Universidad Zaragoza
1542

Trabajo Fin de Grado

Desarrollo de aplicación para seguimiento de jugadores en cantera de futbol

Autor

Jorge Fuertes Orrios

Director

Jesús Gallardo Casero

Escuela Universitaria Politécnica de Teruel

Ingeniería Electrónica y Automática

2023

Desarrollo de aplicación para seguimiento de jugadores en cantera de futbol

Resumen

El uso de los dispositivos móviles ha adquirido una importancia mayúscula en nuestras vidas durante los últimos años, esto ha hecho que dejen de verse como una simple herramienta para realizar llamadas y que pasen a ser necesarios en casi cualquier actividad que realizamos.

Este avance tecnológico hace que cada vez más los equipos busquen una manera informatizada de llevar el control y seguimiento de sus jugadores.

Por tanto, este Trabajo De Final de Grado (TFG) consiste en responder esta necesidad planteada por el CD Teruel mediante el desarrollo de una aplicación para dispositivos móviles proporcionando información sobre el desarrollo de los jugadores de los diferentes equipos de la cantera empleando tecnológicas como la base de datos Room la arquitectura MVVM o los objetos LiveData.

Como finalidad adicional de este TFG, se pretende aprender conocimientos referentes al desarrollo de aplicaciones en Android Studio utilizando tecnologías novedosas que permitan que la aplicación este lo más actualizada posible.

Abstract

The use of mobile devices has acquired a major importance in our lives in recent years, this has made them stop being seen as a simple tool for making calls and have become necessary in almost any activity we do.

This technological advance means that more teams are looking for computerized a way to control and monitor their players.

Therefore, this Final Degree Project (TFG) consist of responding to this need raised by CD Teruel through the development of a mobile application providing information on the development of the different team's players using technologies such as Room database, MVVM architecture or LiveData objects.

As an additional purpose of this TFG, it is intended to learn knowledge related to the development of application in Android Studio using innovative technologies that allow the application to be update as possible.

Tabla de contenido

Tabla de figuras

Listado de Acrónimos

- API: Application Programming Interfaces
- App: Application
- CD: Club Deportivo
- DAO: Data Access Object
- IDE: Integral Development Environment
- Int: Integer
- iOs: iPhone Operating System
- MVVM: Model View ViewModel
- PDA: Personal Digital Assistant
- SQL: Structured Query Language
- TFG: Trabajo Fin de Grado
- UDF: Universal Disk Format
- UTF: Unicode Transformation Format
- UI: User Interface
- XML: Extensible Markup Language

1. Introducción

En el presente Trabajo Fin de Grado (TFG) se ha abordado el desarrollo de una aplicación móvil con la finalidad de que sea usada por el staff de entrenadores de las canteras del Club Deportivo Teruel.

El objetivo de la aplicación es proporcionar a los entrenadores del club información sobre sus equipos para que les sea más fácil tomar decisiones en consecuencia con dicha información.

A continuación, se presenta la memoria y documentación del TFG "Desarrollo de aplicación para seguimiento de jugadores en canteras de futbol".

1.1 Motivación

El fútbol es más que un deporte, es una pasión que une a personas de todas las edades, géneros y culturas de todo el mundo. Las canteras de futbol desempeñan un papel fundamental en el desarrollo de jóvenes talentos, brindándoles la oportunidad de crecer no solo como deportistas sino como personas. Sin embargo, la gestión eficiente y el seguimiento preciso de los jugadores en estas canteras son aspectos que a menudo pueden ser mejorados.

El futbol está en constante evolución, y la búsqueda de nuevos talentos es esencial para mantener la competitividad y el éxito de cualquier equipo o selección nacional. Las canteras de futbol son viveros de talento, donde jóvenes promesas pueden ser moldeadas y preparadas para alcanzar su máximo potencial. Sin embargo, para que esto suceda, es crucial contar con herramientas que faciliten la identificación, seguimiento y desarrollo de estos talentos desde edades tempranas.

A pesar de la importancia de las canteras, muchas enfrentan desafíos significativos en su gestión. La falta de seguimientos efectivos a menudo conduce a la perdida de talento, la falta de continuidad en el desarrollo de jugadores y la dificultad para identificar las áreas de mejora. Además, la presión por obtener resultados inmediatos a veces puede eclipsar la atención necesaria que requiere el desarrollo a largo plazo de los jóvenes futbolistas.

La idea de la realización de este trabajo surge tras conservaciones entre mi tutor e integrantes del staff del club deportivo Teruel y llegaron a la conclusión que el desarrollo de una aplicación destinada al seguimiento de jugadores puede hacer que los entrenadores tengan una herramienta tecnológica que los ayude a favorecer el desarrollo de sus jugadores.

1.2 Objetivos de la aplicación

Este trabajo consiste en el desarrollo de una aplicación para dispositivos móviles Android orientada al seguimiento de diferentes aspectos que conciernen a la progresión de los jugadores de las canteras del Club Deportivo Teruel.

Para el desarrollo de la aplicación se utilizarán tecnologías que son para mí una novedad con el objetivo de explorar en profundidad y obtener conocimientos del desarrollo de aplicaciones móviles.

Por ello, los objetivos del Trabajo Fin De Grado son:

- Desarrollar una aplicación que pueda ser utilizada desde cualquier Smartphone Android.
- Que la aplicación gestione su propia base de datos.
- Permitir realizar operaciones dentro de esta base de datos como pueden ser la inserción, eliminación, actualización y consulta de datos.
- Realizar una sección de estadísticas, con el objetivo de filtrar los datos y mostrarlos de una manera eficiente y fácil de visualizar para el usuario.
- Estudiar tecnologías novedosas que se necesitan implementar en el proyecto y con las que no he trabajado todavía.

1.3 Herramientas y tecnologías

Android Studio

Es el entorno de desarrollo integrado (IDE) oficial para la plataforma Android. Ofrece una amplia gama de herramientas y recursos que simplifican la creación de aplicaciones nativas para dispositivos Android. [1]

Entre las características destacadas se incluyen:

- **Lenguajes de Programación:** Android Studio admite Java y Kotlin como lenguajes principales para el desarrollo de aplicaciones Android. Kotlin, en particular, ha ganado popularidad debido a su concisión y seguridad.
- **Emuladores y Depuración:** Proporciona emuladores de dispositivos Android para probar aplicaciones en diversas configuraciones. Además, ofrece herramientas de depuración robustas.
- **Diseño de Interfaz de Usuario (UI):** Android Studio incluye un editor de diseño visual que facilita la creación de interfaces de usuario atractivas y funcionales.
- **Integración con Google Services:** Permite la integración sencilla de servicios de Google, como Google Maps, Firebase y Google Cloud, en las aplicaciones.

Flutter

Flutter es un *framework* de código abierto desarrollado por Google que se utiliza para crear aplicaciones móviles nativas en plataformas múltiples desde un solo código base. Algunas de sus características más destacadas son:

- **Lenguaje Dart:** Flutter utiliza el lenguaje de programación Dart, que es conocido por su alto rendimiento y su facilidad de aprendizaje.
- **Widgets Personalizables**: Flutter ofrece un amplio conjunto de widgets personalizables que permiten a los desarrolladores crear interfaces de usuario atractivas y consistentes en todas las plataformas.
- **Rápido Desarrollo y Recarga en Caliente**: La característica de recarga en caliente de Flutter permite realizar cambios en tiempo real en la aplicación sin necesidad de reiniciarla, lo que acelera el proceso de desarrollo.
- **Compatibilidad Multiplataforma**: Flutter permite crear aplicaciones para Android, iOS, web y escritorio, lo que simplifica la expansión de una aplicación a diferentes plataformas. [2]

Xcode

Este es un entorno de desarrollo integrado (IDE) desarrollado por Apple que se utiliza principalmente para crear aplicaciones para dispositivos Apple, como iPhone, iPad, Mac y Apple Watch.

- **Lenguaje de Programación**: Tiene varios lenguajes de programación como pueden ser Swift y Objective-C.
- **Diseño de interfaces**: Xcode incluye una interfaz de usuario gráfica que facilita la creación y diseño de interfaces de aplicaciones de manera visual.
- **Depuración y pruebas**: La plataforma proporciona herramientas de depuración y emulación que permiten a los desarrolladores probar sus aplicaciones en simuladores de dispositivos antes de lanzarlas al mercado.
- **Integración de servicios**: Xcode ofrece integración con otros servicios de Apple, como iCloud y Game Center, lo que facilita la creación de aplicaciones que aprovechan estas funcionalidades.
- **Desarrollo multiplataforma**: Con Xcode, puedes crear aplicaciones que funcionen en una variedad de dispositivos Apple, incluyendo iOS, macOS, watchOS y tvOS.
- **Gestión de proyectos**: Xcode permite organizar y gestionar proyectos de desarrollo de aplicaciones de manera eficiente, lo que facilita la colaboración en equipos de desarrollo. [3]

Kotlin

Esta herramienta puede trabajar tanto con el lenguaje Java como con Kotlin, este último ha sido nombrado por Google como lenguaje oficial de Android por lo que ha sido mi elección para llevar a cabo la implementación de la app.

Kotlin aparece en el 2016, pero es a partir del 2017 cuando empieza a coger fuerza ya que es en este momento cuando recibe la ya mencionada oficialidad para el desarrollo de apps de Android.

Es un lenguaje que guarda similitudes con Java a diferencia de que esta combinación de características modernas con una sintaxis concisa. Al tener una sintaxis concisa Kotlin reduce el código respecto a Java lo que hace que el desarrollo sea más eficiente. Además, Kotlin es interoperable con Java, lo que permite a los desarrolladores utilizar bibliotecas de Java existentes en proyectos de Kotlin y viceversa. [4]

SQLite

SQLite es una herramienta de software libre que permite almacenar información en dispositivos empotrados de una forma sencilla, eficaz, potente, rápida y en equipos con pocas capacidades de hardware, como puede ser una PDA o un teléfono. Esta herramienta de software se puede usar tanto en dispositivos móviles como en sistemas de escritorio, sin necesidad de realizar procesos complejos de importación y exportación de datos, ya que existe compatibilidad al 100% entre las diversas plataformas disponibles, haciendo que la portabilidad entre dispositivos y plataformas sea transparente.

Estas son algunas de las características principales de SQLite:

- La base de datos completa se encuentra en un solo archivo.
- Puede funcionar enteramente en memoria, lo que la hace muy rápida.
- Tiene un *footprint* menor a 230KB.
- Es totalmente autocontenida (sin dependencias externas).
- Cuenta con librerías de acceso para muchos lenguajes de programación.
- Soporta texto en formato UTF-8 y UTF-16, así como datos numéricos de 64 bits.
- Soporta funciones SQL definidas por el usuario (UDF).
- El código fuente es de dominio público y se encuentra muy bien documentado. [6]

Room

La biblioteca de persistencias Room brinda una capa de abstracción para SQLite que permite acceder a la base de datos sin problemas y, al mismo tiempo, aprovechar toda la potencia de SQLite.

La biblioteca ayuda a crear una caché de los datos de tu app en un dispositivo que la ejecute. Esta caché, que funciona como la única fuente de confianza de la app, permite que los usuarios vean una copia coherente de información clave en la app, independientemente de si cuentan con conexión a Internet. [5]

Entre las características de Room las más destacadas son:

- **Relaciones entre tablas:** Room facilita la definición de relaciones entre entidades, lo que simplifica el trabajo con base de datos relacionales. Puedes establecer fácilmente relaciones uno a uno, uno a muchos y muchos a muchos.

- **Validación en tiempo de conmutación:** Room realiza comprobaciones en tiempo de compilación de las consultas SQL, lo que ayuda a evitar errores en tiempo de ejecución al escribir consultas incorrectas.

- **Integración con LiveData:** Room se integra bien con LiveData, una clase de la arquitectura de componentes de Android, lo que facilita la actualización automática de la interfaz de usuario cuando los datos en la base de datos cambian.

- **Migraciones automáticas:** Room puede manejar automáticamente las migraciones de la base de datos cuando se realizan cambios en el esquema, lo que simplifica la evolución de la base de datos a medida que la aplicación se actualiza.

MySQL

MySQL es un sistema de gestión de bases de datos que cuenta con una doble licencia. Por una parte, es de código abierto, pero por otra, cuenta con una versión comercial gestionada por la compañía Oracle.

Estas son algunas de las características principales de MySql

- **Arquitectura Cliente y Servidor**: MySQL basa su funcionamiento en un modelo cliente y servidor. Es decir, clientes y servidores se comunican entre sí de manera diferenciada para un mejor rendimiento. Cada cliente puede hacer consultas a través del sistema de registro para obtener datos, modificarlos, guardar estos cambios o establecer nuevas tablas de registros, por ejemplo.
- **Compatibilidad con SQL**: SQL es un lenguaje generalizado dentro de la industria. Al ser un estándar MySQL ofrece plena compatibilidad por lo que si has trabajado en otro motor de bases de datos no tendrás problemas en migrar a MySQL.

