Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Diseño e implementación de un visualizador para Android de la información de los datos de competiciones y carreras de SLOT

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Ventura Vallés, Saúl

Tutor: Carsí Cubel, José Ángel

2018-2019

Resumen

El presente trabajo consiste en el desarrollo de una aplicación móvil desarrollada para dispositivos con sistema operativo Android. Se trata de una aplicación para mostrar los resultados de las competiciones de SLOT, dicha información se encuentra almacenada en ficheros XML y bases de datos Access generadas por el PCLapCounter y ubicadas en un servidor.

Palabras clave: Aplicación móvil, Android, SLOT, XML, Access, PCLapCounter.

Abstract

The current Project consists on the development of a mobile application for Android devices. This is an application to show results of SLOT competitions, this data are storage in XML files and Access data base generated by PCLapCounter and located on a server.

Keywords: mobile app, Android, SLOT, XML, Access, PCLapCounter.

Tabla de contenidos

Índice de figuras

1. Introducción

Para la realización del proyecto, se decidió realizar un visualizador de la información de los datos de las competiciones y carreras de SLOT, atraído al ver el proyecto y a la práctica de este hobby.

1.1. Introducción a las competiciones de SLOT

Para entrar en situación se va a explicar el escenario de la puesta en marcha de dicho proyecto, las carreras de SLOT son carreras de lo que se conoce normalmente como Scalextric aquellos juguetes que todos hemos tenido o deseado de pequeños que constan de un circuito y dos coches con un motor eléctrico que se introducen en los carriles que posee el circuito y mediante un mando puedes controlar la corriente que recibe el coche haciendo que este corra más o menos, proporcionando este juguete a quien lo haya poseído horas de entretenimiento y grandes duelos con los amigos y familiares.

Con el paso del tiempo y conforme hemos crecido estas carreras caseras también han ido creciendo con nosotros llegando a construirse circuitos inmensos y existen locales habilitados con circuitos permanentes donde se puede ir a competir con más gente que comparte este hobby.

Para el control de las carreras ya no basta con llevar la cuenta de cabeza como en casa y se utilizan sistemas de cronometraje informatizados donde se puede gestionar y configurar la carrera programando la duración de la misma y consultar datos como el tiempo por vuelta, el número de vueltas que se lleva, en nuestro caso el programa de gestión de dichas carreras el PCLapCounter[1] con el cual se gestiona y obtienen todos los datos de la carrera y se almacenan en un ordenador.

Estos gestores de carreras son los encargados de monitorizar las carreras y mostrar toda la información por pantalla permitiendo configurar la carrera, y guardar toda la información para posteriormente ver los resultados y generar clasificaciones. En este caso el PCLapCounter presenta la siguiente apariencia:

[1] PCLapCounter (http://www.pclapcounter.be)

Ilustración 1. PcLapCounter

La anterior imagen es de una carrera en curso donde se ve toda la información de la carrera, mostrando los siguientes datos: posición, coche, piloto, vuelta, tiempo de última vuelta, distancia con el primero, mejor vuelta, gasolina, indicador de gasolina, entre otros datos.

Los datos de las carreras se almacenan en bases de datos Access para posteriormente generara gráficas y mostrar la información de la carrera.

