UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

APLICACIÓN PARA LA GESTIÓN Y OBTENCIÓN DE LA INFORMACIÓN PROCEDENTE DE UN SISTEMA DE MEDICIÓN VELOCIDAD EN PRUEBAS DE ATLETISMO

Luis Bermejo Vaquera

Tutor: Clara Pérez Fuster

Cotutor: Fulgencio Montilla Meoro

Trabajo Fin de Grado presentado en la Escuela Técnica Superior de Ingenieros de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Graduado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Curso 2019-20

Valencia, 9 de septiembre de 2020

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
www.etsit.upv.es

VLC/ CAMPUS VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS Gestión ambiental verificada REG.NO. ES-CV- 000030

Resumen

El trabajo que a continuación se presenta consiste en el desarrollo de una aplicación Android para la gestión y obtención de la información procedente de un equipo de medición de velocidad en pruebas de Atletismo. El software desarrollado permitirá al usuario realizar estas mediciones de manera eficiente, así como poder visualizarlas en tiempo real.
El equipo de medida dispondrá de microcontroladores, que se comunicarán entre ellos mediante un módulo de RF (Radio Frecuencia) y a su vez transmitirán la información al dispositivo Android mediante el canal de Bluetooth. La aplicación móvil desarrollada para Android, se encargará por una parte de recibir los datos proporcionados por el equipo y le enviará las instrucciones para su puesta en marcha; por otra parte, después de tratarlos, los subirá a la "Realtime Database" de Firebase. La aplicación permitirá mantener un historial de las carreras, para poder verlas en cualquier momento y se actualizarán en tiempo real. En la base de datos se guardará también la configuración de la carrera, datos tales como la distancia total de la carrera, marcas de tiempo y velocidades medias.

Resum

El treball que a continuació es presenta consisteix en el desenvolupament d'una aplicació Android per a la gestió i obtenció de la informació procedent d'un equip de mesurament de velocitat en proves d'Atletisme. El programari desenvolupat permetrà l'usuari realitzar aquests mesuraments de manera eficient, així com poder visualitzar-les en temps real.

L'equip de mesura disposarà de microcontroladors, que es comunicaran entre ells mitjançant un mòdul de RF (Radio Freqüència) i al seu torn transmetran la informació a el dispositiu Android mitjançant el canal de Bluetooth. L'aplicació mòbil desenvolupada per Android, s'encarregarà d'una banda de rebre les dades proporcionades per l'equip i li enviarà les instruccions per a la seva posada en marxa; d'altra banda, després de tractar-los, els pujarà a la "Realtime Database" de Firebase. L'aplicació permetrà mantenir un historial de les carreres, per poder veure-les en qualsevol moment i s'actualitzaran en temps real. A la base de dades es guardarà també la configuració de la cursa, dades com ara la distància total de la cursa, marques de temps i velocitats mitjanes.

Abstract

The work that is presented below consists of the development of an Android application for the management and obtaining of information from a speed measurement equipment in Athletics events. The software developed will allow the user to perform these measurements efficiently, as well as to be able to view them in real time.

The measuring equipment will have microcontrollers, which will communicate with each other through a RF (Radio Frequency) module and in turn will transmit the information to the Android device through the Bluetooth channel. The mobile application developed for Android will, on the one hand, be in charge of receiving the data provided by the equipment and will send you the instructions for its start-up; on the other hand, after treating them, it will upload them to the Firebase "Realtime Database". The application will allow you to keep a history of the races, to be able to see them at any time and they will be updated in real time. In the database, the configuration of the race, data such as the total distance of the race, time stamps and average speeds will also be saved.

Índice

Tabla de contenido

Capítulo 1. Introducción

1.1 Introducción.

En la actualidad, la tecnología ha pasado a formar parte de nuestro día a día. Esta implementada en innumerables gadgets, desde un SmartWatch, que puede medir: el ritmo cardiaco, los pasos que damos, la geolocalización, etc; a un coche capaz de conducir de forma autónoma o robots súper inteligentes que rozan la inteligencia artificial. En esta última etapa de la evolución tecnológica se ha avanzado a un ritmo exponencial, en poco menos de 100 años ha habido una serie de avances que han cambiado los límites de la tecnología.

En 1977 se lanzó al mercado el Apple II, desde entonces la primera serie de computadoras de producción en masa, se ha ido introduciendo en la vida de la gente hasta tal punto que dependemos de ellas, antes de eso la gente vivía sin teléfonos móviles, ordenadores… En cambio, las nuevas generaciones, al nacer rodeados de aparatos inteligentes crecen con total normalidad de su uso, esto hará que dentro de 30 años prácticamente toda la población del mundo esté acostumbrada a su uso.

En el ámbito deportivo la tecnología ha desarrollado una gran cantidad de dispositivos que permiten controlar el estado físico del deportista; también despiertan gran interés las medidas de tiempos, velocidad, distancia recorrida…, existen gadgets que son capaces de medir tales parámetros, de la exactitud de estas medidas, dependen en muchas ocasiones el que un deportista se clasifique o no en una competición. El coste de estos equipos de medidas no es accesible económicamente para un público mayoritario, y por otra parte cada vez hay más deportistas que están interesados en conocer sus logros, y controlar sus tiempos durante los entrenamientos para poder mejorar y progresar; a la vez que también aumenta el deseo de compartir la información con el resto de miembros del equipo.

En este caso, este proyecto se basará en la obtención de marcas de tiempo para la medición de un corredor de atletismo; así mismo se podrán obtener las estadísticas de las carreras, y disponer con más en detalle en cualquier momento siempre que se tenga internet en el dispositivo móvil.

1.2 Motivación.

Con esta aplicación se pretende colaborar para el avance de las tecnologías en el ámbito deportivo, permitiendo desarrollar un equipo de medida cuyo coste sea asequible para deportistas que sin ser de alto nivel quieran entrenarse en unas condiciones óptimas.

Esta aplicación se integrará con el equipo de medida de velocidad, diseñado como Trabajo Final de Grado de otro compañero, siendo el conjunto de ambos trabajos una colaboración conjunta para el desarrollo del equipo, lo que requiere una gran interrelación entre ambos TFG's, que permitirá aprender a trabajar en equipo y a la vez, servirá para aplicar los conceptos estudiados en la carrera.

1.3 Descripción del equipo de medida de velocidad.

El equipo de medida constará de sensores se colocarán en la pista a la distancia que se desee, marcado el punto de partida y el punto de meta. Cada sensor estará conectado a un microcontrolador, comunicándose entre ellos por RF; y a través del módulo Bluetooth con el móvil, que soportará la aplicación con la que se controlará el equipo y se gestionará la información proporcionada por los sensores; para poder calcular las medidas y realizar las estadísticas.

Capítulo 2. Objetivos

2.1 Justificación.

La justificación del presente trabajo son dos: la primera es de carácter académico y es la obtención del título de Grado en Ingeniería y Servicios de las Telecomunicaciones; la segunda de carácter técnico y social. La justificación técnica porque se trata de desarrollar una aplicación cómoda y amigable para el usuario, ya que controlará el equipo de medida solo con el dispositivo móvil; y social porque se pretende abaratar un producto existente en el mercado y hacerlo más asequible económicamente.

2.2 Objetivos.

El objetivo principal de este proyecto es desarrollar una aplicación en Android sencilla e intuitiva, para controlar un sistema de medida de velocidad en carreras de atletismo, portátil, alimentado por baterías.

A continuación, se presentan los objetivos específicos que deberá cumplir la aplicación:

- Deberá permitir que se introduzca la distancia que se pretende correr y la de los sensores.
- Comprobar que los sensores están bien posicionados.
- Apagar los sensores cuando no se usan para ahorrar consumo y evitar que se descarguen las baterías.
- Un sistema de usuarios con un inicio de sesión y su correspondiente registro, utilizando la Autenticación de Firebase, también dispondrá de la opción de recuperar contraseña.
- Conexión con el sistema de medida mediante Bluetooth, previamente se tendrá que vincular con el dispositivo desde el teléfono móvil.
- Pantalla donde registrar las carreras, se mostrará un cronómetro con las funciones de inicio, pausa y reinicio. Con el botón inicio comenzará el entrenamiento y en tiempo real se irán mostrando las marcas de tiempo enviadas por el sistema de medida. Se finalizará automáticamente la carrera cuando se reciba la última marca.
- Guardar en la base de datos los datos de la carrera una vez finalizada.
- Desde el menú Historial se podrán visualizar todas las carreras guardadas, y ver más en detalle los resultados
- Persistencia de datos, aunque no se disponga de conexión a internet, se guardarán en el teléfono móvil y en cuanto se reestablezca se subirá automáticamente.
- Posibilidad de elegir el idioma de la aplicación entre castellano e inglés.

Capítulo 3. Metodología

Este proyecto tal como se ha comentado en la introducción, forma parte de un trabajo conjunto, y que requiere de una gran colaboración y una correcta comunicación y sincronización entre ellos. El equipo completo requiere por una parte de un diseño Hardware y su implementación y por otra parte de una aplicación para móvil que permita al deportista el uso del equipo.

La aplicación se desarrollará para Android. En concreto se realizará con Android Studio, programa visto en Aplicaciones Telemáticas.

Las fases del trabajo son las siguientes:

- Elección de una base de datos en tiempo real, que permita su acceso a ella en todo momento. Siendo la elegida Realtime Database de Firebase.

- Estructurar la Base de Datos.

- Inicio de sesión con la autenticación de Firebase y registro de usuarios.

- Sistema de comunicación mediante conexión Bluetooth entre el microcontrolador del equipo y el sistema operativo Android del dispositivo móvil.

- Configuración de los comandos necesarios para el entendimiento entre dispositivos.

- Pantalla para medir las actividades.

- Pantalla para visualizar las actividades realizadas.

- Implementar la elección de los idiomas.

- Depuración de errores.

Capítulo 4. Firebase

4.1 Introducción a Firebase

Firebase es una plataforma para el desarrollo de aplicaciones web y móviles desarrollada por Google. Tiene distintos servicios como la autenticación, diferentes bases de datos, hostings, machine learning, analíticas… Se puede disponer de estos recursos mediante varios planes, uno de ellos, el plan Spark, permite utilizar algunas de las funcionalidades de forma gratuita con límite de uso. El otro plan en cambio, se paga por uso, esto viene bien cuando se necesita utilizar una gran cantidad de datos. En este proyecto se utilizará el plan Spark porque para el funcionamiento de la aplicación es suficiente. También cabe destacar que desde el propio Android Studio esta implementado Firebase y se pueden implementar las dependencias y librerías de forma automática.

4.2 Servicios usados

De todas las funcionalidades que ofrece Firebase, las que se necesitan en este caso son: la Authentication, para que haya un inicio de sesión robusto, y una base de datos para poder guardar toda la información. Existen varias versiones de base de datos, ambas no SQL, Realtime Database o Cloud Firestore. Cloud Firestore es el servicio más reciente y tiene más ventajas:

- Aloja los datos en varios centros de datos de distintas regiones, lo que garantiza una escalabilidad global y una confiabilidad sólida.
- Permite escribir datos a través de operaciones de configuración y actualización, así como transformaciones avanzadas, como operadores de arreglos y numéricos.
- Las transacciones pueden leer y escribir datos autómicamente desde cualquier parte de la base de datos.
- Puedes encadenar filtros y combinar filtrado con ordenamiento según una propiedad en la misma consulta.

