de ellos, se redirige a una sala de *chat* existente con ese perfil social, o se abrirá una nueva sala de *chat*.



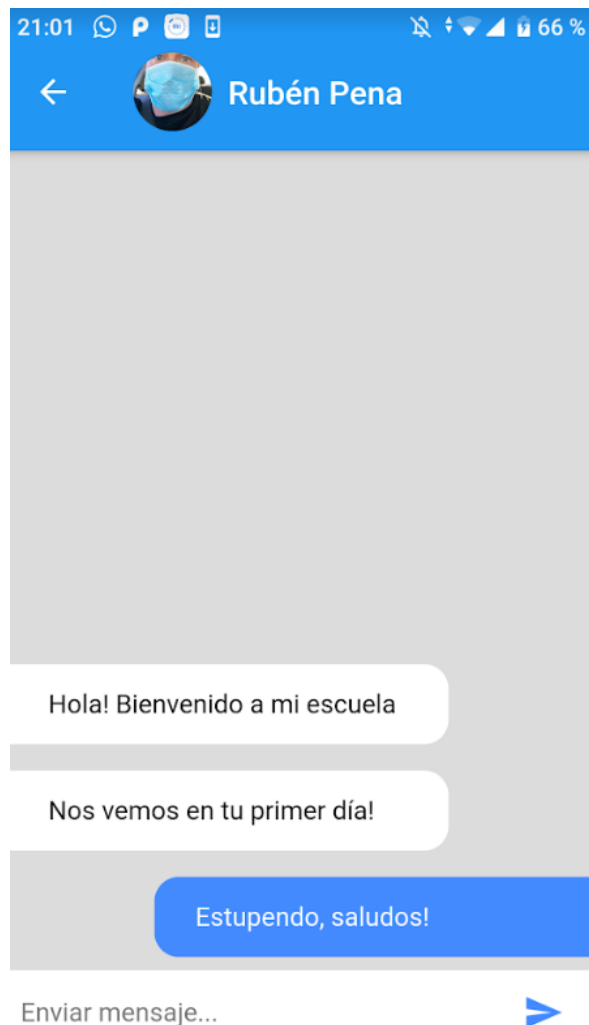
En este caso, la sala de *chat* no tendrá una *ID* autogenerada por *Firebase*, sino que nosotros mismos le asignamos la *ID*, siendo ésta la *ID* del primer perfil social y del segundo, separadas por una '\_'.

Además, empleamos la caché de la aplicación para guardar en ella el objeto de la sala de *chat* seleccionada por el usuario, para acceder a ella de forma mucho más sencilla.

### **RF-13: Chat Room**

Este requisito ha sido desarrollado en su totalidad por Antonio Montaña.

Para cumplimentar este requisito hemos implementado una ventana en la que hemos incluido un *Scaffold* cuya *AppBar* ha sido personalizada para incluir la imagen de perfil del perfil social con el que se está *chateando* en esa sala, y el nombre y apellidos de dicho perfil. En el cuerpo de la vista se encuentra lo interesante, tal y como se puede observar en la siguiente imagen:



En el cuerpo de la vista se encuentra la lista de mensajes que han sido enviados por los distintos perfiles sociales. Para diferenciarlos, los enviados por el perfil social actual están con el fondo azul, mientras que los enviados por el otro perfil social están con el fondo en blanco.

En la parte baja de la vista encontramos el cuadro de texto (Pone "send a message", pendiente de arreglarlo para ponerlo en español...) para enviar mensajes al otro perfil social en la sala de *chat*. Dicho mensaje será enviado cuando el usuario pulse el botón de envío y se almacenará en la base de datos en una colección dentro de la propia sala de *chat*, tal y como se observa en la siguiente imagen:

<a href="#">🏠</a> > <a href="#">chatRooms</a> > <a href="#">8Tna2zShJ4XsA...</a> > <a href="#">messages</a> > <a href="#">VH7exwpUomq...</a>		
<a href="#">☰</a> 8Tna2zShJ4XsANOrJGMn_xHOWDh9...	<a href="#">📄</a> messages	<a href="#">☰</a> VH7exwpUomqvjrG1tBok
<a href="#">+</a> Iniciar colección	<a href="#">+</a> Agregar documento	<a href="#">+</a> Iniciar colección
<a href="#">messages</a> >	<a href="#">VH7exwpUomqvjrG1tBok</a> >	<a href="#">+</a> Agregar campo
<a href="#">+</a> Agregar campo id: "8Tna2zShJ4XsANOrJGMn_xHOWDh96\ users <ul style="list-style-type: none"> <li>0 "8Tna2zShJ4XsANOrJGMn"</li> <li>1 "xHOWDh96WLHB8gTad2Hp"</li> </ul>	XgqXkL8pBGAVmk2c5UxX oTLm5bD62ZX1HAvz5ZsN	message: "Hola! Bienvenido a mi escuela" senderId: "8Tna2zShJ4XsANOrJGMn" sentDate: 14 de agosto de 2020 a las 18:05:00 UTC+2

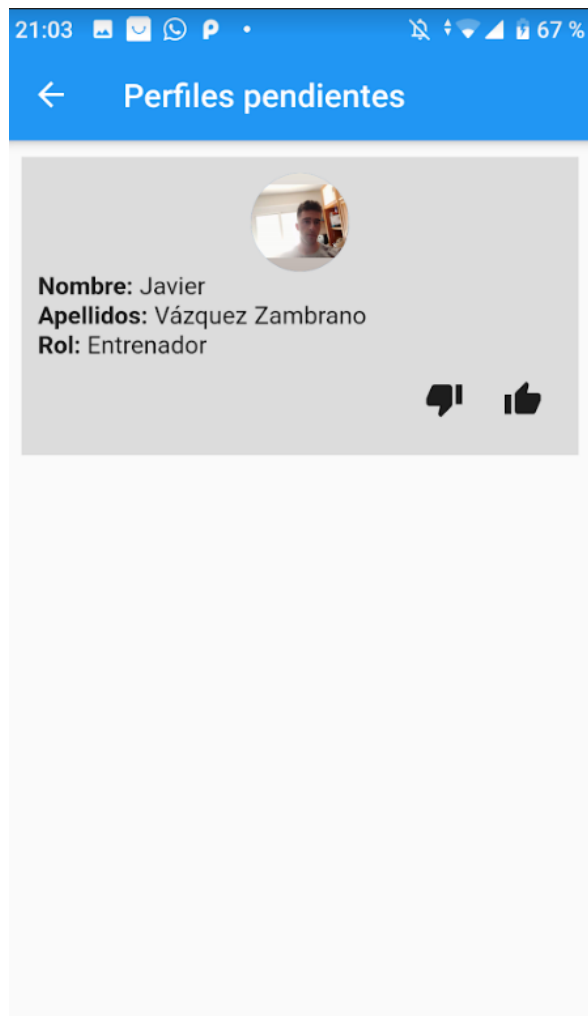
Como se puede observar, se almacena cada mensaje como documentos independientes, para poder trabajar con ellos de forma individual para ponerle el color de fondo y la posición según el emisor del mensaje, además de la fecha para ordenarlos por fecha de envío. Es el único caso en el que se almacena una colección dentro de un documento en nuestra base de datos.

#### RF-14: Aceptar y rechazar perfiles sociales

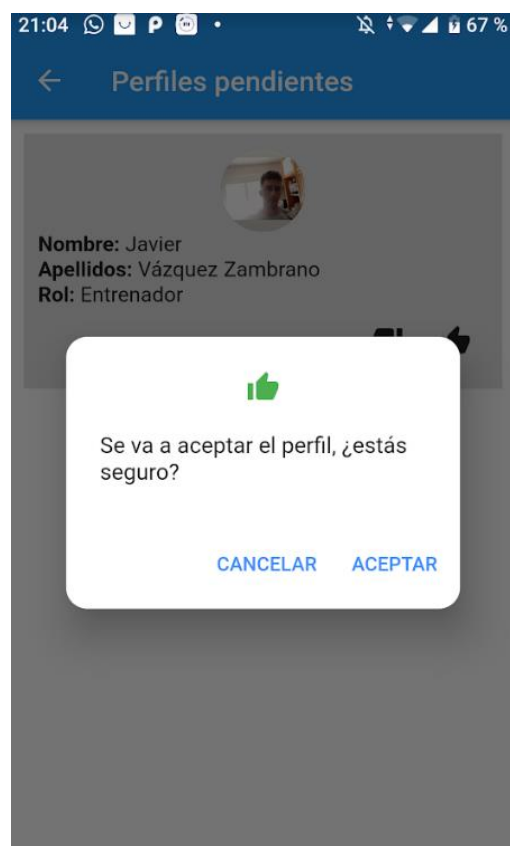
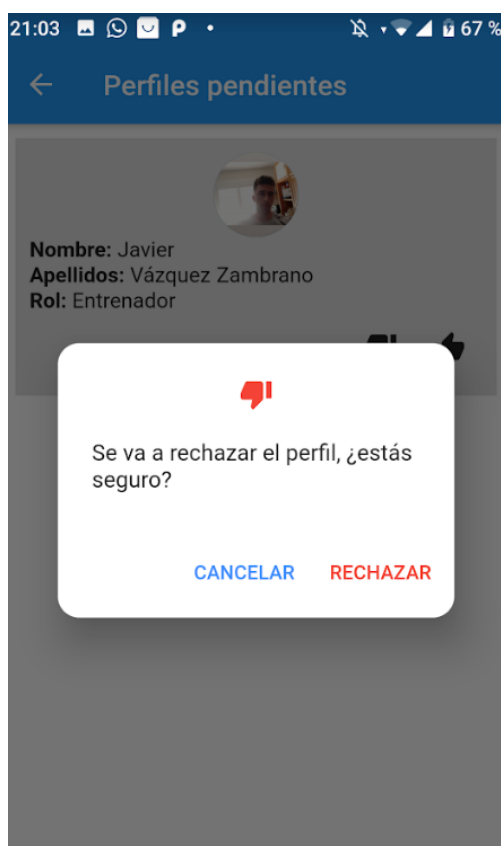
Este requisito ha sido implementado en su totalidad por Antonio Montaña.

Para cumplimentar este requisito hemos desarrollado una vista en la que se incluye una lista de perfiles sociales que estén en estado *pendiente*, de forma que el director de una escuela pueda aceptarlos o rechazarlos según su conveniencia. Para cada perfil social se muestra la imagen de perfil, el nombre y apellidos del perfil social y el rol que desempeñará en la escuela en caso de ser aceptado. La pantalla ha quedado tal que así:





En cada perfil social se muestran dos botones, el pulgar arriba y el pulgar abajo. En función de el botón pulsado, se abrirá un *dialog* para confirmar la operación que quiere realizar el director, tal y como se ve en las siguientes imágenes:



En caso de ser aceptado, el estado del perfil social dado pasará a ser *aceptado* y podrá disfrutar de las funcionalidades de la aplicación dentro de la escuela. En caso contrario, el perfil social será eliminado de la base de datos, ya que de momento consideramos inútil ocupar ese espacio en la base de datos. Más adelante añadiremos funcionalidades con el estado *rechazado*.

### **RF-15: Aceptar y rechazar escuelas**

Este requisito ha sido implementado por Antonio Montaña en su totalidad.

Para cumplimentar este requisito, hemos implementado una lista idéntica a la de aceptar y rechazar perfiles sociales que se vio en el requisito *RF-14*. A esta vista se accede desde el botón habilitado en el menú de administradores, mostrando una lista de escuelas que tengan el estado pendiente, así como el director que ha creado la escuela en el sistema, junto con el logotipo de la escuela y la imagen de perfil del director.

Según el deseo del administrador, deberá pulsar el pulgar arriba para aceptar o el pulgar abajo para rechazar. Tras esto, aparecerán los respectivos *dialogs* para aceptar o rechazar dichas escuelas y perfiles, quedando en estado aceptado en caso de aceptar la escuela y el director, o eliminando de la base de datos las dos instancias en caso de rechazarlas.