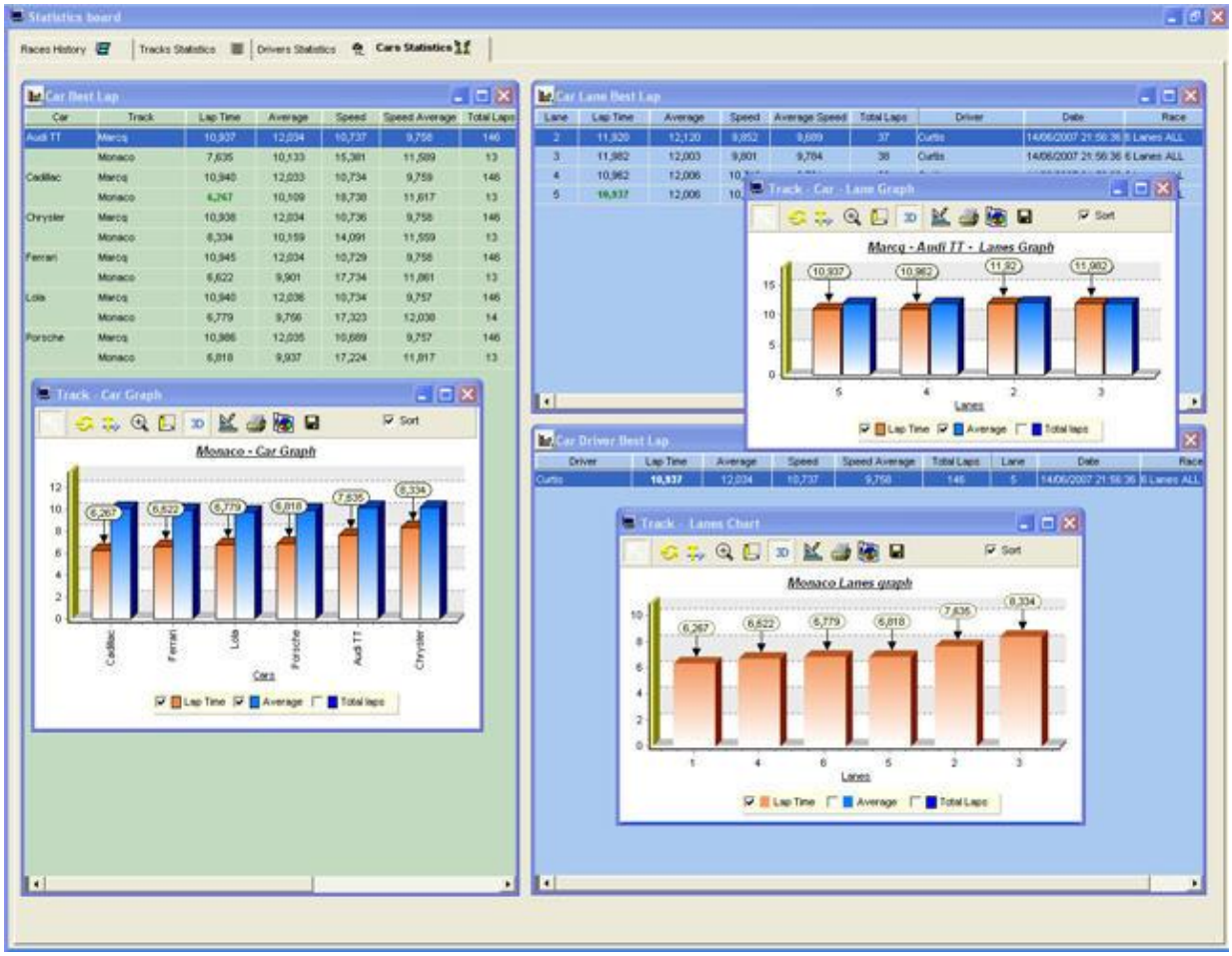

Ilustración 2. Gráficos PCLapCounter

Para la organización de las carreras se utiliza otro programa en este caso es el CarreraSlot[2] con el que se puede organizar la estructura de las carreras, permitiendo configurar competiciones formadas por diferentes carreras y estas carreras a su vez divididas en diferentes fases.

Una vez organizadas las competiciones se puede enlazar los datos de las competiciones de CarreraSlot con las carreras disputadas y monitorizadas con PCLapCounter de esta forma se tienen toda la información organizada para la generación de estadísticas de una determinada competición o carrera.

[2] CarreraSlot(http://www.slotdigital.com/forums/topic/16259-carreraslot/), Pep0n- 2010

A continuación, se muestra la apariencia del programa CarreraSlot.