- **Vistas**: Desde la versión 5.0 de MySQL se ofrece compatibilidad para poder configurar vistas personalizadas del mismo modo que podemos hacerlo en otras bases de datos SQL. En bases de datos de gran tamaño las vistas se hacen un recurso imprescindible.
- **Procedimientos almacenados**. MySQL posee la característica de no procesar las tablas directamente, sino que a través de procedimientos almacenados es posible incrementar la eficacia de nuestra implementación.
- **Desencadenantes**. MySQL permite además poder automatizar ciertas tareas dentro de nuestra base de datos. En el momento que se produce un evento otro es lanzado para actualizar registros u optimizar su funcionalidad.
- **Transacciones**. Una transacción representa la actuación de diversas operaciones en la base de datos como un dispositivo. El sistema de base de registros avala que todos los procedimientos se establezcan correctamente o ninguna de ellas. En caso por ejemplo de una falla de energía, cuando el monitor falla u ocurre algún otro inconveniente, el sistema opta por preservar la integridad de la base de datos resguardando la información. [7]

1.4 Decisión Final

Una vez vistas y analizadas las tecnologías anteriores, la escogida para realizar la aplicación ha sido Android Studio debido a estas razones:

- Android Studio permite un acceso directo a las API nativas de Android, lo que significa que se pueden aprovechar al máximo las características específicas de Android en tus aplicaciones.
- Dado que Android es una plataforma muy popular he podido encontrar gran cantidad de recursos en línea como pueden ser tutoriales o información en la nube factor que me ha sido muy útil durante el desarrollo de la aplicación ya que partir de cero en cuanto al conocimiento de desarrollo de aplicaciones móviles.
- Dentro de los dos lenguajes de programación disponibles en Android Studio he elegido Kotlin ya que Google lo está priorizando respecto a Java.
- Para la base de datos he elegido Room ya que es una opción poderosa y eficiente para la persistencia de datos en aplicaciones Android, mejorando la productividad del desarrollador y garantizando un manejo más seguro y eficaz de la base de datos.
- He decidido utilizar SQLite en vez de MySQL debido a que mi aplicación no requiere de una base de datos demasiado grande y es necesario escribir y leer directamente desde la aplicación.

2. Análisis

La fase de análisis es uno de los pilares fundamentales de todo proyecto de software, es el punto de partida de este TFG a partir del cual se deben asentar las bases del proyecto.

En esta sección de la memoria se mostrará el planteamiento del problema, el público objetivo, las aplicaciones existentes y los requisitos incluyendo los funcionales y los no funcionales.

A partir del planteamiento del problema el público objetivo y las diferentes aplicaciones existentes se plantearán los requisitos.

2.1 Planteamiento del problema

Como se ha comentado en el apartado de motivación el trabajo está destinado a proporcionar una herramienta tecnológica a los entrenadores del club deportivo Teruel.

Las canteras de fútbol son fundamentales para los clubes ya que proporcionan una fuente constante de talento joven y local. Estas canteras no solo ayudan a reducir costos al nutrir al primer equipo con jugadores de desarrollo propio, sino que también fortalecen la identidad y la conexión del club con la comunidad local. Además, contribuyen a la formación de futbolistas profesionales de alta calidad que pueden generar ingresos a través de transferencias a otros equipos, lo que beneficia financieramente al club. Por los motivos expuestos considero que las canteras son un componente esencial en el éxito y la sostenibilidad a largo plazo de los clubes de fútbol.

El auge de las nuevas tecnologías y de los dispositivos móviles abre un mundo de posibilidades para el desarrollo de aplicaciones que permitan el seguimiento de los jugadores jóvenes. En el Play Store de Google podemos ver aplicaciones similares pero la idea era crear una exclusiva para el Club Deportivo Teruel con una interfaz gráfica amigable con el usuario y un diseño con elementos propios del club como son la imagen del escudo en diferentes pantallas un fondo de pantalla en el menú principal donde aparece el campo Pinilla.

2.2 Público Objetivo

La aplicación desarrollada en este TFG está diseñada para ser utilizada por el staff técnico de las canteras del Club Deportivo Teruel. Este público puede ser de todas edades de modo que la aplicación debe ser lo más simple e intuitiva posible para asegurar el correcto uso y la mejor utilidad de esta.

La aplicación podría tener una serie de extensiones que incluyeran más funcionalidades en tal caso la aplicación podría ser utilizada también por los jugadores del club, por lo que esta versión solo está destinada a entrenadores.

2.3 Aplicaciones existentes

En este apartado voy a exponer ciertas aplicaciones similares a la desarrollada disponibles en el Play Store de Google.

Gol Manager

Se trata de una aplicación muy similar a la expuesta en el TFG, que consiste en una serie de funcionalidades estadísticas para el entrenador de un club.

Esta aplicación tiene muchas cosas buenas respecto a la mía como pueden ser una variedad más amplia de funcionalidades la implementación de un calendario donde aparecen registrados los entrenamientos y los partidos.

El que esta aplicación tenga tantas funcionalidades es favorable, pese a que en CanteraControl se ha querido tener una funcionalidad mucho más específica, pero también tiene sus inconvenientes como que el diseño es muy ostentoso y recargado lo que no facilita su uso mientras que en CanteraControl uno de los elementos más importantes de la aplicación es que el diseño es simple e intuitivo.

Esta aplicación tiene otros dos aspectos más negativos, como son la aparición de anuncios constantes lo que hace que la aplicación en si sea bastante tediosa y por último que muchas de las funcionalidades de estas son de pago.

Ilustración 1: -Aplicación GolManager

Bcoach

Esta aplicación se centra en diversas tareas de registro de los entrenadores con el objetivo de facilitar su día a día y poder planear la temporada de su equipo. [8]

Esta aplicación cuenta con una pantalla de inicio que te da la opción de crear un entrenamiento un partido o ir a la pizarra, esta pizarra te sirve para registrar diferentes estadísticas en el tiempo real de partido como pueden ser los goles minutos jugados o tarjetas amarillas funcionalidad que me parece bastante interesante y podría ser implementada en futuras versiones de la aplicación. Cuando finaliza un partido se pueden ver todas las estadísticas que hayan sido anotadas por el entrenador.

Ilustración 2: -Aplicación Bcoach

Esta aplicación tiene dos inconvenientes principales: el primero que es de pago y el segundo que solo sirve para monitorizar un equipo.

2.4 Estructuras

A continuación, se detallan los distintos sistemas de los que hace uso la aplicación:

Estructura de jugadores

La estructura de jugadores permitirá guardar jugadores en la base de datos con sus diferentes parámetros (nombre, posición, dorsal) para que este pueda ser incluido posteriormente en una plantilla.

Estructura de plantillas

La estructura de plantillas permitirá guardar plantillas con sus diferentes parámetros (nombre, categoría, temporada) en la base de datos.

Subestructura de consulta de jugadores en plantilla

La subestructura permitirá al usuario visualizar los jugadores que se encuentren dentro de la plantilla además de poder guardar jugadores dentro de la misma.

Subestructura de entrenamientos

En la subestructura de entrenamientos el usuario podrá añadir entrenamientos a la base de datos del sistema con sus respectivos parámetros (fecha, hora, URL) y cuando los entrenamientos estén dentro de la base de datos realizar diferentes operaciones que detallare más adelante en los requisitos funcionales.

Subestructura de partidos

En la subestructura de partidos el usuario podrá guardar partidos en la base de datos con sus respectivos parámetros (rival, jornada, temporada y URL) y una vez estén dentro realizar diferentes operaciones y consultas que expondré más adelante.

Estructura de estadísticas

La estructura de estadísticas se encargará de toda la funcionalidad referente a la generación y visualización de gráficos, siendo una parte fundamental de la aplicación.

Subestructura de generación de estadísticas

En la subestructura de generación de estadísticas se podrá seleccionar primero una plantilla seguida de una opción referente a los partidos o a los entrenamientos y por último el nombre del jugador.

Subestructura de visualización de estadísticas

Subestructura que se encargará de mostrar el gráfico que genera el subsistema de generación de estadísticas, que permitirá interactuar con el gráfico para resaltar alguno de sus valores o ampliar su tamaño.

2.5 Especificación de requisitos

En este apartado, vamos a especificar las funciones que se podrán realizar dentro del sistema y bajo que restricciones. Para ello vamos a presentar los requisitos funcionales y no funcionales.

2.5.1 Requisitos funcionales

En esta sección detallaremos los requisitos funcionales de la aplicación catalogados por los sistemas y subsistemas que ya se han definido.

RF -1 – Añadir un jugador: La aplicación permitirá añadir un jugador con sus diferentes atributos (nombre, dorsal, posición) a la base de datos.

RF -2 – Añadir una plantilla: La aplicación permitirá añadir una plantilla a la base de datos con sus diferentes atributos (nombre, categoría, temporada).

RF -3 – Añadir jugadores en la plantilla: El usuario podrá incorporar jugadores en la plantilla entre los disponibles en la base de datos que aún no hayan sido incluidos en ninguna plantilla del sistema.

RF -4 – Visualización de jugadores en plantilla: El usuario podrá ver la lista de jugadores que se hayan introducido en cada una de las diferentes plantillas.

RF -5 – Eliminación de jugadores: El usuario podrá borrar las plantillas y con este todos sus elementos asociados (jugadores, partidos, entrenamientos, valoraciones del jugador).

RF -6 – Añadir entrenamientos: El usuario podrá incorporar entrenamientos de las diferentes plantillas con sus atributos asociados (fecha, hora, URL).

RF -7 – Añadir fotos: El usuario podrá cargar fotos de su galería y guardarlas como atributo del entrenamiento.

RF -8 – Añadir Anotaciones: El usuario podrá añadir las anotaciones que considere necesarias a cada entrenamiento.

RF -9 – Eliminar entrenamientos: El usuario podrá eliminar entrenamientos y con este todos sus elementos asociados (jugadores, puntuaciones de jugadores).

RF -10 – Añadir jugadores al entrenamiento: El usuario podrá seleccionar entre los jugadores correspondientes de la plantilla que no se hayan añadido con anterioridad a la base de datos.

RF -11 – Añadir valoraciones a los jugadores: El usuario al mismo tiempo que añade los jugadores al entrenamiento les podrá asociar una puntuación en consecuencia con su rendimiento en el entrenamiento que variará de 1 a 5 estrellas.

RF -12 – Visualización del entrenamiento: En esta pantalla el usuario podrá ver la foto asociada a cada entrenamiento junto con las anotaciones y la lista de los jugadores que hayan acudido al entrenamiento.

RF -13 – Añadir partidos: El usuario podrá incorporar partidos de las diferentes plantillas con sus atributos asociados (rival, jornada, temporada).

RF -14 – Añadir fotos: El usuario podrá cargar fotos de su galería y guardarlas como atributo del partido.

RF -15 – Añadir Anotaciones: El usuario podrá añadir las anotaciones que considere necesarias a cada partido.

RF -16 – Eliminar partidos: El usuario podrá eliminar partidos y con este, todos sus elementos asociados (jugadores, puntuaciones de jugadores).

RF -17 – Añadir jugadores al partido: El usuario podrá seleccionar entre los jugadores correspondientes de la plantilla que no se hayan añadido con anterioridad a la base de datos.

RF -18 – Añadir valoraciones a los jugadores: El usuario al mismo tiempo que añade los jugadores al partido les podrá asociar una puntuación en consecuencia con su rendimiento en el entrenamiento que variará de 1 a 5 estrellas.

RF -19 – Visualización del entrenamiento: En esta pantalla el usuario podrá ver la foto asociada a cada entrenamiento junto con las anotaciones y la lista de los jugadores que hayan acudido al entrenamiento.

RF -20 – Elegir Plantilla: El usuario podrá seleccionar una plantilla para generar datos sobre algún jugador incluido en esta.

RF -21 – Elegir partido o entrenamiento: El usuario podrá seleccionar si quiere generar datos de entrenamientos o partidos.

RF -23 – Elegir jugador: El usuario podrá seleccionar un jugador concreto para generar datos sobre él.

RF -24 – Generar un gráfico: Una vez seleccionados los parámetros, se podrá generar el grafico deseado y aunque haya un gráfico ya generado se podrán seleccionar otros atributos y generar uno distinto.

2.5.2 Requisitos no funcionales

En esta sección se detallarán los requisitos no funcionales con los que debe contar la aplicación.

RNF 1 – Interfaz sencilla e intuitiva: La interfaz de la aplicación deberá ser simple, pero de fácil uso y que no sea necesario pasar mucho tiempo usándola para familiarizarse con ella.

RNF 2 – Interfaz personalizada para el club: La interfaz deberá contener elementos que nos hagan reconocer la aplicación como propia del club deportivo Teruel.

RNF 3 – Fluidez: La aplicación tendrá que ser rápida a la hora de cambiar entre las distintas vistas que la componen, así como a la hora de generar gráficos.

RNF 4 – Funcionamientos en dispositivos Android: La aplicación debe funcionar en dispositivos cuya versión mínima sea Android 7(API 24), esto incluye el 96,7 % de todos los dispositivos.

RNF 5 – Velocidad de acceso a la base de datos: Las acciones que necesiten guardar datos en la base de datos o acceder a ellos tendrán que ser rápidas para no entorpecer la experiencia del usuario.

3. Diseño

En este capítulo vamos a describir todo el proceso de diseño de la aplicación móvil.

El diseño realizado engloba todos los requisitos tanto funcionales como no funcionales descritos en el apartado de análisis.

3.1 Arquitectura general del sistema

La arquitectura de la aplicación será cliente-servidor.