Todas estas ventajas y mucho más están muy bien, pero hay una desventaja que me hace decantar por Realtime Database. En el plan Spark, ofrece un servicio gratuito por GB almacenados y descargados, para esta aplicación no se requiere mucha capacidad. En cambio, en Cloud Firestore la limitación es el número de consultas a realizar.

4.3 Authentication

Firebase Authentication proporciona servicios de backend, SDK fáciles de usar y bibliotecas de IU ya elaboradas para autenticar a los usuarios en su app. Admite la autenticación mediante contraseñas, números de teléfono, proveedores de identidad federada populares, como Google, Facebook y Twitter, y mucho más.

La función FirebaseUI Auth es la que se encarga de realizar la autenticación de los usuarios. También tiene la funcionalidad de recuperación de cuentas en el caso de olvidar la contraseña, mandando un correo electrónico con un enlace para poner una nueva. También es posible mediante SMS.

Cuando un usuario se crea una cuenta, automáticamente se le asigna un Id de Usuario. Ese Id se puede consultar en cualquier momento y será de gran ayuda para la implementación de la base de datos.

Firebase console permite modificar que proveedores de inicio de sesión se utilizan, dominios autorizados y modificar los templetes de los correos enviados.

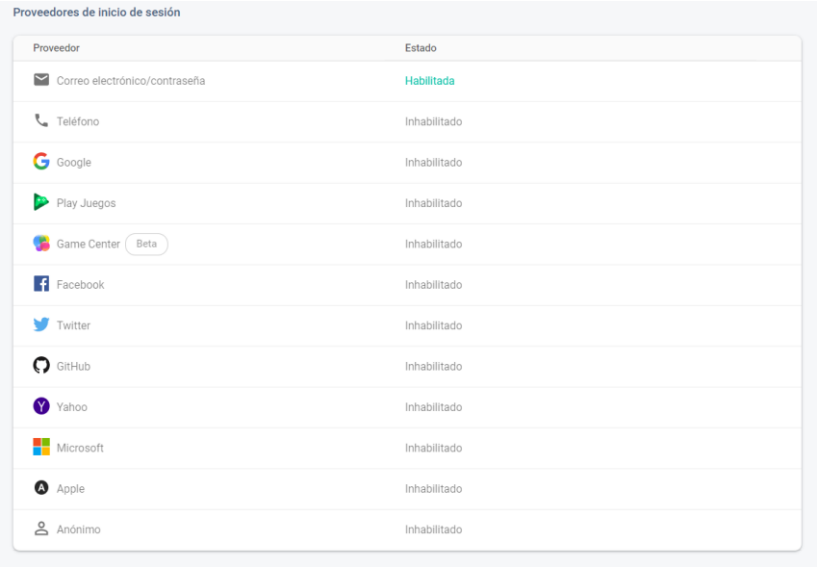

Figura 1. Proveedores de inicio de sesión

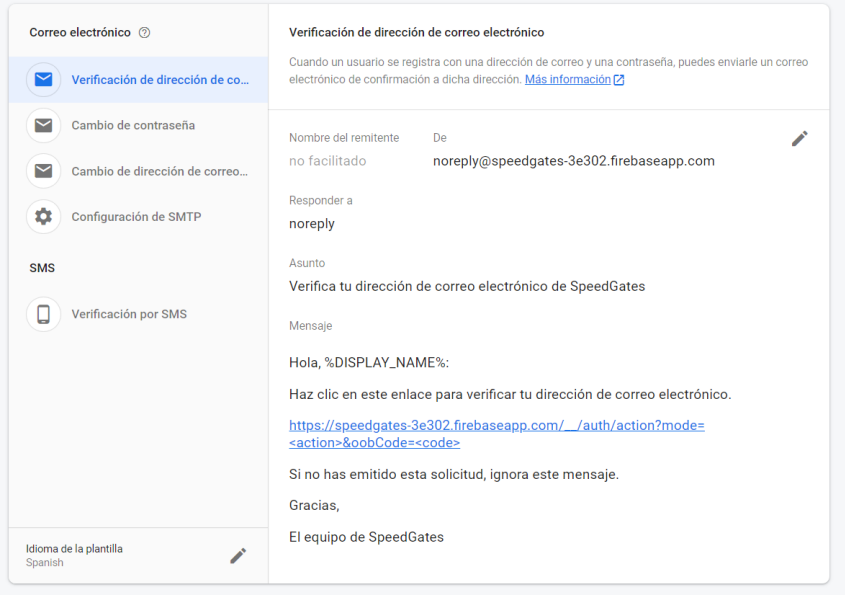

Figura 2. Templetes correos

4.4 Realtime Database

Realtime Database es una base de datos no SQL, alojada en la nube con estructura de datos JSON. Los datos se sincronizan en tiempo real con los usuarios conectados; todos los usuarios comparten una instancia de la base de datos y reciben actualizaciones automáticamente con los datos más recientes. Las funciones clave de Realtime Database son:

- Tiempo real: realiza la sincronización de datos, los usuarios conectados reciben dicha actualización en milisegundos.
- Sin conexión: aunque no se disponga de conexión a internet los datos persisten en el dispositivo. Una vez se tenga conexión, los datos se sincronizarán con la base de datos; es decir, se subirán y descargarán los datos que estaban en espera.
- Acceso desde dispositivos cliente: se puede acceder a la base de datos directamente desde un dispositivo móvil o un navegador web, no es necesario un servidor de aplicaciones. La seguridad y validación de los datos están disponibles mediante reglas de seguridad, reglas basadas en expresiones que se ejecutan cuando se leen o se escriben.
- Escalamiento en varias bases de datos: mediante el plan Blaze (de pago) se pueden satisfacer las necesidades de datos dentro del mismo proyecto de Firebase.

4.4.1 Estructura JSON

La estructura JSON es un formato ligero para la representación de datos en ficheros de texto, son fáciles de leer y escribir tanto por un usuario como por un programa. Es una alternativa a XML debido a su sencillez a la hora de leer los datos, también ofrece una alta velocidad de procesamiento.

JSON puede representar dos tipos de estructuras de datos:

- Un conjunto de pares, los cuales corresponden a la clave y el valor. Separados por el carácter ":".
- Un conjunto ordenado de valores entre corchetes y separados por una coma.

Requiere ser cuidadoso a la hora de introducir los datos, una coma o dos puntos mal ubicados pueden hacer que no funcione correctamente. También cabe destacar que distintos archivos JSON, pueden contener la misma información, pero estructurada de forma distinta.

A continuación, se muestra una estructura simple de datos:

```
{
 "Nombre": "Luis",
"Edad": 23,
 "Aficiones": ["Deporte", "Cine", "Videojuegos"],
"Residencia": "Valencia"
}
```

Figura 3. JSON simple

Se pueden realizar estructuras más complejas incluyendo la forma de padres e hijos. En los que se crea un array donde van anidados los conjuntos de valores. A continuación, se muestra un ejemplo más complejo donde se aplica el sistema de padres e hijos.

```
{
 "trabajadores":
  [
     {
     "Nombre" : "Elena",
     "Edad": 26,
     "Aficiones": ["Música", "Cine"],
     "Residencia": "Valencia"
     },
     {
     "Nombre" : "Luis",
     "Edad": 23,
     "Aficiones": ["Deporte", "Cine", "Videojuegos"],
     "Residencia": "Valencia"
     }
  ]
}
```


Figura 4. JSON complejo

Existe la posibilidad de tener varios archivos JSON con diferentes estructuras, pero que contengan la misma información.


```
{
   "arrayColores":[{
         "nombreColor":"rojo",
         "valorHexadec":"#f00"
      },
      {
         "nombreColor":"verde",
         "valorHexadec":"#0f0"
      },
      {
         "nombreColor":"azul",
         "valorHexadec":"#00f"
      },
      {
         "nombreColor":"cyan",
         "valorHexadec":"#0ff"
      },
      {
         "nombreColor":"magenta",
         "valorHexadec":"#f0f"
      },
      {
         "nombreColor":"amarillo",
         "valorHexadec":"#ff0"
      },
      {
         "nombreColor":"negro",
         "valorHexadec":"#000"
      }
   ]
}
```

```
{
   "arrayColores":[{
         "rojo":"#f00",
         "verde":"#0f0",
         "azul":"#00f",
         "cyan":"#0ff",
         "magenta":"#f0f",
         "amarillo":"#ff0",
         "negro":"#000"
      }
   ]
}
```

```
{
   "rojo":"#f00",
   "verde":"#0f0",
   "azul":"#00f",
   "cyan":"#0ff",
   "magenta":"#f0f",
   "amarillo":"#ff0",
   "negro":"#000"
}
```


Figura 5. JSON diferentes estructuras

No hay una estructura mejor que otra, cada una tiene sus ventajas e inconvenientes, dependiendo de las necesidades de la aplicación una será mejor que otras en cuanto a la facilidad de acceder a los datos y el nº de consultas que se tienen que realizar.

4.4.2 Estructura de la aplicación

La aplicación necesitará almacenar diversos datos de los usuarios, nombres, correos, historial de las carreras, configuraciones de los sensores. Al tratarse de una base de datos no SQL, se necesita

estructurar la base de datos en función de los usuarios, para poder realizar más adelante consultas de datos de forma sencilla. El servicio de autenticación, mencionado anteriormente, proporciona un Id de usuario de forma automática, dicho Id es el que se utilizará como clave para los usuarios.

Identificador	Proveedores	Fecha de creación	Inicio de sesión	UID de usuario
luis@gmail.com		25 ago. 2020	25 ago. 2020	XZcCfUtMf2Vc9kOeOCjwBdWUwK...
jose@gmail.com		7 ago. 2020	21 ago. 2020	xd3kOWziWUMRQ1hX93CI1LZKW...

Figura 6. Usuarios Authentication

Figura 7. Usuarios Database

Como se puede observar, el Id se usa como clave del usuario. Como datos se tiene el Email y el nombre. Las configuraciones de los Arduinos se harán de manera similar.

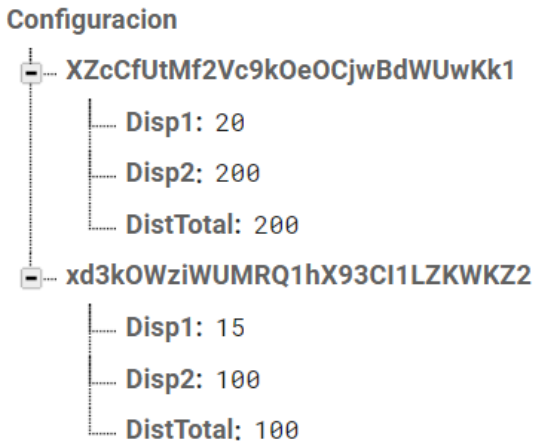

Figura 8. Configuración sensores

Se utilizará como clave de Configuración el Id del usuario, para poder realizar las búsquedas de la manera más sencilla. En la configuración se guardará la distancia a la que se encuentran los diferentes sensores respecto la línea de meta. También se guardará la distancia total del

entrenamiento, aunque deberá ser igual a la distancia del último sensor, que será el que dicta cuando el corredor ha pasado por la línea de meta.