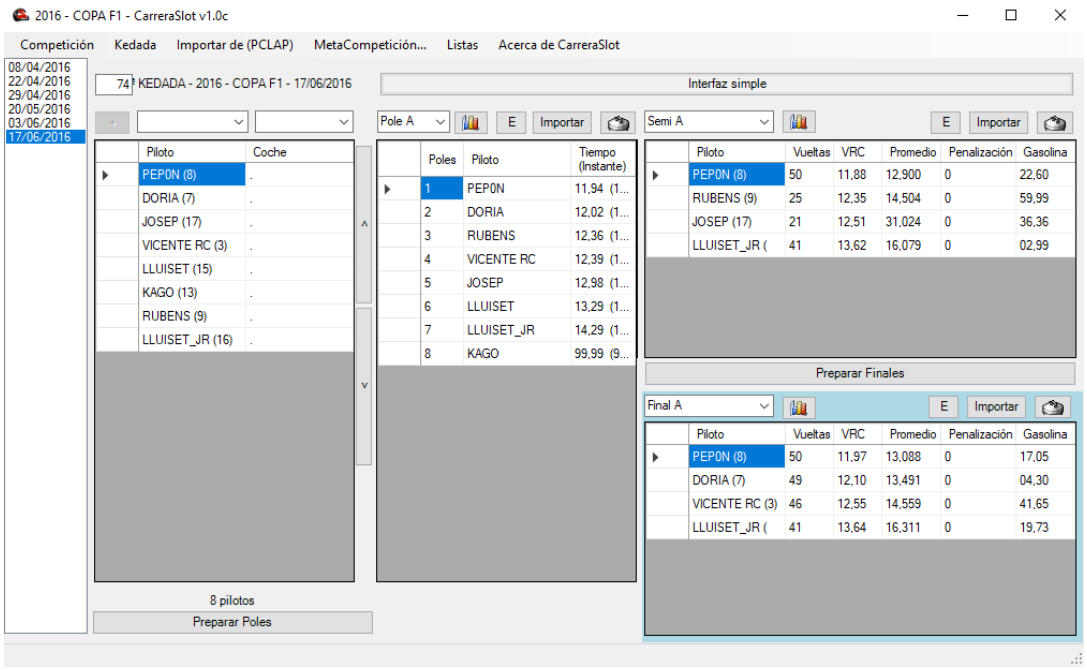

Ilustración 3. CarreraSlot

Aquí se puede ver una competición en este caso COPA F1, donde se encuentran las carreras que la forman a la izquierda nombradas con la fecha en que se realizaron, los pilotos que participaron, el resultado de la pole, el resultado de las semifinales y las finales, el programa permite ir configurando estas semifinales y finales conforme se introducen los datos procedentes de PCLapCounter.

Una vez enlazada las carreras con la base de datos procedente de PCLapCounter el programa CarreraSlot permite generar diferentes gráficas y estadísticas sobre la pole, semifinal o final seleccionada mostrando la siguiente ventana:

