UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de aplicaciones móviles utilizando React Native

TRABAJO FIN DE GRADO

GRADO EN INGENIERÍA TELEMÁTICA

Autor: Sergio Carralero Nuño

Director: Daniel Pérez Berenguer

Codirector: Mathieu Kessler

Cartagena, Septiembre de 2021

Índice

Capítulo I: Introducción

La Fórmula 1 es uno de los deportes más importantes a nivel mundial, llegando a mover grandes masas de aficionados y profesionales. Se trata de la principal competición de automovilismo internacional de los deportes de motor, siendo el más prestigioso y popular del mundo.

Este campeonato es dirigido por la Federación Internacional del Automóvil (FIA), y fue creado en 1950, aunque las carreras automovilísticas comenzaron en Francia bastante antes, en torno al año 1894.

Este primer campeonato dio comienzo en Reino Unido, más concretamente en el circuito de Silverstone, el cual continúa en el calendario de carreras a día de hoy.

Desde entonces, de pasar a ser un deporte conocido por unos pocos fue creciendo hasta llegar al ámbito internacional.

Dentro del campeonato, se pueden distinguir los circuitos más famosos dentro del mundo de la Fórmula 1, los cuales siguen vigentes en el calendario actual. Algunos ejemplos son Silverstone (Reino Unido), Monza (Italia), Hungaroring (Hungría), Mónaco (Principado de Mónaco), Suzuka (Japón) o Interlagos (Brasil).

A día de hoy, el campeonato consta de 23 fechas distribuidas en diferentes países de todo el mundo, en las cuales se disputan entrenamientos, clasificaciones y carreras con la finalidad de obtener al piloto y la escudería ganadores.

1. Objetivos y motivación

La experimentación en arquitectura de servicios web, donde un conjunto de especificaciones tecnológicas basadas en estándares abiertos como HTTP, URL, XML o SOAP, nos proporcionan un modelo de interacción de sistema a sistema, estando íntimamente relacionados con las aplicaciones web.

Gracias a esto, podemos tener una aplicación con un back-end independiente que puede acoplarse a cualquier parte de front-end sin necesidad de cambios, pudiéndose utilizar desde tecnologías de programación web como Angular o ReactJS hasta desarrollo con tecnologías móviles como React Native o Flutter.

El uso de aplicaciones web con front-end y back-end por separado tiene una serie de ventajas que hacen que ésta sea la elección predefinida para la mayoría de desarrollos de tecnologías web y móvil. Entre estas ventajas, destacan las siguientes:

➔ **La rapidez e interactividad fluida que estas aplicaciones consiguen gracias a este desacople.**

➔ **Un back-end que puede ser utilizado por otras aplicaciones o interfaces.**

➔ **El intercambio de información es extremadamente rápido, debido a que los datos se reciben en formato JSON.**

Esto se puede llevar al lado de las tecnologías móviles, de tal forma que tener este tipo de arquitectura nos brinda las ventajas que hemos mencionado, como puede verse en la figura 1.

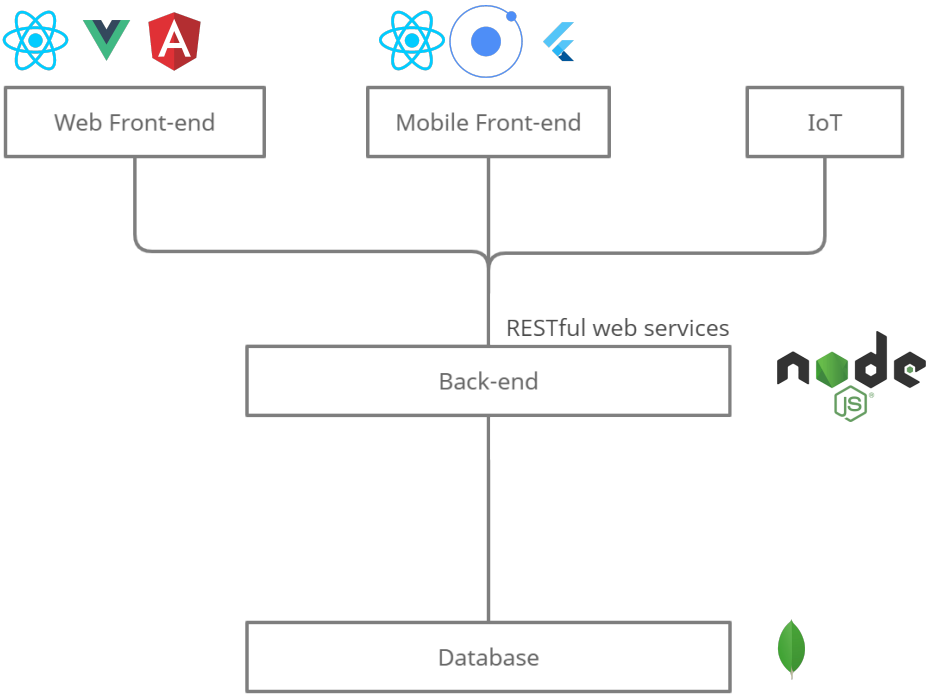


Figura 1. Estructura tradicional de servicios de front-end y back-end desacoplados.
Fuente: Elaboración propia

En el caso de las tecnologías elegidas en este proyecto, las cuales veremos más adelante en profundidad, nuestra arquitectura sufre un cambio que nos facilita el desarrollo del back-end y la base de datos (ver figura 2). Al haber decidido utilizar Firebase (conocido como BaaS, Back end as a Service), esta tecnología nos permite tener a nuestra disposición todo lo necesario para el back-end y la base de datos desde el propio Firebase sin necesidad de la utilización de otras tecnologías como NodeJS para la parte de back-end o como MongoDB para la parte de la base de datos.

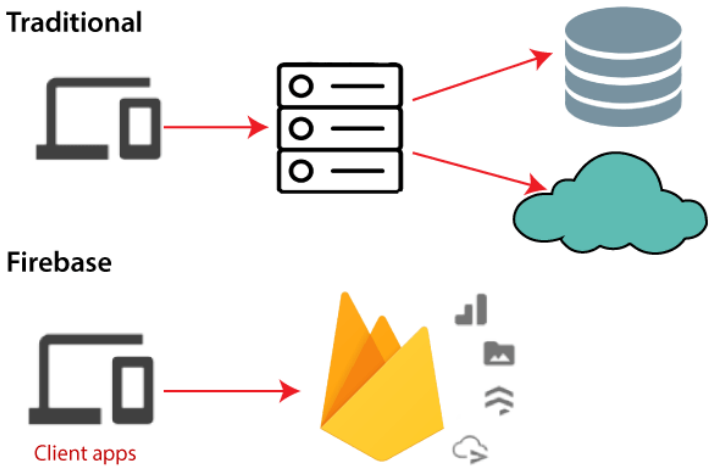


Figura 2. Reestructuración de back-end y DB con el uso de Firebase.
Fuente: javadesde0.com

De esta forma, simplificamos nuestra arquitectura y apostamos por una computación total en la nube, es decir, la disponibilidad de los recursos sin gestión activa por parte del usuario. Esto es una tendencia creciente que cada vez más empresas están aplicando.

Esta computación en la nube tiene una serie de ventajas, las cuales son:

➔ **Integración de servicios de red.**

➔ **Servicios a nivel mundial.**

➔ **Prescindir de instalaciones de dispositivos físicos.**

➔ **Uso eficiente de energía.**

➔ **Portabilidad de la información.**

➔ **Actualizaciones automáticas e implementación más rápida y con menos riesgos.**

Centrándonos en esto, se ha decidido utilizar React Native en el uso de dispositivos multiplataforma en conjunto a un back-end totalmente independiente realizado con Firebase que nos permitirá expandir la aplicación sin límite.

El contexto elegido para experimentar con estas tecnologías es el Campeonato de Fórmula 1, que cada vez es seguido por más gente, y las plataformas para estar informado son bastante escasas de datos o caras.

Así mismo, estas aplicaciones suelen mostrar un diseño bastante desactualizado y suelen ser de difícil entendimiento.

Teniendo en cuenta estos motivos, esta aplicación nace de la necesidad de tener una aplicación de Fórmula 1 gratuita que muestre toda información necesaria para aquellos entusiastas de este deporte sin tener que salir de ella, y que sea sencilla e intuitiva para que todo el mundo pueda usarla sin mayor problema, desde gente joven hasta personas más mayores.

Además, con este proyecto se pretende dar una mayor visibilidad a este deporte y llegar a cualquier parte del mundo con ella, generando un feedback que permita realizar las modificaciones necesarias para tener la aplicación soñada de la Fórmula 1.

2. Organización del proyecto

Este proyecto se ha organizado en cuatro grandes bloques, que se corresponden a las pestañas que contiene la aplicación.

En cada bloque se ha hecho una planificación de los elementos necesarios, así como del peso en la propia aplicación.

Para realizar esta organización se ha hecho uso de una serie de metodologías ágiles las cuales mencionaremos más adelante, y que han facilitado bastante el desarrollo de la aplicación y su testeo.

3. Requerimientos

Esta aplicación tiene una serie de requerimientos mínimos y necesarios para que sea totalmente disfrutable tanto desde un dispositivo iOS como desde un dispositivo Android.

Deberá ser intuitiva y sencilla, con una interfaz muy fluida que permita disfrutar de la experiencia de usuario. Además, debe ser escalable para facilitar el futuro desarrollo continuado de la aplicación o la incorporación de nuevas mejoras o nuevos deportes.

Debido a que la aplicación maneja datos de usuarios, debe ser segura y transparente, y proporcionar los sistemas necesarios de encriptación para el tratamiento de dichos datos.

La aplicación deberá tener las siguientes funcionalidades:

➔ **Carreras del campeonato**
Los usuarios podrán revisar las carreras del campeonato, con la información de cada circuito y ver cuándo serán disputadas. Además, si ya han sido disputadas, se podrán consultar los resultados.

➔ **Puntuaciones actualizadas**
Los usuarios tendrán acceso a las puntuaciones actualizadas del campeonato de pilotos y constructores, así como un desglose de las últimas 5 carreras en las puntuaciones de los pilotos.

➔ **Buscador de pilotos**
Se podrá acceder a un buscador que tras introducir el apellido del piloto mostrará el resultado de todos los campeonatos disputados por éste sin límite de años.

➔ **Información extra**
Los usuarios que deseen información algo más avanzada podrán consultar tanto la información de cada piloto, así como de cada escudería y los tipos de neumáticos de la temporada actual.

➔ **Login y registro de usuarios**
Se habilitará un apartado de perfil en el cual los usuarios tendrán la opción de registrarse para acceder a la activación de las notificaciones. Los usuarios que no se quieran registrar seguirán con pleno acceso a la aplicación, simplemente no podrán activar estos avisos.

➔ **Notificaciones**
Los usuarios que se registren podrán recibir alertas antes de que una carrera vaya a ser disputada.

4. Tecnologías elegidas

En base a estos requerimientos, se ha decidido desarrollar la aplicación utilizando React Native con Firebase.

Para tomar esta decisión, se han tenido en cuenta otras opciones de desarrollo multiplataforma como Flutter o NativeScript. Teniendo en cuenta estas tecnologías, se ha decidido desarrollar en React Native debido a una serie de temas cruciales.

React Native cuenta con una comunidad mayor que las tecnologías anteriormente mencionadas. Además, tiene más tiempo en mercado que Flutter y está bastante más actualizada que NativeScript.