Para guardar los historiales de las carreras, se hará de forma algo diferente; se usará como hijo de Historial el Id del usuario, y a continuación otro hijo con un nº aleatorio, que es el identificativo único de cada carrera. De esta forma las carreras estarán asignadas a un usuario en concreto, esto ayudará a simplificar el muestreo de las mismas.

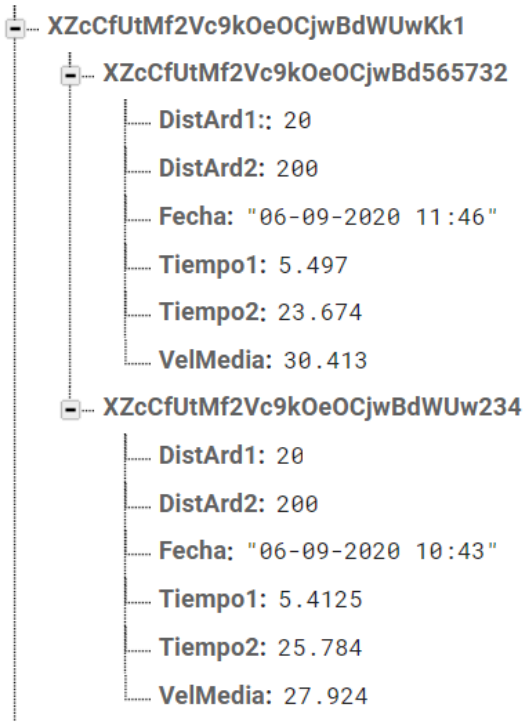

Figura 9. Historial de carreras

Capítulo 5. Aplicación Android

5.1 Descripción de la aplicación

La aplicación dispondrá de una un inicio de sesión, en el que se utilizará Firebase y sus servicios Authentication y Realtime Database. A continuación, constará de una pantalla en la que se mostrarán distintas opciones: start, historial e idiomas. Start es la actividad más compleja de todas, se encargará de conectar mediante Bluetooth el móvil con el sistema de medida. Pulsando el botón de Start comenzará una cuenta atrás enviándole el mensaje Epoch, y se encargará de recibir los datos y mostrar los tiempos por la pantalla. También se podrá acceder a la pantalla para configurar las distancias de los sensores. Y saltando a la actividad historial, se podrán visualizar todos los entrenamientos realizados, de gran utilidad para poder ver el progreso del atleta. En idiomas se podrá elegir entre castellano e inglés, pulsado la bandera correspondiente se cambiará el idioma de la aplicación y retornará a la pantalla principal.

5.2 Entornos de desarrollo integrado

Para poder desarrollar una aplicación para móvil capaz de cumplir las expectativas del proyecto, se necesita el respaldo de un IDE (entorno de desarrollo integrado). Hoy en día existen diversas opciones. Todas ellas sin necesidad de adquirirlas mediante pago. A continuación, se describirán algunas de las disponibles sin coste alguno.

- *Android Studio*: Es una gran opción de cara a desarrollar servicios y aplicaciones. Es el IDE oficial creado por Google y hoy en día es el programa con mayor documentación, muy importante a la hora de aprender. Se pueden encontrar nuevas librerías que se adapten a tu proyecto. Poco a poco va mejorando la fluidez de la misma. Se desarrolla en Java utilizando referencias al Android SDK (Sofware Development Kit), lo cual permite visualizar la interfaz desde una vista de diseño, lo cual resulta muy útil.

- *Eclipse*: Era el IDE que más destacaba para el desarrollo de aplicaciones. Fue la opción oficial que recomendaba Google hasta que le reemplazo Android Studio. Se recomienda utilizar este último frente a Eclipse. La configuración de Eclipse y el flujo de trabajo son muy similares a los de Android Studio. Está pensado para desarrollar en varias plataformas e idiomas, hace que la experiencia de uso sea más lenta y tienden a generan más errores.

- *Xamarin*: Para realizar aplicaciones multiplataforma en las que el código sea el mismo tanto para iOS, como para Windows, Xamarin es el IDE indicado. Creado por Microsoft, viene incluida en Visual Studio. El programa se desarrolla en C#, si bien es cierto que se aleja de la experiencia de desarrollo nativo para Android, es una alternativa a contemplar para aquellos que dominen dicho lenguaje de programación. Dispone de un framework multiplataforma basado en el framework XNA de Microsoft, muy interesante para crear juegos 2D y 3D.

- *AIDE*: Se diferencia con el resto de entornos mencionados dado que se ejecuta directamente en el dispositivo Android. De esta manera es posible desarrollar una aplicación desde el mismo, y no tener que disponer de otro equipo para la creación del proyecto. Su funcionamiento es similar a Android Studio y Eclipse. Incluso existe la posibilidad de trasladar dichos proyectos a este último. Cuenta con bastantes tutoriales para crear aplicaciones sencillas, pero para lecciones más avanzadas tendremos que recurrir a métodos de pago.

- *Pyton*: Popular lenguaje de programación, muy usado para crear aplicaciones Android. Para desarrollarlas se puede hacer mediante librerías tales como PyMob o pgs4a (Pygame Subset for Android). Lenguaje muy accesible y fácil de aprender, considerado muy elegante por muchos desarrolladores. Para aplicaciones complejas no es el programa más recomendable ya que no tiene tantas funcionalidades como Android Studio.

- *B4A*: Es el IDE más recomendable si se busca sencillez y rapidez, no hace falta el uso de Java, se utiliza el lenguaje llamado BASIC (Beginners All Purpose Symbolic Instruction Code), muy fácil de aprender ya que es muy parecido al inglés. Se pueden acceder a las mismas APIs y librerías de Android Studio. Tiene la capacidad de portar fácilmente a un proyecto de B4i, con lo que da la posibilidad de desarrollar aplicaciones para iOS.

Hay más opciones de las mencionadas anteriormente, pero estas son las más utilizadas. De todos estos programas se ha elegido para desarrollar la aplicación Android Studio, por ser el que más documentación tiene, hay muchos tutoriales en foros, en YouTube… Una de las características que destacan, es su integración nativa con GitHub, gracias a ello se podrán guardar las distintas versiones de la aplicación y la elección de las mismas de forma rápida.

5.3 Android Studio

En este apartado se explica el funcionamiento del Programa Android Studio de forma básica, que será el utilizado para el desarrollo de la aplicación objeto de este trabajo. Para empezar, se habrá que crear un nuevo proyecto, y lo que se suele recomendar es seleccionar un modelo de actividad vacía.

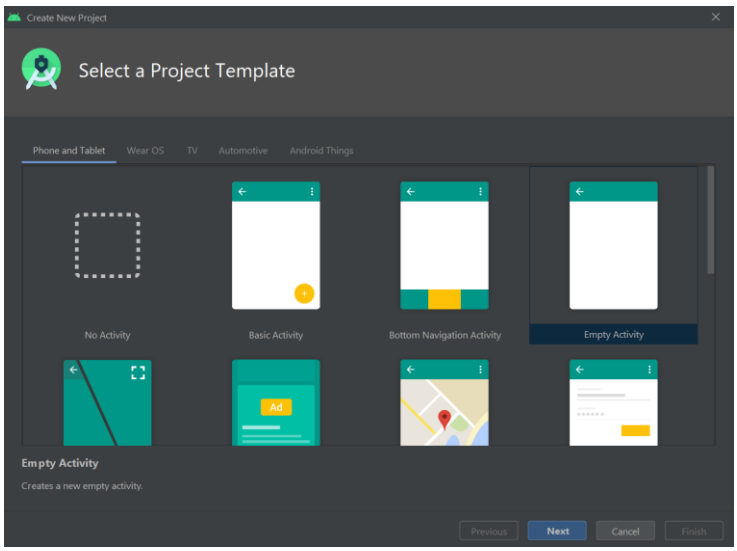

Figura 10. Modelo del proyecto

A continuación, se rellenarán unos campos sobre el proyecto: el nombre del proyecto, el del paquete a utilizar, la localización donde guardarlo, lenguaje de programación, existiendo la posibilidad de realizarlo en Java o Kotlin, este último ha ganado popularidad para desarrollar en Android Studio por su elegancia en los últimos años, por último, la API (Interfaz de programación de aplicaciones) mínima soportada que corresponde con la versión de Android compatible. Hay que tener en cuenta que cuanto más elevado sea el API, menor es el número de dispositivos compatibles.

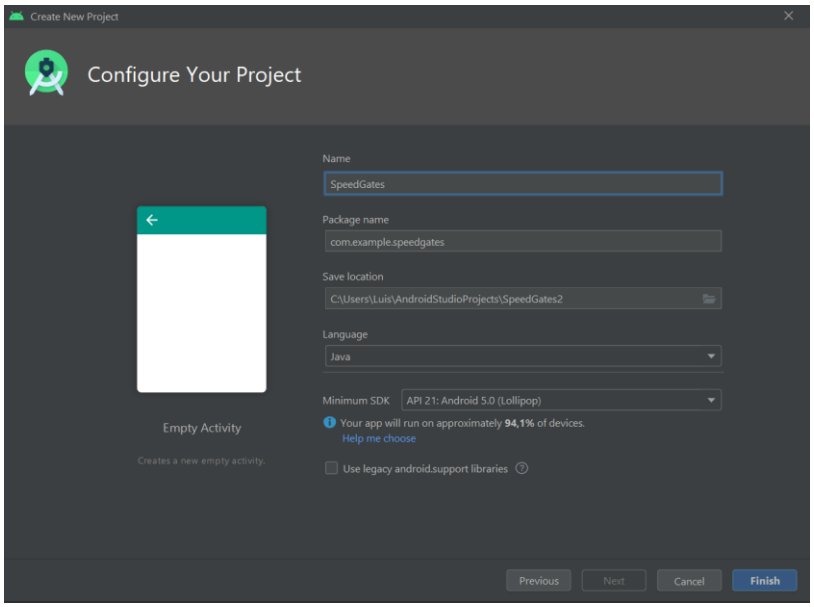

Figura 11. Configuración del proyecto

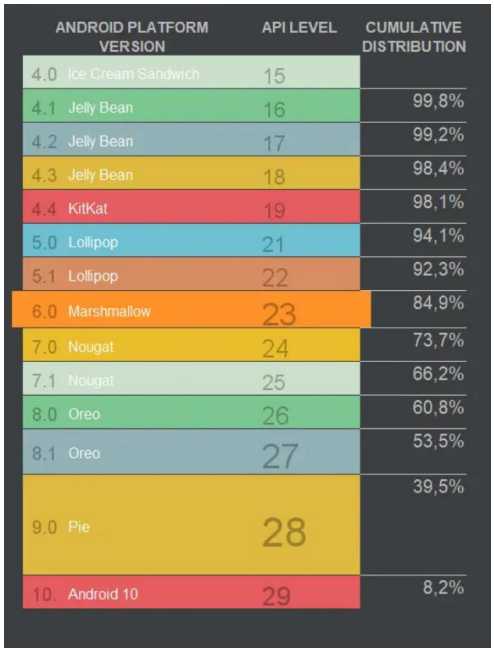

Figura 12. APIs de Android

Disponiendo de esta información, la mejor elección es elegir un API el cual abarque al mayor número de dispositivos, pero sin perder funcionalidades. Para este proyecto se ha elegido el API 5.0 Lollipop, el cual permite ser compatible con el 94,1% de los dispositivos en uso.