En cuanto a la parte del servidor utilizo la biblioteca Room [9] que es una biblioteca de persistencia de datos de Android que proporciona una capa de abstracción sobre SQLite para permitir un acceso más fluido a la base de datos.

Para tratar las fotos que los entrenadores utilizaran para los entrenamientos y los partidos la aplicación utilizara un directorio ubicado en el almacenamiento externo donde se pueden almacenar archivos que pertenecen a la aplicación y que no serán eliminados si el usuario desinstala la aplicación.

Respecto al cliente, se trata de una aplicación en lenguaje Kotlin que será ejecutada desde dispositivos móviles con un sistema operativo Android. Esta aplicación cliente será la encargada de contactar con la base de datos Room para obtener la información necesaria en cada momento.

3.2 Prototipo de interfaces

Los prototipos de interfaces de usuario los utilizare para explorar un diseño de interfaz de usuario alcanzable y adecuado en consonancia con los requisitos no funcionales estipulados anteriormente. Este prototipo de interfaces es una representación visual y

funcional de cómo se verá y comportará la aplicación antes de desarrollarla completamente.

En el caso de este trabajo, las interfaces se han adaptado para sistemas móviles de Android.

La primera interfaz que se muestra al iniciar la app es un *splashscreen* en el que se mostrara el logotipo de las canteras del Club Deportivo Teruel sobre un fondo blanco.

Inmediatamente después se mostrará una pantalla de menú en cuyo fondo se mostrará una imagen proporcionada por el club donde aparece el campo y en el centro el logotipo de la cantera del Club Deportivo Teruel. En esta pestaña también aparecen cuatro botones para darle al usuario diferentes opciones según lo que desee hacer.

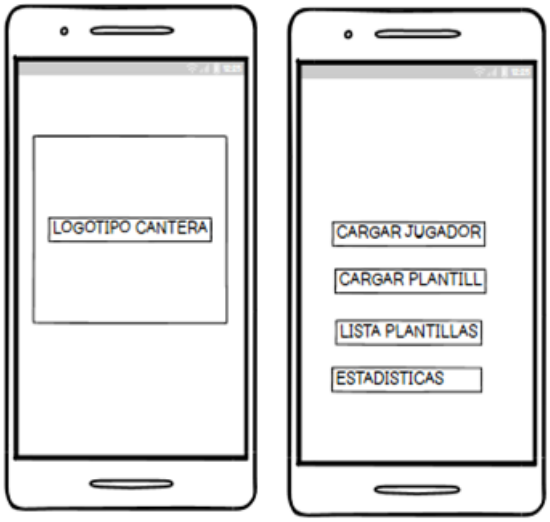


Ilustración 3: Prototipo de Interfaces “SplashScreen”, “MenuActivity”

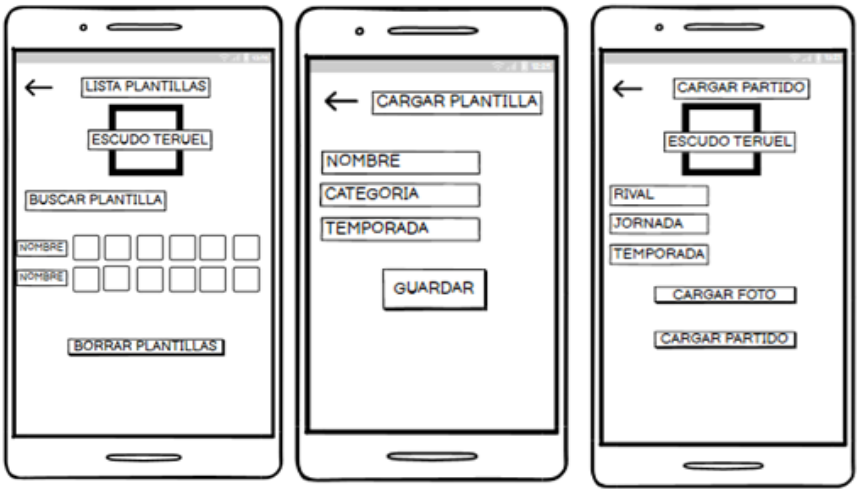


Ilustración 4: -Prototipo de Interfaces “Lista de plantillas”,” Cargar Plantilla”, “Cargar Jugador”

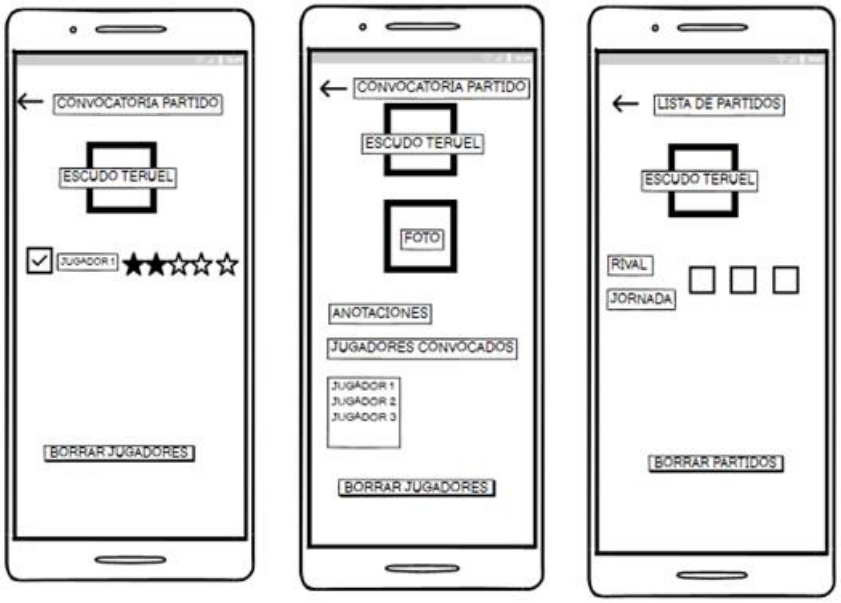


Ilustración 5: - Prototipo de Interfaces “Resumen Partido” “Convocatoria Partido” “Lista de Partidos”

Los cuadrados de la primera imagen de la ilustración 3 se corresponden con iconos que funcionan como botones en la aplicación. Las funcionalidades asociadas a estos botones por orden serían las siguientes: (Añadir Jugadores, Ver Jugadores en la plantilla, Añadir Plantilla, Añadir Entrenamientos, Lista de Entrenamientos, Añadir Partido y Lista de Partidos)

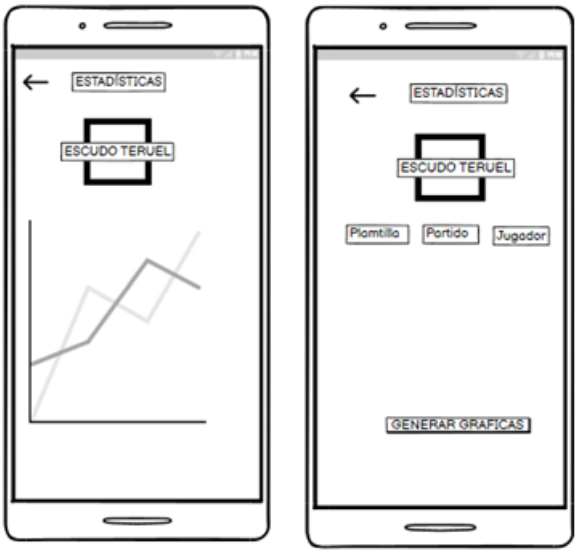


Ilustración 6: - Prototipo de interfaces pantallas de estadísticas

3.3 Diseño de la aplicación

En las estructuras de las aplicaciones Android la lógica de programa ocurre en actividades que son independientes entre sí ya que cada una tiene su propio ciclo de vida. Estas actividades se encargar de llevar todo el flujo del programa haciendo uso de diferentes componentes como son el modelo o los objetos de la interfaz.

Como se ha comentado en la parte de la arquitectura la aplicación es del tipo cliente servidor. La parte del servidor es la que se encarga de la gestión de la base de datos la gestión de la lógica de negocio, la seguridad y protección de datos y la sincronización de datos. La parte cliente debe estar en constante comunicación con el servidor para asegurar la persistencia de los datos.

Existente diferentes maneras de asegurar la persistencia de los datos y la elección de la arquitectura puede ser clave para ello. La arquitectura MVVM ha sido convertida en Google como estándar para implementar aplicaciones Android por lo tanto ha sido mi elección para desarrollar el TFG, esta arquitectura la explicare más adelante en la parte de tecnologías utilizadas.

Una de las principales ventajas de esta arquitectura es el uso de los objetos Live Data la cual es una clase diseñada específicamente para la comunicación entre componentes de la interfaz de usuario y componentes de la clase ViewModel.

La clase Live Data está diseñada para ser reactiva. Esto significa que los objetos Live Data pueden notificar automáticamente a sus observadores cuando los datos subyacentes cambian. Esta clase además está diseñada para ser consciente del ciclo de vida de los componentes Android como actividades y fragmentos. Esto garantiza que los observadores solo reciban actualizaciones cuando el componente este en un estado activo listo para recibir datos evitando así problemas como la fuga de memoria al

desvincular automáticamente los observadores cuando el ciclo de vida del componente no está activo.

La implementación de Live Data en este proyecto se realiza de la forma siguiente.

- Los datos se encuentran almacenados en la base de datos Room esta base de datos contiene varias tablas o entidades diferentes dependiendo del dato que queramos almacenar (jugadores, plantillas, entrenamientos, partidos).
- Existe un componente llamado DAO que se encarga de toda la comunicación con la base de datos es decir realizar consultas inserciones modificaciones y eliminaciones. En la clase ViewModel es donde se instancian y modifican los objetos de tipo Live Data. Para consultar los datos, esta clase implementa un *listener* que, ante cambios en la base de datos modifica el valor de los objetos Live Data manteniendo siempre sincronizados los datos entre cliente y servidor.
- En las actividades se implementa un *observer* a los objetos Live Data, por lo que cuando estos son modificados, las actividades reciben los cambios y pueden actualizar la UI.

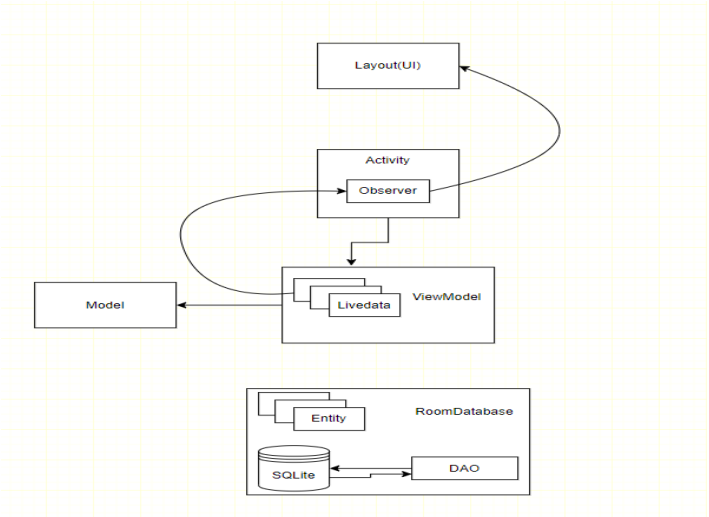


Ilustración 7: -Componentes del sistema

Ilustracion 8: -Diseño de la aplicación

En cuanto a la interfaz de usuario, es bastante simple, con el fin de que la aplicación sea fácil de usar por el usuario. Se ha utilizado una gama de colores pertenecientes a la entidad deportiva del CD Teruel como son el rojo y el azul. Por otra parte, he utilizado unos botones personalizados para las diferentes opciones de la plantilla con el fin de que quedara más estético y poder poner todas las opciones en una misma línea.

En la Ilustración 8 podemos ver el diseño de la aplicación.