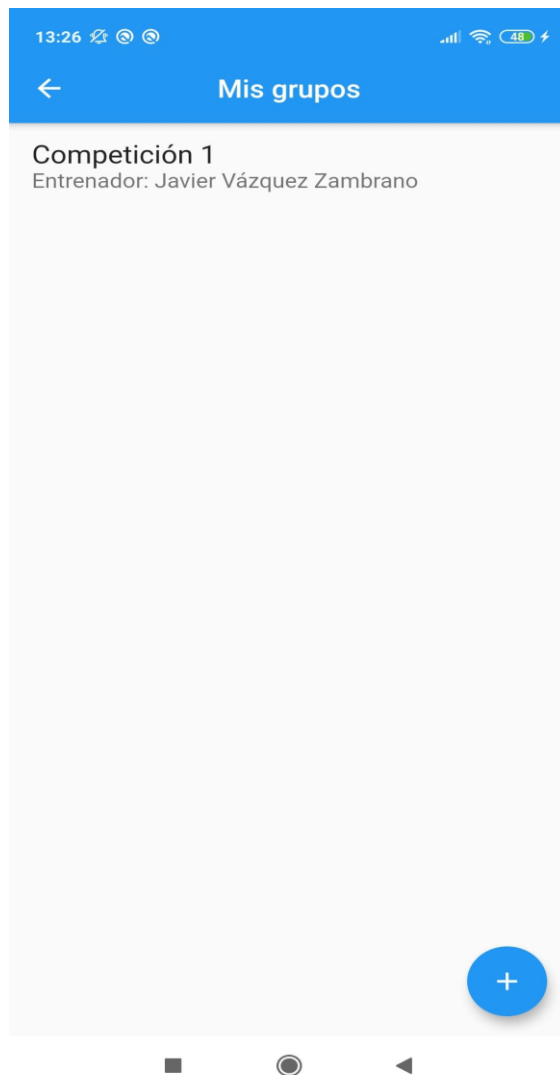
### **RF-16: Información sobre el grupo**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito hemos desarrollado una vista en la que se puede ver una lista de grupos en la que:

- Si es el director de la escuela, le aparecen todos los grupos de la misma, mostrando así el nombre del grupo y el entrenador encargado del mismo. Además, en esta vista le aparece un botón flotante abajo a la derecha que nos lleva a otra vista que servirá para crear un grupo. Para visualizar en detalle un grupo, basta con seleccionar uno. En caso de que no haya grupos para mostrar, aparecerá un mensaje avisando de que no hay nada que mostrar.
- Si es un entrenador, le aparecen solo los grupos en los que él mismo está asignado como entrenador, mostrando así el nombre del grupo y el entrenador encargado del mismo. Para visualizar en detalle un grupo, nuevamente basta con seleccionar uno. En caso de que no haya grupos para mostrar, se le aparece un mensaje avisando de que no hay nada que mostrar.





- En caso de ser un alumno, directamente no le aparece esta vista. Para acceder a la visualización del grupo en el que se encuentra, debe de acceder desde el menú *home* mediante el apartado de "mi grupo". En caso de que no se encuentre actualmente en ningún grupo, se le aparecerá un *dialog* de alerta avisándole de que no pertenece a ninguno.

Una vez nos dirigimos a la vista de detalles de un grupo, podemos observar que se muestran el nombre del mismo, el entrenador asignado y los horarios y alumnos del grupo. Además, si el perfil social que hemos seleccionado al iniciar sesión es el director de la escuela, aparece un botón flotante en la esquina inferior derecha que sirve para editar los datos del grupo.



### **RF-17: Crear grupo**

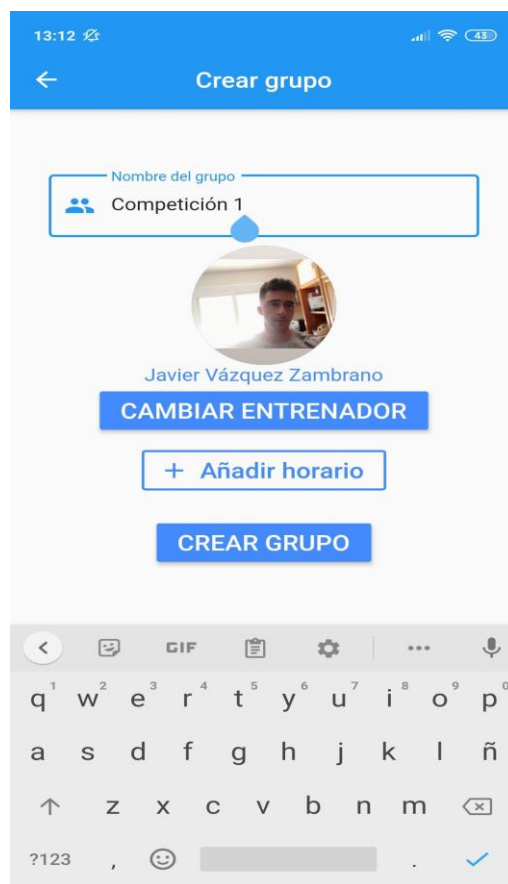
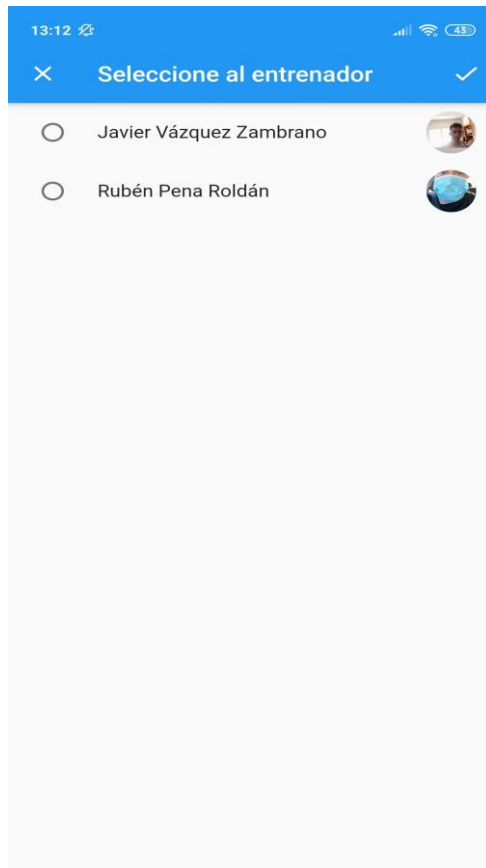
Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito hemos desarrollado una vista la cual aparece si accedemos desde la vista de listado de grupos desde el perfil social del director de la escuela pulsando sobre el botón flotante que aparece en la misma, mencionado anteriormente. En esta vista podemos observar que hay diversos campos a rellenar: el nombre del grupo, el entrenador encargado y los horarios, todos obligatorios.

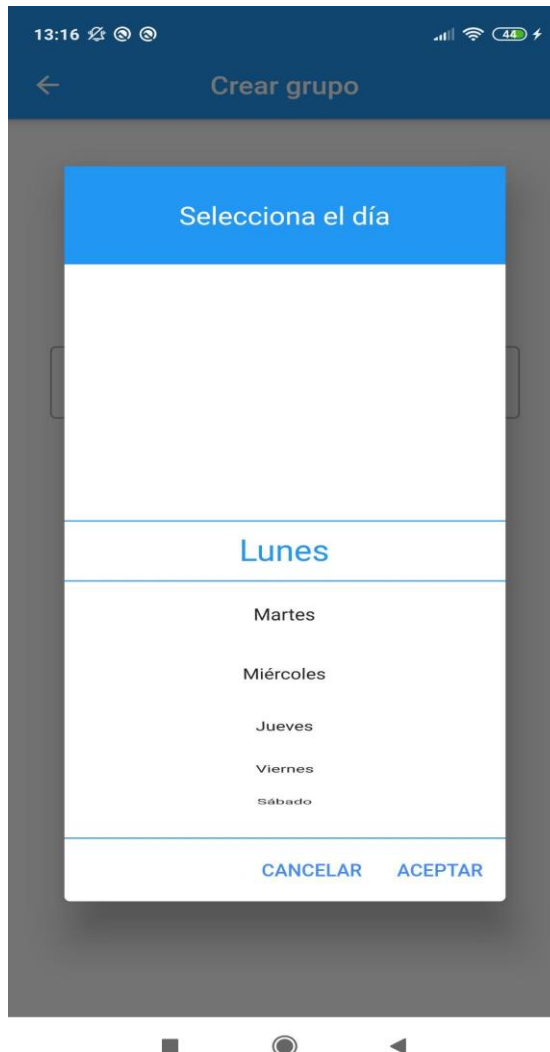


A la hora de seleccionar al entrenador y los horarios ocurre lo siguiente:

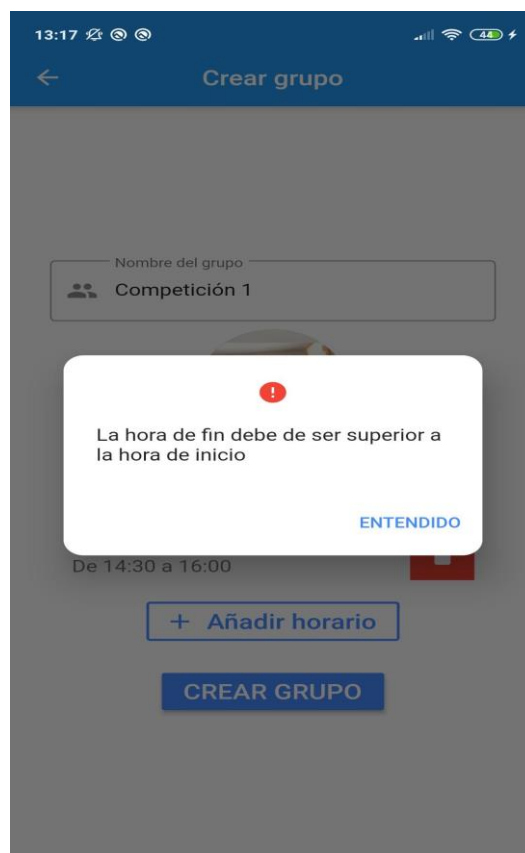
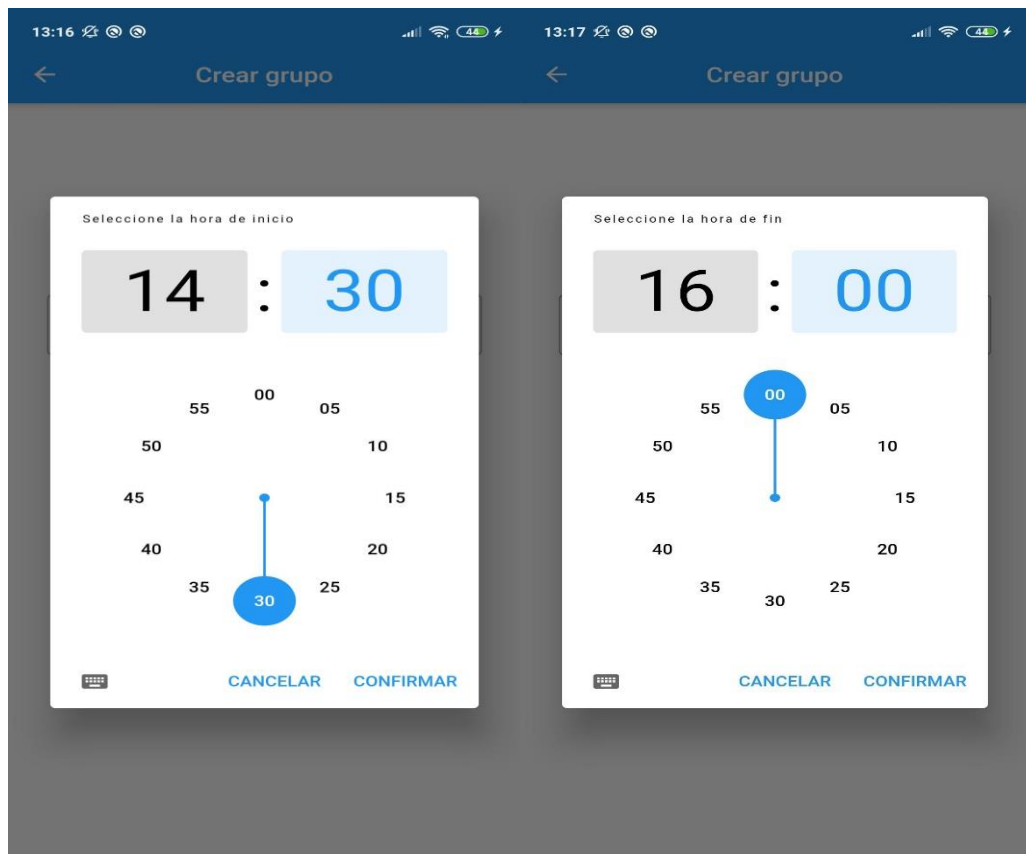
- Para seleccionar al entrenador, vemos que hay un botón para ello, el cual si lo pulsamos, nos lleva a otra vista donde aparece la lista de entrenadores de la escuela y arriba de la misma, un filtro mediante el cual podemos filtrar por el nombre de los entrenadores. Si vemos la parte de arriba de la pantalla, a la izquierda, si le damos a la "X" que aparece, cerramos la vista sin aplicar ningún cambio, y a la derecha si le damos al *check* que aparece, en caso de haber seleccionado algún entrenador, se cierra la vista y nos muestra, en la vista para crear el grupo, la foto del entrenador y su nombre completo, y debajo un botón para cambiar el entrenador, el cual tiene el mismo procedimiento que el de seleccionar entrenador.