Una vez hecho esto ya estará listo para empezar con el proyecto. Se muestra la pantalla principal de Android Studio.

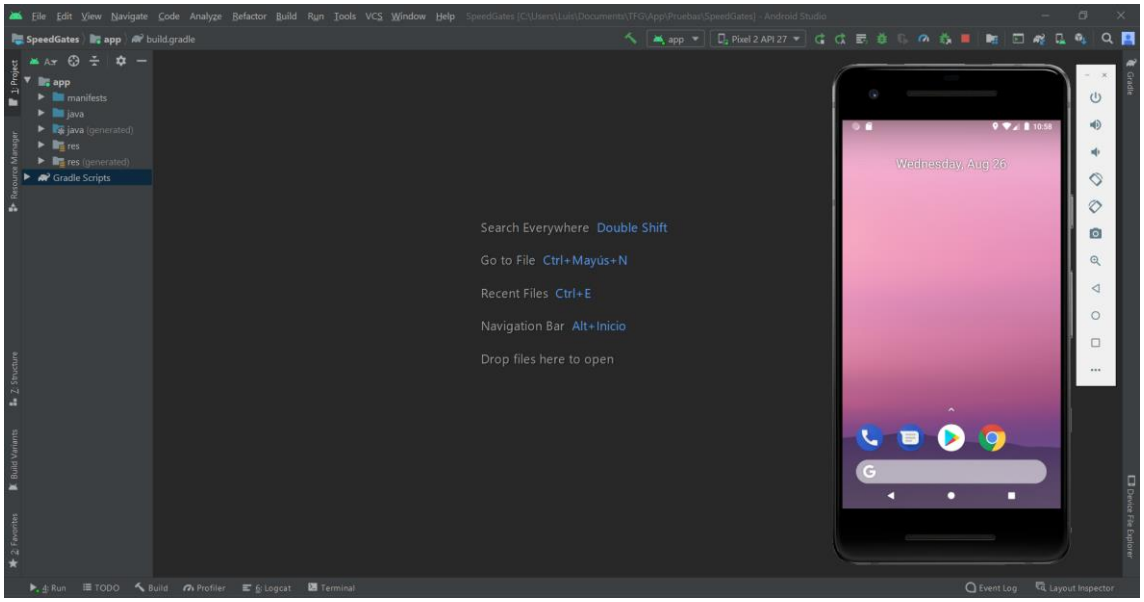

Figura 13. Pantalla principal Android Studio

Por defecto, en la parte izquierda de la pantalla del programa se encuentra la estructura en carpetas del proyecto. Se describirán a continuación las más importantes.

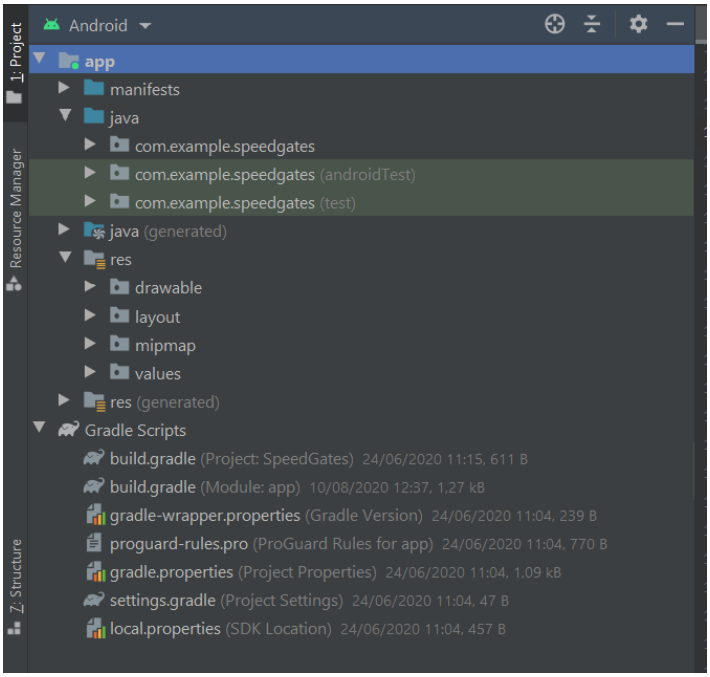

Figura 14. Estructura de las carpetas

En la carpeta *Manifest* se halla el fichero AndroidManifest.xml, es el archivo manifiesto donde se describen la información esencial de la aplicación para las herramientas de creación de Android, el sistema operativo Android y Google Play.

En la carpeta *Java*, seguido de otra con el nombre del paquete del proyecto, se encuentran todos los archivos .java del proyecto.

En *res* podremos encontrar distintas carpetas tales como *drawable*, localización para guardar las imágenes que luego utilizaremos en la aplicación, *layaout*, donde se sitúan los archivos .xml correspondientes a la parte visual de las diferentes pantallas. También encontramos la carpeta *values* donde encontramos otros .xml en los que se guardan los colores, Strings que no sean en inglés y estilos, todos esos para poder luego utilizarlos en el proyecto.

Por último están los *build.grandle*, donde indicarán las versión de SDK utilizada, la API y las librerías utilizas en el proyecto.

Los Layaouts se relacionan con las clases de Java, en ellos se diseñan las interfaces de forma intuitiva, gracias al modo Desing. Pudiendo modificar los objetos que salen en la pantalla y ver en el momento la distribución de las mismas. Esto se traduce a xml, donde también se podrán diseñar las de forma escrita.

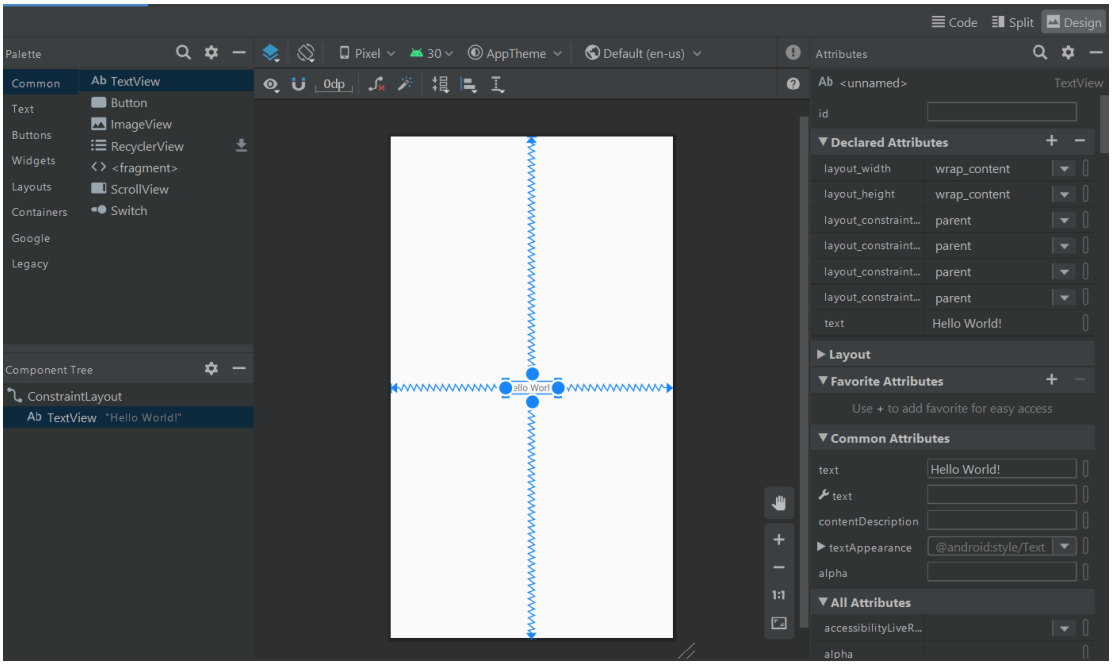

Figura 15. Modo design

En la parte inferior de la pantalla principal, se encuentran las ventanas donde se puede ver el log de compilación, también el log de la aplicación ejecutándose y un terminal. Esas son las más utilizadas.

Figura 16. Ventanas de compilación

5.4 Desarrollo de la aplicación

Las distintas pantallas que compondrán esta aplicación se ha descrito en el apartado anterior. En este apartado se explicará cómo se han desarrollado cada una de ellas y cómo funcionan. Empezando primero por su diseño y luego por la explicación de los métodos.

5.4.1 Login Activity

Como primera pantalla al iniciar la aplicación se mostrará un inicio de sesión donde introducir el correo y la contraseña, con la posibilidad de ir a la actividad de registro y la opción de reestablecer la contraseña.

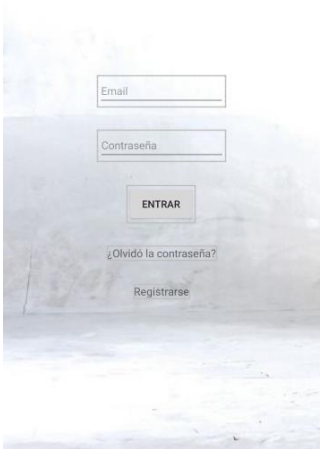

Figura 17. Login Activity

Dispone de dos EditTexts, en los que se introducirán los datos y luego un Button, para una vez introducidos los datos, iniciar la sesión. Si uno de los campos es erróneo, notificará la causa, bien por el email o por la contraseña. Además, hay dos TextViews, uno de ellos para reestablecer la contraseña, el cual mostrará una vista para poder poner el email y enviar un enlace de recuperación. El otro enviará a la actividad de Registrar.

Se declararán todos los objetos necesarios, incluyendo uno de FirebaseAuth llamado fAuth, servirá para utilizar las ventajas de la Autenticación. Luego se relacionarán con los del diseño en el OnCreate. Más adelante se explicarán las funciones de cada método.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_login);

    mEmail = findViewById(R.id.EditTextEmail);
    mPassword = findViewById(R.id.EditTextPassword);
    fAuth = FirebaseAuth.getInstance();
    mLoginBtn = findViewById(R.id.buttonSignIn);
    mCreateBtn = findViewById(R.id.TextToRegister);
    mForgotPass = findViewById(R.id.ForgetPass);
```

Figura 18. Relaciones de objetos Login

En todas las actividades se tendrán que relacionar las variables con los objetos del Layaout, por tanto, en las siguientes actividades, esta parte se obviará.

IsConected:

Encargado de saber si la aplicación ya ha iniciado la sesión previamente. Compara el objeto que devuelve el usuario activo con null. Si no está vacío significa que se podrá pasar directamente a la pantalla principal donde se muestraránn las opciones.

```
private void isConected() {
    if(fAuth.getCurrentUser() != null){
        startActivity(new Intent(getApplicationContext(), MainActivity.class));
        finish();
    }
}
```

Figura 19. Método isConected

LoginBtn:

Método que se activa cuando se presiona el Button de inicio de sesión. Consta de varias partes. La primera de ella comprueba si los campos introducidos son correctos. Devuelve error si el correo o el email está vacío, o si la contraseña es de menos de 6 caracteres.