3.4 Diagramas

3.4.1 Diagrama de clases

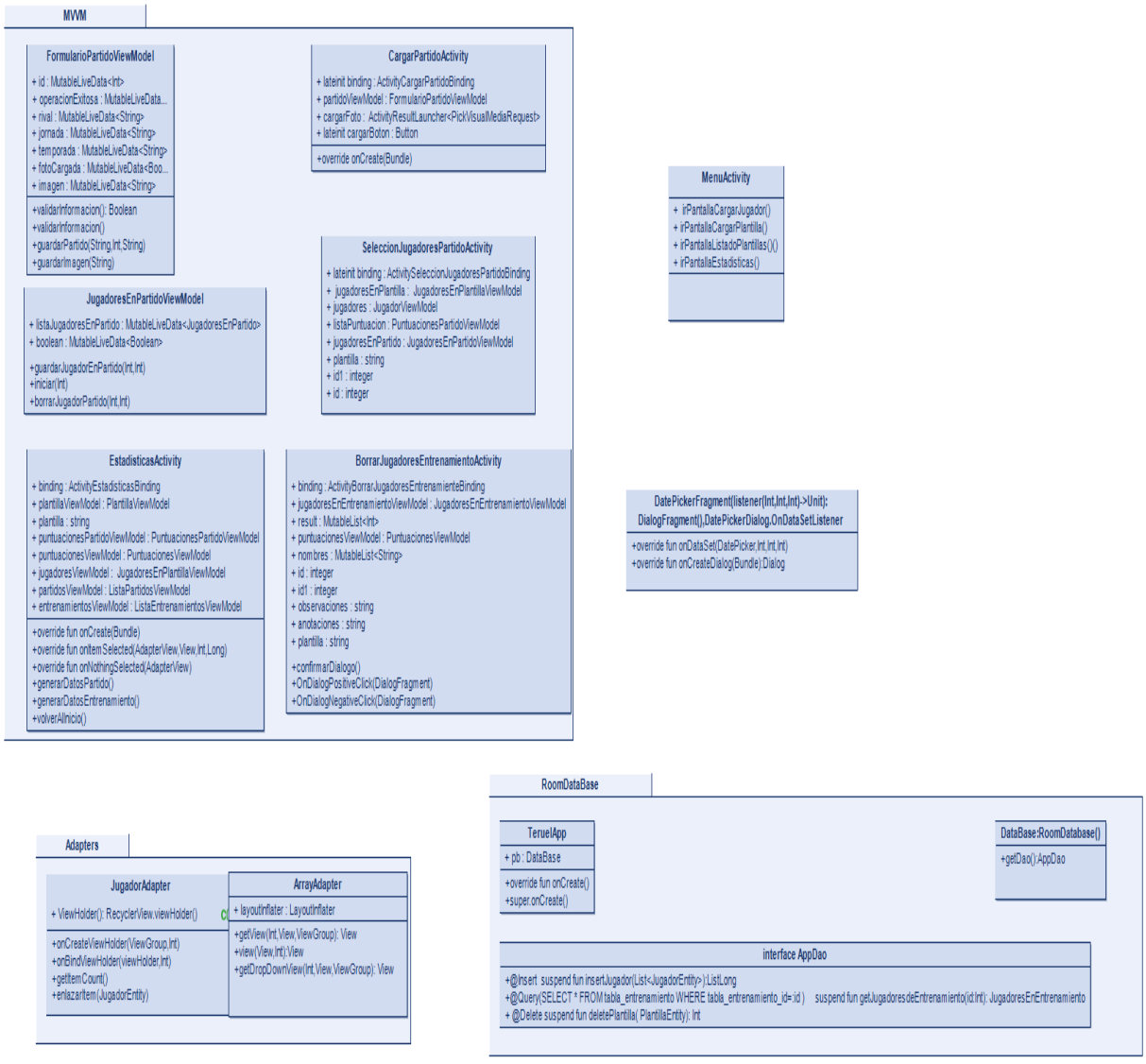


Ilustración 9: -Diagrama de clases

En la Ilustración 9 podemos ver el diagrama de clases de la aplicación. Este diagrama esta resumido y muestro una selección de clases dentro de la estructura de la aplicación para que se observe visualmente.

En la sección MVVM muestro por una parte el 'FormularioPartidoViewModel' que contiene los métodos que se utilizaran en 'CargarPartidoActivity' a través del objeto partidoViewModel de tipo viewModels.

Los cambios que hace el usuario en la interfaz de usuario se guardan en el objeto *binding* a través de un *adapter*.

Las actividades 'SelecciónJugadoresActivity', 'BorraJugadoresActivty' y 'EstadisticasActivty' siguen una lógica parecida cambiando los métodos que aplicamos en los diferentes ViewModels y Adapters.

Los Adapters sirven para gestionar elementos de tablas de las diferentes entidades de la base de datos que se representan en la interfaz de usuario a través de RecyclerViews.

El funcionamiento de RoomDataBase lo explicare en su correspondiente apartado.

3.4.2 Diagramas de actividades

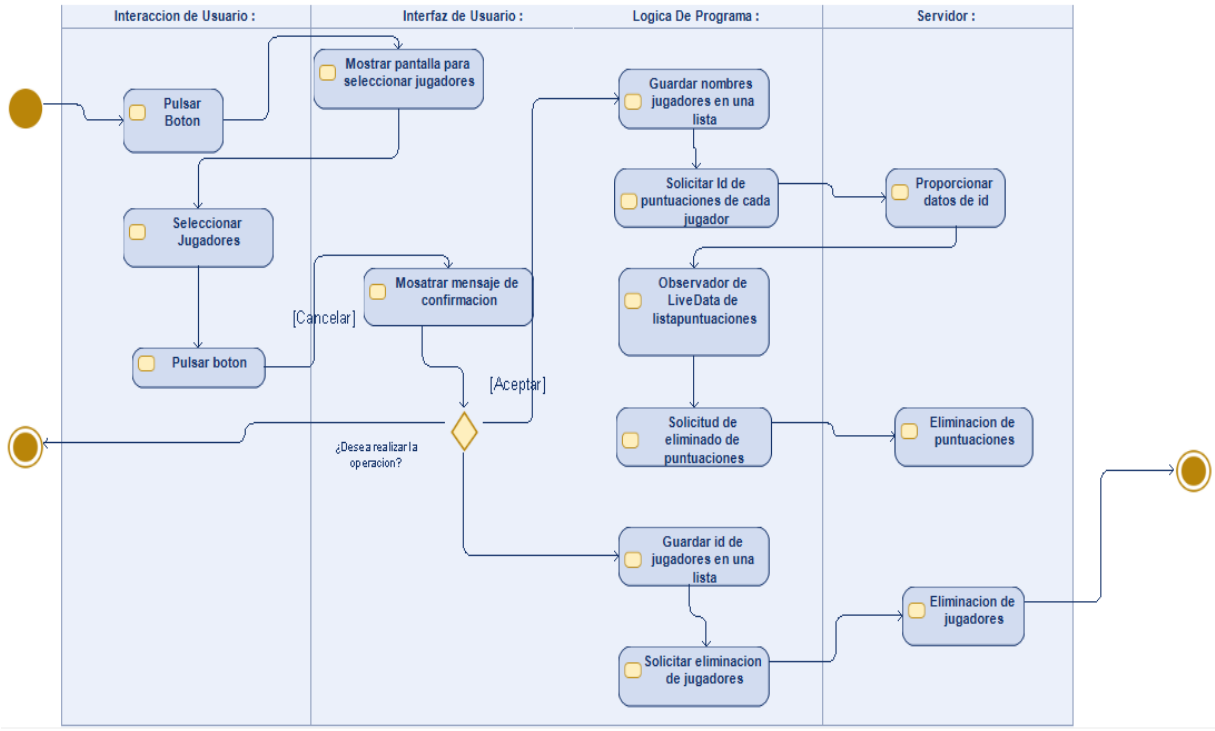


Ilustración 10: -Diagrama de actividades "Eliminación de Jugador de un entrenamiento"

En la Ilustración 10 podemos observar cómo se relacionan los diferentes procesos que se llevan a cabo en cada una de las partes (Interacción del Usuario, Interfaz de Usuario, Lógica de Programa y Servidor).

Me parece importante comentar la importancia de que la Interfaz de Usuario muestre un mensaje de confirmación sobre si se desea realizar la operación ya que al ser un borrado de jugador es una operación delicada que lleva consigo un proceso largo en caso de que el usuario se haya equivocado en su decisión.

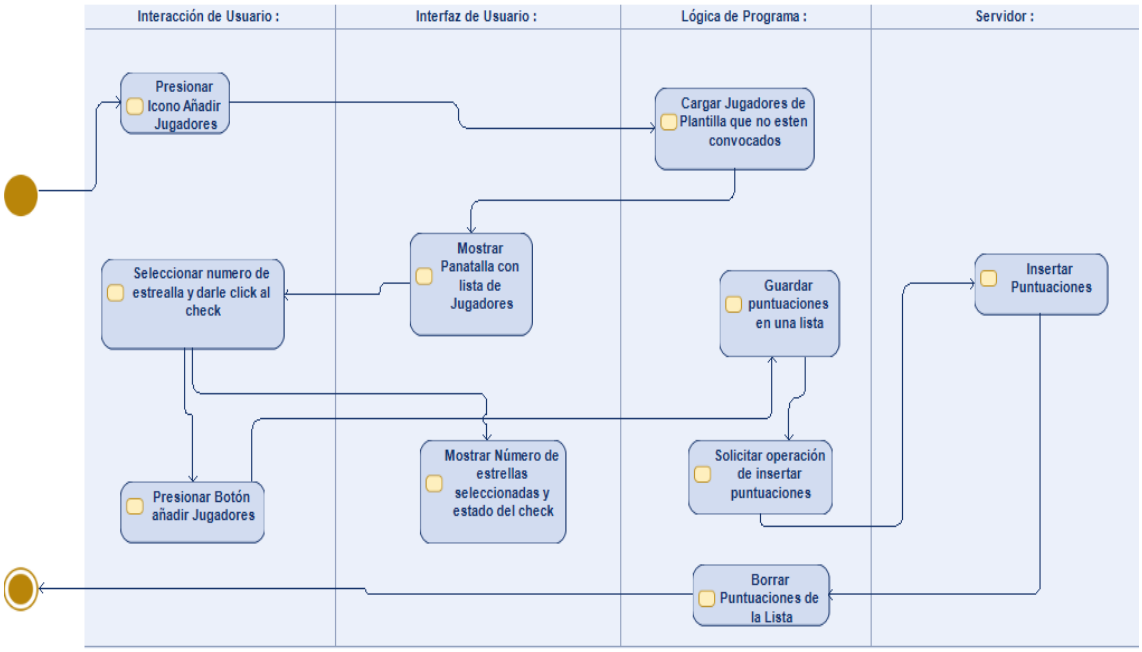


Ilustración 11: - Insertar puntuaciones en jugadores de partido

En la Ilustración 11 vemos el proceso para añadir una puntuación a un jugador es importante recalcar que este proceso es el mismo en el caso de que se quiera añadir una puntuación para un jugador en un entrenamiento o en un partido.

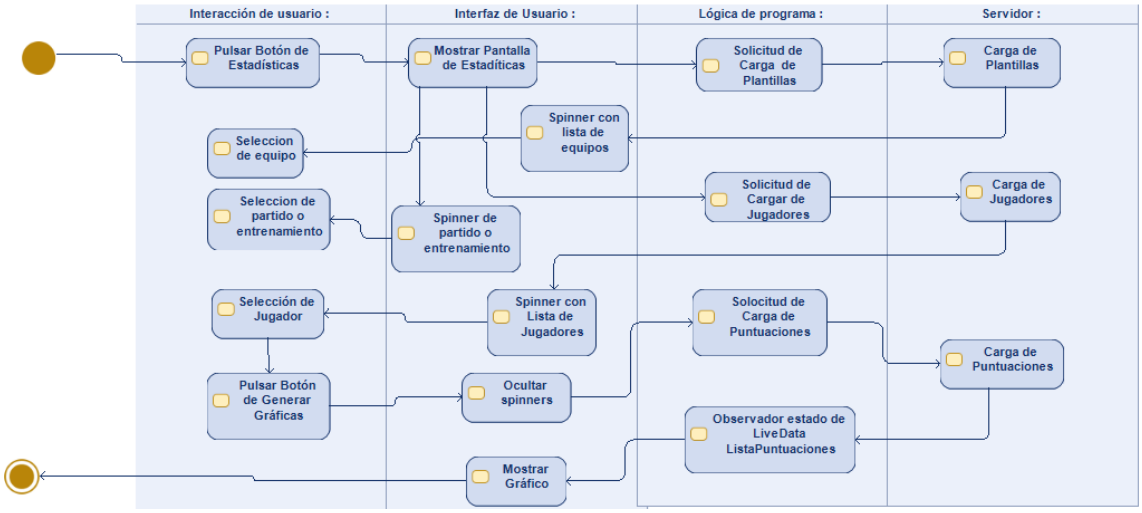


Ilustración 12: -Generar gráfico de estadísticas

En la Ilustración 12 se puede observar el procedimiento para generar una gráfica de valoración de jugadores de una plantilla partidos o entrenamientos a los que hayan

asistido la diferencia de seleccionar uno u otro se encuentra en la opción marcada por el usuario en el *spinner* comentado.