Esta tecnología a día de hoy es la más usada para el desarrollo móvil multiplataforma, seguida de cerca por Flutter.

Además, la parte de backend de la aplicación se ha decidido desarrollar con Firebase, debido a su gran implicación y adaptación en tecnologías móviles, junto con una API REST de la que obtendremos los datos de Fórmula 1.

Figura 3. Logotipos de React Native y Firebase
Fuente: Medium
(https://medium.com/5bayt/authentication-with-google-and-firebase-in-react-native-e00c46fd048c)

Esto implica que el lenguaje en el que se va a desarrollar toda la aplicación será JavaScript.

A continuación, se describen las tecnologías mencionadas:

➔ **React Native:** Se trata de un framework JavaScript utilizado para crear aplicaciones móviles nativas multiplataforma, y se basa en ReactJS. Es una de las tecnologías móviles más utilizadas para desarrollar el front-end, ya que es respaldada por una gran comunidad. Este framework ha sido desarrollado por Facebook, es gratuito y de código abierto. Además, entre sus ventajas podemos destacar la facilidad de aprendizaje, ya que su lenguaje es JavaScript.

➔ **Firebase:** Se trata de una plataforma en la nube que permite el desarrollo de aplicaciones web y móvil, con una gran implicación en esta última. Gracias a que permite crear una base de datos de manera sencilla y rápida para cualquier plataforma, esta herramienta está en auge. Está desarrollado por Google y se puede utilizar de forma gratuita, permitiendo junto con la centralización de sus servicios la creación de una base de datos robusta. Proporciona bases de datos no relacionales (NoSQL).

Esta tecnología es denominada BaaS, esto es, Backend as a Service, y nos permite prescindir de la utilización de una API personalizada (backendless). Este es un concepto novedoso impulsado por Google para que la tendencia cada vez sea más la utilización de una sola tecnología para sustituir la parte de back-end y base de datos (ver figura 2).

Algunos de los grandes beneficios del uso de BaaS son el sistema de almacenamiento en la nube que ofrece, las funcionalidades realtime, la ejecución de código back-end, y librerías que permiten una sencilla utilización de este servicio.

Cabe destacar que Firebase no sólo ofrece una base de datos, si no que también da la opción de utilizar toda clase de servicios de analíticas, autenticación, notificaciones push o in-app, además de las ya mencionadas bases de datos y realtime.

➔ **API REST:** Se trata de una interfaz de programación de aplicaciones que permite la interacción con servicios web de RESTful. A su vez, un servicio web es una tecnología que utiliza un conjunto de protocolos para el intercambio de datos entre aplicaciones. Si ahondamos en los servicios web RESTful, son unos servicios basados en la arquitectura REST que se basa en recursos almacenados en un servidor, los cuales son solicitados por el cliente.

Existen una serie de operaciones que se pueden realizar con este tipo de servicios, como pueden ser la obtención de recursos (GET), la creación de recursos (POST), la actualización de recursos (PUT) o la eliminación de éstos (DELETE).

5. Herramientas de desarrollo

Para la realización de este proyecto y debido a la multitud de funcionalidades implementadas, se ha decidido utilizar las siguientes herramientas de desarrollo:

➔ **Git y Github:** Es un sistema de control de versiones que permite tener una copia del código fuente en la nube. Su propósito es llevar un registro de los cambios en los ficheros de un proyecto para coordinar el trabajo.

➔ **Trello:** Es una aplicación para gestión de proyectos que permite organizar las tareas a realizar. Permite subdividir las tareas a realizar en cada parte del proyecto.

➔ **Postman:** Es una plataforma para consumir y construir APIs. Permite ver el JSON que se va a recibir de la llamada a la API y así poder manejar los datos de una mejor forma.

Capítulo II: Estructura de la app

La estructuración del proyecto es una de las partes más importantes, ya que si se realiza de la forma adecuada permite una gran escalabilidad de cara al futuro de la aplicación.

Como se ha mencionado con anterioridad, disponemos de cuatro grandes bloques que dividiremos entre sí. Estos bloques se corresponden con las funcionalidades de cada pantalla.

Además, existen una serie de sub bloques que contienen las funciones compartidas por toda la aplicación.

1. Estructura del proyecto

En la estructura de la aplicación podemos diferenciar las siguientes carpetas:

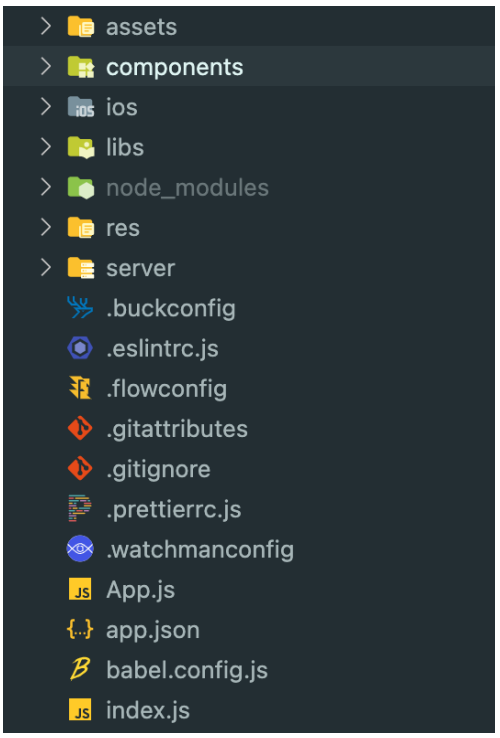


Figura 4. Organización de carpetas.
Patrón modular: https://es.reactjs.org/docs/faq-structure.html

Esta organización se ha realizado de esta manera siguiendo el patrón modular, también denominado como estructuración por funcionalidad o rutas, en el cual tendremos en nuestra carpeta *'Components'* las distintas pantallas divididas por función. Aun así, puede utilizarse cualquier otro debido a que React Native no especifica una forma correcta para organizar los ficheros, como sí que ocurre con Angular.

A continuación, analizaremos los directorios y sus contenidos:

➔ **Assets:** dentro de este sub bloque encontramos las imágenes y recursos utilizados, como pueden ser el icono de la aplicación, o las imágenes incluidas en cada carrera.

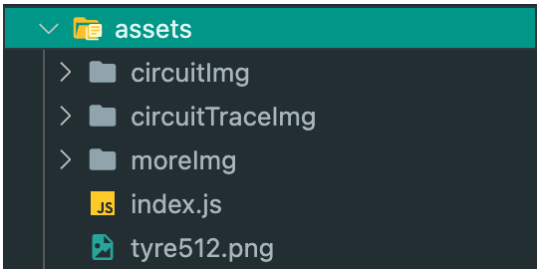


Figura 5. Contenido carpeta 'assets'

➔ **Components:** dentro de esta carpeta se encuentran los cuatro grandes bloques de los que hemos hablado. Contiene los archivos necesarios para crear cada pantalla, divididos por secciones.

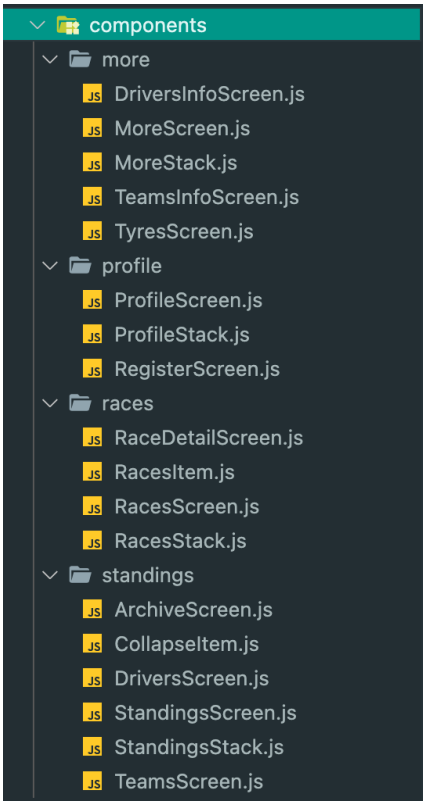


Figura 6. Contenido carpeta 'components'

➔ **Libs:** contiene los archivos comunes de la aplicación. Estos archivos contienen un número considerable de líneas de código.

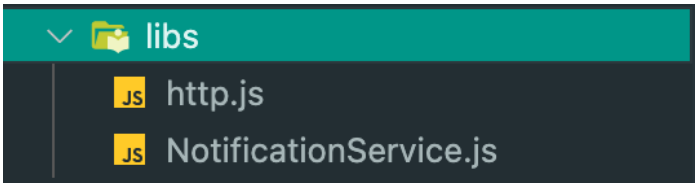


Figura 7. Contenido carpeta 'libs'

➔ **Res:** contiene un único fichero común para toda la aplicación, pero que a diferencia de la carpeta anterior, se trata de un recurso de estilo. Éste contiene los colores personalizados utilizados en el proyecto.

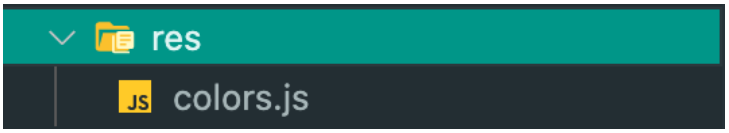


Figura 8. Contenido carpeta 'res'

➔ **Server:** en esta carpeta se encuentra el fichero que se encarga de la parte de backend con Firebase, donde se encuentran las funciones necesarias para realizar el login y registro.

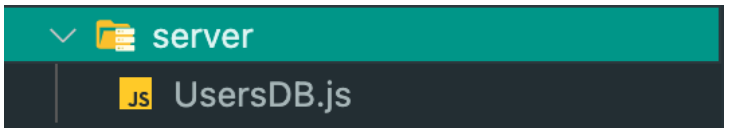


Figura 9. Contenido carpeta 'server'

2. Pantallas y sus funciones

Nuestra aplicación se compone de un total de 11 pantallas, organizadas en 4 pestañas:

Figura 10. Barra de navegación

➔ **Races:** pestaña principal que se muestra al entrar en la aplicación. Contiene un listado de los grandes premios del campeonato. Se divide en las siguientes pantallas:

- ❖ **Races Stack:** pantalla que se encarga de la navegación de esta pestaña, permitiendo que al clickar en una carrera, se pueda redirigir al usuario a los detalles de ésta.

- ❖ **Races Screen:** pantalla que se encarga de mostrar la lista scrollable de carreras. Esta lista realiza un scroll automático hasta la carrera que se vaya a disputar o se esté disputando, de forma que siempre esté visible para el usuario sin necesidad de buscarla en el listado.

- **Races Item:** pantalla que se encarga de mostrar lo que contiene cada gran premio de la lista mostrada en ‘Races Screen’, así como de la redirección al gran premio seleccionado. Éstos se muestran en forma de tarjetas clickables con una opacidad diferente dependiendo de si el gran premio ha sido disputado o no.

- **Race Detail Screen:** pantalla que se encarga de mostrar la información de la carrera seleccionada, con los datos del gran premio y los horarios. En caso de que un gran premio ya haya sido disputado, se muestran los resultados de la clasificación y la carrera en forma de desplegable.