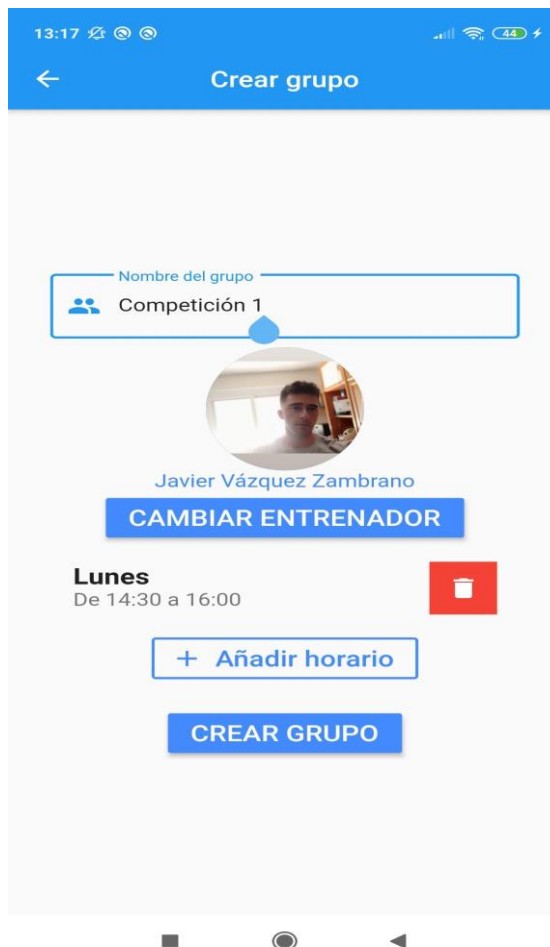


- Para seleccionar los horarios de las clases del grupo, debemos de pulsar el botón para ello, el cual nos muestra para empezar un *dialog* para elegir el día de la semana. Si le damos a confirmar, nos dirige al siguiente *dialog* que es para seleccionar la hora de inicio de la clase y al confirmar, nos aparecerá el último *dialog* que es para seleccionar la hora de fin de la clase. En caso de que en este último *dialog* le demos a aceptar y la hora de fin sea anterior a la de inicio, saltará un *dialog* de error y no se guardaría el nuevo horario. Si en cualquiera de los *dialogs* para el día de la semana y las horas de inicio y fin le damos a cancelar, desaparecerá ese *dialog* y no ocurrirá nada. Al crear el horario, aparecerá arriba del botón, pudiendo crear más horarios o eliminar los ya creados mediante el botón que se habilita a la derecha de los horarios.

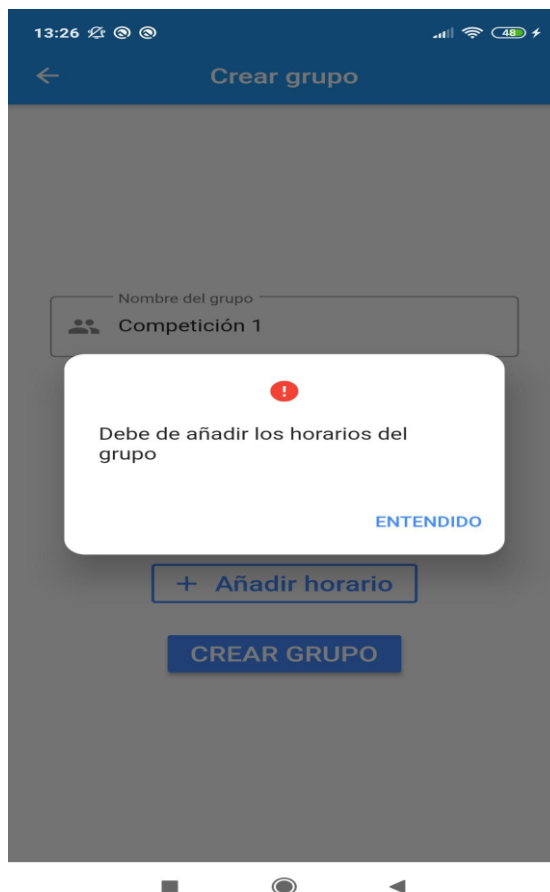








Para crear el grupo, debemos pulsar el botón flotante de abajo a la derecha, el cual en caso de estar todo correcto (nombre relleno, entrenador seleccionado y al menos un horario), nos devolverá a la vista de grupos donde aparecerá el nuevo grupo creado. En caso de haber algún error, nos mostrará algún mensaje de error mediante un *dialog* indicando el error correspondiente.



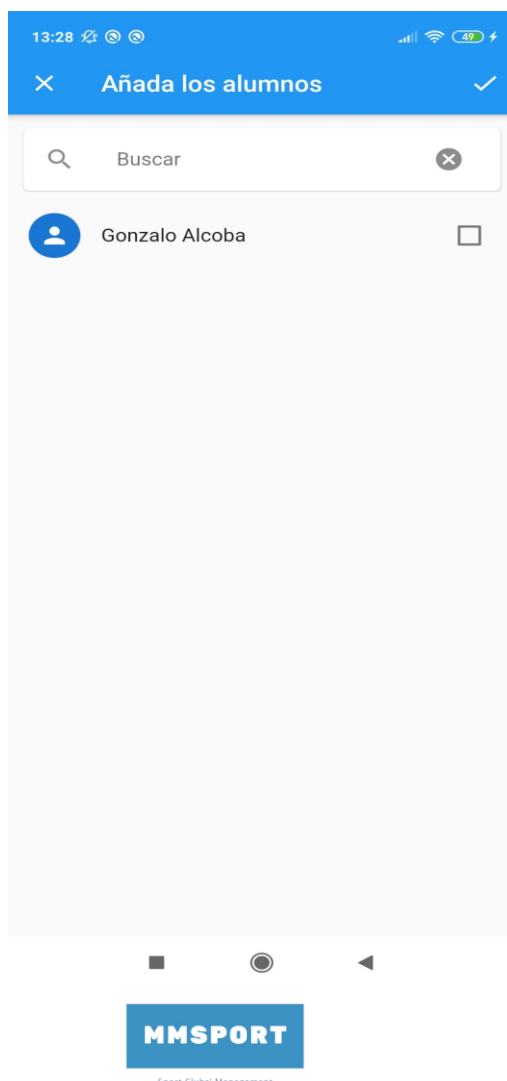
Hay que destacar que, para elegir los horarios, se ha implementado una librería llamada *flutter\_material\_pickers*, que se puede encontrar [aquí](#). Con esto, se nos facilita muchísimo la tarea para elegir los días de la semana en los que se celebrarán los entrenamientos.

### **RF-18: Editar grupo**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

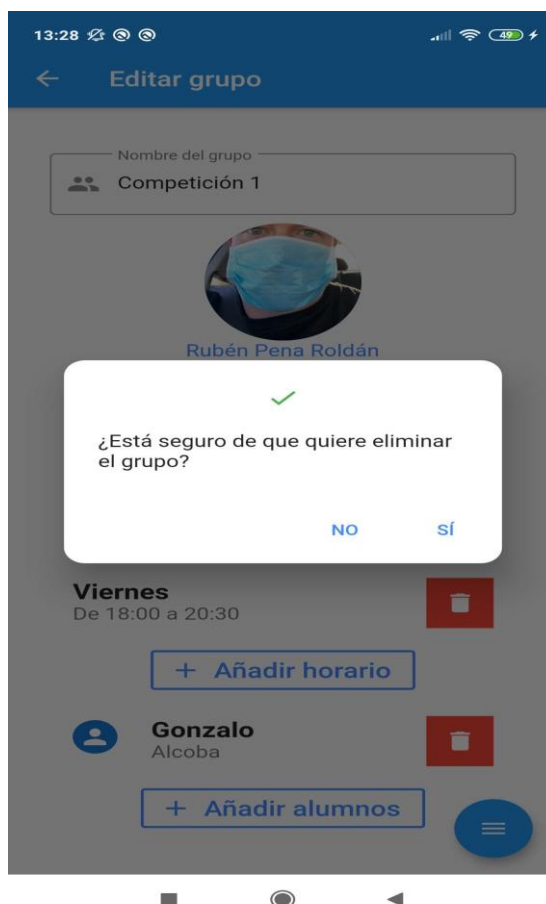
Para cumplimentar este requisito hemos desarrollado una vista a la cual se accede desde la pantalla de visualización de detalles de un grupo, dándole al botón de editar que aparece solamente al haber seleccionado el perfil social del director de la escuela en el inicio de sesión, mencionado anteriormente. La funcionalidad es similar a la de la vista para crear un grupo.

Además, nos aparece un botón mediante el que podemos seleccionar los alumnos del grupo. A la hora de seleccionar los alumnos, es similar a la funcionalidad para seleccionar el entrenador, con la diferencia de que solo nos aparecen los alumnos que no pertenecen a dicho grupo y, además, podemos seleccionar a más de un alumno. Al confirmar pulsando el *check* de la pantalla para confirmar los alumnos seleccionados, volveremos a la pantalla de editar y arriba del botón para seleccionar los alumnos nos aparecerá la lista de alumnos que actualmente están seleccionados para el grupo, pudiendo borrarlos de la misma mediante el botón rojo que aparece a la derecha de cada uno.





Por último, si le damos al botón flotante que aparece en la esquina inferior derecha, se nos abrirá un menú de botones hacia arriba y a la izquierda de cada uno aparecerá el nombre de la acción que realiza. Si le damos al botón de abajo del todo con una "X", cierra el menú. Para guardar los cambios hechos en el grupo, con el menú de botones desplegado, seleccionamos el botón de "guardar" y en caso de que no haya ningún error, nos volverá a mostrar los datos del grupo con los cambios hechos. En caso de que queramos volver a la visualización del mismo sin aplicar cambios solo tenemos que seleccionar el botón de "cancelar".



### **RF-19: Asignar y eliminar alumnos/as de un grupo**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

La implementación de este requisito ya ha sido explicada en el requisito anterior, es decir, el *RF-18*, por lo que no hay necesidad de desarrollar este punto.

### **RF-20: Eliminar grupo**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito, dentro del menú de botones en la vista de edición de grupo mencionada anteriormente en el requisito *RF-18*, basta con seleccionar el botón de "eliminar". Al hacerlo, nos saldrá un diálogo para confirmar que queremos eliminarlo. En caso de eliminarlo, nos devolverá a la pantalla del listado de los grupos de la escuela.

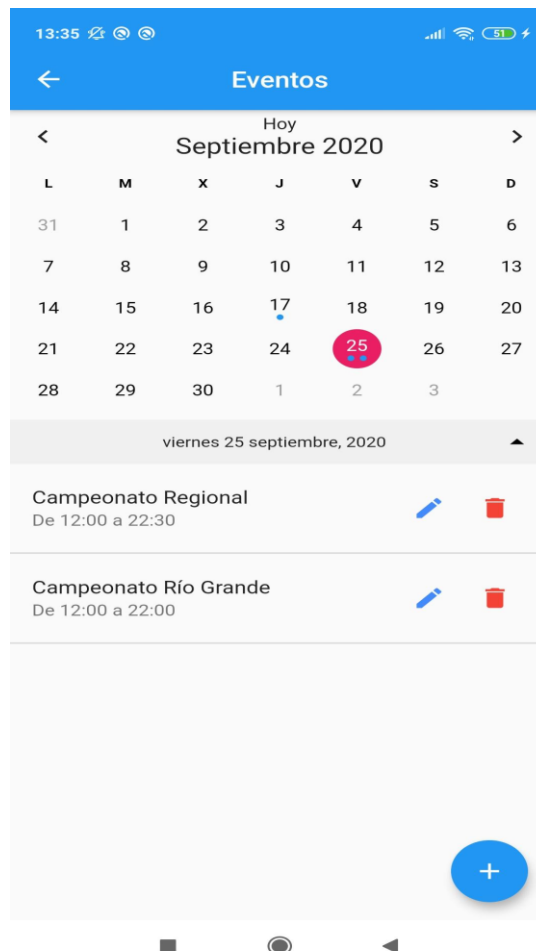
### **RF-21: Información del calendario de eventos**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito, hemos desarrollado una vista a la cual se accede desde el menú principal de cualquier perfil social que pertenezca a la escuela, pulsando en "calendario". Al hacerlo, nos aparecerá una vista de un calendario en la parte superior de la pantalla, la cual al principio nos mostrará la vista del mes actual y con el día actual seleccionado. Si nos fijamos en los días que muestra el calendario, podemos observar que en caso de que haya algún evento ese día, aparecerá un puntito debajo del número. En caso de que haya un solo evento, aparecerá un puntito. En caso de ser dos los eventos aparecerán dos, y si son tres o más aparecerán tres.

Al seleccionar un día que tenga eventos, en la parte de abajo de la pantalla se mostrará mediante una lista los eventos para ese día, mediante el nombre del mismo y el horario.

Hay que destacar el uso de la librería *flutter\_clean\_calendar*, que nos ayuda a implementar el calendario que se verá en la siguiente imagen. Podemos encontrar su documentación [aquí](#).



## RF-22: Crear eventos

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito, hemos desarrollado una vista a la cual se accede desde la vista del calendario de eventos, pulsando el botón flotante de abajo a la derecha que aparece al haber seleccionado el perfil social del director de la escuela. Al hacerlo, se nos abre una pantalla en la que rellenamos el nombre del evento, el día que se realizará y la hora de inicio y de fin del mismo.