3.4.3 Diagrama de base de datos

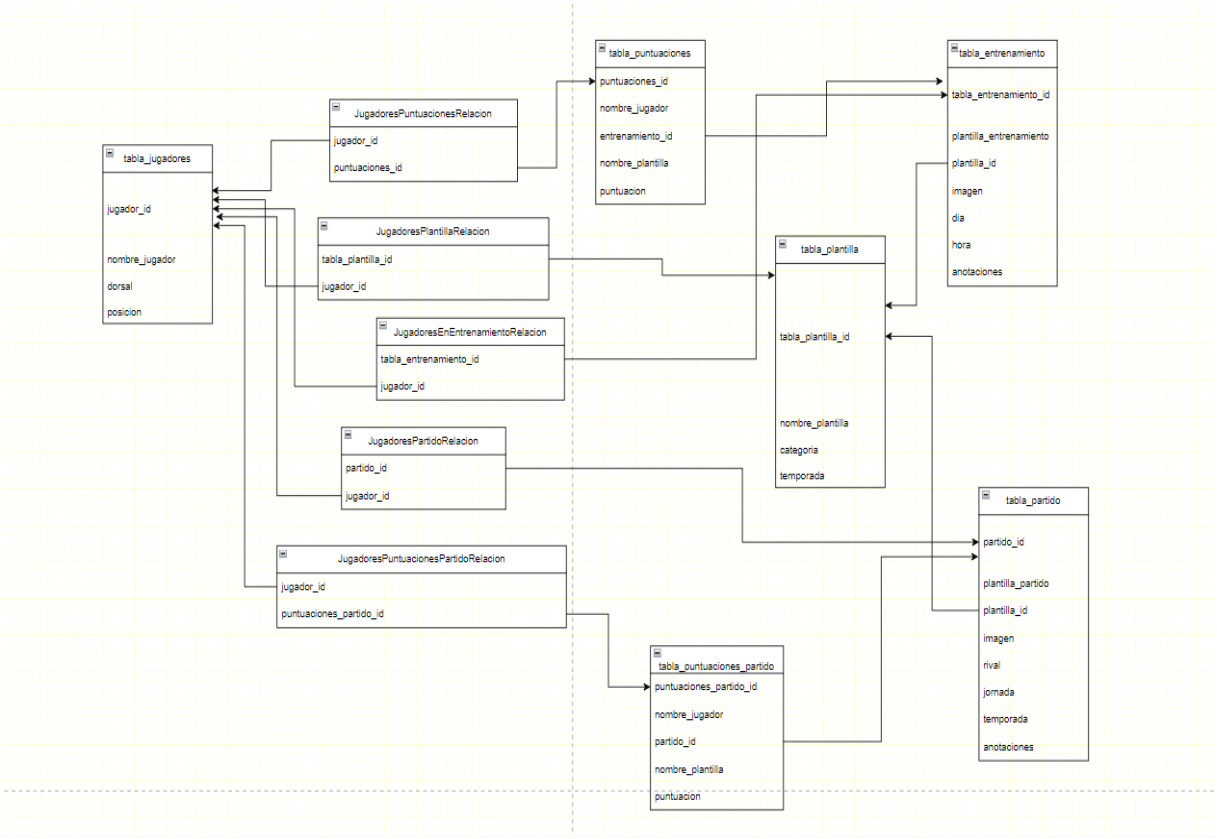


Ilustración 13: -Diagrama de la base de datos

Como se puede ver en la ilustración 13 hay una serie de columnas de las diferentes tablas de las bases de datos que apuntan a otras, a continuación, voy a explicar cómo funcionan a nivel interno.

Las columnas que voy a exponer son tabla_plantilla_id y plantilla_id la primera de tabla_plantilla y la segunda de tabla_entrenamiento.

Antes de explicar esto tengo la necesidad de aclarar que son los *intent*, ya que los voy a utilizar en el ejemplo.

Un *intent* es un objeto que proporciona vinculación en tiempo de ejecución entre componentes separados, como dos actividades. El *intent* representa la intención que tiene una app de realizar una tarea. Puedes usar *intents* para varias tareas; pero, en este ejemplo el *intent* se utiliza para iniciar otra actividad. [10]

```
class ViewHolder(view: View) : RecyclerView.ViewHolder(view) {
    var binding = ItemPlantillaBinding.bind(view)
    var context = view.context
    fun enlazarItem(p: PlantillaEntity) {
        binding.Plantilla.text = "${p.nombre}"
```

Ilustración 14: -Código de enlazar ítem de la clase PlantillasAdapter

Como podemos ver en la Ilustración 14, la clase ViewHolder pertenece a la clase Adapter y la función enlazar ítem se encarga de que por cada elemento del recyclerView se asocien a este todas las columnas de la clase PlantillaEntity.

```
binding.btnCrearEntrenamiento.setOnClickListener { it: View!
    val intent = Intent(context, CargarEntrenamientoActivity::class.java)
    intent.putExtra(Constantes.ID_PLANTILLA, p.tabla_plantilla_id)
    intent.putExtra(Constantes.NOMBRE_PLANTILLA, p.nombre)
    context.startActivity(intent)
```

Ilustración 15 -Código del botón CrearEntrenamiento de la clase Plantillas Adapter

Como podemos ver en la Ilustración 15, posteriormente cuando el usuario le da al botón de crear el entrenamiento se crea una nueva actividad (CargarEntrenamientoActivity) a través del *intent*, gracias al método putExtra() la primaryKey tabla_plantilla_id se guardar en las Constante ID_PLANTILLA para poder ser utilizada en la actividad comentada.

```
fun guardarEntrenamiento(s: String,id:Int,i:String) {
    if (validarInformacion() ) {
        operacionExitosa.value=true
        val pEntrenamiento =
            EntrenamientoEntity( tabla_entrenamiento_id: 0, s,id,i, dia.value!!, hora.value!!, anotaciones: "")
        viewModelScope.launch { this: CoroutineScope
            val result = withContext(Dispatchers.IO) { this: CoroutineScope
                TeruelApp.pb.getDao().insertEntrenamiento(
                    arrayListOf(
                        (pEntrenamiento)
                    )
                )
            }


        }

    }
```

Ilustración 16: -Parte de Código para guardar un entrenamiento de la clase EntrenamientoViewModel

Como se pude ver en el método mostrado en la ilustración 16 se procede a la inserción del entrenamiento en la base de datos pasándole como parámetro un id que se corresponde con plantilla_id.

```
val id = intent.getIntExtra(Constantes.ID_PLANTILLA, defaultValue: 0)
```

```
entrenamientoViewModel.guardarEntrenamiento(nombre!!,id,entrenamientoViewModel.imagen.value!!)
```

Ilustración 17: -Partes de Código de la clase CargarEntrenamientoActivity

En el primero de los recortes de la ilustración 17 asignamos el valor de la constante ID_PLANTILLA procedente de la pantalla de ListadoPlantillasActivity a una variable utilizando el método getIntExtra() que posteriormente pasaremos como parámetro a la función guardarEntrenamiento.

4 Desarrollo del sistema

Una vez completada la fase de análisis y diseño del proyecto, la siguiente etapa es implementar el sistema. En este se define como ha sido el desarrollo de los apartados más relevantes de todo lo especificado en el diseño, utilizando las herramientas escogidas al comenzar el proyecto. Para los ficheros de diseño se ha utilizado XML.

4.1 Biblioteca Room

Como ya he mencionado anteriormente he utilizado Room para el diseño de la base de datos.

Room es una de las bibliotecas de Android Jetpack, un conjunto de librerías para seguir las prácticas recomendadas por Google a la hora de desarrollar aplicaciones para Android.

Esta biblioteca funciona proporcionando una capa de abstracción para una base de datos SQLite creando una memoria cache en el dispositivo en el que estés utilizando CanteraControl de forma que la base de datos y el acceso a ella es local, lo que supone que los usuarios podrán acceder a la información de su equipo independientemente si tienen conexión a internet o no.

Para declarar una tabla en la base de datos mediante Room, lo único que debe hacerse es sobre una clase de Kotlin añadir una anotación como que se observa en la siguiente imagen, que es una de las clases y tablas de CanteraControl.

```
@Entity(tableName = "tabla_jugadores")
data class JugadorEntity(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="jugador_id") val jugador_id:Int=0,
    @ColumnInfo(name="nombre_jugador")val nombre:String,
    @ColumnInfo(name="dorsal")val dorsal:String,
    @ColumnInfo(name="posicion")val posicion:String,
)
```

Ilustración 18: -Código con las columnas de la tabla_jugadores

Como se puede ver en la ilustración 18, tenemos distintas etiquetas cada una de ellas se utiliza para una determinada función:

- **@Entity =** Con esta etiqueta hacemos que una clase se convierta en una tabla de nuestra base de datos.
- **@ColumnInfo=** Esta etiqueta la utilizamos para las columnas de las tablas y hace que podamos elegir un nombre de tablas distinto a las variables asociadas a ellas.
- **@PrimaryKey**= La utilizamos para registrar de manera única cada registro de nuestra tabla. Como podemos observar en el código esta clave es autogenerada y se genera cada vez que se crea un registro.

Para controlar una base de datos mediante Room serán necesarios dos ficheros:

Fichero database

```
@Database(
    entities = [JugadorEntity::class, PlantillaEntity::class, JugadoresPlantillaRelacion::class,
        PartidoEntity::class, EntrenamientoEntity::class, JugadoresEnEntrenamientoRelacion::class,
        PuntuacionesEntity::class, JugadoresPuntuacionesRelacion::class, JugadoresPuntuacionesPartidoRelacion::class,
        PuntuacionesPartidoEntity::class,JugadoresPartidoRelacion::class],
    version = 1
)
abstract class DataBase : RoomDatabase() {
    abstract fun getDao(): AppDao
}
```

Ilustración 19: -Código para crear la base de datos y los métodos DAO asociados a ella

Como se pude ver en la Ilustración 19, la función de esta clase es declarar la base de datos y las clases que compondrán sus tablas, además añadimos un método abstracto

con el que obtenemos una instancia del objeto DAO cuya utilidad voy a explicar a continuación.

Fichero DAO

En este fichero se almacenarán las diferentes operaciones a través de las cuales interactuamos con la base de datos, estas operaciones pueden ser de los diferentes tipos:

Consulta:

```
@Query("SELECT * FROM tabla_plantilla")
suspend fun getAllPlantillas(): List<PlantillaEntity>
```

Ilustración 20: -Código de método de consulta a la base de datos para obtener la lista de plantillas

En la ilustración 20 se pude observar el código de la clase Dao para solicitar el conjunto de plantillas de la base de datos.

Inserción:

```
@Insert
suspend fun insertPlantilla(plantilla: List<PlantillaEntity>):List<Long>
```

Ilustración 21: -Código de método de inserción de una plantilla en la base de datos.

En la ilustración 21 se pude observar el código de la clase Dao para insertar una plantilla en nuestra base de datos.

Eliminación:

```
@Delete
suspend fun deletePlantilla(plantilla: PlantillaEntity): Int
```

Ilustración 22: -Código de método de eliminación de plantilla en nuestra base de datos

En la ilustración 22 se pude observar el código de la clase Dao para eliminar una plantilla en nuestra base de datos.

4.1.1 Relaciones entre entidades

Cuando se trabaja con bases de datos en aplicaciones Android es común que haya entidades que estén relacionadas entre sí de diferentes maneras:

Uno a uno: A cada elemento de la entidad principal solo le corresponde uno de la entidad secundaria y viceversa.

Uno a muchos: A cada elemento de la entidad le corresponden muchos de la entidad secundaria pero un elemento de la entidad secundaria solo puede ir asociado a uno de la principal.

Muchos a muchos: A cada elemento de la entidad principal le corresponden muchos de la secundaria y viceversa.

En nuestra aplicación las relaciones entre entidades son de muchos a muchos y voy a explicar la forma en que las he modelado.

```
@Entity(primaryKeys=["tabla_entrenamiento_id","jugador_id"])
data class JugadoresEnEntrenamientoRelacion(
    val tabla_entrenamiento_id:Int,
    val jugador_id:Int)
```

Ilustración 23: - Columnas de la tabla JugadoresEnEntrenamientoRelacion

```
data class JugadoresEnEntrenamiento(
    @Embedded val entrenamiento: EntrenamientoEntity,
    @Relation(
        parentColumn = "tabla_entrenamiento_id",
        entityColumn = "jugador_id",
        associateBy = Junction(JugadoresEnEntrenamientoRelacion::class)
    )
    val jugadores:List<JugadorEntity>
)
```

Ilustración 24: -Código con los elementos de la clase JugadoresEnEntrenamiento

Estas dos clases combinadas de la ilustración 23 y 24 están diseñadas para manejar una relación compleja de muchos a muchos entre entrenamientos y jugadores en una base de datos de Room.

Estas dos clases se utilizan para representar una relación compleja de muchos a muchos entre los jugadores y los entrenamientos en una base de datos Room. La información del entrenamiento se incrusta en la tabla de JugadoresenEntrenamiento y la relación

entre las dos entidades se gestiona a través de la tabla de enlace JugadoresEnEntrenamientoRelacion.

4.2 Navegación entre pantallas

Para la navegación entre pantallas de la aplicación se ha utilizado ViewBinding la cual es una característica de Android Studio que simplifica la forma en que los desarrolladores de aplicaciones Android interactúan con las vistas (UI) en sus de vistas para cada archivo de diseño XML. A diferencia de findViewById que era la forma tradicional de obtener referencias a las vistas en el pasado, ViewBinding genera automáticamente clases de enlace de vistas para cada archivo de diseño XML en tu proyecto, lo que facilita el acceso a las vistas y elimina la necesidad de escribir código de búsqueda manual.

Para habilitar ViewBinding en mi proyecto he seguido los siguientes pasos:

1. Agregar la configuración de ViewBinding en mi archivo 'build.gradle' (módulo de la aplicación).

```
buildFeatures{
    viewBinding=true
}
```

Ilustración 25: -Implementación de la biblioteca viewBinding

2. Android Studio genera automáticamente las clases de enlace de vista para tus archivos XML de diseño
   Una vez habilitado ViewBinding, accedo a las vistas de mi diseño XML de la siguiente manera.

```
// Acceso a el fichero activity_cargar_plantilla.xml
binding = ActivityCargarPlantillaBinding.inflate(layoutInflater)
setContentView(binding.root)
// Acceso aun elemento de la vista
val nombre=binding.etNombre
```

Ilustración 26: -Código para acceso a una variable de un fichero

A continuación, voy a mostrar de forma gráfica la navegación entre las diferentes pantallas de la aplicación.