Figura 11. Pantalla principal

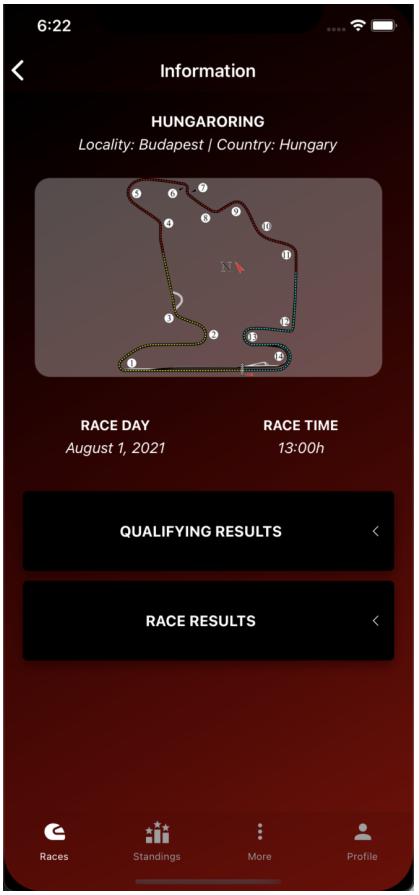


Figura 12. Detalle de gran premio

➔ **Standings:** pestaña que contiene la puntuación de los pilotos y equipos en cualquier punto del campeonato, además de un sistema de búsqueda por piloto que muestra sus resultados en todos los años de Fórmula 1. Se divide en las siguientes pantallas:

- ❖ **Standings Stack:** pantalla que se encarga de la navegación de esta pestaña, permitiendo la navegación de la pantalla principal de resultados al buscador de pilotos.

- ❖ **Standings Screen:** pantalla que se encarga de la navegación entre las pestañas de 'DRIVERS' y 'TEAMS', mostrando un resultado u otro dependiendo del apartado en el que estemos.

- ❖ **Drivers Screen:** pantalla que se encarga de mostrar la lista de pilotos mediante elementos expandibles. Esta pantalla se encargará de enviar dicha información al fichero 'Collapsible Item'.

- ❖ **Teams Screen:** pantalla que se encarga de mostrar la lista de equipos con el mismo estilo de los elementos expandibles. Esta pantalla se encargará de enviar dicha información al fichero 'Collapsible Item', pero en este caso la lista no será desplegable.

- ❖ **Collapsible Item:** elemento encargado de mostrar la lista de pilotos como un expandible, de tal manera que si clickamos sobre un piloto podamos ver la puntuación y posición obtenida en los últimos 5 grandes premios. Si el expandible no es desplegado, se mostrará la puntuación total del piloto o equipo hasta el momento.

- ❖ **Archive Screen:** pantalla encargada de las búsquedas de resultados por pilotos. Si el usuario busca el apellido de un piloto (este apellido es reconocido por la FIA como el id de piloto), se mostrarán todos los resultados obtenidos en cada una de las temporadas en las que haya participado en la Fórmula 1, mostrando así su puntuación final, posición final, nombre completo y escudería a la que pertenecía en ese campeonato.

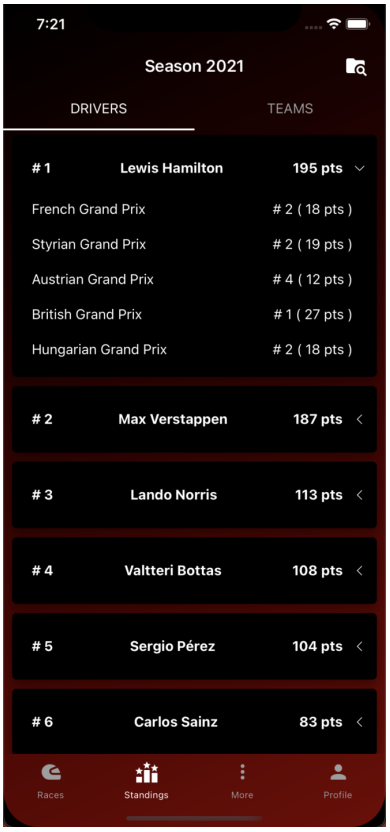


Figura 13. Puntuación de pilotos

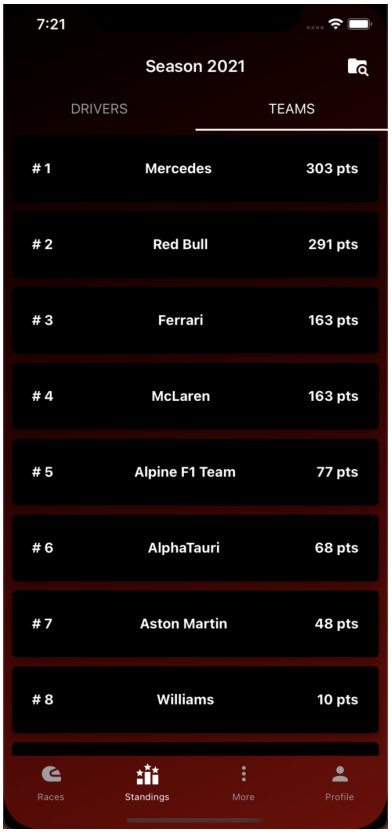


Figura 14. Puntuación de equipos

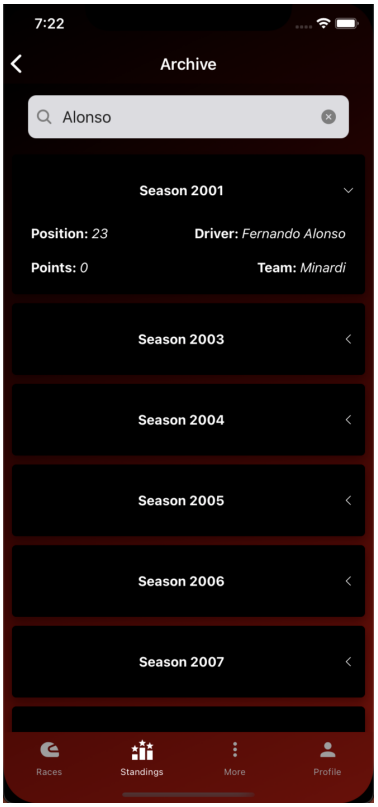


Figura 15. Buscador de pilotos

➔ **More:** pestaña que contiene la información extra de la aplicación, esto es, la información adicional necesaria de los pilotos, equipos y neumáticos. Se compone de las siguientes pantallas:

- ❖ **More Stack:** pantalla encargada de la navegación de esta pestaña, permitiendo redirigir a las pantallas de información de pilotos y equipos, e información de neumáticos.

- ❖ **More Screen:** pantalla que contiene una serie de tarjetas verticales con imagen de fondo que dividen las pantallas contenidas en esta pestaña. Se encarga de la navegación entre éstas.

- ❖ **Drivers Info Screen:** pantalla que muestra al usuario la información adicional de cada piloto de la temporada actual en formato de tarjetas horizontales, ordenado alfabéticamente por apellido.

- ❖ **Teams Info Screen:** pantalla que muestra al usuario la información adicional de cada equipo de la temporada actual en formato de tarjetas horizontales, ordenado alfabéticamente por nombre de escudería.

- ❖ **Tyres Screen:** pantalla que contiene todos los compuestos de neumáticos usados en la temporada actual, con una breve descripción de cuándo se usa cada uno y cómo están catalogados.

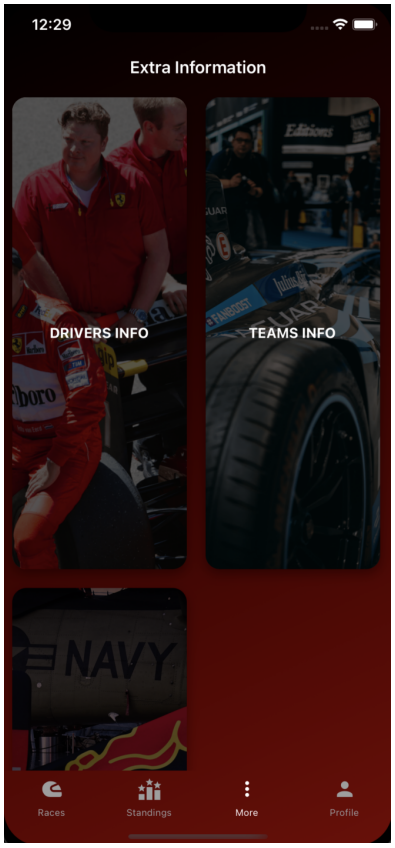


Figura 16. Pestaña More

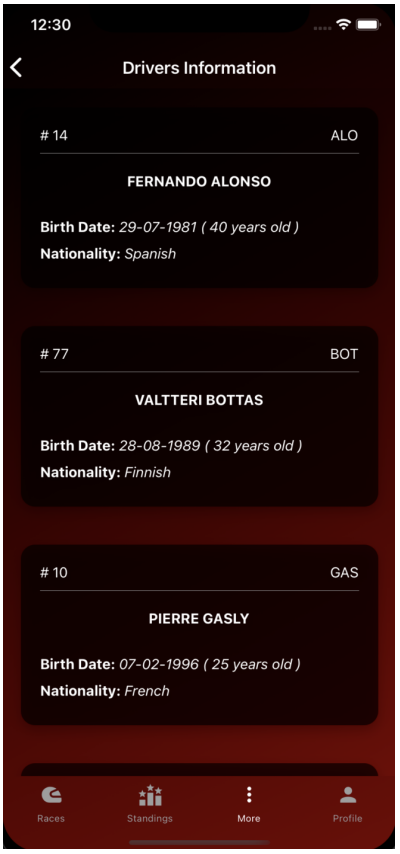


Figura 17. Información adicional de pilotos

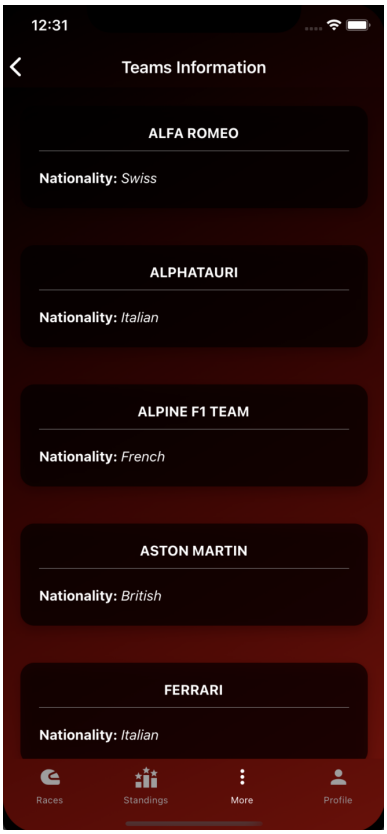


Figura 18. Información adicional de equipos

Figura 19. Información de neumáticos

➔ **Profile:** pestaña que contiene el login y registro del usuario, así como las opciones de perfil una vez se ha iniciado sesión. Se compone de las siguientes pantallas:

- ❖ **Profile Stack:** pantalla encargada de la navegación de esta pestaña, permitiendo cambiar entre las pantallas de login y registro y redirigir a los ajustes del usuario una vez se ha iniciado sesión con éxito.

- ❖ **Profile Screen:** esta pantalla muestra tanto la pantalla de ajustes de usuario como el login. Si no existe usuario logueado, se mostrará la pantalla de login, desde la cual se podrá ir a la pantalla de registro si el usuario no dispone de cuenta en la aplicación. En el caso de que el usuario ya esté logueado, se muestra la pantalla de configuración de notificaciones.