13:30 13:30 49%

← Crear evento

Evento

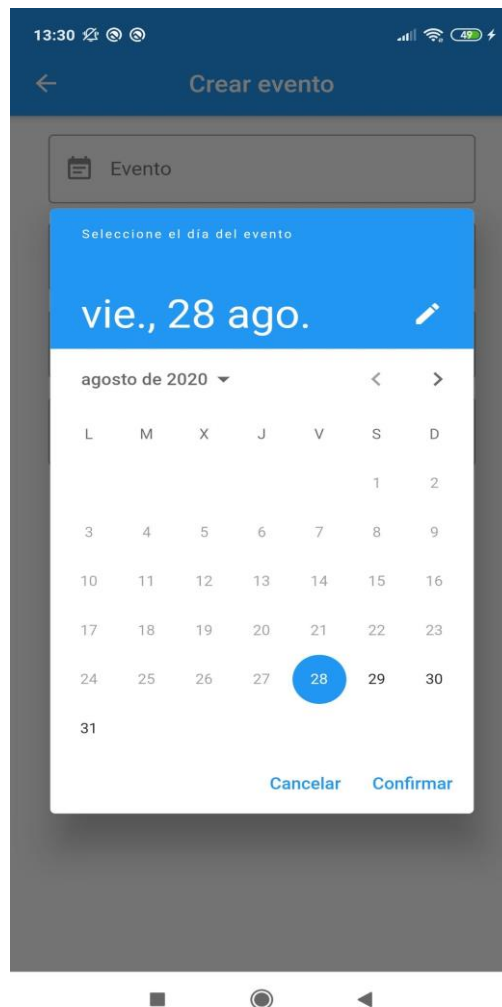
Día del evento

Hora inicio

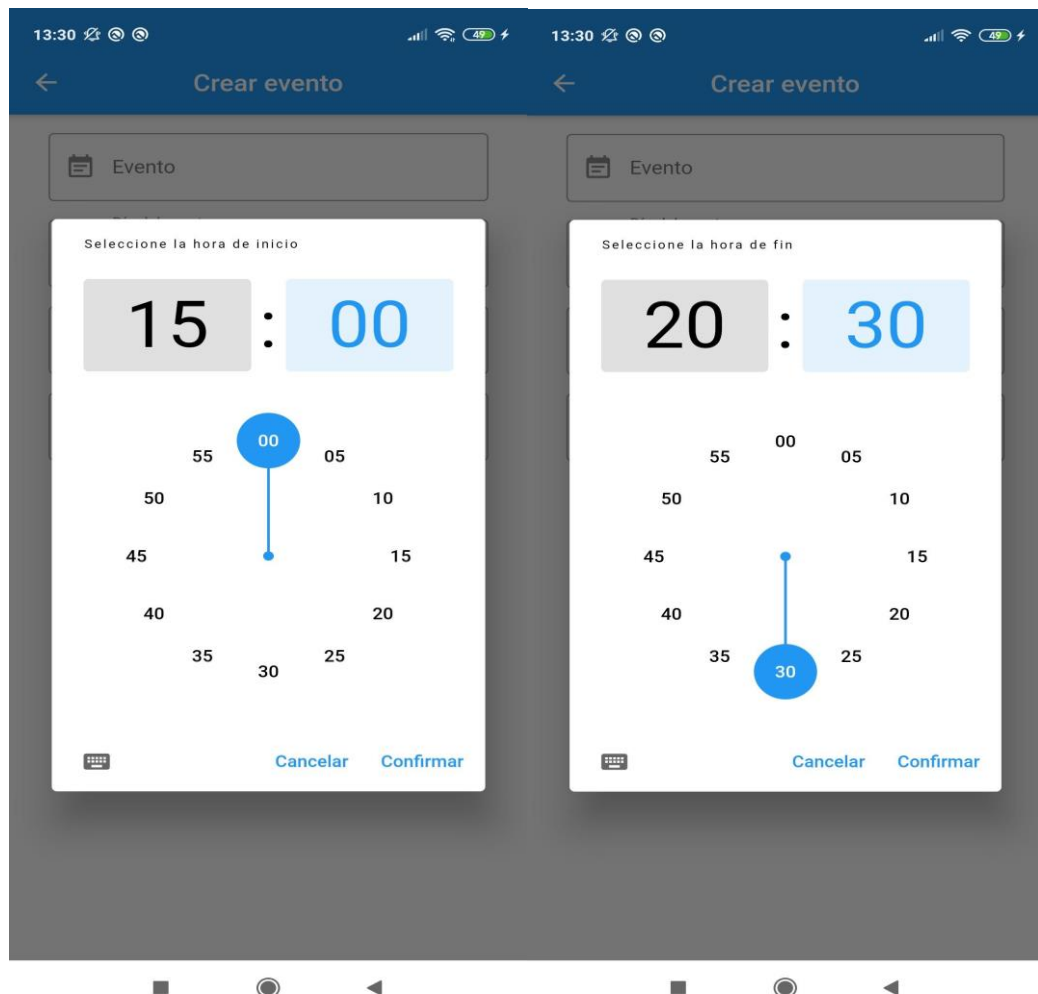
Hora fin

CREAR EVENTO

Para seleccionar el día se muestra un *dialog* con un calendario en el que debemos seleccionar un día, el cual debe ser como muy temprano el día siguiente al actual.



Para seleccionar las horas de inicio y fin, solamente tenemos que pulsar en los apartados correspondientes y se nos abrirá un *dialog* para seleccionar las horas. En caso de que la hora de inicio sea superior a la hora fin se nos mostrará un *dialog* de error.



Para guardar el evento, basta con pulsar el botón flotante de abajo a la derecha, y en caso de no haber errores nos devolverá a la pantalla del calendario de eventos con el nuevo evento creado. En caso de que haya errores (que falten campos por rellenar) se nos mostrará un *dialog* de error como en el requisito RF-17.

### **RF-23: Editar eventos**

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito, basta con seleccionar el día del evento que queremos editar en el calendario de eventos y al mostrarlo en la lista de los eventos de ese día, habiendo seleccionado el perfil social del director de la escuela, nos aparecerá un icono con la forma de un lápiz azul. Si le damos, nos aparece una pantalla idéntica a la de crear eventos, mencionada anteriormente, funcionando exactamente igual que la pantalla de crear eventos. En caso de que el evento ocurra en el día actual, no se podrá editar el evento.

13:41 100% 54%

← Editar evento

Evento  
Campeonato Regional

Día del evento  
25-09-2020

Hora inicio  
12:00

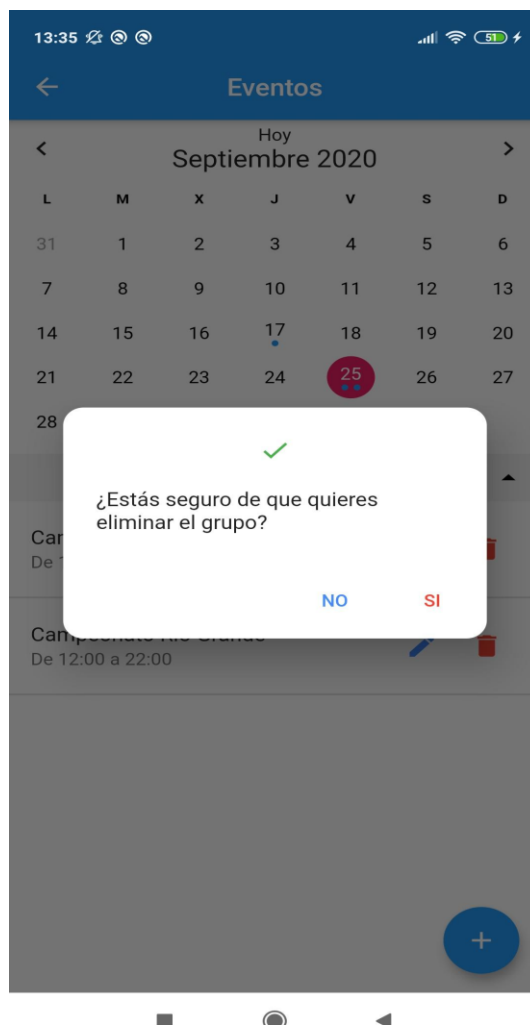
Hora fin  
22:30

GUARDAR CAMBIOS

### RF-24: Eliminar eventos

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

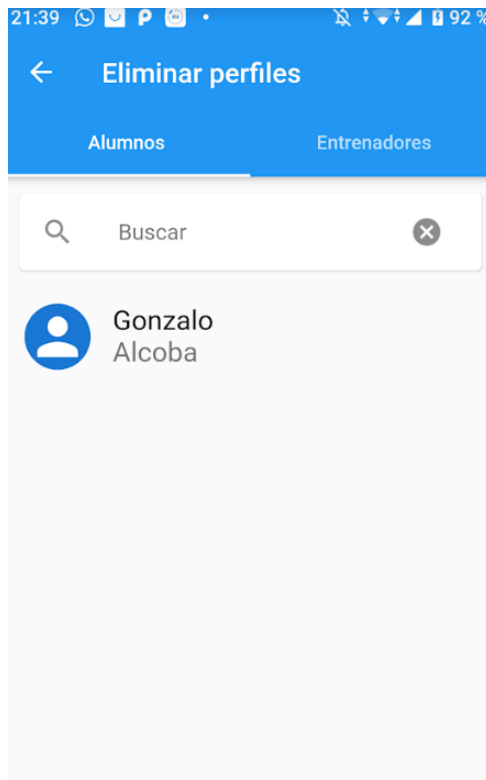
Para cumplimentar este requisito, basta con seleccionar el día del evento que queremos editar en el calendario de eventos y al mostrarlo en la lista de los eventos de ese día, habiendo seleccionado el perfil social del director de la escuela, nos aparecerá un icono con la forma de un cubo de basura rojo. Al pulsarlo, nos aparecerá un *dialog* para confirmar que queremos eliminarlo. Al confirmar, se eliminará el evento y por tanto ya no se mostrará en el calendario de eventos.



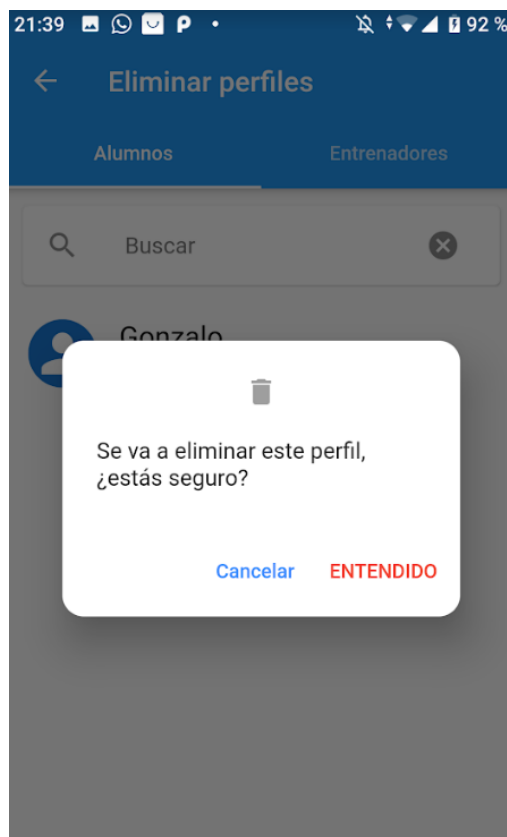
### **RF-25: Eliminar perfiles sociales de una escuela**

Este requisito ha sido implementado en su totalidad por Antonio Montaña.

Para cumplimentar este requisito se ha implementado una vista a la cual se accede mediante el botón habilitado en el menú de director de escuela. Una vez se accede, se muestran dos pestañas con listas distintas cada una, diferenciando entre los alumnos de la escuela y los entrenadores, para poder filtrar más rápido según el perfil que se desea eliminar. Además, se ha implementado un buscador para facilitar más aún esta labor. La ventana ha quedado tal que así:



Como se puede observar para cada perfil social se muestra su imagen de perfil junto con su nombre y apellidos. Al pulsar en uno de los perfiles sociales, se abrirá un *dialog* para confirmar la eliminación de dicho perfil social. Al confirmar la eliminación, quedará eliminada toda la información que incumbe a dicho perfil social de la base de datos.



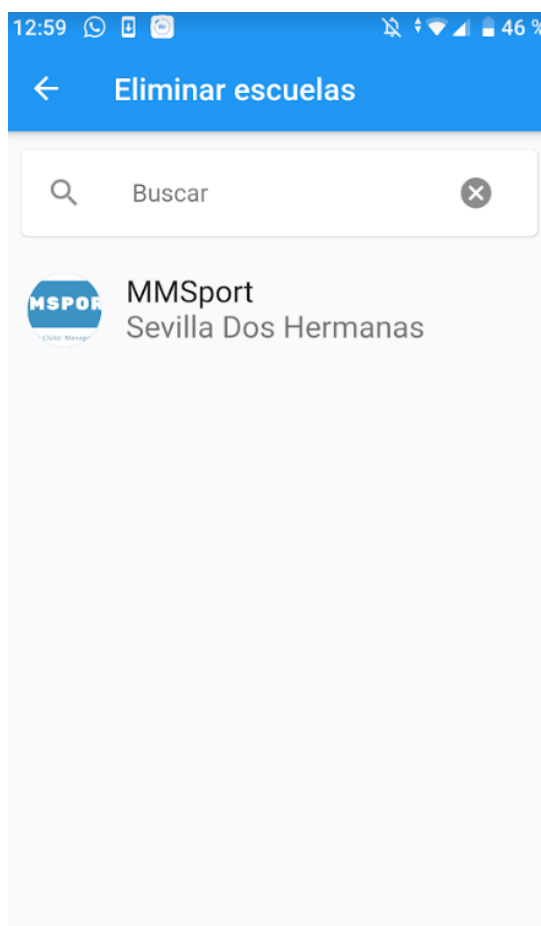
## RF-26: Banear/desbanear usuarios del sistema

Esta funcionalidad no se ha implementado porque se ha decidido dejarla para su implementación en un futuro. El motivo es que en la consola de la base de datos ya tenemos la opción de inhabilitar y rehabilitar cuentas de usuario, por lo que decidimos dejarla para más tarde debido a que es innecesaria a corto plazo, pudiendo aprovechar ese tiempo en otras tareas mucho más importantes.