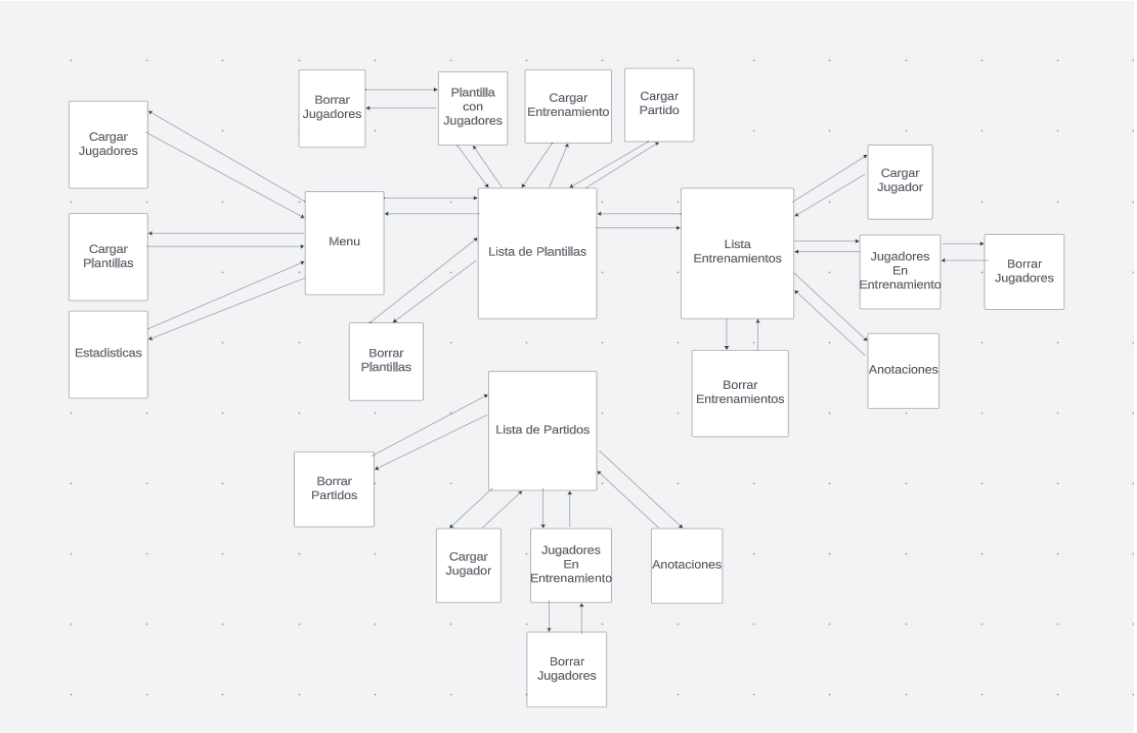


Ilustración 27: -Navegación entre las distintas pantallas de la aplicación.

4.3 Tecnologías y Bibliotecas utilizadas

4.3.1 Arquitectura MVVM

La arquitectura MVVM se basa en los siguientes componentes clave:

- **Model:** El Modelo representa la capa de datos de la aplicación. Contiene la lógica de negocio, los datos y las operaciones relacionadas con la manipulación de datos. Puede incluir acceso a bases de datos, servicios web, almacenamiento local y cualquier fuente de datos necesaria para la aplicación.
- **View:** La Vista representa la interfaz de usuario (UI) de la aplicación. Es responsable de mostrar los datos del Modelo y de capturar las interacciones del usuario, como toques en la pantalla, clics de botones, etc. Sin embargo, la Vista no contiene lógica de negocio; su papel principal es mostrar la información y las interacciones al usuario.
- **ViewModel**: El ViewModel actúa como un intermediario entre el Modelo y la Vista. Su principal responsabilidad es proporcionar los datos necesarios desde el Modelo a la Vista de una manera que sea fácil de mostrar en la interfaz de usuario. También gestiona el estado de la interfaz de usuario y las acciones del usuario. El ViewModel generalmente se comunica con el Modelo para obtener o actualizar datos, y luego expone estos datos a la Vista a través de propiedades observables.

4.3.2 Biblioteca DataBinding

La biblioteca DataBinding permite vincular los elementos de la interfaz de Usuario (UI) de una aplicación Android directamente a los datos subyacentes en el modelo de datos. Esto simplifica y mejora la forma en que se manejan los datos en la aplicación al eliminar gran parte del código de enlace manual que normalmente se necesita. Además, DataBinding es especialmente útil cuando se implementa el patrón de arquitectura MVVM como es mi caso ya que simplifica la comunicación entre la vista y el modelo.

Para habilitar DataBinding al proyecto he seguido los siguientes pasos:

1. Agregar la configuración de DataBinding en mi archivo 'build.gradle' (módulo de la aplicación).

```
buildFeatures{
    dataBinding true
}
```

Ilustración 28: -Implementación de la biblioteca DataBinding

2. Vincular los elementos de la interfaz de usuario a variables.

```
<data>
    <variable
        name="modelo"
        type="com.example.tfg.cargarplantilla.FormularioPlantillaViewModel" />
```

Ilustración 29: -Código para vincular elementos de la interfaz de usuario a variables

3. Utilizar DataBinding en las vistas.

```
<EditText
    android:id="@+id/etNombre"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:hint="NOMBRE"
    android:text="@={modelo.nombre}"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/Linear"
    />
```

Ilustración 30: -Código para configurar el EditText de la interfaz de usuario

4. Configurar el enlace de datos con el ViewModel en la actividad.
   La declaración de la variable plantillaViewModel está fuera del método onCreate() mientras que la asignación de plantillaViewModel al *model* está dentro.

```
val plantillaViewModel: FormularioPlantillaViewModel by viewModels()
```

```
binding.modelo = plantillaViewModel
```

Ilustración 31: -Código para enlazar datos del ViewModel en la Actividad

4.4 Implementación de la base de datos

Para garantizar la persistencia de los datos y almacenar toda la información relacionada con los puntos de interés era necesario disponer de una base de datos. Tal y como se ha explicado en la fase de diseño se ha utilizado Room para almacenar los datos.
Para la implementación de la base de datos Room en mi diseño he seguido los siguientes pasos:

1. Agregar las dependencias de Room en mi archivo ‘’build.gradle” (módulo de la aplicación).

```
//Room
implementation "androidx.room:room-ktx:2.4.0"
kapt "androidx.room:room-compiler:2.4.0"
```

Ilustración 32: -Código para implementar la biblioteca Room

2. Definir entidades en la base de datos.

```
@Entity(tableName = "tabla_plantilla")
data class PlantillaEntity(
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name="tabla_plantilla_id") var tabla_plantilla_id:Int=0,
    @ColumnInfo(name="nombre_plantilla")val nombre:String,
    @ColumnInfo(name="categoria")val categoria:String,
    @ColumnInfo(name="temporada")val temporada:String)
```

Ilustración 33: Código con las columnas de la tabla_plantilla

3. Crear una interfaz DAO para definir las operaciones de acceso a la base de datos.

```
@Dao
interface AppDao {
    @Insert
    suspend fun insertJugador(jugadores: List<JugadorEntity>): List<Long>
```

Ilustración 34: Código para declarar la clase AppDao

4. Crear la base de datos Room.

```
@Database(
    entities = [JugadorEntity::class, PlantillaEntity::class, JugadoresPlantillaRelacion::class,
        PartidoEntity::class, EntrenamientoEntity::class, JugadoresEnEntrenamientoRelacion::class,
        PuntuacionesEntity::class, JugadoresPuntuacionesRelacion::class, JugadoresPuntuacionesPartidoRelacion::class,
        PuntuacionesPartidoEntity::class,JugadoresPartidoRelacion::class],
    version = 1
)
```

Ilustración 35: Código para crear la base de datos y los métodos DAO asociados a ella

5. Configurar y obtener una instancia de la base de datos en la aplicación.

```
abstract class DataBase : RoomDatabase() {
    abstract fun getDao(): AppDao
}
```

Ilustración 36: Código para obtener instancia de la base de datos

```
class TeruelApp: Application() {
    companion object {
        lateinit var pb: DataBase

    }

    override fun onCreate() {
        super.onCreate()
        pb = Room.databaseBuilder( context: this, DataBase::class.java,  name: "tabla_datos")
            .fallbackToDestructiveMigration().build()
    }
}
```

Ilustración 37: Código para inicializar la base de datos

6. Manejo de hilos y corrutinas:
   Las operaciones de acceso a la base de datos deben realizarse en hilos o corrutinas diferentes para no bloquear el hilo principal de la interfaz de usuario.

```
fun obtenerEntrenamiento(dia: String) {
    viewModelScope.launch { this: CoroutineScope
        Entrenamiento.value = withContext(Dispatchers.IO) { this: CoroutineScope
            TeruelApp.pb.getDao().getEntrenamientoDePlantilla(dia)
        }
    }
}
```

Ilustración 38: Código para obtener un entrenamiento según un día especifico

4.5 Utilización de Live Data

La utilización de objetos LiveData es un tema importante en este proyecto, ya que permite tener una interfaz de datos reactiva ante los cambios realizados en la base de datos, de modo que la persistencia de datos es siempre óptima. Live Data es una clase de contenedor de datos observables. A diferencia de un observable regular, LiveData está optimizado para ciclos de vida, lo que significa que respeta el ciclo de vida de otros componentes de las apps, como actividades, fragmentos o servicios. Esta optimización garantiza que LiveData solo actualice observadores de componentes de apps que tienen un estado de ciclo de vida activo.

Para la utilización de Live Data en mi proyecto he seguido los siguientes pasos:

1. Crear instancia LiveData para contener un tipo de datos determinado. Esta instancia la he creado en las clases del tipo ViewModel.

```
class FormularioJugadorViewModel: ViewModel(){
        var id = MutableLiveData<Int>()
        var nombre = MutableLiveData<String>()
        var dorsal = MutableLiveData<String>()
        var posicion = MutableLiveData<String>()
        var operacionExitosa = MutableLiveData<Boolean>()
```

Ilustración 39: -Código con la declaración de diferentes variables LiveData

En la Ilustración 39 he querido reflejar esta parte del código ya que pertenece a la misma clase (FormularioJugadorViewModel) que utilice para explicar el uso de la biblioteca DataBinding.
En esta clase las dos tecnologías están relacionadas ya que DataBinding vincula elementos de la interfaz de usuario, Edit Text en este caso a variables y posteriormente son almacenadas en objetos LiveData.

```
    private fun validarInformacion(): Boolean {
        return !(nombre.value.isNullOrEmpty()||
                 dorsal.value.isNullOrEmpty() ||
                 posicion.value.isNullOrEmpty()
                 )
    }
```

Ilustración 40: -Código para comprobar si la información del usuario es correcta

Esta función expuesta en la Ilustración 40 comprueba el estado de los LiveData y si alguno de ellos está vacío devuelve un False.

```
fun guardarJugador() {
    if (validarInformacion()) {
        operacionExitosa.value = true
        var pJugador = JugadorEntity( jugador_id: 0,nombre.value!!,dorsal.value!!, posicion.value!!)
        viewModelScope.launch { this: CoroutineScope  val result = withContext(Dispatchers.IO) { this: CoroutineScope  TeruelApp.pb.getDao().insertJugador(
            arrayListOf(
                (pJugador)
                        )
                    )
                }


            }
        }

    else{
        operacionExitosa.value=false
    }
}
```

Ilustración 41: -Código para guardar un jugador

Como podemos observar en esta función de la Ilustración 40, si la función comentada anteriormente nos ha devuelto un True esto significa que los datos del jugador son válidos por lo que actualizo el valor del objeto LiveData operacionExitosa a True y procedo a hacer la operación en segundo plano de la inserción del jugador a la base de datos.

2. Observar los cambios del objeto Live Data desde la actividad y actualizar la interfaz de usuario en consecuencia.

```
binding.etbtnGuardar.setOnClickListener { it: View!
    jugadorViewModel.guardarJugador()
    jugadorViewModel.operacionExitosa.observe( owner: this, Observer{ it: Boolean!
        if (jugadorViewModel.operacionExitosa.value == true) {
            Toast.makeText(
                 context: this,
                 text: "El jugador se ha guardado correctamente",
                Toast.LENGTH_SHORT
            ).show()
        jugadorViewModel.borrarDatos()

        }
        else
            Toast.makeText(
                 context: this,
                 text: "El jugador no se  ha guardado correctamente debido a que todos los campos deben estar completos",
                Toast.LENGTH_SHORT
            ).show()
        jugadorViewModel.borrarDatos()

    })
}
```

Ilustración 42: -Código para guardar jugador cuando el usuario presiona un botón

En el momento que el usuario presiona el botón de guardar el observador comprobara el estado del LiveData operacionExitosa en el caso de que sea true mostrara el mensaje “el jugador se ha guardado correctamente” en caso contrario “El jugador no se ha guardado correctamente debido a que todos los campos deben estar completos”.

4.6 Creación de la interfaz de usuario

Un diseño de la estructura de interfaz de usuario en una aplicación, como, por ejemplo, en una actividad. Todos los elementos del diseño se crean usando la jerarquía de objetos View y ViewGroup. Una View suele mostrarse en un elemento que el usuario puede ver y con el que puede interactuar. Por su parte, ViewGroup es un contenedor invisible que define la estructura de diseño de View y otros objetos ViewGroup.