- ❖ **Register Screen:** pantalla encargada de mostrar el formulario de registro en caso de que el usuario no disponga de cuenta o quiera tener otra además de la suya. Al registrarse, será redirigido a la pantalla de login.

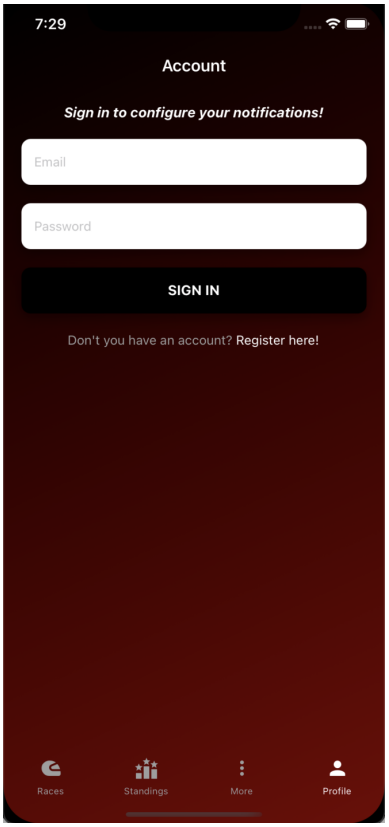


Figura 20. Pantalla de Login

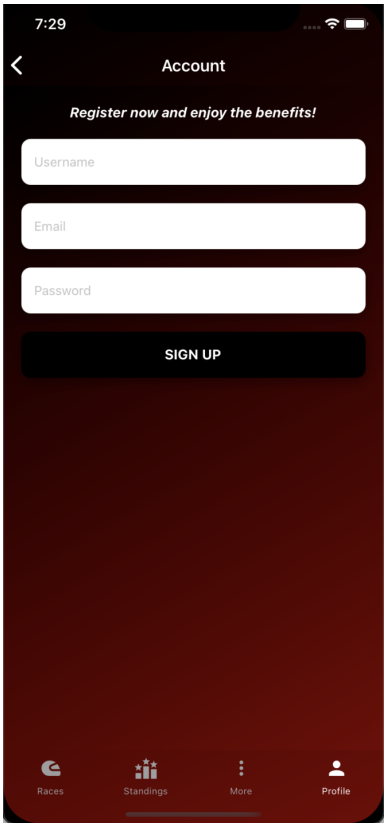


Figura 21. Pantalla de registro

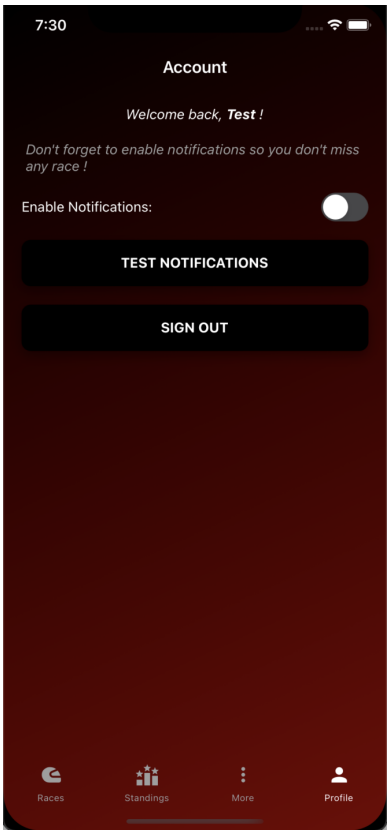


Figura 22. Pantalla de ajustes

Dependiendo del tipo de contenido que queramos añadir en cada sección, usaremos clases para renderizar el contenido de la pantalla en la aplicación (como pueden ser las pantallas denominadas ‘Screen’) o variables para los ficheros que necesitan retornar la navegación de las pantallas o ciertas funciones (como pueden ser las pantallas denominadas ‘Stack’).

3. Diseño

Para realizar el diseño de la aplicación se ha utilizado Figma, una aplicación que permite diseñar UI/UX y prototipar nuestra aplicación.
Además, se ha tenido en cuenta el color emblemático de la Fórmula 1: el rojo.

Dado que actualmente el modo oscuro está presente en casi todas las aplicaciones y es preferido por bastantes usuarios, se ha decidido hacer un degradado de negro a rojo en diagonal para darle ese dinamismo y sensación de ‘velocidad’ a la aplicación.

Este degradado se ha realizado ajustando los colores a mano, hasta encontrar una combinación que cuadrase con la idea principal del proyecto.

Con este fondo, los textos son de color blanco y los textos con menos énfasis de un gris claro personalizado (al igual que el color rojo utilizado).

```
1   const colors = {      Sergio Carralero Nuño, 6 months ago • Main colors file added
2     white: '#ffffff',
3     black: '#000000',
4     lightGrey: '#9e9e9e',
5     red: '#aa0000',
6   };
7
8   export default colors;
```

Figura 23. Colores definidos en la aplicación

Para la selección de iconos se ha utilizado la librería '*react-native-vector-icons*', sobre la cual hablaremos más adelante. Se han utilizado los estilos de iconos de Material Design.

En cuanto a la tipografía, se ha elegido la letra predefinida en dispositivos iOS y Android, de tal forma que es el propio sistema operativo el que lo define.

Así mismo, se ha escogido un tamaño de letra de 15 puntos para los textos normales, y de 18 puntos para los textos que necesitan ser resaltados, como títulos o secciones.

4. Splash Screen e icono

A la hora de presentar la aplicación, es importante personalizar tanto el icono como la pantalla de carga de ésta, también conocida como Splash Screen.

Para la personalización del icono, deberemos seguir una serie de pasos:

➔ **Obtener los distintos tamaños de iconos para iOS/Android:** para obtener los tamaños de iconos utilizaremos una herramienta gratuita llamada *'App Icon Generator'*, la cual nos devuelve todos los tamaños de imagen necesarios cargando el icono elegido para nuestra aplicación.

➔ **Agregar las carpetas generadas de iconos de iOS/Android al proyecto:** dependiendo del sistema operativo, tendremos que añadir las carpetas de iconos generadas a los directorios *'Images.xcassets'* para iOS y *'src/main/res'* para Android.

En nuestro proyecto, hemos elegido el icono de una rueda para simbolizar la velocidad de los coches de Fórmula 1.

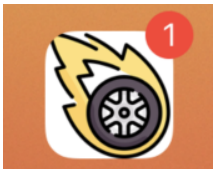


Figura 24. Icono de la aplicación

Para la realización de la pantalla de carga deberemos seguir los siguientes pasos:

➔ **Añadir el icono a utilizar en las mismas carpetas en las que hemos añadido el icono de iOS/Android.**

➔ **Caso iOS:**

- ❖ Añadir la imagen al fichero *'LaunchScreen.storyboard'* y colocarla a nuestro gusto

➔ **Caso Android:**

- ❖ Crear el fichero *'background_splash.xml'* al directorio *'android/app/src/main/res/drawable'* y añadir el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<layer-list xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:drawable="@color/blue"/>

    <item
        android:width="200dp"
        android:height="200dp"
        android:drawable="@mipmap/icon"
        android:gravity="center" />

</layer-list>
```

Figura 25. Carga de Splash Screen en Android

- Al terminar de añadir este código, simplemente deberemos añadir la llamada de este fichero al archivo *'AndroidManifest.xml'*.

- Para terminar, crearemos el fichero *'SplashActivity.java'* en el cual añadiremos el siguiente código:

```java
package com.splashexample; // make sure this is your package name

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

public class SplashActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
        finish();
    }
}
```

Figura 26. Configuración archivo SplashActivity en Android

Como resultado final obtenemos la siguiente pantalla de carga:

Figura 27. Splash Screen finalizada

5. Recursos utilizados

En la aplicación podemos encontrar una serie de imágenes dependiendo de la pantalla en la que nos encontremos (obtenidas desde unsplash, sin copyright). Estas imágenes están divididas en tres carpetas dentro de 'assets':

➔ **Carpeta 'circuitImg':** contiene las imágenes de las tarjetas horizontales de la pantalla 'Races'.

➔ **Carpeta 'circuitTraceImg':** contiene las imágenes de los trazados de la pantalla 'Race Detail'.

➔ **Carpeta 'moreImg':** contiene las imágenes de las tarjetas verticales de la pantalla 'More'.

Además, en la carpeta raíz de ‘assets’ podemos encontrar el icono de la aplicación y un fichero llamado ‘*index.js*’. En este fichero se encuentra una variable encargada de realizar los ‘imports’ necesarios para llamar a cada imagen. A estas importaciones se les asigna una clave para poder referirse a ellas de manera individualizada, sin necesidad de cargar todas las imágenes.

```
You, 4 days ago | 1 author (You)
1  const images = {      You, 4 days ago • Adding images to race screen
2    driversScreen: require('./moreImg/drivers.png'),
3    teamsScreen: require('./moreImg/teams.png'),
4    tyresScreen: require('./moreImg/tyre.png'),
5    albert_park: require('./circuitImg/albert_park.png'),
6    americas: require('./circuitImg/americas.png'),
```

Figura 28. Variable que inicializa las imágenes en *‘index.js’*

De esta manera, las imágenes se importan en un fichero propio una sola vez, y no en cada pantalla en la que sea necesario. En su lugar, cuando una pantalla necesita alguna simplemente se importa el fichero *‘index.js’* y se selecciona la clave de la imagen necesitada de la variable *‘images’*.