```
String email = mEmail.getText().toString().trim();
String password = mPassword.getText().toString().trim();

if(TextUtils.isEmpty(email)){
    mEmail.setError("El correo es necesario");
    return;
}

if(TextUtils.isEmpty(password)){
    mPassword.setError("La contraseña es necesaria");
    return;
}

if(password.length() < 6){
    mPassword.setError("La contraseña debe ser de al menos 6 caracteres");
    return;
}
```

Figura 20. Comprobar campos

Si los datos son correctos se utilizará el objeto fAuth para autenticar al usuario. Se le aplicará un addOnCompleteListener, para tratar tanto cuando sale bien como cuando ocurre un error.

fAuth.signInWithEmailAndPassword(email,password)

Para escribir el correo de la contraseña olvidada se utilizará un AlertDialog. Se mostrará el campo para introducirlo y un botón para enviar.

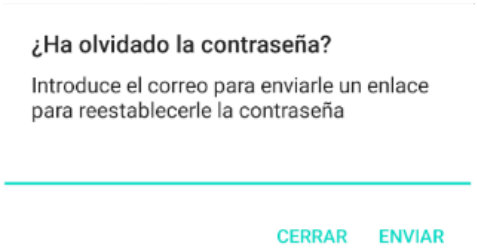

Figura 21. Reestablecer la contraseña

5.4.2 Registro Activity

Esta actividad es parecida a la de Login, tiene tres EditTexts donde introducir los datos de registro: nombre, correo electrónico y contraseña. Un TextView encargado de dirigirnos a la actividad Login y luego un Button para enviar los datos a Authentication para crear el usuario y a RealTime Database para guardarlo. A continuación, se mostrará la pantalla y se explicarán los métodos usados.

Figura 22. Registro Activity

isConected:

Es el mismo método que en la actividad anterior, revisa si hay un usuario activo, en ese caso afirmativo se redirige a la pantalla principal.

Registrar:

Este método es parecido a LoginBtn. Su función es recoger los datos introducidos, comprobar si son correctos. Después de eso, utilizará el objeto Auth para crear los usuarios usando también addOnCompleteListerner. Se llamará a un método para guardar al usuario en la base de datos.

fAuth.createUserWithEmailAndPassword(email,password)

GetUid:

Método auxiliar que devuelve el Id del usuario que tiene asignado Authentication de Firebase

GuardaUsuario:

Este método utilizará los servicios de Realtime Database de Firebase y también utilizará otra clase modelo llamada Post, que devuelve un HashMap, al pasarle los datos del Usuario. Este HashMap es el que se envía a la base de datos, se hace de esta forma porque ya están ordenados los datos y no hay que introducirlos uno a uno.

```
public Map<String, Object> toMap() {
    HashMap<String, Object> result = new HashMap<>();
    result.put("Email", Email);
    result.put("Nombre", Nombre);

    return result;
}
```

Figura 23. HashMap

Para enviar los datos se tendrá que utilizar un objeto de la clase DatabaseReferece, con nombre myRef, a este se le tendrá que decir en que parte de la base de datos se desea guardar el HashMap. Tal como se explicó en la estructura de la aplicación, tendrá que ser guardado en Usuarios. A continuación, se pondrá el id de usuario llamando a getUid. Y finalmente se insertarán los datos.

```
private void guardaUsuario(String Email, String Nombre) {

    Post post = new Post(Email, Nombre);
    Map<String, Object> postValues = post.toMap();
    myRef.child("Usuario").child(getUid()).setValue(postValues);
    }
```

Figura 24. Método guardaUsuario

5.4.3 Main Activity

Desde esta sencilla actividad se podrá acceder las actividades de Start, historial e idiomas. Se dispondrá de 3 Buttons para cambiar de actividad, y otro más para cerrar la sesión del usuario utilizando el objeto Auth.

fAuth.signOut()

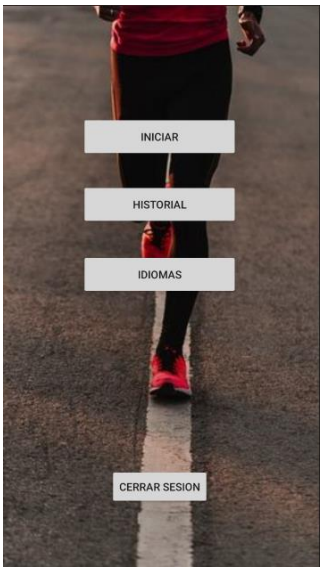

Figura 25. Main Activity

5.4.4 Start Activity

Esta es la actividad principal de la aplicación y la más compleja de todas. Desde ella se podrán medir los entrenamientos. Para ello lo primero es pulsar el ImageButton con el símbolo del Bluetooth, redirigirá DispositivosVinculados, donde se podrá conectar con los dispositivos vinculados previamente. Otro ImageButton servirá para redirigir a la actividad StartSettings para poder configurar distintos parámetros, más adelante se explicarán estas dos actividades.

Figura 26. Start Activity

Se mostrará un cronometro con precisión de milisegundos que dispondrá de tres Buttones: start, pause y restart. También habrá cuatro TextView, dos de ellos mostrarán las marcas de tiempo

obtenidas por el sistema de medida, el tercero indicará la velocidad y el último servirá para mostrar una cuenta atrás cuando se inicie la carrera.

Hasta que no esté conectado el equipo con el dispositivo, no dejará pulsar ninguno Button. Habrá un modo de alineación mediante un Switch, se tendrá que poner el modo Alinear, para el correcto funcionamiento de la aplicación los sensores tienen que estar perfectamente alineados con los catadióptricos, ya que, si los sensores no estuvieran detectando el haz reflejado, considerarían que el haz de luz está interrumpido todo el rato y no sería posible medir la actividad. Por ahorro de energía hasta que no reciben la orden de Prestart no se deben activar los sensores. A continuación, se explicarán los métodos utilizados:

OnResume:

Después de pulsar el símbolo de Bluetooth, se obtendrá la dirección MAC del arduino Master de la actividad DispositivosVinculados, cuando se regreses a la actividad se entrará en el modo onResume, el cual estará listo para crear un objeto BluetoothDevice (device), utilizando un BluetoothAdapter (btAdapter) y la MAC obtenida. De esta forma se conectarán el móvil con el equipo de medida. Luego se creará un objeto de la clase ConnectedTherad (MyConexionBT), pasándole el objeto btSocket. Este constructor será el encargado de recibir y enviar los datos por Bluetooth, también se utilizará la clase Message mediante un Handler para caracterizar los estados del Bluetooth: Escuchando, conectando, conectado, conexión fallida y mensaje recibido.

```
public void onResume()
{
    super.onResume();

    Intent intent = getIntent();
    address = intent.getStringExtra(DispositivosVinculados.EXTRA_DEVICE_ADDRESS);
    if(address==null){
        Toast.makeText(getBaseContext(), text: "No se ha conectado a ningun" +
                " dispositivo", Toast.LENGTH_LONG).show();
    }else {
        //Setea la direccion MAC
        BluetoothDevice device = btAdapter.getRemoteDevice(address);

        try {
            btSocket = createBluetoothSocket(device);
        } catch (IOException e) {
            Toast.makeText(getBaseContext(),
                    text: "La creacción del Socket fallo", Toast.LENGTH_LONG).show();
        }
        // Establece la conexión con el socket Bluetooth.
        try {
            btSocket.connect();
        } catch (IOException e) {
            try {
                btSocket.close();
            } catch (IOException e2) {
            }
        }

        if (btSocket.isConnected()) {
            MyConexionBT = new ConnectedThread(btSocket);
            MyConexionBT.start();
            Toast.makeText(getBaseContext(),
                    text: "Conexion Realizada", Toast.LENGTH_SHORT).show();
            Message message = Message.obtain();
            message.what = STATE_CONNECTED;
            handlerBT.sendMessage(message);
        } else{
            Toast.makeText(getBaseContext(),
                    text: "La conexión falló", Toast.LENGTH_SHORT).show();
            Message message = Message.obtain();
            message.what = STATE_CONNECTION_FAILED;
            handlerBT.sendMessage(message);
        }
    }
}
```

Figura 29. Método onResume

ConnectedThread:

Esta clase es muy importante para la utilización del Bluetooth. Contiene varios constructores dentro de ella, por eso se le aplica extends Thread. El primer método se encarga de crear dos buffers, un InputStream y un OutputStream, y los relaciona con el socket que le pasará el onCreate. Se utilizarán para enviar y recibir los datos del Bluetooth. Los dos métodos que se encargarán de hacerlo son:

Run:

Encargado de recibir los mensajes, se mantiene escuchando al InputStream mientras la conexión esté activa. Cuando entra un mensaje por el Bluetooth se manda al handlerBT el mensaje recibido y el case del handler, en este caso será STATE_MASSAGE_RECEIVED debido a que ha recibido un mensaje. Más adelante se explicará este case, que es el más complejo de los que hay.

```
//Recibir datos del Bluetooth
public void run()
{
    // se mantiene escuchando el InputStream mientras el socket está conectado
    while (true) {
        try {
            byte[] buffer = new byte[128];
            String readMessage;
            int bytes;
            if (mmInStream.available()>2) {
                try {
                    // Lee del InputStream
                    bytes = mmInStream.read(buffer);
                }catch (IOException e) {
                    Log.e(TAG,  msg: "disconnected", e);
                    break;
                }
                // Envia los bytes obtenidos al handler
                handlerBT.obtainMessage(STATE_MESSAGE_RECEIVED, bytes, arg2: -1,buffer).sendToTarget();
            }
            else {
                SystemClock.sleep( ms: 100);
            }
        } catch (IOException e) {

            e.printStackTrace();
        }
    }
}
```

Figura 30. Método run del bluetooth.

Write:

Se encarga de escribir en el OutStream el mensaje, para que directamente se envíe por Bluetooth. Todos los mensajes enviados tienen que tener el formato byte array para mandarse correctamente. En caso de producirse un error se mostrará un Toast comentando que la conexión falló.

```
public void write(byte[] datos) {
    try {
        mmOutStream.write(datos);
    }
    catch (IOException e) {
        //si no es posible enviar datos se cierra la conexión
        Toast.makeText(getBaseContext(), text: "La Conexión fallo", Toast.LENGTH_LONG).show();
        finish();
    }
}
```

Figura 31. Método write

ModoAlinear:

Con el método modo alinear permitirá encender los sensores, para posicionarlos correctamente. El móvil estará escuchando por si recibe algún paquete por Bluetooth, si pasa eso alguno de los sensores estará mal colocado. Los sensores tienen una luz indicando que está correctamente posicionado, lo cual facilita bastante la alineación. Cuando se sale del modo alinear, hay que volver a apagar los sensores y estará listo el equipo para empezar la carrera.