Los objetos View suelen llamarse widgets y pueden ser una de las muchas subclases, como Button o TextView. Los objetos ViewGroup se denominan generalmente *layouts* y pueden ser de muchos tipos que proporcionan una estructura diferente, como LinearLayout y Constraintlayout [11].

La interfaz de usuario en Android se define típicamente utilizando archivos de diseño XML que describen la disposición y el aspecto visual de los elementos de la interfaz de usuario, como botones, campos de texto, imagen etc. Estos archivos de XML se encuentran en la carpeta de recursos de la aplicación.

Como se ha comentado anteriormente, los widgets se encuentran contenido en los layouts, estos pueden ser de muchos tipos, los escogidos para el desarrollo de la aplicación han sido los ConstraintLayout y los LinearLayout predominando los segundos sobre los primeros.

LinearLayout es un grupo de vistas que alinea todos los elementos secundarios en una única dirección, de manera vertical u horizontal. Puedes especificar la dirección del diseño con el atributo android:orientation [12].

ConstraintLayout te permite crear diseños grandes y complejos con una jerarquía de vistas plana (sin grupos de vistas anidadas). Es similar a RelativeLayout en cuanto a que se presentan todas las vistas de acuerdo con las relaciones entre las vistas del mismo nivel y el diseño de nivel superior, pero es más flexible que RelativeLayout y más fácil de usar con el editor de diseño de Android Studio[13].

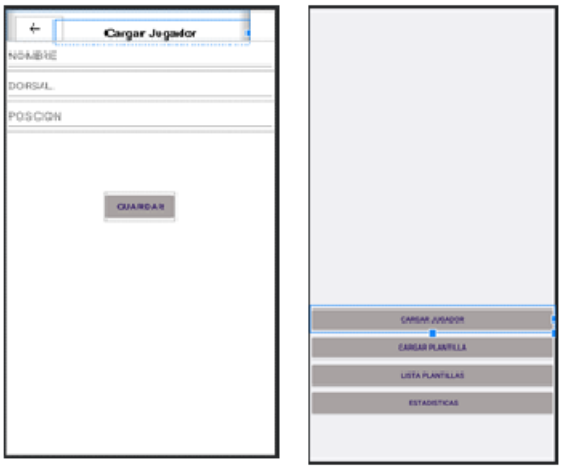


Ilustración 43: Uso de ConstraintLayout y LinearLayout

A continuación, voy a comentar los aspectos más relevantes e inusuales de lo que ha sido la implementación de las interfaces en la aplicación. Con inusuales me refiero a los elementos que se salen de lo que sería común y sencillo como por ejemplo colocar elementos simples en los layouts.

En la aplicación todos los datos que se muestran como ítems en una lista dinámica que nos permite desplazarnos hacia arriba y abajo a través de esta son tratados con RecyclerView.

RecyclerView facilita que se muestren de manera eficiente grandes conjuntos de datos. Tú proporcionas los datos y defines el aspecto de cada elemento, y la biblioteca RecyclerView creará los elementos de forma dinámica cuando se los necesite.

Como su nombre lo indica, RecyclerView recicla esos elementos individuales. Cuando un elemento se desplaza fuera de la pantalla, RecyclerView no destruye su vista. En cambio, reutiliza la vista para los elementos nuevos que se desplazaron y ahora se muestran en pantalla. Esto mejora en gran medida el rendimiento y la capacidad de respuesta de tu app y reduce el consumo de energía [14].

El RecyclerView se encargar de inflar la vista, pero en la pantalla que muestra la lista de lugares es necesario utilizar un *adaptor*. Un *adaptor* es un objeto de una clase que implementa la interfaz Adapter. Este actúa como un enlace entre un conjunto de datos y un adaptador de vista. El conjunto de datos puede ser cualquier cosa que presente datos de una manera estructurada, en este caso se trata de una lista de objetos JugadorEntity. Un adaptor coge un conjunto de datos y los recorre generando una vista para cada uno de los registros que contenga el conjunto.

Los adaptadores pueden mostrar grandes conjuntos de datos muy eficientemente, ya que cargan solamente los objetos View que están listos en pantalla o que están a punto de moverse en la pantalla. De esta manera, la memoria consumida por un adaptador puede ser constante e independiente del tamaño del conjunto de datos. [15]

Primero es necesario crear un fichero XML que defina la vista en la que el adaptador cargará la información de cada registro del conjunto de datos.

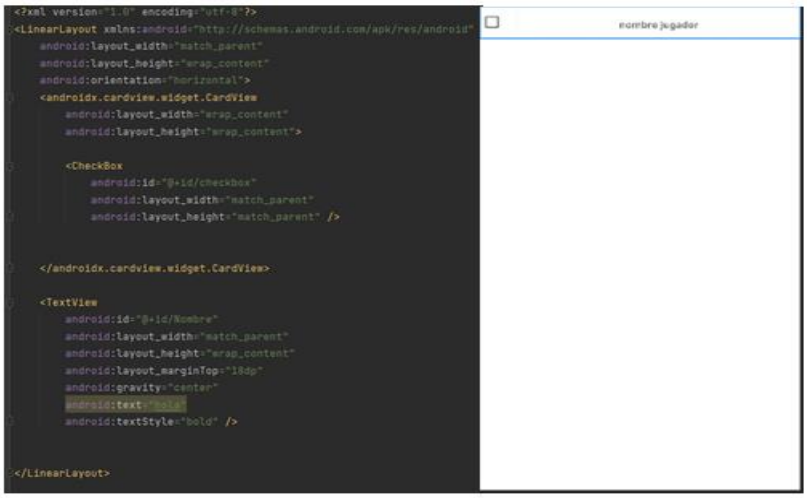


Ilustración 44: -Vista en la que el adaptador cargara cada elemento

Posteriormente se implementa el adaptador sobrescribiendo los métodos onCreateViewHolder(), getItemCount() y onBindViewHolder() para definir manualmente de que lista se deben tomar los datos, a que vista se van a inflar los datos y a que elementos de dicha vista va a ir cada dato.

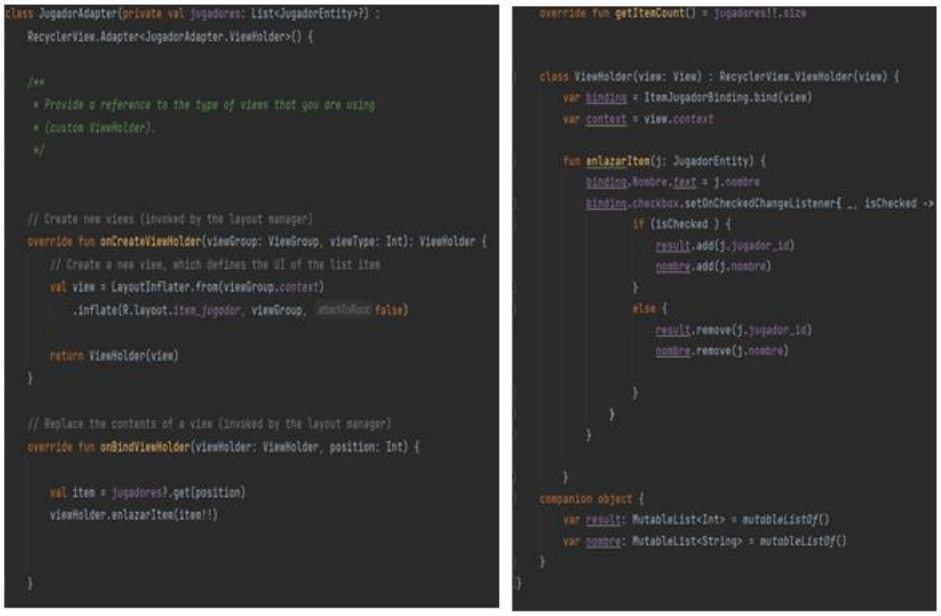


Ilustración 45: Clase JugadorAdapter

Voy a explicar lo que realiza la función enlazarItem porque es común a todas las clases *adapter* y tiene una importancia bastante notable en la aplicación.
Esta función coge el elemento nombre del fichero ítem jugador, imagen expuesta anteriormente y le da el valor de la columna nombre perteneciente a la entidad

JugadorEntity. En el fichero también existe un elemento llamado CheckBox y dentro de esta función implementamos un listener que nos avisa de cada vez que ha cambiado el estado del CheckBox, en el caso de que este activado se van a añadir a la lista result y al nombre el valor de las columnas id y nombre respectivamente mientras que si esta desactivado lo que hace es borrar estos elementos de las listas.
Como esta función la utilizamos dentro del onBindViewHolder se va a aplicar a cada elemento de nuestra lista dinámica.

En la pantalla de estadísticas se deben mostrar las opciones de las plantillas o los jugadores a seleccionar, en esta ocasión ya no podemos utilizar los RecyclerView ya que estas opciones deben situarse dentro de Spinners el procedimiento ha sido el siguiente.

El fichero XML creado es bastante similar al empleado en el RecyclerView lo que si tiene cambios significativos es el uso del adaptador para este caso.

```
class ArrayAdapter(context: Context, opciones: List<String>) :
    ArrayAdapter<String>(context, resource: 0,opciones) {

    private var layoutInflater= LayoutInflater.from(context)

    @SuppressLint("ViewHolder","InflateParams")
    override fun getView(position: Int, convertView: View?, parent: ViewGroup): View {
        val view: View = layoutInflater.inflate(R.layout.item_plantilla_est,  root: null,  attachToRoot: true)
        return view(view,position)
    }

    private fun view(view: View, position: Int): View {
        val opciones:String=getItem(position)?:return view
        val binding=ItemPlantillaEstBinding.bind(view)
        binding.nombrePlantillaEst.text=opciones
        return view
    }

    override fun getDropDownView(position: Int, convertView: View?, parent: ViewGroup): View {
        var cv=convertView
        if(cv==null)
            cv=layoutInflater.inflate(R.layout.item_plantilla_est,parent, attachToRoot: false)
        return view(cv!!,position)
    }
}
```

Ilustración 46: -Clase ArrayAdapter

Como podemos ver en la ilustración 46la clase getView se encarga de devolver la vista que se utilizara para representar el elemento seleccionado en el Spinner. Aquí se infla la vista desde el fichero 'item_plantilla_est.xml' y se establece el texto correspondiente en el elemento de la vista.
La función View se utiliza para configurar los elementos de la vista para cada posición en el Spinner. Aquí se enlaza el texto de la lista de opciones a un elemento de texto en la vista.

Por último, la función getDropDownView se encarga de devolver la vista que se utilizara para representar cada elemento en la lista despegable del Spinner. Si la vista es nula, se infla la vista desde el fichero y luego se establece el texto correspondiente.

4.7 Biblioteca MP Android chart

Para mostrar gráficos en nuestras aplicaciones de Android la biblioteca más utilizada es MPAndroidChart esta es una biblioteca de gráficos muy potente y fácil de usar. [16]

Para utilizar esta biblioteca tenemos que añadir el siguiente repositorio al build.gradle (Project:TFG).

```
allprojects {
    repositories {
        maven { url 'https://jitpack.io' }
```

Ilustración 47: -Repositorio de la biblioteca MPAndroidChart

Además, debemos añadir la siguiente línea de código al build gradle(Module:App).

```
implementation 'com.github.PhilJay:MPAndroidChart:v3.1.0'
```

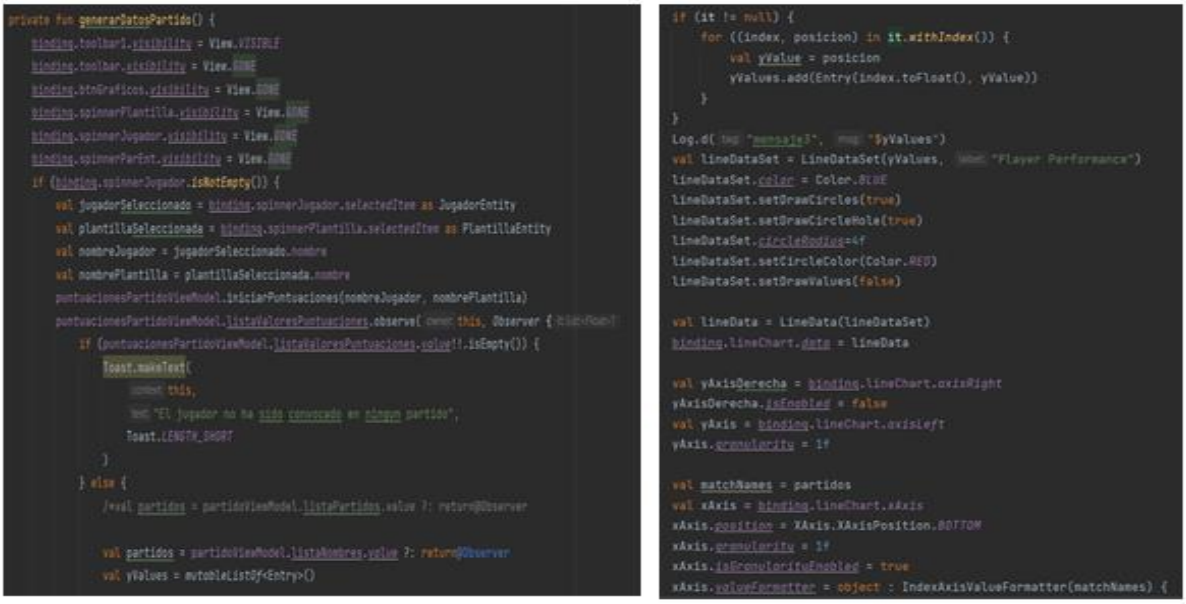
Ilustración 48: -Implementación de la biblioteca MPAndroidChart

Una vez que podemos utilizar la biblioteca MPAndroidChart lo primero que hacemos es crear un elemento MPAndroidChart en el fichero activity_estadisticas.

```
<com.github.mikephil.charting.charts.LineChart
    android:id="@+id/lineChart"
    android:layout_width="match_parent"
    android:layout_height="match_parent"/>
```

Ilustración 49: -Elemento LineChart del fichero 'activity_estadísticas'

A continuación, vamos a utilizar este objeto lineChart en la clase EstadisticasActivity.