```
13  import colors from '../../res/colors';
14  import images from '../../assets/index';      You, 4 days ago • Adding images to ra
15
    You, 4 days ago | 1 author (You)
16  class MoreScreen extends Component {
17    render() {
18      return (
19        <SafeAreaView style={styles.safeAreaContainer}>
20          <ScrollView ref={(ref) => (this.scrollViewRef = ref)}>
21            <View style={styles.container}>
22              <TouchableHighlight
23                onPress={() => this.props.navigation.navigate('DriversInfo')}
24                style={styles.touchableBtn}>
25                <ImageBackground
26                  source={images.driversScreen}
27                  style={styles.backgroundImg}>
28                  <View style={styles.opacityRaces}>
29                    <Text style={styles.touchableRaceTitle}>Drivers Info</Text>
30                  </View>
31                </ImageBackground>
32              </TouchableHighlight>
```

Figura 29. Ejemplo de importación de imágenes y llamada por clave

6. Elementos destacables

En este proyecto se han decidido usar cierto tipo de funcionalidades que le dan un estilo diferenciador al resto de aplicaciones de este estilo. Entre ellas, se pueden destacar las siguientes:

- ➔ **Scroll hacia la carrera siguiente o actual:** se ha añadido una funcionalidad que permite hacer un scroll hacia el gran premio actual o siguiente cuando el usuario abre la aplicación. De esta manera, el usuario no tiene que buscar por toda la lista qué carrera toca ese fin de semana.
- ➔ **Registro con cifrado de contraseña:** se ha utilizado una librería de encriptación de contraseñas para que la seguridad del usuario prevalezca al registrarse e iniciar sesión. De esta manera, las contraseñas están protegidas tanto en la aplicación en sí como en el servidor de backend de Firebase utilizado.

- ➔ **Búsqueda:** se ha decidido incorporar una búsqueda que permita al usuario una mayor libertad a la hora de consultar datos sin tener que salir de la aplicación.

- ➔ **Posibilidad de activar las notificaciones:** mediante un registro en la aplicación se da la posibilidad al usuario de activar las notificaciones para que sea avisado antes de que una carrera vaya a comenzar. De esta manera, se generará un interés al usuario por registrarse en nuestra aplicación con el fin de no perderse ningún gran premio.

- ➔ **Uso de API REST junto con un backend personalizado:** en esta aplicación se ha optado por el uso de las distintas formas de obtener datos en una aplicación. De esta manera, obtendremos los datos de la Fórmula 1 consultando a una API Oficial y a su vez consultaremos un backend para el inicio de sesión y registro de los usuarios.

En los siguientes capítulos entraremos más en detalle de cada uno de estos elementos a tener en cuenta, además del resto de contenido de la aplicación.

Capítulo III: Desarrollo de la app

1. CocoaPods

React Native es un framework que, a parte de basarse en sus propios componentes, se apoya bastante en multitud de librerías externas para conseguir elementos que no están incluidos de manera nativa. De esta forma, se consigue expandir este framework hasta el punto que el programador desea o necesita.

Para permitir todo este uso de librerías, necesitaremos hacer uso de CocoaPods para la aplicación de iOS. CocoaPods es un gestor de dependencias de Xcode que nos permite agregar los frameworks que necesitemos, facilitándonos la tarea de incluirlos de uno en uno en nuestro proyecto desde Xcode.

Por este motivo, cada vez que instalemos una librería necesitaremos actualizar los Pods de iOS para que la aplicación compile con el siguiente comando:

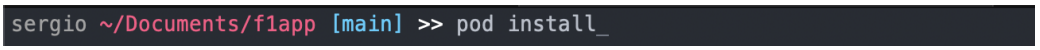


Figura 30. Comando de instalación de librerías externas en CocoaPods

Existen multitud de gestores a parte de CocoaPods, como puede ser Carthage, pero Cocoa permite una gestión bastante más automatizada y sencilla, puesto que nos genera directamente un fichero desde el que trabajar con nuestro proyecto sin necesidad de crearlo nosotros.

En Android, en cambio, la instalación de librerías externas se hace de manera automática en React Native.

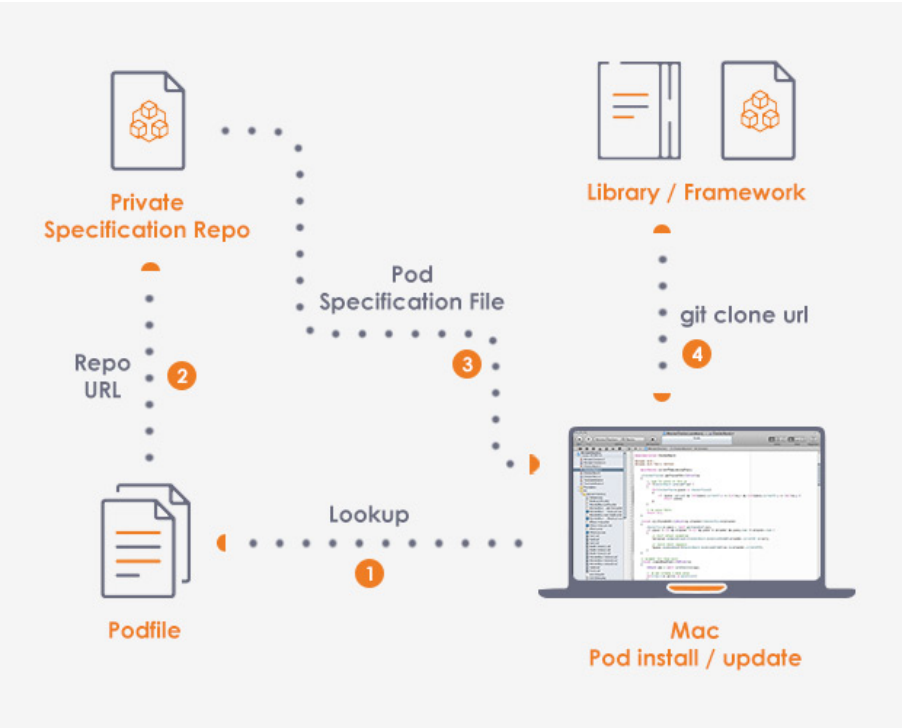


Figura 31. Funcionamiento de CocoaPods al instalar una librería externa. Fuente: DZone (https://dzone.com/articles/private-pod-support-using-cocoapods-in-ios-1)

2. Librerías utilizadas

Para acomodar todas las funcionalidades y diseño requerido para nuestra aplicación, se han añadido una serie de librerías que nos permiten el uso de ciertos elementos que no están disponibles de manera nativa en React Native, a parte de las librerías predefinidas de React Native usadas.

Estas librerías son las siguientes:

➔ **React:** librería nativa de React Native que permite la utilización de clases y constantes. Debe estar presente en todos los ficheros como primer elemento de la cabecera.

➔ **React Native:** librería nativa de React Native que permite la utilización de los componentes de este framework, tales como botones, listas o imágenes. En esta librería hacemos uso de los siguientes componentes:

- ❖ **Flatlist:** muestra una lista scrollable que renderiza un número determinado de elementos.
- ❖ **Activity Indicator:** muestra un indicador de carga con movimiento circular.
- ❖ **Image:** muestra una imagen o recurso.
- ❖ **Image Background:** muestra una imagen de fondo sobre la cual se pueden incluir textos u otros recursos.
- ❖ **Scroll View:** muestra una vista sobre la que se puede hacer scroll.
- ❖ **Status Bar:** controla el estado de la barra de estado del dispositivo.
- ❖ **Switch:** interruptor que cambia de estado ON a OFF y viceversa.

- **Text:** muestra un texto con o sin estilo.

- **Text Input:** muestra un campo de texto rellenable que puede utilizarse en formularios. El texto es recogido por las funciones del propio componente.

- **View:** elemento fundamental de React Native. Es un contenedor de componentes que soporta múltiples estilos de vistas. El resto de los componentes deben estar definidos dentro de una vista.

- **Safe Area View:** contenedor de componentes que permite acotar el área de trabajo en los dispositivos iOS con notch. Totalmente necesario cuando se trabaja con este tipo de dispositivos para comenzar las vistas por debajo de este notch.

- **Touchable Highlight:** contenedor en forma de clickable que permite crear un botón con propiedades más extensas que las que tiene ese elemento. Permite resaltar el botón al seleccionarlo.

- **Touchable Opacity:** contenedor en forma de clickable que permite crear un botón con propiedades más extensas que las que tiene ese elemento. Permite disminuir la opacidad del botón al seleccionarlo.

➔ **React Navigation:** librería externa de gran importancia, pues es la que nos permite navegar entre las pantallas de la aplicación de diversas maneras. En esta librería hacemos uso de los siguientes componentes:

- **Stack:** navegación que permite abrir una pantalla nueva encima de la anterior, con un botón para volver a la pantalla anterior. Este tipo de navegación es muy usada para consultar los detalles de un elemento (en nuestro caso, al seleccionar un gran premio, abrir el buscador, abrir alguna pantalla de la pestaña ‘More’, o abrir la pantalla de registro).

- ❖ **Bottom Tabs:** navegación que crea una barra inferior con iconos que permite cambiar entre las distintas pantallas añadidas (en nuestro caso, tenemos ‘Races’, ‘Standings’, ‘More’ y ‘Profile’). Éstas se presentan sin botón para volver atrás.

- ❖ **Material Top Tabs:** navegación que permite configurar varias pestañas en una misma pantalla, permitiendo el desplazamiento por ellas con un gesto de deslizamiento (en nuestro caso, en la pantalla de ‘Standings’ a la hora de seleccionar entre ‘Drivers’ o ‘Teams’).

➔ **React Native Vector Icons:** librería externa que nos permite la utilización de multitud de iconos totalmente customizables. Incluye bastantes librerías de iconos, pero en nuestro haremos uso de las siguientes:

- ❖ **Material Icons.**

- ❖ **Material Community Icons.**

➔ **React Native Linear Gradient:** librería externa que nos permite crear gradientes completamente personalizables (en nuestro caso, el fondo de la aplicación).

➔ **MomentJS:** librería externa que permite manipular fechas y horas de forma sencilla (en nuestro caso, se ha utilizado para calcular si un gran premio se ha disputado, se está disputando o se va a disputar, así como de darle formato a la fecha de un gran premio en el detalle de éste).

➔ **React Native Collapsible View:** librería externa que permite el uso de elementos expandibles para mostrar información adicional (en nuestro caso, todos los desplegables de la aplicación).

➔ **React Native Elements:** librería externa que añade componentes adicionales a los predefinidos en React Native, además de dar más opciones de personalización y obtención de datos para los elementos ya predefinidos en éste. En esta librería hacemos uso de los siguientes componentes:

- ❖ **Icon:** componente que trabaja en conjunto a la librería anteriormente mencionada *'React Native Vector Icons'*. Nos permite añadir iconos en cualquier sitio de nuestra aplicación (en nuestro caso, el icono para acceder al buscador).

- ❖ **Divider:** componente que genera una línea divisoria horizontal o vertical.

- ❖ **Search Bar:** componente que genera una barra de búsqueda que recoge los datos de forma directa.

➔ **React Native Push Notifications:** librería externa que trabaja con la generación de notificaciones locales y remotas.

3. Configuraciones previas

Es importante destacar que si no se poseyera un Mac no se puede programar la aplicación para iOS. Como en nuestro caso esto no supone problema alguno, la configuración previa va orientada al sistema operativo macOS.

A la hora de comenzar nuestro proyecto, hay que realizar una serie de configuraciones que nos permitan crear y simular sin errores. Para ello, hay dos opciones para empezar a utilizar React Native:

➔ **React Native Expo:** esta configuración incluye un conjunto de herramientas llamado Expo, el cual permite realizar aplicaciones de forma sencilla sin tener que hacer cambios en los archivos correspondientes de iOs y Android. No es necesario ni Xcode ni Android Studio, pero las librerías que se pueden utilizar son restringidas.

➔ **React Native CLI:** esta es la configuración que hemos usado en nuestra aplicación, ya que consiste en la utilización de React Native sin limitación alguna. Esta configuración es más complicada, pues es necesario configurar ciertos ficheros en Xcode y Android Studio, con sus respectivos lenguajes.

Después de escoger la opción con la que queremos trabajar en React Native, lo siguiente que debemos hacer es instalar *Homebrew*.

Homebrew es un gestor de paquetes de para macOS. Para su instalación, en la consola de comandos escribiremos lo siguiente:

```
$ /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

Figura 32. Instalación de Homebrew

Esta instalación es necesaria para instalar tanto *Node* como *Watchman*.

Node es un entorno de ejecución para JavaScript que incorpora *NPM*, el cual utilizaremos para la instalación de las librerías externas del proyecto.

Watchman es una herramienta creada por Facebook que permite el control de cambios en los archivos del sistema. Es recomendable instalarlo para obtener un mejor rendimiento de React Native.

La instalación de estas herramientas se realizarán en la consola de comandos de la siguiente forma:

```
brew install node
brew install watchman
```

Figura 33. Instalación de Node y Watchman

A continuación, instalaremos tanto Xcode como Android Studio para dar soporte a ambos sistemas operativos.