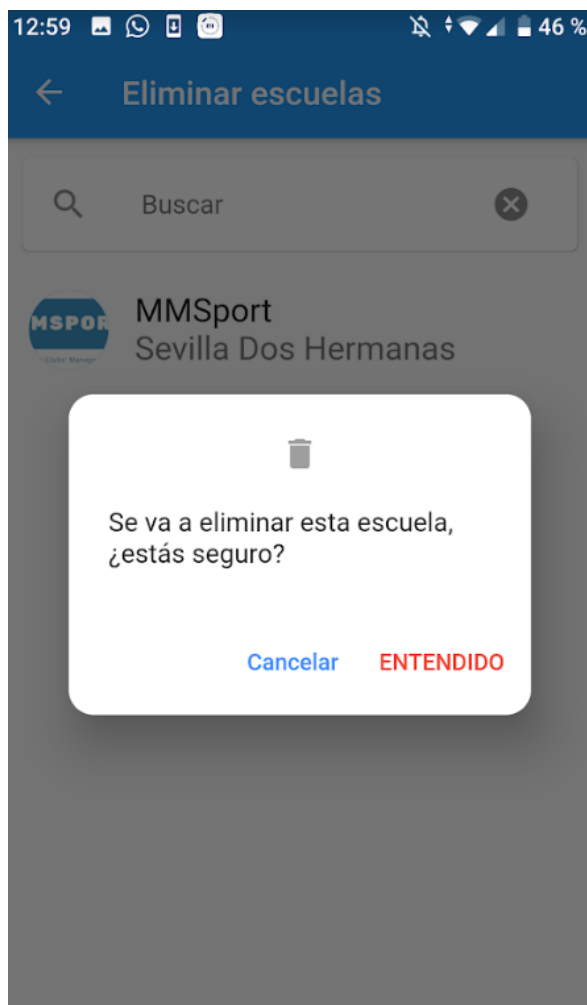
## RF-27: Eliminar escuelas del sistema

Este requisito ha sido implementado por Antonio Montaña en su totalidad.

Para cumplimentar este requisito, se ha implementado una vista similar a la de eliminar perfiles sociales. Para acceder a dicha vista hay que iniciar sesión como administrador, y una vez en el menú principal, pulsar en el botón habilitado para dicha función. Se mostrará la lista tal y como se puede apreciar en la siguiente imagen:



Al igual que en el requisito RF-25, al pulsar en una de las escuelas, se abrirá un *dialog* en el que se podrá confirmar la eliminación de la escuela, quedando toda la información que referencia a dicha escuela eliminada de la base de datos.



### RF-28: Confirmar correo electrónico

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito se ha usado la información proporcionada por *FirebaseAuth*, que se encarga expresamente de enviar un correo electrónico a la dirección introducida por el usuario en el momento del registro. *Firebase* también proporciona la plantilla del correo electrónico, que contendrá un enlace que deberá ser pulsado por el usuario para confirmar su dirección de correo electrónico. Hasta que no se confirme esta dirección, no podrá iniciar sesión en el sistema, aunque, en caso de que sea un director de escuela, podrá crear la misma en el momento del registro, pero no más tarde hasta haber validado su dirección.

En caso de que el usuario quiera iniciar sesión sin haber confirmado su correo, le aparecerá un *dialog* con un error recordándole que debe confirmar la dirección.

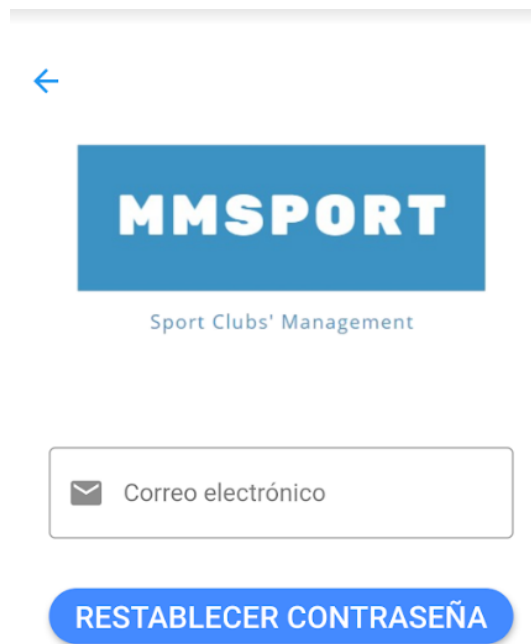
### RF-29: Restablecer contraseña

Este requisito ha sido implementado por Pablo Vázquez en su totalidad.

Para cumplimentar este requisito, se ha implementado un botón en la vista de inicio de sesión clásico en el que, si el usuario olvida su contraseña, será redirigido a una vista en la que podrá introducir su correo electrónico. *Firebase* enviará automáticamente un correo electrónico que contendrá un enlace para que el usuario lo



pulse y, con una ventana proporcionada por el propio *Firebase*, podrá cambiar su contraseña.



## 3.8. Pruebas del sistema

Hablar sobre las pruebas del sistema, poniendo ejemplos de los *tests* unitarios, de integración y de los widgets, y los *tests* de aceptación.

### 3.8.1. Tests unitarios

Los *tests* unitarios son pruebas que consisten en comprobar el correcto funcionamiento de métodos individuales, funciones que se encargan de realizar algún cálculo o comprobación. Estas pruebas han sido ejecutadas gracias al paquete *test*, dado por el propio *framework* que se puede encontrar [aquí](#). En este proyecto se han realizado *tests* unitarios como el ejemplo que vamos a mostrar a continuación:

```
test('Validación de email - Email inválido', () {  
  bool aux = FormValidators.validateValidEmail("prueba");  
  expect(aux, false);  
});  
  
test('Validación de email - Email correcto', () {  
  bool aux = FormValidators.validateValidEmail("prueba@prueba.com");  
  expect(aux, true);  
});
```

Estos *tests* comprueban el correcto funcionamiento de la función que valida el campo *email* de algunos formularios. En el primero se introduce una dirección inválida, esperando un resultado fallido del *test*. En el otro, se introduce una dirección correcta, esperando un resultado correcto al finalizar la ejecución de la prueba.

### 3.8.2. Tests de aceptación

Aquí se muestran todas las pruebas de aceptación pasadas al proyecto para comprobar su correcto funcionamiento. Las pruebas tendrán el siguiente formato:

Título	Inicio de sesión con campos vacíos
Pasos	En la ventana de inicio de sesión, pulsar en "Iniciar sesión" con los campos del formulario vacíos.
Resultado esperado	Deben aparecer errores en los dos campos del formulario.
Resultado obtenido	Correcto

Título	Inicio de sesión con e-mail inválido
Pasos	En la ventana de inicio de sesión, rellenar el campo de correo electrónico poniendo "prueba" y una contraseña cualquiera y pulsar en iniciar sesión.
Resultado esperado	Debe aparecer un error en el campo de correo electrónico que diga que el correo es inválido.
Resultado obtenido	Correcto

Título	Inicio de sesión con e-mail que no exista
Pasos	En la ventana de inicio de sesión, rellenar el correo electrónico con uno válido pero que no exista en la aplicación y una contraseña cualquiera y pulsar en iniciar sesión.
Resultado esperado	Debe aparecer una ventana de error que diga que el correo no existe.
Resultado obtenido	Correcto

Título	Inicio de sesión con contraseña corta
Pasos	En la ventana de inicio de sesión, rellenar el correo electrónico con uno que exista en la aplicación y una contraseña corta.

Resultado esperado	Debe aparecer una ventana de error que diga que la contraseña es demasiado corta.
Resultado obtenido	Correcto

<b>Título</b>	<b>Inicio de sesión con contraseña errónea</b>
Pasos	En la ventana de inicio de sesión, rellenar el correo electrónico con uno que exista en la aplicación y una contraseña errónea.
Resultado esperado	Debe aparecer una ventana de error que diga que la contraseña es errónea.
Resultado obtenido	Correcto

<b>Título</b>	<b>Inicio de sesión válido</b>
Pasos	En la ventana de inicio de sesión, rellenar el formulario con datos correctos.
Resultado esperado	Debe validar el formulario y redirigir a la vista de selección de escuelas.
Resultado obtenido	Correcto

<b>Título</b>	<b>Entrar al registro</b>
Pasos	En la ventana de inicio de sesión, pulsar en el botón "Regístrate".
Resultado esperado	Debe redirigir al formulario de registro.
Resultado obtenido	Correcto

<b>Título</b>	<b>Registro con los campos vacíos</b>
Pasos	En la ventana de registro, pulsar en "Registrarse" sin rellenar ningún campo.
Resultado esperado	Debe mostrar los errores en todos los campos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Registro con correo electrónico inválido</b>
Pasos	En la ventana de registro, rellenar el campo del e-mail con "prueba" y pulsar en "Registrarse".
Resultado esperado	Debe mostrar un error que indique que el correo es inválido en el campo del e-mail.
Resultado obtenido	Correcto

<b>Título</b>	<b>Registro con contraseña demasiado corta</b>
Pasos	En la ventana de registro, rellenar el campo del e-mail con correo electrónico correcto y contraseña de menos de 6 caracteres y pulsar en "Registrarse".
Resultado esperado	Debe mostrar un error que indique que la contraseña es demasiado corta en el campo de la contraseña.
Resultado obtenido	Correcto

<b>Título</b>	<b>Registro con contraseñas distintas</b>
Pasos	En la ventana de registro, rellenar el campo del e-mail con correo electrónico correcto y contraseña correcta, pero distinta en el campo de confirmación y pulsar en "Registrarse".
Resultado esperado	Debe mostrar un error que indique que las contraseñas son distintas.
Resultado obtenido	Correcto

<b>Título</b>	<b>Registro correcto sin escuela</b>
Pasos	En la ventana de registro, rellenar bien el formulario e indicar que no se tiene escuela.
Resultado esperado	Debe mostrar un <i>dialog</i> de confirmación y redirigir a la ventana de inicio de sesión.
Resultado obtenido	Correcto

<b>Título</b>	<b>Registro correcto con escuela</b>
Pasos	En la ventana de registro, rellenar bien el formulario e indicar que se tiene escuela.

Resultado esperado	Debe redirigir a la ventana de creación de escuela.
Resultado obtenido	Correcto

<b>Título</b>	<b>Creación de escuela - Campos vacíos</b>
Pasos	En la ventana de creación de escuela, pulsar en "Siguiete" sin rellenar ningún campo.
Resultado esperado	Debe mostrar los errores en todos los campos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Creación de escuela - Foto vacía</b>
Pasos	En la ventana de creación de escuela, pulsar en "Siguiete" con los campos rellenos pero sin seleccionar ninguna foto como logotipo.
Resultado esperado	Debe mostrar un <i>dialog</i> que indique que falta la fotografía.
Resultado obtenido	Correcto

<b>Título</b>	<b>Creación de escuela - Formulario correcto</b>
Pasos	En la ventana de creación de escuela, pulsar en "Siguiete" con los campos rellenos y con una foto como logotipo.
Resultado esperado	Debe redirigir a la siguiente página del formulario, la creación del director.
Resultado obtenido	Correcto

<b>Título</b>	<b>Creación de director - Campos vacíos</b>
Pasos	En la ventana de creación de director, pulsar en "Finalizar" sin rellenar ningún campo.
Resultado esperado	Debe mostrar los errores en todos los campos menos en el segundo apellido, que es opcional.
Resultado obtenido	Correcto

<b>Título</b>	<b>Creación de director - Foto vacía</b>
Pasos	En la ventana de creación de director, pulsar en "Siguiente" con los campos rellenos pero sin seleccionar ninguna foto como imagen de perfil.
Resultado esperado	Debe mostrar un <i>dialog</i> que indique que falta la fotografía.
Resultado obtenido	Correcto

<b>Título</b>	<b>Creación de director - Formulario correcto</b>
Pasos	En la ventana de creación de director, pulsar en "Siguiente" con los campos rellenos y con una foto como logotipo.
Resultado esperado	Debe mostrar un <i>dialog</i> de confirmación sobre la creación de la escuela y el director, y al pulsar en entendido, redirigir a la ventana de inicio de sesión.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de búsqueda de escuelas para inscribirse</b>
Pasos	En la ventana de mis escuelas, pulsar en el botón "+".
Resultado esperado	Comprobar que redirige a la ventana de búsqueda de escuelas.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de búsqueda de escuelas - Filtrado</b>
Pasos	En la ventana de búsqueda de escuelas, pulsar en la lupa y filtrar por nombre, provincia y municipio.
Resultado esperado	Comprobar que el filtrado muestra correctamente las escuelas que tiene que mostrar según el filtro.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de búsqueda de escuelas - Selección</b>
Pasos	En la ventana de búsqueda de escuelas, seleccionar cualquier escuela.