```
private void ModoAlinear() {
    mModoAlinear.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if (mModoAlinear.isChecked()){
                final String StringConfig = "33";
                byte[] Config = StringConfig.getBytes();
                MyConexionBT.write(Config);
                contador++;
            }else{
                String stringReset ="22";
                byte[] Reset = stringReset.getBytes();
                MyConexionBT.write(Reset);
                contador=0;
            }
        }
    });
```

Figura 32. Método ModoAlinear

IniciarCarrera:

Cuando se pulse el botón start, primero comprobará si está el modo alinear activo, en caso afirmativo no permitirá pulsarlo. Si ha sido pulsado anteriormente no dejará tampoco hasta que no se pulse el Button reinicio. Si no se cumplen ninguna de las anteriores se enviará un mensaje de PreStart por el Bluetooth gracias al método write. Este mensaje consistirá en un String con el número “11”, el cual se transformará a byte array para el correcto envío. Con este mensaje el dispositivo de medida activará los sensores para que estén listos para el entrenamiento.

A continuación, se pondrá en visible el TextView del contador y se iniciará una cuenta atrás llamando al método IniciarCuentaAtrás.

```
private void IniciarCarrera() {
    mStart.setOnClickListener((view) -> {
            //Bluetooth
            if(status.getText()=="Conectado") {
                if (contador == 0) {

                    final String StringPreStart = "11";
                    byte[] PreStart = StringPreStart.getBytes();
                    MyConexionBT.write(PreStart);
                    mContador.setVisibility(View.VISIBLE);
                    inciarCuentaAtras();
                } else {
                    if (mModoAlinear.isChecked()) {
                        Toast.makeText( context: Start.this,  text: "Quite el modo de prueba si quiere empezar la actividad", Toast.LENGTH_SHORT).show();
                    }
                    else{
                        Toast.makeText( context: Start.this,  text: "Restablezca la carrera si quiere empezar de nuevo", Toast.LENGTH_SHORT).show();
                    }
                }
            } else {
                Toast.makeText( context: Start.this,  text: "Conecte con el dispositivo", Toast.LENGTH_SHORT).show();
            }
    });
}
```

Figura 33. Método IniciarCarrera

IniciarCuentaAtras:

Este método es el encargado de mostrar una cuenta atrás en la pantalla de la aplicación. Esto se consigue gracias al elemento CountDownTimer, a este le asignan dos parámetros. El primero de ellos es la duración total de la cuenta atrás. El segundo es el intervalo de ejecución del método onTick. Cuando se finaliza la cuenta atrás se ejecutará el método onFinish. En este caso se le asignarán los valores de 3000 y 1000 respectivamente, expresado en milisegundos. A continuación, se explicarán las acciones de los métodos mencionados:

OnTick:

Encargado de mostrar por pantalla los segundos restantes de la cuenta atrás. Esto lo realizará cogiendo el parámetro de milisegundos restantes y una división entre 1000 para obtener los segundos.

OnFinish:

Una vez finalizada la cuenta atrás este método se encargará de realizar las siguientes funciones. En primer lugar, pone el TextView del contador en invisible. Después obtiene el Epoch en milisegundos mediante la función Instant.now().toEpochMilli(), este dato está en formato Long, se transformará en String, a continuación en byte array y lo enviará por Bluetooth con el método write.

Por último, tendrá como función iniciar el cronómetro, haciendo uso un constructor llamado runnable al cual le pasará el parámetro del tiempo actual en milisegundos por medio de la función SystemClock.uptimeMillis() con la ayuda de un handler.

```
private void inciarCuentaAtras() {
    new CountDownTimer( millisInFuture: 3000,  countDownInterval: 1000){

        @SuppressLint("SetTextI18n")
        @Override
        public void onTick(long l) {
            long segundosPendientes=(l/1000)+1;
            mContador.setText(Long.toString(segundosPendientes));
        }

        @RequiresApi(api = Build.VERSION_CODES.O)
        @Override
        public void onFinish() {
            mContador.setVisibility(View.INVISIBLE);

            now = Instant.now().toEpochMilli();
            EpochtimeStart = Long.toString(now);
            Epochtime = EpochtimeStart.substring(5).trim();
            Log.d( tag: "Epoch", Epochtime);
            byte[] Epoch = Epochtime.getBytes();
            MyConexionBT.write(Epoch);


            //Chrono
            StartTime = SystemClock.uptimeMillis();
            handler.postDelayed(runnable,  delayMillis: 0);
            mReset.setEnabled(false);
            contador++;
            mContador.setText("3");
        }
    }.start();
}
```

Figura 34. Método iniciarCuentaAtras

Runnable:

Este constructor, como se ha mencionado anteriormente tiene la función de hacer funcionar al cronómetro. Para ello utilizará un método run el cual no tendrá ningún tipo de espera y se ejecutará cada milisegundo. Irá restando el tiempo actual con el tiempo calculado al finalizar la cuenta atrás. Después de ello realizará la conversión a minutos, segundos y milisegundos y lo insertará en el TextView del cronómetro el tiempo.

```
public Runnable runnable = new Runnable() {

    public void run() {

        MillisecondTime = SystemClock.uptimeMillis() - StartTime;
        UpdateTime = TimeBuff + MillisecondTime;
        Seconds = (int) (UpdateTime / 1000);
        Minutes = Seconds / 60;
        Seconds = Seconds % 60;
        MilliSeconds = (int) (UpdateTime % 1000);
        mChronometer.setText("" + Minutes + ":"
                + String.format("%02d", Seconds) + ":"
                + String.format("%03d", MilliSeconds));
        handler.postDelayed( r: this,  delayMillis: 0);
    }
};
```

Figura 35. Método run del cronómetro

Reset:

Método llamado al pulsar el Button de reinicio, este botón solo estará activo si se pulsa previamente el Button de pausar. En primer lugar, se limpiarán todos los TextViews insertando una cadena vacía. A continuación, se reestablecerán todos los valores del cronómetro a "0L", ya que se tratan de datos Long y en el TextView del cronómetro se introducirá una cadena de la siguiente forma: "00:00:00". Para finalizar se creará un mensaje de reinicio con formado string que será tranformado en byte array para enviarlo por el Bluetooth.

```
private void Reset() {
    mReset.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {

            contador=0;

            msg_box.setText("");
            msg_box2.setText("");
            msg_box3.setText("");

            MillisecondTime = 0L ;
            StartTime = 0L ;
            Seconds = 0 ;
            Minutes = 0 ;
            MilliSeconds = 0 ;
            mChronometer.setText("00:00:00");
            String stringReset ="22";
            byte[] Reset = stringReset.getBytes();
            MyConexionBT.write(Reset);

        }
    });
}
```

Figura 36. Método Reset

Pause:

Método ejecutado cuando se pulsa el Button pausar el cronómetro. Se realiza aplicando al handler la función removeCallbacks y pasándole el nombre del constructor, en este caso runnable. A continuación, se activa el Button reinicio.

```
private void Pause() {
    mPause.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            handler.removeCallbacks(runnable);
            mReset.setEnabled(true);
        }
    });
}
```

Figura 37. Método Pausar

HanderlBT:

Mencionado anteriormente, este método tiene un switch con 5 posibles casos.: Escuchando, conectando, conectado, conexión fallida y mensaje recibido. Todos ellos menos el último escribe en un TextView el estado en que se encuentra.

En caso de recibir un mensaje, es muy distinto a los demás estados, se tendrá que comprobar si está el modo alinear activo. En ese caso mostrará un Toast mostrando que algunos de los dispositivos no están correctamente colocados. En caso de recibir un mensaje y no darse la condición anterior, este mensaje contendrá el número del sensor y el Epoch el cual se tendrá que tratar.

En primero lugar se extraerá la información mediante la función substring. Para crear dos String que contengan tanto el número del sensor como el del Epoch. Después de esto se convertirá el número Epoch en un dato entendible, segundos y milisegundo. Esto se realizará restando el Epoch inicial con el recibido, como resultado se obtendrán los milisegundos que han pasado desde el comienzo de la carrera. Con una simple división entre mil se obtendrá el formato deseado.

```
String StringArd = Mensaje.substring(0, 2);
String aux = Mensaje.substring(2).trim();
```

Figura 38. Conversión de datos

```
Config Dispositivos = new Config();
double Disp1 = Dispositivos.getDisp1();
double Disp2 = Dispositivos.getDisp2();
```

Figura 39. Obtención de distancias

La obtención de las distancias, las cuales habrá que introducir previamente en la actividad StartSettings, se realizará mediante una referencia de la base de datos, apuntando al hijo Configuración y luego al Id del usuario. Se aplicará un addListenerForSinglesValueEvent que generará un método onDataChange y otro onCancelled para obtener los resultados.

A continuación, habrá un Switch con dos casos posibles, dependiendo del número del sensor. En el caso de recibir el mensaje del primero de ellos se mostrará en un TextView la marca de tiempo y se guardará en una variable, para poder utilizarla más adelante.

En el segundo case se realizarán bastantes más acciones. En primer lugar, se pausará el cronómetro aplicando removeCallbacks al handler y también se activará el Button de reinicio. A continuación, se mostrará la marca por el TextView y se guardará también la variable. Después se calculará la velocidad media de la carrera, esto se hará cogiendo la distancia del último sensor, siendo la misma que la distancia total de la carrera, dividiéndolo por el tiempo total y multiplicando por 3,6 para tenerlo en kilómetros por hora, y se mostrará en otro TextView. Antes de subir los datos a la base datos, se obtendrá la fecha y hora del entrenamiento mediante un método llamado obtenerfecha. Por último, se subirán todos los datos a la base de datos de Firebase.

```
case 2:
    //Parar chrono
    handler.removeCallbacks(runnable);
    mReset.setEnabled(true);
    msg_box2.setText(Marcadetiempo);
    tiempo2 = Marcadetiempovel;
    contador2 = 0;
    //Calcular VelMedia en Km/h
    double velocidadaux = (Disp2 / Marcadetiempovel)*3.6;
    String velocidad = Double.toString(velocidadaux);
    msg_box3.setText(velocidad);
    //Obtener Fecha
    String Fecha = obtenerfecha();
    //Insertar los datos en el Hashmap
    historial.put( k: "DistArd1", Disp1);
    historial.put( k: "Tiempo1", tiempo1);
    historial.put( k: "DistArd2", Disp2);
    historial.put( k: "Tiempo2", tiempo2);
    historial.put( k: "VelMedia", velocidadaux);
    historial.put( k: "Fecha",Fecha);
    //Subir los datos
    myRef.push().setValue(historial);
    break;
```

Figura 40. Recepción y envío de datos

OnDataChange:

Este método es muy similar al que se verá en la actividad StartSettings. El método recibe un objeto del tipo Datasnapshot, con la ayuda de la clase Config, conformada por setters y getters, guardará en dos variables las distancias de los sensores, para utilizarlas más adelante.

```
public void onDataChange(@NonNull DataSnapshot snapshot) {
    Config config = snapshot.getValue(Config.class);

    if (!(config == null)) {
        Disp1 = config.getDisp1();
        Disp2 = config.getDisp2();
    }
}
```

Figura 42. Recibir datos distancia

OnCancelled:

Método para indicar que la recepción de los datos ha fallado, mostrará un Toast comentando que ha habido algún error de conexión a Internet.

ObtenerFecha:

Este método es utilizado para devolver la fecha y hora actual. Para ello se utilizará un objeto tipo Date, al cual se le asignará un formato (Día - Mes - Año Hora:Minuto). Se guardará en una variable String y devolverá dicha variable.