Después de estas instalaciones, podremos crear nuestro proyecto con el siguiente comando:

```
npx react-native init AwesomeProject
```

Figura 34. Creación de un proyecto

Con este comando tendremos en la carpeta del proyecto los archivos necesarios para comenzar, con las correspondientes carpetas de iOS y Android, y el fichero llamado *'App.js'*, en el cual empezaremos a construir la aplicación.

Para simular en iOS y en Android, utilizaremos los comandos:

```
npx react-native run-ios
```

Figura 35. Simular en iOS

```
npx react-native run-android
```

Figura 36. Simular en Android

4. Optimizaciones

Para mantener la fluidez de la aplicación, se han decidido tomar una serie de medidas de cara a la optimización del rendimiento, sobre todo teniendo en cuenta la cantidad de imágenes que se tienen que ir cargando.

Para que el rendimiento no se vea lastrado por esto, se han realizado las siguientes acciones:

➔ Optimizaciones en Flatlists

- **Evitar usar funciones de flecha en la propiedad *'renderItem'*:** se han movido las funciones de estos elementos fuera de la función de renderizado, de tal manera que no se volverá a crear cada vez que se llame a la función de renderizado.

- **Añadir la propiedad *'initialNumToRender'* a la lista:** se define cuántos elementos se van a renderizar por primera vez, ahorrando mucha carga de datos.

- **Definir la *'key'* o *'keyExtractor'* en el componente del elemento:** utilizar una clave identificatoria del elemento que vamos a mostrar evitará volver a renderizar los elementos que ya se han mostrado con anterioridad.

- **Utilización de *'maxToRenderPerBatch'*, *'getItemLayout'* o *'windowSize'*:** en determinadas listas necesitaremos acotar el número de elementos a renderizar para que no haya desgaste de recursos y la lista sepa cuándo ha terminado de mostrar información.

➔ Optimizaciones en Collapsible Views

- **Utilización de la propiedad *'renderChildrenCollapsed'*:** esta propiedad viene activada por defecto, y es la encargada de renderizar el contenido de una vista expandible antes de expandir. En nuestro caso, hemos desactivado esta opción con el fin de permitir una mayor velocidad de navegación entre expandibles.

➔ Optimizaciones en imágenes

- **Reducción del tamaño de las imágenes:** con el fin de agilizar la precarga de las imágenes de la aplicación, se ha reducido el tamaño de las imágenes hasta coincidir con el tamaño de los botones de la lista, que es nuestro tamaño necesario. De esta forma, se siguen viendo de manera correcta y cargan con gran velocidad.

- **Compresión de las imágenes:** tras reducir el tamaño de las imágenes, éstas aún podían bajar más de peso sin perder calidad comprimiéndolas. Con esto, obtenemos la misma calidad de imagen con una velocidad de carga perfecta.

5. Notificaciones

Una parte importante de toda aplicación móvil es el uso de las notificaciones para mantener informado al usuario cuando éste lo desee.

En nuestro caso, se ha decidido añadir este sistema para avisar al usuario cuando un gran premio vaya a comenzar. Este sistema está disponible sólo si el usuario se registra e inicia sesión en la aplicación.

Para la realización de esta funcionalidad, se ha creado un fichero específico llamado *'NotificationService.js'* que permite controlar el tipo de notificación que vamos a recibir, qué contenido debe llevar y en qué momento se va a mostrar.

Se ha realizado con una librería específica debido a que de otra manera en iOS es necesario un token FCM. Este token sólo puede obtenerse con una cuenta pagada del programa de Apple Developers.

De esta forma, utilizando esta librería obtenemos las notificaciones del mismo modo tanto en iOS como en Android, con los estilos de cada sistema operativo y sin necesidad de estos tokens.

6. Búsqueda

La búsqueda de pilotos es otro pilar de la aplicación. Gracias a ella, los usuarios pueden consultar los resultados de sus pilotos favoritos en cada temporada.

Para su realización, se ha utilizado el componente '*SearchBar'* de la librería externa '*React Native Elements'*. Gracias a este componente, podemos obtener el valor del campo conforme va cambiando, de tal manera que la búsqueda sólo se realiza cuando el usuario pulsa la tecla de intro.

Se ha decidido hacer de esta manera para evitar resultados erróneos si el usuario introduce un apellido que empieza de forma común al de otro piloto. A pesar de realizarlo de esta manera, la variable de estado de la clase contendrá cada cambio de valor del campo de búsqueda, actualizándose a cada letra que se escriba.

La búsqueda se lleva a cabo en base al apellido del piloto. Cuando el usuario comienza una búsqueda, se activa el icono de loading mientras el parámetro de ésta es enviado junto con la petición de la API REST para obtener la información del piloto requerido.

En caso de que no exista ningún piloto, no se mostrará ningún resultado. Si por el contrario sí que existe, se mostrarán una serie de desplegables que tendrán como título la temporada a la que se refiere la información obtenida. Dentro de cada temporada, encontraremos los puntos, posición, nombre completo del piloto y escudería de esa temporada.

Para esta búsqueda no ha hecho falta la creación de ningún servicio intermedio, puesto que el propio componente es el que maneja las funciones necesarias para obtener la información introducida.

Capítulo IV: Obtención de datos y backend

7. Librerías utilizadas

A parte de las librerías utilizadas para la aplicación en general, se han usado una serie de librerías para la parte de backend que permiten proporcionar la seguridad necesaria y el almacenamiento de datos.

Estas librerías son:

➔ **React Native Firebase:** librería externa oficial recomendada para el uso de Firebase en React Native. Esta librería proporciona todos los paquetes que el usuario necesite para el desarrollo e implementación de las funcionalidades de Firebase.

En nuestro caso, utilizaremos el paquete de Firebase que incluye Cloud Firestore, usada para el uso y la creación de la base de datos de nuestra aplicación.

➔ **React Native CryptoJS:** librería externa que contiene una colección creciente de algoritmos criptográficos estándar y seguros implementados en JavaScript utilizando las mejores prácticas y patrones. Son algoritmos rápidos y tienen una interfaz simple y consistente.

En nuestra aplicación, utilizaremos esta librería para encriptar y desencriptar las contraseñas de los usuarios en el registro y login con el estándar AES.

8. Configuraciones previas

A la hora de comenzar el desarrollo del backend y la obtención de datos de la API, será necesaria la instalación de las librerías mencionadas en el apartado anterior a través de la consola de comandos.

En el caso de Firebase, serán necesarios una serie de prerrequisitos a la hora de usar la librería. Lo primero de todo será instalarla con la consola de comandos, y tras esto deberemos seguir los siguientes pasos:

➔ **Abrir la consola de Firebase.**

➔ **Añadir una nueva aplicación de Android/iOS.**

➔ **Descargar ficheros JSON de Google Services.**

➔ **Configurar credenciales de Firebase.**

A la hora configurar las credenciales, dependiendo de la plataforma en la que estemos (Android o iOS), habrá que añadir ciertas líneas de código en los ficheros *'build.gradle'* o *'AppDelegate.m'* respectivamente.

Los ficheros JSON descargados de Firebase son los que permiten tener la configuración de Firebase en nuestro proyecto, por lo que sin estos ficheros no funcionarán las librerías.

9. API REST

Los datos de Fórmula 1 de nuestra aplicación son obtenidos a través de un servicio de API REST oficial (http://ergast.com/mrd/). Este servicio nos permite realizar una serie de peticiones que hemos adaptado según necesitamos en este proyecto, como obtener la lista de temporadas, los calendarios de carreras, o los resultados de clasificación.

Para poder comprobar estas peticiones y hacer un mejor uso de ellas, hemos decidido utilizar la aplicación Postman, la cual nos permite comprobar el funcionamiento de estas peticiones y ver los datos que devuelve.

Con el fin de tener las peticiones organizadas para encontrarlas y testearlas con facilidad, hemos creado una carpeta que contiene las peticiones utilizadas en la aplicación.

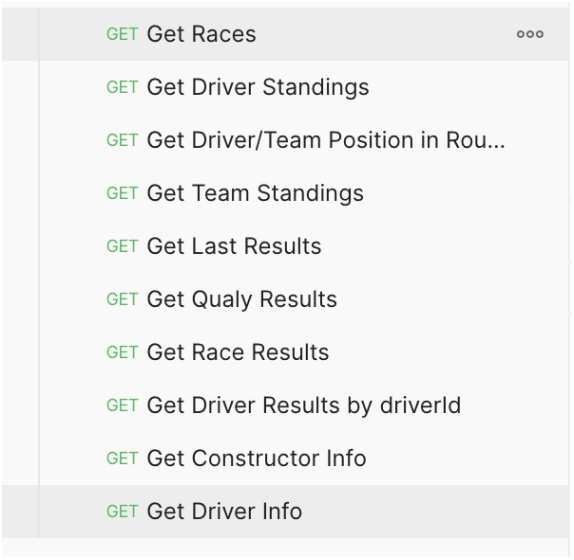


Figura 37. Peticiones API REST desde Postman

Las peticiones son recibidas en formato JSON, y se extraen los datos necesarios en cada pantalla que sea necesaria dicha información. Para la obtención de este JSON se ha creado el fichero *'http.js'* en la carpeta *'libs'* del proyecto. Este archivo contiene la estructura de petición de los datos de la API para poder manejarlos desde las pantallas.

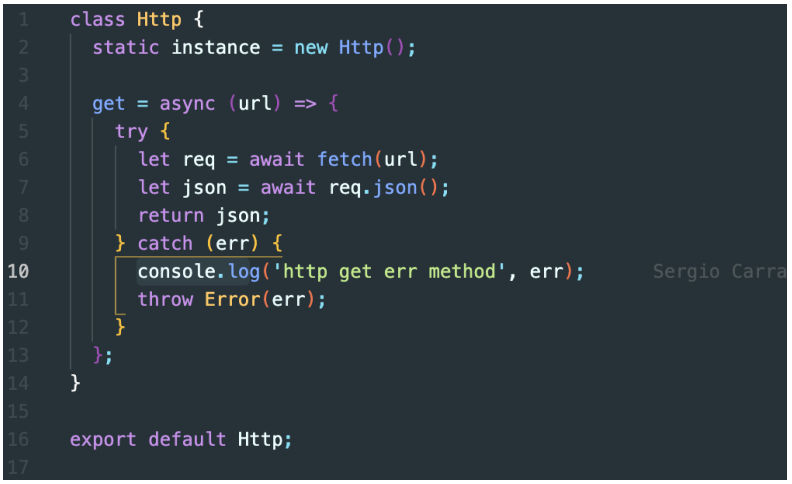


Figura 38. Función HTTP para obtener los datos de la API REST

Se trata de una petición GET asíncrona que nos devolverá los datos que pidamos en las distintas pantallas o un error si no se han podido extraer dichos datos.

En la pantalla de ‘Races’ tenemos un ejemplo sobre cómo se realizan las peticiones utilizando esta petición HTTP:

Figura 39. Ejemplo de petición de datos a la API REST

Lo primero que debemos hacer es crear una variable de estado en la que guardaremos toda la información que obtengamos, así como el estado *'loading'*, que se encargará de mostrar un icono de carga si los datos están siendo obtenidos. Esta variable de estado se actualiza a tiempo real conforme se cargan los datos de la API.