Resultado esperado	Debe redirigir a la ventana de creación de perfil social.
Resultado obtenido	Correcto

<b>Título</b>	<b>Inscripción en una escuela - Campos vacíos</b>
Pasos	En la ventana de inscripción a una escuela, enviar el formulario con los campos vacíos.
Resultado esperado	Debe mostrar los errores correspondientes en los campos, excepto el segundo apellido que es opcional.
Resultado obtenido	Correcto

<b>Título</b>	<b>Inscripción en una escuela - Foto vacía como entrenador</b>
Pasos	En la ventana de inscripción a una escuela, enviar el formulario con los campos rellenos, con la selección de entrenador pero sin foto de perfil.
Resultado esperado	Debe mostrar un error porque la foto es obligatoria para el entrenador.
Resultado obtenido	Correcto

<b>Título</b>	<b>Inscripción en una escuela - Foto vacía como alumno</b>
Pasos	En la ventana de inscripción a una escuela, enviar el formulario con los campos rellenos, con la selección de alumno pero sin foto de perfil.
Resultado esperado	Debe mostrar un <i>dialog</i> de confirmación y redirigir a la vista de mis escuelas.
Resultado obtenido	Correcto

<b>Título</b>	<b>Inscripción en una escuela - Foto seleccionada como entrenador</b>
Pasos	En la ventana de inscripción a una escuela, enviar el formulario con los campos rellenos, con la selección de entrenador y con foto de perfil.

Resultado esperado	Debe mostrar un <i>dialog</i> de confirmación y redirigir a la vista de mis escuelas.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de mis escuelas - Botón de añadir perfil</b>
Pasos	En la ventana de mis escuelas, pulsar el botón de "añadir perfil social".
Resultado esperado	Debe redirigir a la ventana de inscripción a la escuela.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de mis escuelas - Desplazamiento de la lista</b>
Pasos	En la ventana de mis escuelas, teniendo más de una escuela asociada al usuario, desplazar la lista de izquierda a derecha.
Resultado esperado	Comprobar que se desplaza bien por toda la lista, y que también cambia el "punto" del índice debajo de la lista.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de mis escuelas - Selección de una escuela</b>
Pasos	En la ventana de mis escuelas, seleccionar una de las escuelas del usuario.
Resultado esperado	Comprobar que redirige a la ventana de selección de perfiles sociales.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de selección de perfil - Datos correctos</b>
Pasos	En la ventana de selección de perfil, comprobar que los datos de los perfiles sociales están realmente en esa escuela.
Resultado esperado	Que sean correctos
Resultado obtenido	Correcto



<b>Título</b>	<b>Ventana de selección de perfil - Desplazamiento de la lista</b>
Pasos	En la ventana de selección de perfil, con una lista de más de un perfil social, desplazar la lista de izquierda a derecha.
Resultado esperado	Comprobar que la lista se desplaza correctamente, y que también cambia el "punto" debajo de la lista según la posición.
Resultado obtenido	Correcto

<b>Título</b>	<b>Ventana de selección de perfil - Selección de perfil</b>
Pasos	En la ventana de selección de perfil, seleccionar uno de los perfiles sociales.
Resultado esperado	Comprobar que redirige al menú principal.
Resultado obtenido	Correcto

<b>Título</b>	<b>Menú principal del director</b>
Pasos	En la ventana de menú principal, comprobar que salen los botones siguientes: <i>Chats</i> , Mis grupos, Aceptar/rechazar perfiles, Editar perfil, Eliminar perfiles, Calendario.
Resultado esperado	Que salgan los botones.
Resultado obtenido	Correcto

<b>Título</b>	<b>Menú principal del alumno</b>
Pasos	En la ventana de menú principal, comprobar que salen los botones siguientes: <i>Chats</i> , Mi grupo, Calendario, Editar perfil.
Resultado esperado	Que salgan los botones.
Resultado obtenido	Correcto

<b>Título</b>	<b>Menú principal del entrenador</b>
Pasos	En la ventana de menú principal, comprobar que salen los botones siguientes: <i>Chats</i> , Mis grupos, Calendario, Editar perfil.

Resultado esperado	Que salgan los botones.
Resultado obtenido	Correcto

<b>Título</b>	<b>Menú principal - Redirecciones</b>
Pasos	En la ventana de menú principal, comprobar que las redirecciones al pulsar los botones son correctas en cada caso.
Resultado esperado	Que cada botón redirija a su ventana correspondiente.
Resultado obtenido	Correcto

<b>Título</b>	<b>Menú principal - Menú <i>PopUp</i></b>
Pasos	En la ventana de menú principal, abrir el menú <i>PopUp</i> de la esquina superior derecha y comprobar las redirecciones de los cuatro botones.
Resultado esperado	Que salgan los botones y que cada uno redirija a su ventana correspondiente.
Resultado obtenido	Correcto

<b>Título</b>	<b><i>Chats</i> - Ventana principal - <i>Chats</i> abiertos</b>
Pasos	En la ventana principal de <i>chats</i> , comprobar que salen correctamente los <i>chats</i> abiertos en la base de datos.
Resultado esperado	Que salgan los <i>chats</i> abiertos del perfil social seleccionado.
Resultado obtenido	Correcto

<b>Título</b>	<b><i>Chats</i> - Ventana principal - Usuarios</b>
Pasos	En la ventana principal de <i>chats</i> , comprobar que salen correctamente los perfiles sociales de la escuela respecto a la base de datos.
Resultado esperado	Que salgan los perfiles sociales de la misma escuela que el perfil social seleccionado.

Resultado obtenido	Correcto
--------------------	----------

<b>Título</b>	<b>Chats - Ventana principal - Cambio entre pestañas</b>
Pasos	En la ventana principal de <i>chats</i> , comprobar que se cambia de pestaña correctamente tanto pulsando en el título como deslizando el dedo.
Resultado esperado	Que cambie de pestaña en los dos casos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Chats - Ventana principal - Seleccionar una sala</b>
Pasos	En la ventana principal de <i>chats</i> , seleccionar una sala de <i>chat</i> , es decir, algún perfil social.
Resultado esperado	Redirige a la sala de <i>chat</i> correspondiente.
Resultado obtenido	Correcto

<b>Título</b>	<b>Chats - Sala de <i>chat</i> - Mensajes</b>
Pasos	En una sala de <i>chat</i> , comprobar que, en caso de haber ya mensajes en esa sala, se despliegan correctamente.
Resultado esperado	Comprobar que los mensajes se muestran bien.
Resultado obtenido	Correcto

<b>Título</b>	<b>Chats - Sala de <i>chat</i> - Envío</b>
Pasos	En una sala de <i>chat</i> , comprobar que se envía un mensaje y se muestra correctamente.
Resultado esperado	Comprobar el envío de mensajes y su despliegue en pantalla.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Acceso desde director y entrenador</b>
Pasos	Acceder a la ventana de mis grupos como director y entrenador
Resultado esperado	Comprobar que se muestra un listado de grupos, al director se le muestran todos los grupos de la escuela y al entrenador los grupos en los que está asignado.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Crear grupo</b>
Pasos	En el listado de grupos para el director, hay un botón para acceder al formulario de creación de grupo.
Resultado esperado	Al pulsar el botón, debe redirigir al formulario.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Crear grupo - Seleccionar entrenador</b>
Pasos	En la creación de grupo, al pulsar en el botón de seleccionar entrenador, se debe mostrar un listado con los entrenadores que se pueden asignar al grupo que se está creando.
Resultado esperado	El entrenador debe salir como asignado al grupo.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Crear grupo - Añadir horario</b>
Pasos	En la creación de grupo, al pulsar en el botón de añadir horario, se debe mostrar un formulario para añadir un horario al grupo que se está creando.
Resultado esperado	El horario debe quedar añadido al formulario.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Crear grupo - Eliminar horario</b>
---------------	--

Pasos	En la creación de grupo, al pulsar en el botón que aparece al lado de un horario previamente añadido, se debe eliminar dicho horario.
Resultado esperado	El horario debe desaparecer del formulario.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Crear grupo - Guardar</b>
Pasos	En la creación de grupo, al pulsar en el botón de guardar.
Resultado esperado	El grupo debe quedar guardado en la base de datos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Acceso desde alumno</b>
Pasos	Acceder a la ventana de mi grupo como alumno
Resultado esperado	Comprobar que se muestra directamente la información del grupo en el que está dicho alumno.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Seleccionar grupo del listado</b>
Pasos	En el listado de grupos del director o entrenador, seleccionar uno de los grupos.
Resultado esperado	Debe redirigir a la ventana de información de dicho grupo.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Información del grupo</b>
Pasos	Al mostrar la información del grupo, se debe mostrar el nombre, el entrenador asignado, el horario del grupo y los alumnos integrantes.
Resultado esperado	Se deben mostrar los datos correctamente.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Activar modo edición</b>
Pasos	Al mostrar la información del grupo, siendo director, pulsar el botón para activar la edición del grupo.
Resultado esperado	Se debe cambiar el formulario a modo edición.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Editar grupo</b>
Pasos	En el modo edición grupo, cambiar toda la información del grupo.
Resultado esperado	Comprobar que se ha cambiado la información del grupo.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Añadir y eliminar integrantes al grupo</b>
Pasos	En el modo edición grupo, añadir y eliminar integrantes del grupo para comprobar su buen funcionamiento.
Resultado esperado	Comprobar que se ha cambiado el grupo una vez guardada la información al añadir y eliminar integrantes.
Resultado obtenido	Correcto

<b>Título</b>	<b>Grupos - Eliminar grupo</b>
Pasos	En el modo edición grupo, eliminar el grupo.
Resultado esperado	Comprobar que se ha eliminado correctamente el grupo.
Resultado obtenido	Correcto

<b>Título</b>	<b>Aceptar y rechazar perfiles - Listado</b>
Pasos	Desde el menú principal como director, acceder a la ventana de aceptar y rechazar perfiles sociales a la escuela de dicho director.

Resultado esperado	Se debe redirigir al listado de perfiles sociales pendientes.
Resultado obtenido	Correcto

<b>Título</b>	<b>Aceptar y rechazar perfiles - Aceptar</b>
Pasos	En la ventana de aceptar y rechazar perfiles sociales, aceptar algún perfil pendiente.
Resultado esperado	Debe aparecer un <i>dialog</i> para confirmar la decisión, y una vez aceptada, se debe haber quitado de la lista dicho perfil, estando aceptado.
Resultado obtenido	Correcto

<b>Título</b>	<b>Aceptar y rechazar perfiles - Rechazar</b>
Pasos	En la ventana de aceptar y rechazar perfiles sociales, rechazar algún perfil pendiente.
Resultado esperado	Debe aparecer un <i>dialog</i> para confirmar la decisión, y una vez aceptada, se debe haber quitado de la lista dicho perfil, estando rechazado y eliminado de la base de datos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Editar perfil social - Ventana</b>
Pasos	En el menú principal, al pulsar en el botón de editar perfil social.
Resultado esperado	Debe redirigir al usuario al formulario de edición del perfil social.
Resultado obtenido	Correcto

<b>Título</b>	<b>Editar perfil social - Cambiar imagen</b>
Pasos	En la ventana de edición de perfil social, debemos cambiar la imagen pulsando en ella y eligiendo otra cualquiera.
Resultado esperado	Se debe mostrar la imagen nueva en el perfil.
Resultado obtenido	Correcto

<b>Título</b>	<b>Editar perfil social - Campos vacíos</b>
Pasos	En la ventana de edición de perfil social, debemos vaciar los campos y enviar el formulario.
Resultado esperado	Deben mostrarse los errores en los campos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Editar perfil social - Guardar</b>
Pasos	En la ventana de edición de perfil social, debemos cambiar los datos que queramos y guardar.
Resultado esperado	Debe guardarse el estado del perfil social en la base de datos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Aceptar y rechazar escuelas - Listado</b>
Pasos	Desde el menú principal como administrador, acceder a la ventana de aceptar y rechazar escuelas.
Resultado esperado	Se debe redirigir al listado de escuelas con estado pendiente.
Resultado obtenido	Correcto

<b>Título</b>	<b>Aceptar y rechazar escuelas - Aceptar</b>
Pasos	En la ventana de aceptar y rechazar escuelas, aceptar alguna escuela pendiente.
Resultado esperado	Debe aparecer un <i>dialog</i> para confirmar la decisión, y una vez aceptada, se debe haber quitado de la lista dicha escuela, estando aceptada la escuela y el director de la misma.
Resultado obtenido	Correcto