```
private String obtenerfecha() {
    Date date = new Date();
    DateFormat fechaHora = new SimpleDateFormat( pattern: "dd-MM-yyyy HH:mm");
    String fecha = fechaHora.format(date);
    return fecha;
}
```

Figura 41. Método obtenerfecha

5.4.5 DispositivosVinculados Activity

Esta actividad será la encargada de conectarse con el dispositivo de medida. Se mostrará una pantalla con los dispositivos vinculados que tiene el dispositivo, con sus respectivas direcciones MAC. Tan solo con pulsar al dispositivo deseado intentará conectarse con él. A continuación, se muestra los métodos utilizados.

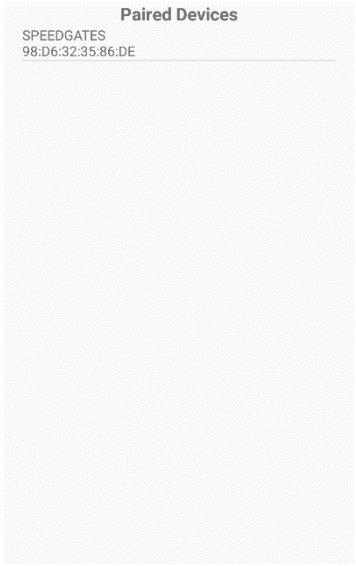

Figura 41. DispositivosVinculados Activity

VerificarEstadoBT:

En primer lugar, se creará un adaptador de Bluetooth; si el adaptador es null significa que el dispositivo tiene un módulo Bluetooth. A continuación, se comprueba si está activo el Bluetooth; si no está activo lanzará un Intent para activarlo. Mostrando por pantalla la figura 42.

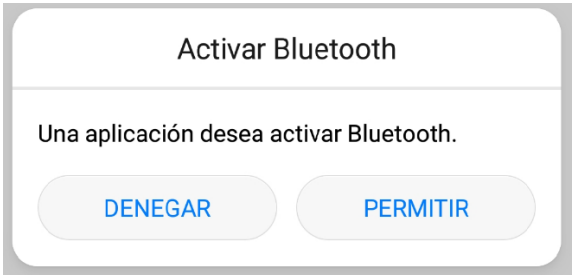

Figura 42. Activar Bluetooth

```
private void VerificarEstadoBT() {
    // Comprueba que el dispositivo tiene Bluetooth y que está encendido.
    mBtAdapter= BluetoothAdapter.getDefaultAdapter();
    if(mBtAdapter==null) {
        Toast.makeText(getBaseContext(), text: "El dispositivo no soporta Bluetooth", Toast.LENGTH_SHORT).show();
    } else {
        if (mBtAdapter.isEnabled()) {
            Log.d(TAG, msg: "...Bluetooth Activado...");
        } else {
            //Solicita al usuario que active Bluetooth
            Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent, requestCode: 1);
        }
    }
}
```

Figura 43. Método VerificarEstadoBT

OnResume:

Este método llamará a VerificarEstadoBT, visto anteriormente. Luego creará un ArrayAdapter, con ayuda de un layaut auxiliar llamado dispositivo_encontrados, dicho array contendrá la lista de los dispositivos bluetooth vinculados. Con la ayuda de un ListView se mostrará por pantalla dicho array de dispositivos. Cuando sea clicada esta lista se activará el método onItemView.

OnItemView:

Método encargado de extraer la información del dispositivo. Se guardará en un String llamado info toda la información del dispositivo, ya que la dirección MAC son los últimos 17 caracteres, con la ayuda del método substring se guardará en otro String llamado address. Si la dirección no es nula, se utilizará un Intent para volver a la actividad de Start, al cual se le añadirá el address extraído.

```
private AdapterView.OnItemClickListener mDeviceClickListener = new AdapterView.OnItemClickListener() {
    public void onItemClick(AdapterView av, View v, int arg2, long arg3) {

        // Obtener la dirección MAC del dispositivo, que son los últimos 17 caracteres en la vista
        String info = ((TextView) v).getText().toString();
        String address = info.substring(info.length() - 17);

        finishAffinity();

        if (address!=null){

        // Realiza un intent para iniciar la siguiente actividad
        // mientras toma un EXTRA_DEVICE_ADDRESS que es la dirección MAC.
        Intent intend = new Intent( packageContext: DispositivosVinculados.this, Start.class);
        intend.putExtra(EXTRA_DEVICE_ADDRESS, address);

        startActivity(intend);}
    }
};
```

Figura 44. Método onItemClick

5.4.6 StartSettings Activity

Esta parte de la aplicación se encargará de recabar la información sobre la distancia de la carrera y la de los sensores. Se muestran 3 EditText, para insertar la distancia total de la carrera y la distancia entre los dos sensores (inicial y final), y dos Buttons. Uno para guardar los datos en la base de datos y en la propia aplicación para usarlos a la hora de calcular los tiempos de la actividad, y otro para regresar a la actividad Start.

Figura 45. Start Settings Activity

Lo primero que hace la actividad es leer en la base de datos si previamente había alguna configuración guardada, si es así, los introducirá dentro de los EditText para que sea visible, y en caso de desear otros datos, simplemente se tendrá que modificar y darle a actualizar. En ese

momento el programa comprobará que los datos son correctos. Comprueba si hay algún campo vacío y la distancia del último sensor y la total de la actividad tendrán que ser iguales, ya que marcará la llegada a meta. Los dos métodos importantes son:

ComprobarDatos:

Método encargado de leer la configuración de la base de datos e introducirlos en los EditText, utilizaremos un addListenerForSinglesValueEvent como en la actividad Start, este evento lee los datos una sola vez, puesto que es algo que no va a estar cambiando constantemente, a no ser que el usuario guarde otros datos. Al usarse, se crear un método onDataChange, el objeto tipo DataSpapshot contendrá todos los datos de la base de datos. Cogerá los datos del snapshot mediante la clase Config.

```
private void ComprobarDatos(DatabaseReference myRef) {
    myRef.addListenerForSingleValueEvent(new ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot) {
            Config config = snapshot.getValue(Config.class);

            if (!(config == null)) {
                DistTotal = config.getDistTotal();
                Disp1 = config.getDisp1();
                Disp2 = config.getDisp2();

                mDistTotal.setText(String.valueOf(DistTotal));
                mDisp1.setText(String.valueOf(Disp1));
                mDisp2.setText(String.valueOf(Disp2));
            }

        }
```

Figura 46. Método ComprobarDatos

GuardarConfig:

Este método tendrá como función guardar todos los parámetros en la base de datos. Para empezar, recogerá los valores que haya en los EditText. Realizará las comprobaciones mencionadas anteriormente, que no haya valores nulos y que la distancia de la carrera sea la misma a la del último sensor. Seguidamente se creará un HashMap para de la clase auxiliar Config, al cual se le añadirán los valores. Se utilizará una referencia que apuntará a la base de datos, en específico, a Configuracion y luego al hijo con el Id del usuario. Dicho Id se obtendrá con el método auxiliar getUid, visto anteriormente.

```
Map<String,Object> config = new HashMap<>();
config.put( k: "Disp1",sDisp1);
config.put( k: "Disp2",sDisp2);
config.put( k: "DistTotal",sDistTotal);
myRef.setValue(config);
```

Figura 47. Subir valores de configuración

5.4.7 Historial Activity

Desde esta actividad se podrán visualizar todas las carreras realizadas. Para poder hacerlo se ha elegido un RecyclerView, una versión más avanzada y flexible del ListView. Ya que un ListView convencional, debido a sus limitaciones, no permite mostrar una lista de elementos con un dato por elemento. Gracias al RecyclerView, más complejo de utilizar, se podrán mostrar los elementos que se desee con infinidad de datos. Para utilizarlo se necesitará de una actividad auxiliar, un adaptador, el cual le dote de los atributos deseados. A continuación, se explicarán los métodos utilizados y más adelante la actividad adaptadora.

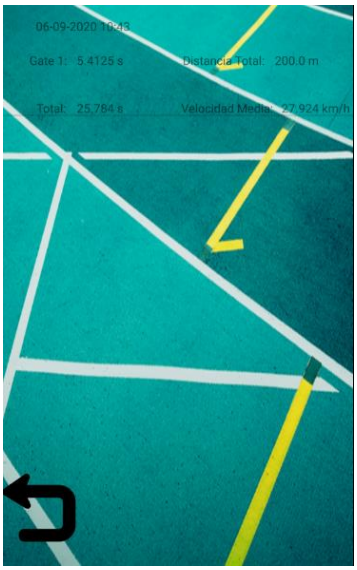

Figura 48. Historial Activity

OnCreate:

Método en el cual se definen todas las variables y se asigna el RecyclerView el Layaout donde va a trabajar. Esto se hace mediante setLayaoutManaget. Después habrá un OnclickListener escuchando si se pulsa el Button VolverAtras. En ese caso se volverá al MainActivity. Después se obtendrá la referencia de la base datos, en este caso apuntará a Historial y después al Id del Usuario, el Id se obtendrá con el método getUid visto anteriormente. Esta referencia tendrá todas las actividades realizadas por el usuario. A continuación, se le aplicará un addValueListener donde se crearán los métodos onDataChange y onCancelled.

OnDataChange:

Este método es el encargado de recoger los datos de la base de datos de Firebase, funciona de forma diferente al método onDataChange de la actividad StarSettings y Start. En primer lugar, creará una lista donde guardar los datos. Se recogerá con un for el snapshot debido a que tendrá diferentes hijos, como se muestra en la Figura 9, se recogerán en un objeto de la clase modelo Historial, un constructor de setters y getters, y por último se guardarán en la lista creada anteriormente. A continuación, se creará un objeto perteneciente al adaptador, al cual se le pasará la lista y por último se le asignará al RecyclerView.

```
public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
    list = new ArrayList<Historial>();
    for(DataSnapshot dataSnapshot1: dataSnapshot.getChildren())
    {
        Historial h = dataSnapshot1.getValue(Historial.class);
        list.add(h);
    }
    Log.d( tag: "Listadedatos", String.valueOf(list));
    adapter = new MyAdapter( c: HistorialList.this,list);
    recyclerView.setAdapter(adapter);
}
```

Figura 49. OnDataChange Historial

OnCancelled:

Método para indicar que ha fallado la consulta debido a la conexión a internet, se mostrará un Toast indicándolo.

5.4.8 MyAdapter Activity

Esta actividad servirá para hacer funcionar correctamente el RecyclerView. Debe extender la clase RecyclerView.Adapter para que se use una instancia de RecyclerVIew.ViewHolder. A continuación, se mostrarán los métodos utilizados.

OnCreateViewHolder:

Este método devolverá el ViewHolder con el layaout historial_view. En este layaout se definirá la vista de cada entrenamiento.

```
public MyViewHolder onCreateViewHolder(@NonNull ViewGroup parent, int viewType) {
    return new MyViewHolder(LayoutInflater.from(context).inflate(R.layout.historial_view,parent, attachToRoot: false));
}
```

Figura 50 . Método onCreateViewHolder

OnBindViewHolder:

Método encargado de establecer los objetos en el ViewHolder. Se utilizará los getters de la clase auxiliar Historial para obtener los valores de los sensores, la velocidad media y los tiempos.