También haremos uso de una variable denominada *'_isMounted'*, la cual nos permitirá saber si una petición está siendo ejecutada en el caso que necesitemos realizar más de una. Esto permite que cuando una petición termina, el componente es destruído para evitar problemas de rendimiento en la aplicación.

Por último, tenemos la llamada asíncrona a nuestra función encargada de obtener los datos de la API a través del fichero HTTP mencionado anteriormente.

Una vez la información ha sido recibida, se envía al estado y se quita el indicador de carga de la aplicación.

Gracias al uso de funciones tales como *'componentWillUnmount'*, *'componentDidMount'* y las variables de estado, la obtención de información a través de la API se realiza de manera rápida sin comprometer el rendimiento de la aplicación.

10. Autenticación con Cloud Firestore (Firebase)

En este proyecto hemos decidido incorporar un servicio de autenticación de usuarios con Firebase. Con esto permitimos el registro de usuarios y generamos un beneficio, en este caso la activación de notificaciones.

Para realizar esta autenticación, hemos elegido la tecnología de Firebase denominada Cloud Firestore. Se trata de una base de datos para el desarrollo de aplicaciones móviles.

Nació de la base de datos originaria de Firebase, denominada Realtime Database. Aprovecha lo mejor de ésta con un modelo de datos nuevo y más intuitivo, además de la gran escalabilidad que permite.

Para la configuración de esta base de datos se ha creado un fichero específico en la carpeta *'server'* del proyecto. Dentro de ésta, encontraremos el fichero *'UsersDB.js'*, el cual contiene la configuración de las peticiones a la base de datos de agregar y obtener usuarios.

Cabe mencionar que al tratarse de una base de datos en la nube, no se ha requerido ninguna inicialización de Firebase en los archivos de la aplicación más allá que la instalación de la propia librería.

De esta forma, lo único necesario es crear la base de datos con la función 'collection' para subir los datos.

11. Medidas de seguridad implementadas

A la hora de la creación de usuarios en la base de datos, se ha tenido en cuenta el hasheo de contraseñas para dar un nivel de seguridad a nuestra aplicación. Para ello, se ha utilizado la librería externa *'CryptoJS'*.

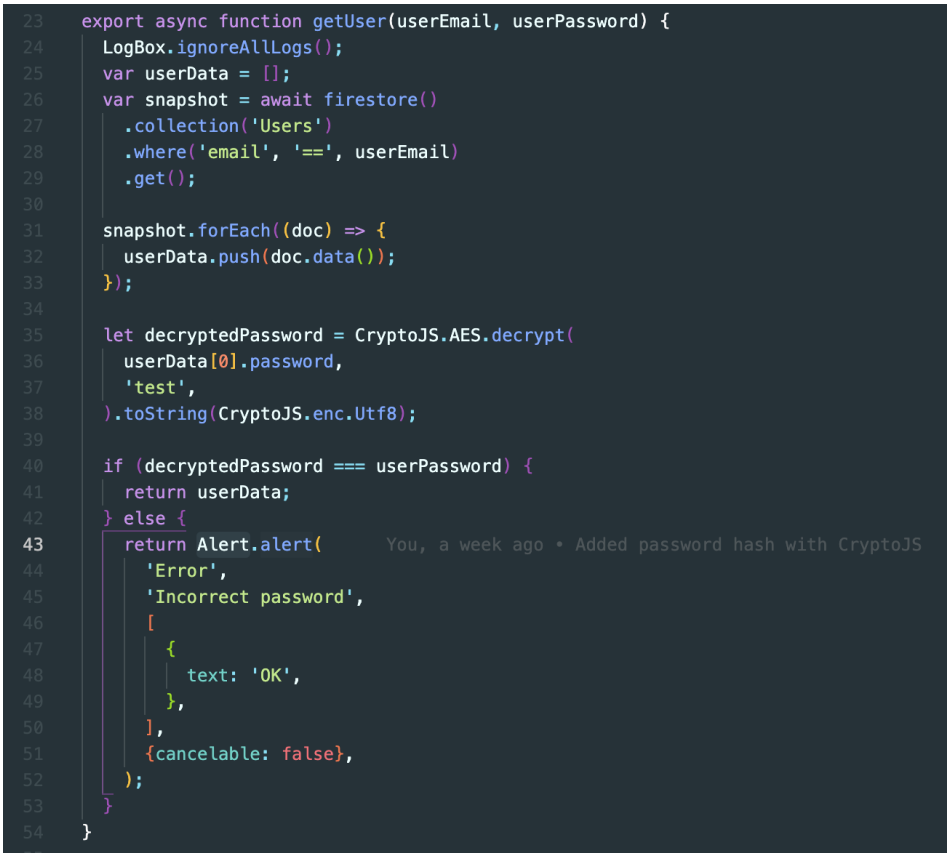
```
export function addUser(user, addComplete) {
  LogBox.ignoreAllLogs();
  firestore()
    .collection('Users')
    .add({
      name: user.name,
      email: user.email,
      password: CryptoJS.AES.encrypt(user.password, 'test').toString(),
      createdAt: firestore.FieldValue.serverTimestamp(),
    })
    .then((snapshot) => snapshot.get())
    .then((userData) => {
      addComplete(userData.data());
    })
    .catch((err) => console.log(err));
}
```

Figura 40. Función de crear usuarios en la base de datos Cloud Firestore

Esta librería permite hashear con palabras clave y con diferentes tipos de codificación. En nuestro caso, utilizaremos la codificación AES. Este hasheo se realiza antes de enviar los datos a Firestore, de tal manera que no queda constancia alguna de la clave sin hashear.

A la hora de realizar el login del usuario, y como no puede haber dos usuarios con un mismo email, obtendremos los datos de usuario con este email, y comprobaremos si la clave proporcionada en el formulario de login es igual a la clave de la base de datos desencriptada.

Si ambas coinciden, se devuelve la información del usuario completa y se realiza el inicio de sesión, pero si no coinciden salta una alerta sin la devolución de dichos datos.



```
export async function getUser(userEmail, userPassword) {
  LogBox.ignoreAllLogs();
  var userData = [];
  var snapshot = await firestore()
    .collection('Users')
    .where('email', '==', userEmail)
    .get();

  snapshot.forEach((doc) => {
    userData.push(doc.data());
  });

  let decryptedPassword = CryptoJS.AES.decrypt(
    userData[0].password,
    'test',
  ).toString(CryptoJS.enc.Utf8);

  if (decryptedPassword === userPassword) {
    return userData;
  } else {
    return Alert.alert(      You, a week ago • Added password hash with CryptoJS
      'Error',
      'Incorrect password',
      [
        {
          text: 'OK',
        },
      ],
      {cancelable: false},
    );
  }
}
```

Figura 41. Obtención de un usuario dado su email y contraseña

En el caso de la petición de obtener usuarios, se realiza de manera asíncrona para no ralentizar la aplicación y que los datos vayan llegando conforme se obtengan sin necesidad de esperar. En cambio, la petición de agregar usuario se realiza de forma síncrona.

Si accedemos a la consola de Cloud Firestore, podemos apreciar que los usuarios registrados tienen la contraseña hasheada de forma correcta.

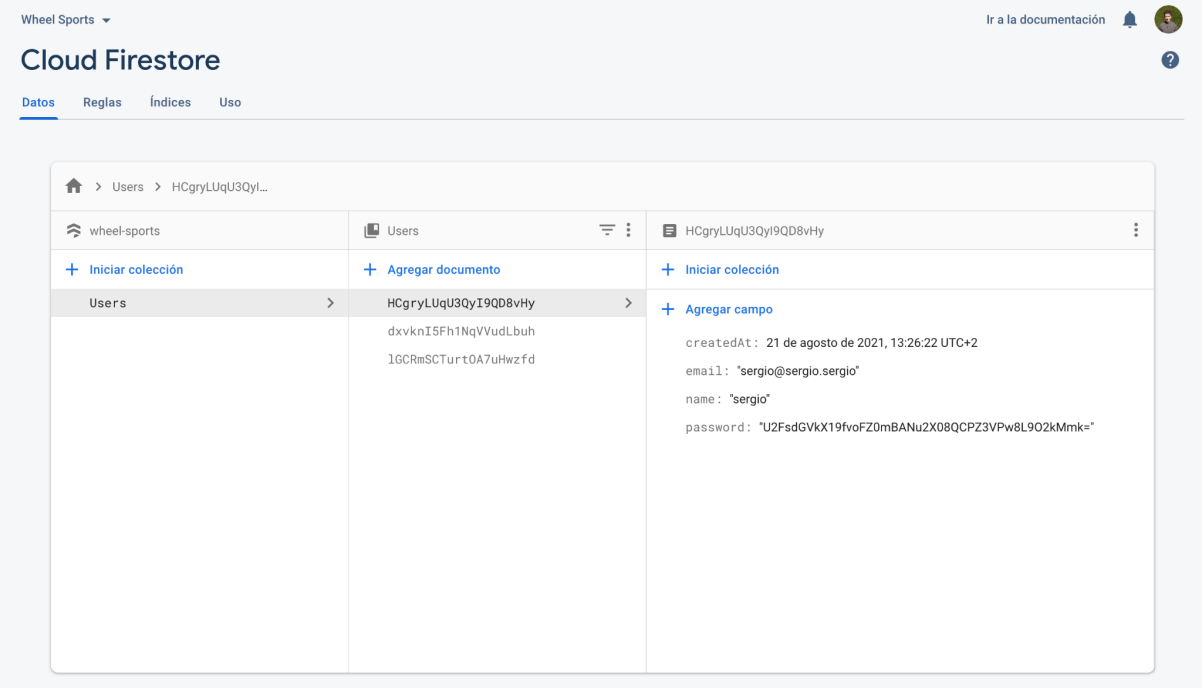


Figura 42. Consola de Cloud Firestore con contraseñas hasheadas

Además de esto, para realizar la validación del correo electrónico introducido en el registro se ha utilizado una función regex que permite los emails de la forma XXX@XXX.XXX . Se ha realizado de esta forma debido a que React Native no incluye ninguna manera de comprobar si los campos de un formulario cumplen ciertas validaciones.

```
31  validateEmail = (email) => {
32    const re = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
33    return re.test(email);
34  };
```

Figura 43. Validación de email con Regex

A su vez, se realizan comprobaciones de que los campos no estén vacíos o tengan la longitud correcta (por ejemplo, el campo contraseña debe tener una longitud superior a 3) a la hora de enviar el formulario de registro. Si algún campo está vacío o no cumple la validación, se muestra una alerta indicando al usuario que debe rellenarlo de forma correcta antes de enviarlo.

Una vez se complete el registro de forma exitosa, se muestra un mensaje de bienvenida y se redirige al usuario al inicio de sesión.