<b>Título</b>	<b>Aceptar y rechazar escuelas - Rechazar</b>
---------------	---



Pasos	En la ventana de aceptar y rechazar escuelas, rechazar alguna escuela pendiente.
Resultado esperado	Debe aparecer un <i>dialog</i> para confirmar la decisión, y una vez aceptada, se debe haber quitado de la lista dicha escuela, estando rechazada la escuela y el director de la misma, siendo eliminados de la base de datos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Información</b>
Pasos	En la ventana de menú principal, pulsar en el botón para acceder al calendario de eventos.
Resultado esperado	Debe redirigir al usuario a la ventana del calendario de eventos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Crear eventos</b>
Pasos	En la ventana del calendario de eventos, como director, debe haber disponible un botón que lleve a un formulario de creación de eventos. Pulsar sobre él.
Resultado esperado	Debe redirigir al usuario a la ventana de creación de eventos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Crear eventos - Campos vacíos</b>
Pasos	En la ventana de creación de eventos, dejar los campos vacíos y enviar el formulario.
Resultado esperado	Debe aparecer el error correspondiente en cada campo del formulario.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Crear eventos - Fecha anterior al presente</b>
---------------	---

Pasos	En la ventana de creación de eventos, insertar una fecha anterior al momento actual.
Resultado esperado	Debe aparecer un error que indique que la fecha no puede ser anterior al presente.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Crear eventos - Guardar</b>
Pasos	En la ventana de creación de eventos, enviar el formulario con datos correctos.
Resultado esperado	Se debe guardar el evento en la base de datos, redirigir al usuario a la ventana del calendario y mostrarse dicho evento en el calendario.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Editar eventos</b>
Pasos	En la ventana del calendario de eventos, como director, en cada evento posterior a la fecha actual se ofrecerá un botón para editarlo, pulsar sobre él.
Resultado esperado	Se debe redirigir al usuario a la ventana de edición de eventos.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Editar eventos - Campos vacíos</b>
Pasos	En la ventana de edición de eventos, dejar los campos vacíos y enviar el formulario.
Resultado esperado	Deben mostrarse los errores en los campos del formulario.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Editar eventos - Fecha anterior al presente</b>
---------------	--

Pasos	En la ventana de edición de eventos, insertar una fecha anterior al presente y enviar el formulario.
Resultado esperado	Debe mostrarse un error que indique que no se puede insertar una fecha anterior al presente.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Editar eventos - Guardar</b>
Pasos	En la ventana de edición de eventos, modificar algún campo respecto al original y guardar.
Resultado esperado	Debe guardarse el evento y mostrarse modificado en el calendario.
Resultado obtenido	Correcto

<b>Título</b>	<b>Calendario de eventos - Eliminar eventos</b>
Pasos	En la ventana del calendario de eventos, como director, en cada evento posterior a la fecha actual se ofrecerá un botón para eliminar dichos eventos. Pulsar sobre uno de ellos.
Resultado esperado	Se debe mostrar un <i>dialog</i> de confirmación y una vez aceptado, eliminar el evento.
Resultado obtenido	Correcto

<b>Título</b>	<b>Eliminar perfiles sociales de la escuela - Listado de perfiles</b>
Pasos	En el menú principal como director, pulsar sobre el botón de "eliminar perfiles".
Resultado esperado	Se debe redirigir al usuario a la ventana de eliminación de perfiles, mostrando un listado con todos los perfiles sociales de su escuela.
Resultado obtenido	Correcto

<b>Título</b>	<b>Eliminar perfiles sociales de la escuela - Eliminar alumno</b>
Pasos	En la ventana de eliminación de perfiles, pulsar en uno de los alumnos y aceptar el <i>dialog</i> correspondiente.
Resultado esperado	El alumno será eliminado de la escuela y de la base de datos, junto con su imagen de perfil y sus <i>chats</i> .

Resultado obtenido	Correcto
--------------------	----------

<b>Título</b>	<b>Eliminar perfiles sociales de la escuela - Eliminar entrenador con grupo asignado</b>
Pasos	En la ventana de eliminación de perfiles, pulsar en uno de los entrenadores que tenga grupo asignado.
Resultado esperado	Debe aparecer un <i>dialog</i> que indique que no se puede eliminar el entrenador, primero tiene que cambiar el entrenador de su grupo.
Resultado obtenido	Correcto

<b>Título</b>	<b>Eliminar perfiles sociales de la escuela - Eliminar entrenador sin grupo asignado</b>
Pasos	En la ventana de eliminación de perfiles, pulsar en uno de los entrenadores sin grupo asignado y aceptar el <i>dialog</i> correspondiente.
Resultado esperado	El entrenador será eliminado de la escuela y de la base de datos, junto con su imagen de perfil y sus <i>chats</i> .
Resultado obtenido	Correcto

<b>Título</b>	<b>Eliminar escuelas - Listado de escuelas</b>
Pasos	En el menú principal como administrador, pulsar sobre el botón de "eliminar escuelas".
Resultado esperado	Se debe redirigir al usuario a la ventana de eliminación de escuelas, mostrando un listado con todas las escuelas del sistema.
Resultado obtenido	Correcto

<b>Título</b>	<b>Eliminar escuelas - Eliminar escuela</b>
Pasos	En la ventana de eliminación de escuelas, pulsar en una de las escuelas y aceptar el <i>dialog</i> correspondiente.
Resultado esperado	La escuela será eliminada del sistema con toda la información que eso conlleva.
Resultado obtenido	Correcto

## 3.9. Instalación del sistema

### 3.9.1. Desarrolladores

Para instalar el entorno de desarrollo del sistema Windows, se deben hacer los siguientes pasos (extraído de [aquí](#)):

1. Descargar el [SDK de Flutter](#) y extraer el archivo en la ruta donde se desea instalar el *framework*.
2. Actualizar las variables de entorno del sistema. Para ello, dirígete a Panel de control -> Sistema y seguridad -> Sistema -> Ajustes avanzados. En la ventana abierta, debemos pulsar en Variables de entorno..., y aquí debemos incluir la variable de nombre **PATH**, y de valor debemos añadir la ruta extraída antes hasta *flutter/bin*. Para verificar la instalación, debemos abrir un terminal y ejecutar el comando `flutter doctor`.
3. Descargar [Android Studio](#) e instalarlo, y una vez instalado, abrirlo y dirigirte a Configuración -> Plugins y seleccionar el plugin de *Flutter*.

Con esto, ya tenemos instalado *Flutter*. Tan solo nos faltaría importar el proyecto y... ¡Listo!

## 3.10. Manual de usuario

Este manual será añadido en un futuro a nuestra aplicación, de forma que en el primer inicio de un usuario en nuestra aplicación, reciba esta guía para entender mejor el funcionamiento de la aplicación.

**¡Bienvenido! Gracias por elegir *MMSport* para hacer de tu gestión de escuelas deportivas más sencilla.** Vamos a realizar una guía rápida por las distintas funcionalidades de la aplicación.

Cuando abrimos *MMSport* por primera vez, entraremos en la ventana de inicio de sesión. Podemos iniciar sesión con una cuenta existente o registrarnos con una nueva cuenta de usuario.

# MMSPORT

Sport Clubs' Management

 Correo electrónico

 Contraseña

INICIAR SESIÓN

[Olvidé mi contraseña](#)

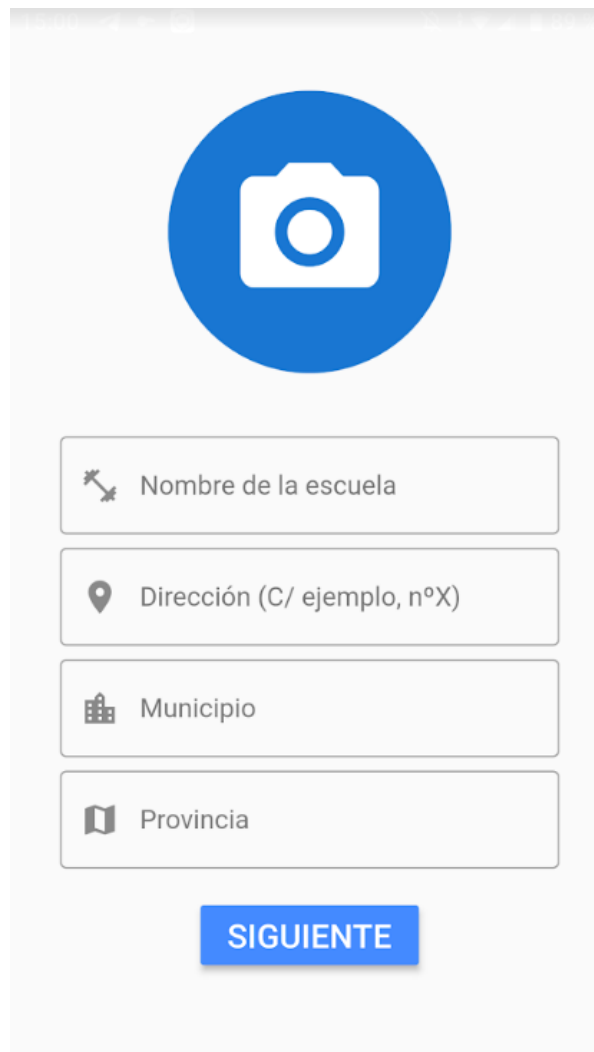
¿Aún no te has registrado?

[¡Regístrate!](#)

En la pantalla de registro, podemos seleccionar si somos dueños de una escuela o no. En caso de que queramos crear una escuela en el sistema, seremos redirigidos a una nueva pantalla en la que debemos ingresar los datos de la escuela y posteriormente los datos de nuestro perfil que actuará como director de la escuela en nuestra aplicación.

MMSPORT

Sport Clubs' Management



The screenshot shows a mobile application interface for creating a school profile. At the top, there is a large blue circular icon containing a white camera symbol. Below this, there are four input fields, each with a small icon on the left and a text label on the right:

- First field: A key icon followed by the text "Nombre de la escuela".
- Second field: A location pin icon followed by the text "Dirección (C/ ejemplo, nºX)".
- Third field: A building icon followed by the text "Municipio".
- Fourth field: A book icon followed by the text "Provincia".

At the bottom of the form, there is a blue rectangular button with the white text "SIGUIENTE".

Una vez enviada la petición para crear la escuela, deberemos esperar a que los administradores del sistema acepten la petición.

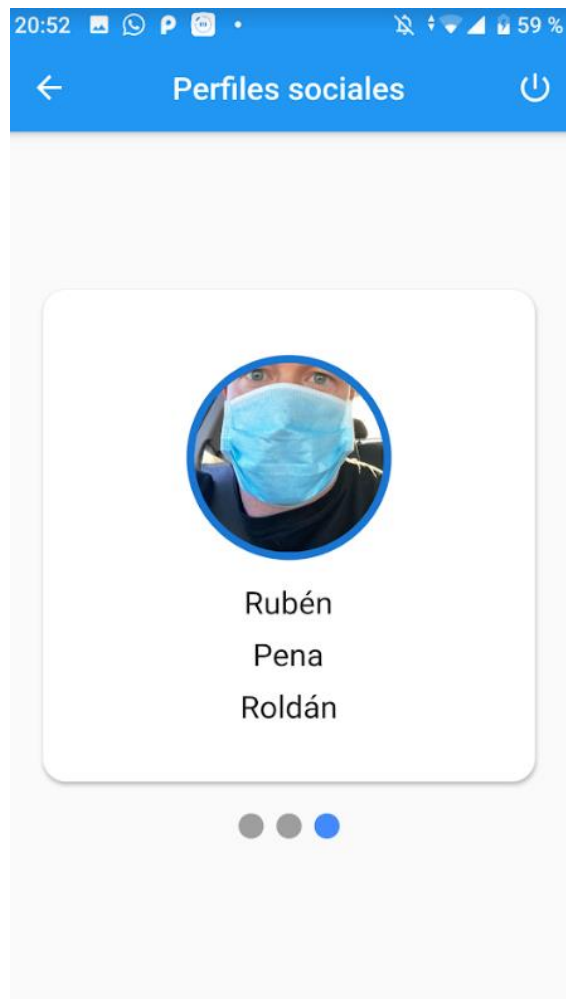
Cuando iniciemos sesión, nos saldrá un listado de las escuelas que tenemos disponibles en nuestra cuenta de usuario. Podemos seleccionar una de ellas o por el contrario, inscribirnos en una escuela pulsando en el botón de la esquina inferior derecha.



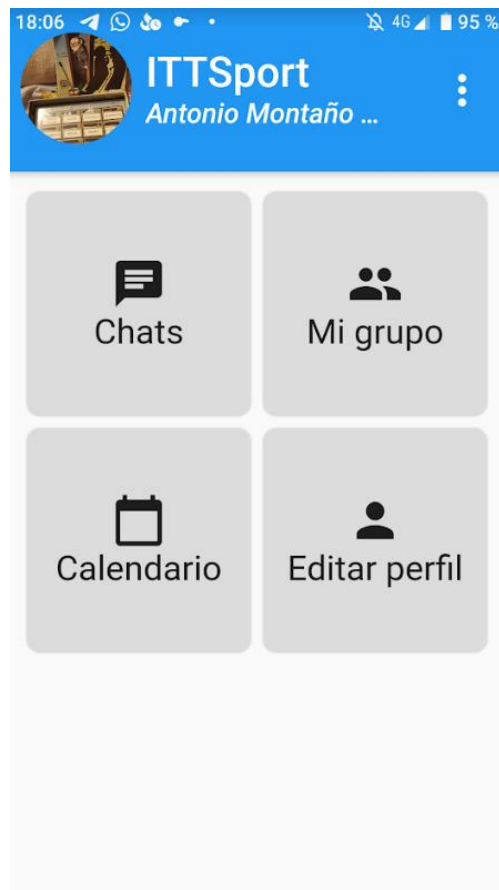
Si nos inscribimos en una escuela, nos llevará a una lista de escuelas donde están todas las escuelas disponibles en la aplicación, pudiendo filtrar en la lista. Una vez encontremos la escuela que queremos, pulsamos y nos llevará a un formulario de inscripción, el cual debemos rellenar. Una vez relleno, debemos esperar a que el director acepte la petición.