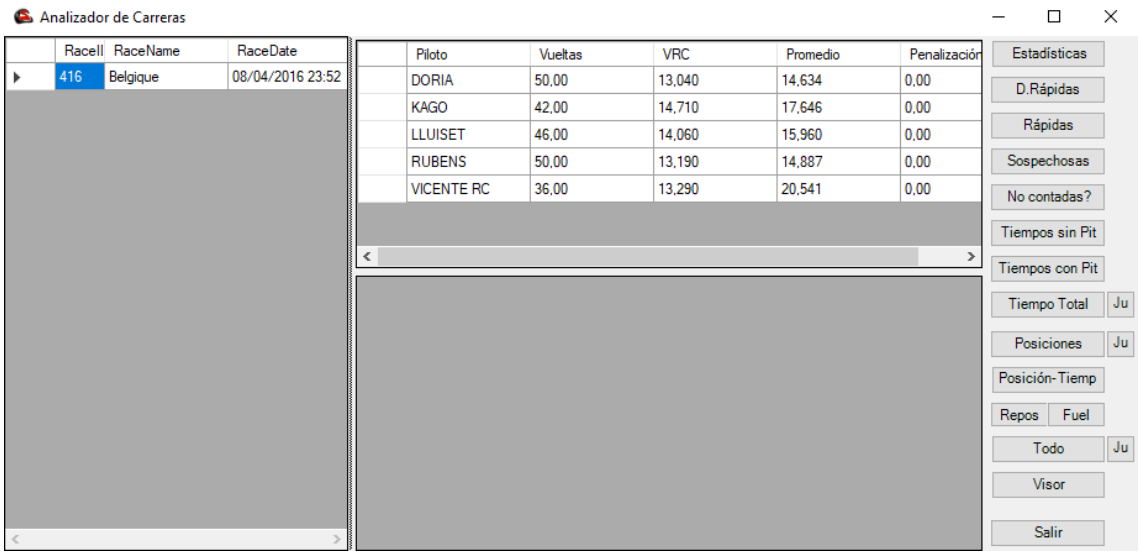

Ilustración 4. Analizador de carreras

1.2. Objetivos del proyecto

El proyecto realizado parte de la base del programa CarreraSlot, en el cual se muestran y se procesar los datos de las competiciones y carreras realizadas, permitiendo importar los datos del PCLapCounter, y de esta forma mostrar los resultados.

Con el uso intensivo de los Smartphone se plantea la posibilidad de poder tener acceso a los datos de las carreras y competiciones por lo que surge la idea de realizar una aplicación móvil que lea los datos de estas carreras y muestre las estadísticas en el móvil permitiéndonos ver los resultados con mayor tranquilidad una vez terminada la carrera, para ello los resultados de las carreras se encuentran en un servidor en el que se publican los resultados de las competiciones realizadas en su club.

1.3. Estructura de la memoria

La memoria está estructurada en diferentes secciones o apartados, conforme se vaya avanzando en la lectura de estas secciones se profundizara y se exponen los procesos seguidos para el desarrollo de la aplicación, principalmente la memoria se estructura en 6 apartados.

En el siguiente capítulo se a habla de los requisitos de la aplicación, mostrando una descripción del producto y comentando los requisitos tecnológicos que plantea nuestra solución, así como un análisis de la investigación previa realizada para desarrollar y comprender el funcionamiento y estructura utilizada en los datos a mostrar.

En el capítulo 3 contiene el análisis del proyecto donde se encuentran los requisitos se realizará un análisis de casos de uso para conocer los requisitos de nuestra aplicación.

El capítulo 4 se centra en los requisitos de diseño que se desean obtener en nuestra aplicación, así como la estructura utilizada y una explicación de grafica del funcionamiento de nuestra aplicación.

En el capítulo 5 hace referencia a como se ha implementado nuestra solución, así como los detalles y tecnologías utilizadas para desarrollar nuestro proyecto.

El capítulo 6 se exponen las conclusiones, así como una lista de posibles trabajos futuros para realizar.

Además, cerrando el proyecto se presentarán una serie de apéndices para mejorar la comprensión del proyecto, así como un breve manual para la configuración de la aplicación y su funcionamiento.

2. Requisitos

En el capítulo actual se realiza una descripción de los requisitos de nuestra aplicación, este análisis de requisitos se realizará mediante el estándar IEEE 830-1998.

2.1. Introducción

En este apartado se van a examinar todos los requisitos que necesitamos y que la aplicación tiene que cumplir para obtener el resultado deseado para ello, para ello van a ver las diferentes situaciones que aparecen y todas las necesidades que debe de cubrir nuestra aplicación, para comenzar se define el propósito de nuestra aplicación, posteriormente, una vez establecido el propósito, se analiza el ámbito del sistema conociendo el entorno que rodea la aplicación y estableciendo cada una de las funcionalidades.