```
public void onBindViewHolder(@NonNull MyViewHolder holder, int position) {
    holder.dist.setText(historiales.get(position).getDistArd2()+" m");
    holder.velmedia.setText(historiales.get(position).getVelMedia()+" km/h");
    holder.tiempo2.setText(historiales.get(position).getTiempo2()+" s");
    holder.tiempo1.setText(historiales.get(position).getTiempo1()+" s");
    holder.fecha.setText(historiales.get(position).getFecha());
}
```

Figura 51. Método onBindViewHolder

OnItemCount:

Método encargado de devolver la cantidad de elementos que tiene la lista. Esta función se hace devolviendo historiales.size.

5.4.9 Idiomas Activity

En esta actividad se podrá elegir el idioma de la aplicación, inglés y castellano. Se mostrará la bandera de los respectivos países mediante dos Buttons, al ser pulsados nos redirigirá al MainActivity de nuevo.

Figura 52. Idiomas Activity

A destacar en esta actividad es el uso de ficheros string.xml, se necesitarán tantos archivos como idiomas a introducir. Mediante el editor de idiomas que implementa Android Studio es fácil realizar las traducciones. En el texto a mostrar, en vez de poner el String de la palabra en concreto, instancias el String del archivo string.xml, y dependiendo del LocaleCode, que es el código del idioma a usar. Mediante un método se cambiará el idioma en función del botón pulsado.

Key	Default Value	Spanish (es)
start	Start	Iniciar
history	History	Historial
language	Language	Idiomas
sing_out	Sing out	Cerrar sesión
choose_language	Choose language	Elige Idioma

Figura 53. Editor de idiomas

```
private void setAppLocale(String localeCode) {

    Resources res = getResources();
    DisplayMetrics dm = res.getDisplayMetrics();
    Configuration conf = res.getConfiguration();
    conf.setLocale(new Locale(localeCode.toLowerCase()));
    res.updateConfiguration(conf,dm);
}
```

Figura 54. Método setAppLocale

Capítulo 6. Instrucciones de uso

Durante la memoria de este proyecto se ha ido explicando cómo utilizar la aplicación, en este apartado se reunirán todas ellas para la fácil utilización del usuario.

6.1 Instrucciones

- Desde su dispositivo móvil vincúlese con el sistema de medida por medio del Bluetooth, para ello vaya ajustes, Bluetooth, busque dispositivos y selecciones vincular con el dispositivo llamado SpeedGates.

- Cree una cuenta con su correo electrónico y una contraseña de al menos 6 caracteres.

- Seleccione desde el menú el idioma deseado.

- Desde el menú start pulse el símbolo de ajustes e introduzca los parámetros de la carrera.

- Desde el menú start, presione el símbolo del Bluetooth y seleccione el dispositivo SpeedGates para conectarse a él.

- Pulse el botón de start para comenzar a medir.

- Desde el menu historial para visualizar las estadísticas.

Capítulo 7. Conclusiones y posibles mejoras

7.1 Conclusiones

Al inicio de este Trabajo Fin de Grado se ha marcado como objetivo principal crear una aplicación que permita el control y uso de un sistema de medición de tiempos en pistas de Atletismo; con la posibilidad de configurar los distintos sensores y la distancia de la carrera para personalizar las necesidades de cada corredor. Ha sido necesario tener conocimientos de programación en el lenguaje de programación Java para poder realizar la aplicación en Android Studio. También hubo que aprender el manejo y funcionamiento de una base de datos noSQL, ya que la estructura JSON es bastante diferente a una estructura de tablas en SQL, que es la que se explica en las asignturas cursadas.

Los retos a superar en este proyecto han sido muchos, ya que, el uso de las funcionalidades de Firebase al principio era muy complejo, tales como programar que funcionará correctamente, a la vez que funcionarán conjuntamente la Autenticación y la Realtime Database de Firebase, ya que las dos funcionan independientemente. Gracias a la función *obtener el Id del usuario* se ha podido estructurar la base de datos. Otro de los apartados más complicados ha sido la conexión Bluetooth, al principio, se mandaban los mensajes, y llegaban fraccionados, con caracteres desconocidos o solo parte del mensaje, lo que resultaba imposible tratarlosIgualmente complicada fue la actividad Historial, ya que el uso del RecyclerView era algo nuevo y pese a ser algo implementado en Android Studio se necesitaba de una clase adaptador para su correcto funcionamiento.

Finalmente, y tras superar todas las dificultades, se ha conseguido realizar una interfaz sencilla a la vez que funcional e intuitiva; que incluye iconos correspondientes a la funcionalidad que implementa, por ejemplo, el bluetooth o los ajustes.

Se puede decir que los objetivos marcados se han cumplido sobradamente consiguiendo una aplicación integrada en el Equipo de medida de velocidad para pruebas de atletismo, de fácil manejo y que permitirá comercializar el equipo con un coste asequible para los deportistas amateurs. También ha servido para conseguir el objetivo académico; porque gracias al trabajo realizado al desarrollar este proyecto he aprendido a implementar muchas tecnologías y la correcta relación entre ellas ha sido lo más importante; a la vez que he desarrollado capacidades como el trabajo en equipo y el autoaprendizaje, ambas muy importantes para enfrentarse al mundo laboral.

7.2 Mejoras futuras

La aplicación podría tener algunas mejoras que la llevarán al siguiente nivel. La primera de ella sería utilizar un sistema de medida que diferenciara distintos corredores en la carrera, con ello se podrían hacer entrenamientos grupales.

Actualmente solo se puede ver el progreso personal sin la posibilidad de compartirlo. Una mejora considerable sería la implementación de una comunidad, donde los usuarios agreguen a sus amigos y puedan ver las carreras de los demás, valorarlas y comentarlas.

Otra mejora interesante sería además de ver el historial, tener la posibilidad de visualizar de forma gráfica el progreso del corredor construido a partir de los diferentes entrenamientos.

También que la interfaz fuera más bonita y moderna, con la implementación de Fragments u otros elementos, resultaría una experiencia mejor.

Por último, sería promocionar la aplicación en Google Play para que las personas que estén interesadas en el servicio puedan acceder a él.

Capítulo 8. Presupuesto.

En esta parte de la memoria se explicará cuanto podría costar el desarrollo de esta aplicación en Android. Es una estimación ya que en el desarrollo se ha invertido tiempo en la investigación y búsqueda de información de cómo hacerla.

En este caso ha sido un mes y medio de trabajo a jornada completa de desarrollo puro. Esto equivalen a 40 horas a la semana, más comúnmente 8 horas de lunes a viernes, a un precio de mano de obra de 10€ la hora. Por tanto, el cómputo global sería de:

$$Mano\ de\ obra = 40 \text{ horas x } 6 \text{ semanas x } 10€ = 2.400\ €$$

El programa utilizado, Android Studio, es de código abierto, no es necesario pagar ninguna licencia para su uso, por tanto, el gasto en software es de 0 €.

La utilización de Realtime Database de Firebase puede conllevar un gasto si se superara el límite de uso gratuito que establecido, 1 GB de almacenamiento en total y 10 GB de descarga de datos al mes.

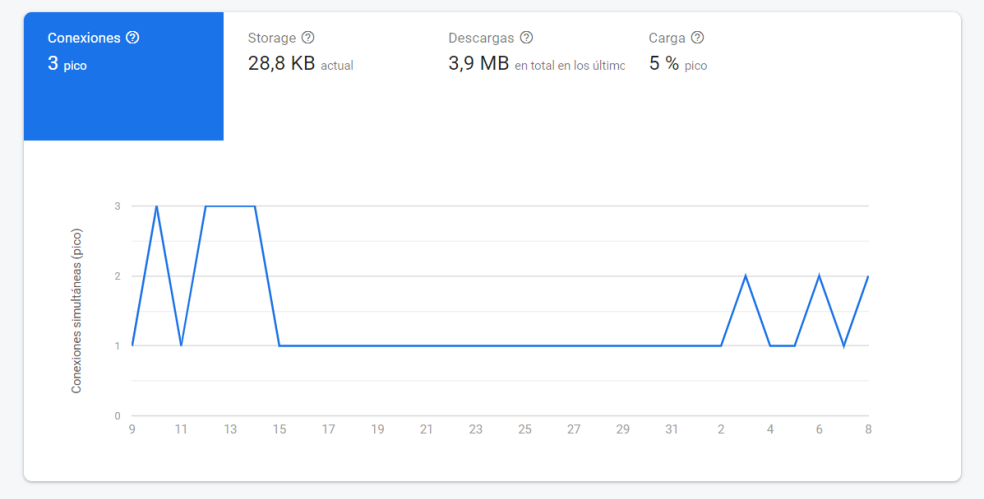

Figura 55. Uso de datos Firebase

Este es el uso en los últimos 30 días de la base de datos. En este caso no se prevé llegar al límite, ya que la aplicación no utiliza prácticamente los recursos de la base de datos, los datos almacenados es sólo texto.

En el caso de Firebase Authentication el límite de usuarios es de 10.000 por mes, por tanto, también vamos a asumir coste nulo. En el caso que se llegara a estas cifras, la base de datos tendría un precio de 5 $ por 1 GB de almacenamiento y 1 $ por GB descargados. En Authentication sería de 0,01$ por verificación desde Estados Unidos, Canadá e India y 0,06 $ para el resto de países.

En conclusión, el presupuesto total sería la mano de obra: **2.400 €**

Capítulo 9. Bibliografía

[1] Documentación Authentication de Firebase: https://firebase.google.com/docs/auth/android/custom-auth?hl=es

[2] Documentación Realtime Database de Firebase: https://firebase.google.com/docs/database

[3] Documentación formato JSON: https://blog.openalfa.com/introduccion-al-formato-json

[3] Documentación de la clase Date de Java: https://docs.oracle.com/javase/7/docs/api/java/util/Date.html

[4] Documentación de consultas de programación: https://es.stackoverflow.com

[5] Documentación del RecyclerView en Android: https://developer.android.com/guide/topics/ui/layout/recyclerview

[7] Documentación clase Countdown Timer en java: https://codigoscript.com/2015/04/16/countdowntimer-programar-una-cuenta-atras-ejecutando-codigo-en-intervalos

[8] Documentación clase String de Java: https://javadesdecero.es/clases/string/

[9] Documentación del Epoch : https://www.tutorialspoint.com/java/lang/system_currenttimemillis.htm

[10] Documentación Bluetooth: http://solderer.tv/data-transfer-between-android-and-arduino-via-bluetooth/

[11] Documentación sobre el diseño de la aplicación: https://developer.android.com/guide/topics/ui/declaring-layout

[11] Documentación sobre añadir idiomas: https://google-developer-training.github.io/android-developer-advanced-course-concepts/unit-3-make-your-apps-accessible/lesson-5-localization/5-1-c-languages-and-layouts/5-1-c-languages-and-layouts.html

[12] Documentación sobre constructores: https://javabasico.osmosislatina.com/curso/varcons.htm

[13] Documentación sobre Handler: https://developer.android.com/reference/android/os/Handler