Cuando seleccionemos la escuela en la vista de "mis escuelas", encontraremos un listado con los perfiles que tiene nuestro usuario en dicha escuela.

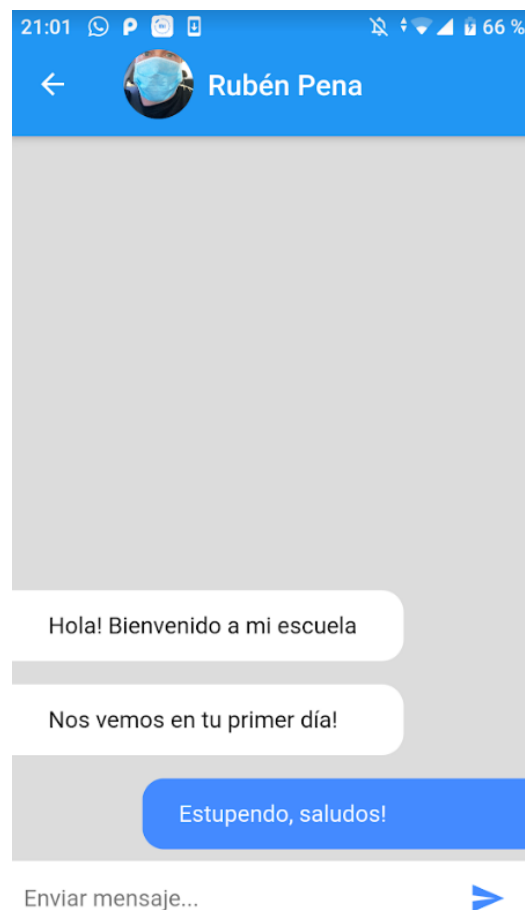
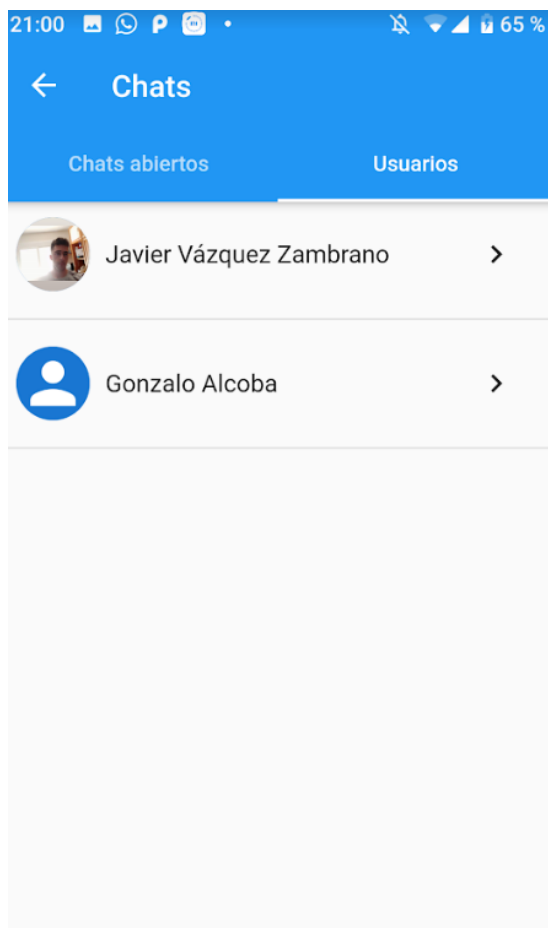




Cuando seleccionemos nuestro perfil social, entraremos en el menú principal, distinto para cada rol dentro de la escuela.



En el caso de los *chats*, podemos entrar y *chatear* con cualquier persona que esté inscrita en la misma escuela. Simplemente debemos ir a la pestaña de usuarios dentro de la ventana principal de *chats* y seleccionarlo.

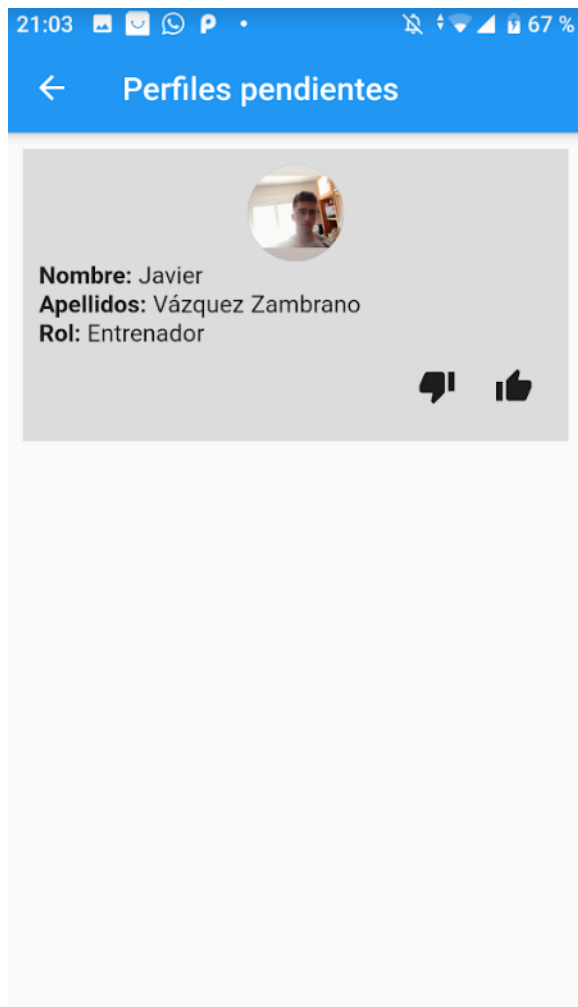


En "mi grupo" podemos acceder a la información del grupo de entrenamiento al que estamos asignados, con el horario, el entrenador y los alumnos asignados al mismo.

En el calendario, podemos ver los eventos en los que está envuelto nuestro grupo de la escuela, que consistirán en posibles entrenamientos especiales, torneos, partidos amistosos, etc.

También podremos editar nuestro perfil social en cualquier momento accediendo al botón para dicha función.

Si sois directores, podréis aceptar y rechazar los perfiles sociales pendientes en el botón que nos llevará a la lista para dicha funcionalidad.



También podréis crear nuevos grupos o gestionar los que tengáis ya creados en el apartado "mis grupos".

Esperamos que disfrutéis de la aplicación, y para cualquier consulta, no dudéis en contactar con nosotros. ¡Muchísimas gracias, y buen entrenamiento!

---

## 4. Conclusiones

---

### 4.1. Problemas encontrados y soluciones aportadas

A continuación, enumeramos los problemas destacables a los que nos hemos tenido que enfrentar, y las soluciones que hemos aportado:

- El problema más importante y destacado sin duda es el problema del que hablamos en la introducción, que es el problema con las versiones nativas de *Android* e *iOS*. Sin duda es algo a lo que nos hemos tenido que enfrentar con, por desgracia, unas cuantas horas ya gastadas en el desarrollo del proyecto. Por suerte, con la implementación del *framework Flutter*, todo se ha quedado ahí y hemos podido reaccionar a tiempo para cumplir con el desarrollo del proyecto.
- Otro de los problemas a los que nos hemos enfrentado ha sido la pandemia producida por el Covid-19. Sin duda, la pandemia ha trastocado las vidas de todos, a algunos en mayor manera, y en otros por suerte en menos. Nosotros somos privilegiados por poder trabajar y estudiar de forma telemática sin mucho problema, pero hay otros que no. Ese es el caso de las escuelas deportivas, nuestros futuros clientes. Por culpa de esto no hemos estado podido estar tan involucrados en la búsqueda de clientes para nuestro proyecto, a excepción de nuestro antiguo entrenador, que siempre ha estado apoyándonos cuando hemos necesitado ayuda con el desarrollo. Además, hemos alcanzado un acuerdo con él para promover y extender el uso de la aplicación a otras escuelas deportivas, algo que desde aquí queremos agradecerle.
- Algo que hemos tenido que resolver ha sido enfrentarnos a *Apple*, ya que, aunque *Flutter* compile tanto para *Android* como para *iOS*, *Apple* obliga a poseer una serie de certificados personales y de desarrollo que obligan a comprar la licencia de desarrollador de *Apple*. Esto nos ha perjudicado a la hora de entregar la aplicación al *Product Owner*, pues no poseemos ni *Mac* ni *iPhone* para poder probar en condiciones la aplicación y hemos tenido que depender de él para este hecho.

### 4.2. Cumplimiento de los objetivos

Consideramos que el desarrollo de nuestro *TFG* ha sido una gran experiencia que nos ha servido para introducirnos al enorme mundo lleno de posibilidades que es el desarrollo móvil, o la movilidad. Teníamos muchas ganas de adentrarnos en este mundo, por lo que ha sido algo muy motivador, además de una gran experiencia a nivel profesional, pues hemos desarrollado nuestra primera aplicación completamente desde "cero".

A nivel de cumplimiento de objetivos, creemos que hemos alcanzado los objetivos generales que nos propusimos al principio del proyecto, aunque es verdad que, si no nos hubiéramos encontrado con el problema del desarrollo de las aplicaciones nativas, probablemente podríamos haber ampliado un poco más la funcionalidad, o

haber pulido algo mejor el diseño, pero en general estamos muy satisfechos por nuestra reacción ante este problema y el nivel de acabado de la aplicación.

También hemos alcanzado un buen nivel de acabado como para publicar la aplicación una vez esté finalizada la implementación de la monetización, cuyo modelo consistirá en un sistema de suscripción mensual por cada escuela inscrita en el sistema. Queremos ofrecer la aplicación a la mayor cantidad de escuelas posible, para poder maximizar nuestros ingresos con una buena base de usuarios.

En cuanto al aprendizaje, también creemos que hemos aprendido muchísimo durante este proyecto, ya que hemos descubierto nuevas tecnologías como *Android* y *Flutter*, siendo este último el que mejor hemos aprendido con diferencia debido a la cantidad de horas dedicadas y a la pequeña curva de aprendizaje que tiene, y sin obviar otras tecnologías como *Codemagic* para la gestión de la integración continua. También hemos aprendido a documentar mejor los desarrollos del proyecto, a llevar mejor la gestión de las tareas y del tiempo desde el principio del proyecto hasta el final, siendo al principio un desastre absoluto, y acabando con una muy buena gestión.

### 4.3. Futuras posibles implementaciones

Esta aplicación tiene las herramientas necesarias para la gestión eficiente de una escuela deportiva. Sin embargo, por suerte, se puede ampliar de multitud de formas que queremos incluir en un futuro. Enumeramos algunas posibilidades, siendo una de ellas muy importante a muy corto plazo:

- Queremos implementar en cuanto tengamos algo de *feedback* de nuestro entrenador, que va a usar la aplicación a partir de septiembre, es decir, cuando comience la nueva temporada de tenis, un sistema de monetización para poder obtener beneficios por el uso de nuestra aplicación. No lo hemos considerado "crucial" hasta acabar el proyecto por dos motivos: el primero ha sido la pandemia, que nos ha impedido contactar con clientes de una manera eficiente; y el segundo, que nuestro único cliente al principio va a ser nuestro entrenador, y como agradecimiento por su ayuda y apoyo, queremos que el uso de la aplicación para él sea gratuito. Por estos motivos, no hemos implementado aún esto, y es algo que queremos hacer ahora que se va a volver a la "normalidad" y el deporte vuelve a ser cada vez más "como antes".
- Nos gustaría implementar notificaciones a los usuarios cuando les lleguen nuevos mensajes de *chat*, o se acepte algún perfil social. Básicamente, notificaciones *push* que les avise de algún evento de su interés.
- También queremos introducir algún sistema que notifique a los usuarios cuando su entrenamiento está en riesgo por culpa del clima. Por ejemplo, introducir una *API* que detecte cuando hay un 50% de posibilidades de lluvia, y que los notifique con un aviso que les haga estar atentos a algún cambio en su escuela.
- Otra muy buena mejora sería introducir localizaciones mediante *Google Maps* para ubicar con mayor facilidad las escuelas deportivas del sistema.

En definitiva, hay múltiples funcionalidades que se pueden introducir para ampliar la aplicación, así que siempre estaremos dispuestos a escuchar el *feedback* de los usuarios e incluir lo que sea necesario.

## 4.4. Bibliografía

Comparación entre *React Native*, *Ionic* y *Flutter*