2.1.1. Propósito

El propósito de la aplicación es informar de los resultados de las competiciones de SLOT realizadas en el club, Además de mostrar de forma detallada estadísticas de las carreras realizadas como pueden ser vueltas rápidas, vueltas sospechosas, información de los repostajes y la variación de posiciones en el transcurso de la carrera.

2.1.2. Ámbito del sistema

La aplicación desarrollada recibe el nombre de CarreraSlot, el nombre procede de la aplicación de escritorio de la cual se leen los datos de las competiciones y kedadas.

Esta aplicación permite añadir URL's que contienen competiciones o metacompeticiones para posteriormente poder visionar los datos de estas, los datos que podemos observar son los siguientes:

Meta competición:

- Competiciones que la forman
- Pódium de cada Competición disputada

Competición:

- Clasificación general
- Recuento de poles
- Recuento de victorias
- Recuento de vueltas rápidas de carrera
- Vueltas rápidas en clasificación
- Vueltas rápidas en carrera

Kedada:

- Resultado de la kedada
- Resultado de la pole
- Resultados de las semifinales disputadas
- Resultado de las Finales disputadas

Pole:

- Vueltas Rápidas
- Vueltas Sospechosas

Semifinal Y finales:

- Vueltas rápidas
- Vueltas sospechosas
- Posiciones
- Repostajes
- Visor

2.1.3. Definiciones, Acrónimos y Abreviaturas

Termino	Descripción
SLOT	Juego formado por coches propulsados por un motor eléctrico y un circuito de plástico con carriles para introducir los coches y mediante un regulador de voltaje independiente por carril hacerlos funcionar
Carril	Hendidura en las pistas de plástico que forman el circuito que posee un conductor eléctrico a cada lado
Pista	Sección de plástico que posee los carriles y conexiones en cada extremo para conectar con otras pistas y formar circuitos
Circuito	Agrupación de pistas conectadas formando un circuito cerrado por el que circulan los coches de SLOT
Meta competición	Agregado de competiciones
Competición	Conjunto de kedadas disputadas
Kedada	Día que se disputa una carrera
Carrera	Una pole, semifinales o finales

2.1.4. Referencias

- 830-1998 - IEEE Recommended Practice for Software Requirements Specifications
- Especificación de Requisitos según el estándar de IEEE 830 (https://www.fdi.ucm.es/profesor/gmendez/docs/is0809/ieee830.pdf)

2.2. Descripción General

La aplicación está basada en los datos proporcionados por otras dos aplicaciones que son PCLapCounter y CarreraSlot, por lo que los datos a procesar ya tienen un formato y un tipo establecidos, Este proyecto está ambientado en un entorno de competición por lo que la información a mostrar son resultados y estadísticas de las carreras disputadas.

2.2.1. Perspectiva del Producto

La aplicación muestra los datos almacenados en un servidor web, estos datos se encuentran en ficheros con una estructura XML y en una base de datos Access procedentes de las aplicaciones nombradas anteriormente, por lo que tenemos que la aplicación tiene que ser capaz de acceder y leer dichos datos.

2.2.2. Funciones del Producto

La aplicación permitirá agregar competiciones y navegar a través de estas siendo capaz de mostrar información de la competición como son los resultados de los pilotos, las victorias, las poles, los diferentes tipos de vueltas rápidas y las kedadas que la forman.

Además, también permite ver de manera gráfica los resultados de cada una de las kedadas obteniendo gráficas con las vueltas rápidas de la carrera, la variación de las posiciones, los repostajes e incluso un visor donde poder reproducir la carrera.

2.2.3. Características de los Usuarios

Los usuarios de la aplicación son los integrantes de un club de slot por lo que ya están familiarizados con las estructuras de las competiciones por lo que la aplicación se centra en mostrar estadísticas y datos.