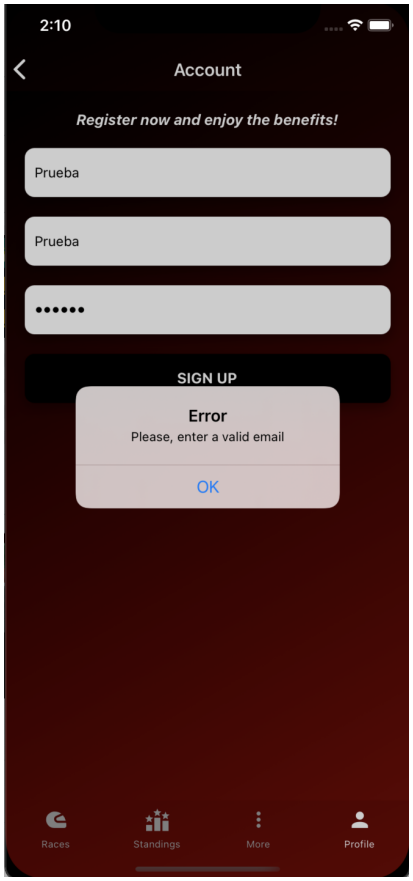


Figura 44. Validación de registro

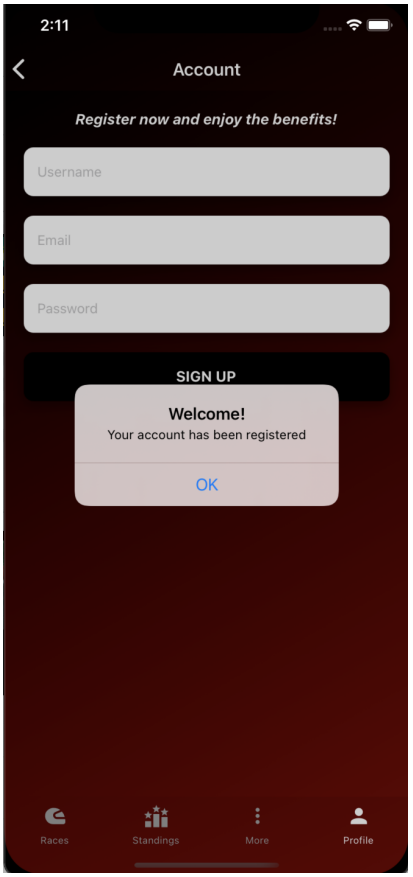


Figura 45. Registro realizado con éxito

Capítulo V: Publicación de la app

12. Publicación en AppStore

Para la publicación de la aplicación en la tienda de Apple, aparte de tener una cuenta del programa de Apple Developers, necesitaremos seguir una serie de pasos:

➔ **Habilitar ATS (App Transport Security):** funcionalidad de seguridad introducida por Apple a partir de iOS 9 que rechaza todas las peticiones HTTP que no son enviadas a través de HTTPS. Por defecto, en el proyecto en local de React Native, esta medida de seguridad viene desactivada, por lo que antes de publicar la aplicación hay que reactivarla de la siguiente forma:

- ❖ **Ir al archivo *'Info.plist'* de la carpeta *'ios/'* del proyecto**
- ❖ **Eliminar la entrada *'NSExceptionDomains'***
- ❖ **Cambiar el valor de la entrada *'NSAllowsArbitraryLoads'* a false**

En el caso en que la aplicación necesite recibir peticiones de conexiones no seguras HTTP, se puede configurar para permitir ciertas excepciones con *'NSExceptionDomain'*, pero no es recomendable ya que expondremos la seguridad de la aplicación.

➔ **Configurar esquema Release (esquema de lanzamiento):** será necesario crear un esquema de Release puesto que nuestra aplicación empieza con un esquema de prueba por defecto. Para ello, debemos realizar la siguiente configuración:

- ❖ **Abrir carpeta *'ios/'* del proyecto desde Xcode.**
- ❖ **Ir a *'Product - Scheme - Edit Scheme'*.**

- ❖ **Seleccionar *'Run'* en el menú lateral.**

- ❖ **En el desplegable *'Build Configuration'* seleccionar *'Release'*.**

➔ **Compilar la aplicación:** después de esta configuración, solo necesitaremos compilar la aplicación desde Xcode seleccionando *'Product - Build'*. Se nos mostrará un menú de progreso en el que podremos ver el estado de nuestra compilación y una vez termine tendremos la opción de subir nuestra aplicación a Apple para que la comiencen a revisar.

Una vez revisada y aceptada, será subida a la Store automáticamente por Apple.

13. Publicación en PlayStore

Para la publicación de la aplicación en la tienda de Google necesitaremos seguir los siguientes pasos:

➔ **Generar una key de subida:** esta clave nos servirá para firmar la aplicación antes de subirla a la Store, por lo que es de vital importancia. Podemos generarla utilizando el siguiente comando:

```
sergio ~/Documents/f1app [main] >> keytool -genkeypair -v -keystore my-upload-key.keystore -alias my-key-alias -keyalg RSA -keysize 2048 -validity 10000
```

Figura 46. Generando clave privada

➔ **Configurar las variables de Gradle:** deberemos configurar las variables de Gradle para que firme nuestra aplicación. Seguiremos los siguientes pasos:

- ❖ **Colocar el archivo generado *'my-upload-key.keystore'* en el directorio *'android/app'* de nuestro proyecto.**

❖ **Editar el archivo *'gradle.properties'* y añadir lo siguiente:**

```
MYAPP_UPLOAD_STORE_FILE=my-upload-key.keystore
MYAPP_UPLOAD_KEY_ALIAS=my-key-alias
MYAPP_UPLOAD_STORE_PASSWORD=*****
MYAPP_UPLOAD_KEY_PASSWORD=*****
```

Figura 47. Configuración de variables de Gradle

➔ **Añadir configuración de firma a la configuración de Gradle:** el último paso para terminar las configuraciones de publicación en PlayStore es editar el fichero *'build.gradle'* situado en la carpeta *'android/app/'* de nuestro proyecto para añadir las siguientes líneas de código:

```
...
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            if (project.hasProperty('MYAPP_UPLOAD_STORE_FILE')) {
                storeFile file(MYAPP_UPLOAD_STORE_FILE)
                storePassword MYAPP_UPLOAD_STORE_PASSWORD
                keyAlias MYAPP_UPLOAD_KEY_ALIAS
                keyPassword MYAPP_UPLOAD_KEY_PASSWORD
            }
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
}
...
```

Figura 48. Configuración de firma de Gradle

➔ **Generar el archivo de Release AAB:** para generar el fichero de distribución de la aplicación deberemos escribir el siguiente comando dentro de la carpeta *'android'* del proyecto:

```
./gradlew bundleRelease
```

Figura 49. Release de Android

De esta forma, tendremos listo el fichero .apk para su subida a la Store de Google.

Capítulo VI: Conclusiones y futuro de la app

1. Futuras mejoras

Tras terminar este proyecto, tenemos una aplicación totalmente funcional y preparada para ser publicada en las Stores de los diferentes sistemas operativos, pero se van a tener en cuenta una serie de mejoras de cara al futuro desarrollo de la aplicación:

- ➔ **Añadir otro apartado en la pestaña 'More' que permita a los usuarios avanzados acceder a información técnica de la temporada, tales como normativas.**

- ➔ **Añadir un apartado de dudas en donde se explique a la gente recién llegada a este deporte las tecnologías de los coches de Fórmula 1 y sus funciones.**

- ➔ **Búsqueda de pilotos para obtener su información personal y de escudería desde que empezó la Fórmula 1.**

- ➔ **Añadir un apartado de noticias para ver la última hora de este deporte.**

- ➔ **Posibilidad de añadir un chat que permita a los usuarios registrados discutir sobre las carreras en vivo.**

- ➔ **Posibilidad de modo claro en la aplicación.**

- ➔ **Implementación de diferentes idiomas, con posibilidad de seleccionarlos por el usuario.**

2. Futuro de la app

Como se puede observar en las futuras mejoras, el proyecto va a mejorar bastante de cara a un futuro próximo, de forma que podamos ofrecer una aplicación cada vez más completa dependiendo del feedback que recibamos de los usuarios.

El desarrollo de la aplicación va a ser continuado, publicándose en las diferentes Stores para que sea disfrutable por todo el mundo.

3. Conclusiones finales

Tras terminar este proyecto, se ha comprobado la facilidad que ofrece el framework React Native a la hora de desarrollar aplicaciones nativas multiplataforma sin necesidad de programar en Java o Swift.

El gran número de librerías permite realizar cualquier tipo de aplicación que se nos pueda ocurrir de una manera disfrutable, sencilla y rápida. Con el paso de los años, veremos como la gran comunidad de React Native sigue creciendo y ofreciendo más y más opciones junto con otros frameworks que vayan creciendo exponencialmente como Flutter.

Esto supone cuestionarse si para el uso de una aplicación de forma nativa es necesario programar en cada lenguaje de los distintos sistemas operativos por separado o si por el contrario, con herramientas como éstas se puede dar al usuario esa experiencia nativa tan buscada.

Capítulo VII: Bibliografía

Todos los recursos en línea se han consultado a inicio de septiembre 2021.

➔ Campeonato Fórmula 1: https://es.wikipedia.org/wiki/F%C3%B3rmula_1
➔ Introducción a la F1: https://laguiaformulera.com/2020/02/07/una-breve-introduccion-a-la-f1/
➔ React Native vs Flutter: https://www.itdo.com/blog/react-native-vs-flutter-cual-es-mejor-para-mi-producto/
➔ Best DB for React Native: https://blog.codemagic.io/choosing-the-right-database-for-react-native-app/
➔ Qué es React Native: https://www2.deloitte.com/es/es/pages/technology/articles/que-es-react-native.html
➔ Qué es una API REST: https://www.redhat.com/es/topics/api/what-is-a-rest-api
➔ Información de Git: https://es.wikipedia.org/wiki/Git
➔ Trello: https://trello.com/es
➔ Funcionamiento de Trello: https://www.expertosnegociosonline.com/que-es-trello-para-que-sirve/
➔ Git docs: https://git-scm.com/
➔ Github: https://github.com/
➔ Firebase docs: https://firebase.google.com/?hl=es
➔ React Native Docs: https://reactnative.dev/
➔ Stack Overflow: https://stackoverflow.com/
➔ App Icon Generator: https://appicon.co/
➔ App icon with React Native: https://www.instamobile.io/react-native-tutorials/react-native-app-icon-ios-android/
➔ Splash screen in React Native: https://medium.com/@appstud/add-a-splash-screen-to-a-react-native-app-810492e773f9
➔ React Native Firebase docs: https://rnfirebase.io/
➔ API utilizada: http://ergast.com/mrd/
➔ React Navigation docs: https://reactnavigation.org/
➔ Unsplash: https://unsplash.com/
➔ CocoaPods docs: https://cocoapods.org/
➔ React Native Vector Icons docs: https://github.com/oblador/react-native-vector-icons
➔ React Native Linear Gradient docs: https://github.com/react-native-linear-gradient/react-native-linear-gradient
➔ MomentJS docs: https://momentjs.com/

➔ React Native Collapsible View docs: https://github.com/Eliav2/react-native-collapsible-view
➔ React Native Elements docs: https://reactnativeelements.com/
➔ React Native Push Notifications docs: https://github.com/zo0r/react-native-push-notification
➔ Homebrew docs: https://brew.sh/index_es
➔ NodeJS docs: https://nodejs.org/es/
➔ Watchman docs: https://facebook.github.io/watchman/docs/install.html
➔ What is FCM token: https://www.raywenderlich.com/20201639-firebase-cloud-messaging-for-ios-push-notifications
➔ CryptoJS docs: https://cryptojs.gitbook.io/docs/
➔ Cloud Firestore docs: https://firebase.google.com/docs/firestore
➔ How to add app icon: https://www.instamobile.io/react-native-tutorials/react-native-app-icon-ios-android/
➔ What is Firebase (BaaS): https://medium.com/firebase-developers/what-is-firebase-the-complete-story-abridged-bcc730c5f2c0