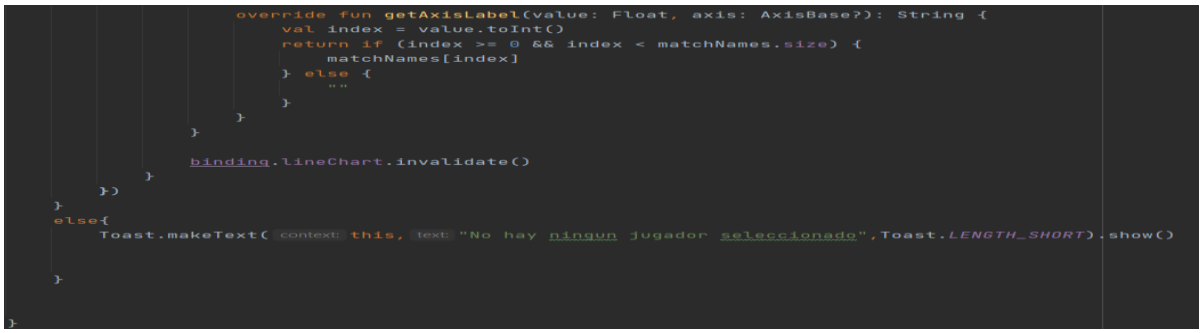


Ilustración 50: -Función generarDatosPartido de la clase EstadisticasActivity

Lo primero que hace la función es ocultar los elementos de la interfaz de usuario correspondientes a la elección de opciones como son los *spinners*, a continuación, se verifica si el usuario ha elegido un jugador si es correcto a través de puntuacionesPartidoViewModel se solicita la consulta a la base de datos de la lista de puntuaciones del jugador seleccionado.

Se comprueba si se ha encontrado una lista de puntuaciones en caso negativo la aplicación mostrara un mensaje por pantalla para que el usuario sepa que el jugador seleccionado no ha sido convocado a ningún partido.

Si el usuario ha seleccionado un jugador el proceso es el siguiente:

- Se obtiene una lista de nombres de partidos desde patidoViewModel.
- Se crean una lista de valores de entrada (‘Entry’) para el grafico de líneas.
- Si la lista it (‘listaValoresPuntuaciones’) no es nula se itera sobre ella con el índice y el valor correspondiente. Luego, se agrega cada valor a la lista ‘yValues’.
- Se configura un conjunto de datos de línea (‘lineDataSet’) con los valores de ‘yValues’ y se establecen varias propiedades entre ellas el color o el radio de los círculos.

- Se crea un objeto de 'LineData' a partir del conjunto del conjunto de datos en línea y se asigna el grafico 'lineChart'en la interfaz de usuario.
- Se configuran los ejes y el formato de los nombres de los partidos en el eje x del grafico usando los nombres de los partidos obtenidos anteriormente.

4.8 DatePickerFragment y TimePickerFragment

Estas dos clases las utilizo para que en el momento de que el usuario cree un partido o un entrenamiento y quiera seleccionar una fecha y una hora aparezca un calendario y un reloj respectivamente.

```
class DatePickerFragment(val listener:(day:Int,month:Int,year:Int) ->Unit): DialogFragment(),
    DatePickerDialog.OnDateSetListener {


    override fun onDateSet(p0: DatePicker?, year: Int, month: Int, dayOfMonth: Int) {
        listener(dayOfMonth,month,year)
    }

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val c= Calendar.getInstance()
        val day=c.get(Calendar.DAY_OF_MONTH)
        val month=c.get(Calendar.MONTH)
        val year=c.get(Calendar.YEAR)

        val picker=DatePickerDialog(activity as Context, listener: this,year,month,day)
        return picker
    }
}
```

Ilustración 51: -Código de la clase DatePickerFragment

Esta clase extiende de 'DialogFragment' y utiliza la interfaz 'DatePickerDialog.OnDataSetListener'.
La clase acepta una función anónima como parámetro 'listener' que toma tres argumentos enteros: 'day', 'month' y 'year'. Esta función se invocará cuando el usuario elija una fecha en el selector de fecha.
Se implementa en el método 'onDataSet' que se llama cuando el usuario elige una fecha en ele selector de fecha. Este método invoca la función 'listener' con los valores seleccionado entre los comentados anteriormente.
Se implementa el método 'onCreateDialog' para crear y configurar un 'DatePickerDialog' con la fecha actual como fecha predeterminada para mostrar al usuario.
El 'DatePickerDialog' se crea con la actividad actual como contexto (activity as context) y se le asigna un oyente de fecha que es esta instancia en la clase 'DatePickerFragment'.

```
class TimePickerFragment(val listener:(String)->Unit) :DialogFragment(), TimePickerDialog.OnTimeSetListener {
    override fun onTimeSet(p0: TimePicker?, hourOfDay: Int, minute: Int) {
        listener("$hourOfDay:$minute")
    }

    override fun onCreateDialog(savedInstanceState: Bundle?): Dialog {
        val calendar=Calendar.getInstance()
        val hour=calendar.get(Calendar.HOUR_OF_DAY)
        val minute=calendar.get(Calendar.MINUTE)
        val dialog=TimePickerDialog(activity as Context, listener: this,hour,minute, is24HourView: false )
        return dialog
    }
}
```

Ilustración 52: -Código de la clase TimePickerFragment

Como podemos ver en la ilustración 53 esta clase es muy similar a la anterior lo único que cambia es los argumentos del listener.

5. Pruebas de la aplicación

Este apartado de la memoria está dedicado a las diferentes versiones que ha tenido la aplicación derivadas de las necesidades manifestadas por el club deportivo Teruel.

Este proyecto requiere de una evaluación final que permita evaluar los objetivos que se han conseguido y cuáles son los puntos débiles del mismo para tratar de mejorarlos en futuras versiones de la aplicación.

La aplicación constaba de una primera versión que fue presentada a parte del staff técnico del Club Deportivo Teruel donde surgieron por su parte una serie de problemas y una falta de funcionalidades que voy a comentar a continuación.

En cuanto a los problemas la mayor parte eran referentes al diseño de la interfaz de usuario, estos eran los siguientes:

- En el menú principal no se veían del todo bien los botones sobre el fondo seleccionado, además de esto había otra serie de botones como el de ‘guardar plantilla’ de la pantalla ‘crear plantilla’ que tampoco se veían de forma clara.
- Al crear un jugador o una plantilla cuando se le daba al botón crear este no te conducía a ninguna pantalla además no se borrar lo escrito en el formulario.
- En las pantallas a las que se dirigía el usuario a partir de la pantalla ‘lista de plantillas’ no había ningún encabezado explicativo de que se hacía en esa pantalla, debido a que se accedía a dichas pantallas a través de un icono podía no quedar claro la funcionalidad de esas pantallas.
- Después de añadir jugadores o borrarlos de una plantilla estos seguían en la pantalla dándole al usuario la opción de añadirlos o borrarlos otra vez.
- Cuando la aplicación pedía una fecha o una hora no existía ni un calendario ni un reloj para poder facilitar al usuario el ingreso de dichos datos.
- En toda la aplicación faltaban elementos visuales que hicieran reconocer la aplicación como perteneciente al Club Deportivo Teruel.

La falta de funcionalidades de la primera versión de la aplicación eran las siguientes:

- No existía ningún sistema de evaluación del desempeño de los jugadores en entrenamientos o en partidos.
- El club solicito que se añadiera alguna forma de poder tener en la aplicación un lugar donde almacenar información sobre el planteamiento previo de partidos y entrenamientos.
- Cuando se eliminaba un jugador de la base de datos este no se eliminaba de los partidos o entrenamientos a los que estuviera convocado.

La segunda versión de la aplicación y la expuesta en este TFG incorpora una serie de funcionalidades y un cambio el diseño de una interfaz de usuario que sirven para satisfacer las necesidades planteadas por el club.

Cabe destacar que los elementos incluidos en la interfaz de usuario que se incluyeron para reconocer la identidad del CD Teruel fueron proporcionados por Víctor Muñiz uno de los responsables de la cantera del club. Los elementos incluidos son los mostrados anteriormente a excepción de una *splashscreen* con el escudo de la cantera.

En cuanto a las funcionalidades se cumplieron con éxito los requisitos planteados a la primera versión.

6. Conclusiones

En este TFG se han alcanzado los objetivos y requisitos planteados al comienzo del mismo. Se ha desarrollado una aplicación sencilla e intuitiva que aporta a los entrenadores de la cantera del CD Teruel una herramienta tecnológica que les facilita el seguimiento del desarrollo de sus jugadores.

Además, he adquirido un gran número de conocimientos de un área que me interesaba (la programación) y sobre una nueva tecnología y un nuevo lenguaje (Android Studio y Kotlin).

6.1 Valoración personal

A nivel personal este proyecto ha supuesto un gran reto para mí ya que es un área del conocimiento que si bien había trabajado en la carrera no tenía nada que ver con lo que me enfrentaba.

En lo que se refiere al desarrollo de la aplicación, ha requerido mucha dedicación, meses de mucho trabajo y muchos momentos de frustración. Diferentes etapas del proceso de

la aplicación me han exigido mucho esfuerzo y recopilación de información como puede ser las relaciones entre entidades y el tratamiento de las puntuaciones y su posterior reflejo en la parte de las estadísticas, pero en gran parte gracias a la ayuda de mi tutor las he podido sacar adelante.

Nunca había trabajado con Android Studio y el hecho de utilizar el lenguaje Kotlin, aunque este es el recomendado por los desarrolladores de Android para crear nuevas aplicaciones tiene la desventaja de que la información disponible en internet para problemas concretos es mucho menor de la que se puede encontrar en otros lenguajes que tienen mucho más recorrido.

En resumen, quiero exponer que gracias a este trabajo he aprendido que, aunque es una tecnología desconocida y nueva para mí con ilusión esfuerzo y trabajo se puede aprender y conseguir cualquier cosa, también he aprendido que las cosas que realmente merecen la pena requieren su tiempo y un saber lidiar con la frustración enseñanzas que hoy en día están en abandono debido a la inmediatez y la ley del mínimo esfuerzo.

Referencias

[1] Introducción a Android Studio
https://developer.android.com/studio/intro?hl=es-419
[2] ¿QUÉ ES FLUTTER ?
https://www.startechup.com/es/blog/what-is-flutter/
[3] Ventajas y desventajas de Xcode
https://keepcoding.io/blog/ventajas-y-desventajas-de-xcode/
[4] Kotlin vs Java
https://openwebinars.net/blog/kotlin-vs-java/
https://keepcoding.io/blog/ventajas-y-desventajas-de-xcode/
[5] Biblioteca de persitencia Room
https://developer.android.com/topic/libraries/architecture/room?hl=es-419
[6] SQLite: La Base de Datos Embedida
https://sg.com.mx/revista/17/sqlite-la-base-datos-embebida
[7] Qué es MySQL: Caracteristicas y ventajas
https://openwebinars.net/blog/que-es-mysql/
[8] Objetivo Analista
https://objetivoanalista.com/conoce-bcoach-la-pizarra-tactica-que-destaca-entre-analistas-profesionales/
[9] Android Developers, Diseños:
https://developer.android.com/guide/topics/ui/declaring-layout?hl=es-419
[10] Android Developers ,Como crear una UI responsiva con ConstraintLayout:
https://developer.android.com/training/constraint-layout?hl=es-419
[11] Android Developers, Diseño Lineal:
https://developer.android.com/guide/topics/ui/layout/linear?hl=es-419
[12] Android Developers, Crear lista dinámicas con RecyclerView:
https://developer.android.com/guide/topics/ui/layout/recyclerview?hl=es-419
[13] Android desde Cero:entender Adaptadores y Adaptador Vista
https://code.tutsplus.com/es/android-desde-cero-entender-adaptadores-y-adaptador-vista--cms-26646t
[14] Android Develepers, Cómo compilar un intent
https://developer.android.com/training/basics/firstapp/starting-activity?hl=es-419
[15] Como utilizar MPAndroidChart en AndroidStudio
https://medium.com/@codingInformer/how-to-use-mpandroidchart-in-android-studio-c01a8150720f