2.2.4. Restricciones

La aplicación requiere de un sistema operativo Android para poder ser instalada y ejecutada, así como mínimo una API 19 Android 4.4 (kitkat) y al tratarse de una aplicación que toma los datos de un servidor externo también se precisa de una red de datos en el terminal donde se ejecuta la aplicación

2.2.5. Suposiciones y Dependencias

Esta aplicación depende de un servidor externo donde se publica los resultados de las competiciones y kedadas del club de SLOT.

2.3. Requisitos Específicos

2.3.1. Requerimientos funcionales

Introducción y modificación URL:

El usuario tendrá que introducir una URL que contenga una competición válida para comenzar a utilizar la aplicación.

Consultar competiciones o metacompeticiones:

Mediante las URL proporcionadas por el usuario se cargan las competiciones y el usuario puede proceder a navegar por la aplicación mostrando los datos seleccionados.

2.3.2. Interfaces Externas

La interfaz de usuario viene determinada por el terminal móvil donde se instale la aplicación en la mayoría de los casos suele ser una pantalla comprendida entre 4.4” a 5.5” permitiendo la lectura de los datos de una forma ágil y cómoda, la interacción con la aplicación se realizará a través de la pantalla táctil del terminal.

2.3.3. Funciones

A continuación, se explica el comportamiento del sistema ante la interacción del usuario y como se desarrolla cada función para ello se han clasificado las acciones según su objetivo.

Inicio de la aplicación

Al iniciar la aplicación esta prepara las bases de datos para el correcto funcionamiento del programa cargando las bases de datos de años pasados y comprobando el estado de la base de datos del año en curso y descargando una nueva versión si es necesario.

Introducción URL

Los usuarios deberán introducir una URL que contenga una competición o una metacompetición valida, la aplicación explorará dicha URL obteniendo la competición o metacompetición y la introducirá en una lista donde el usuario.

Selección de una metacompetición

El usuario accede a una metacompetición establecida en el programa para ver la clasificación de las competiciones que integran la meta competición pudiendo observar el nombre de la competición, el pódium de cada competición con los puntos obtenidos y el número de kedadas que forman la competición.
En el caso de que la meta competición este formada por competiciones y metacompeticiones el usuario será dirigido a una nueva vista donde tendrá una lista con el nombre de las competiciones y metacompeticiones que forman la meta competición que desea observar.

Selección de una Competición

Al acceder a una competición establecida en el programa para ver la clasificación de las kedadas que forman dicha competición, obteniendo una clasificación por kedada con los puntos obtenidos por los pilotos y la diferencia de puntos al primer clasificado, así como resaltado en colores el poleman de la kedada, la vuelta rápida.

Selección de una kedada

El usuario accede a una kedada para observar la clasificación los resultados de la kedada en detalle pudiendo observar resultados de pole, semifinales, finales y general de la kedada.

Selección de una pole

El usuario al acceder a la pole position puede observar el número máximo de vueltas y el tiempo con el que se ha obtenido la pole además de otras opciones para visualizar las vueltas rápidas, vueltas sospechosas, y un visor para reproducir la pole position.

Selección de una semifinal

El usuario al acceder a la semifinal seleccionada y puede observar el número máximo de vueltas y la vuelta rápida de la semifinal además de otras opciones para visualizar las vueltas rápidas, vueltas sospechosas, posiciones, repostajes y el visor.

Selección de una final

El usuario al acceder a la final seleccionada y puede observar el número máximo de vueltas y la vuelta rápida de la final además de otras opciones para visualizar las vueltas rápidas, vueltas sospechosas, posiciones, repostajes y el visor.

Las funciones descritas a continuación tienen lugar dentro de una pole, semifinal o final:

Gráfico de vueltas rápidas

Al seleccionar el usuario la opción de vueltas rápidas obtiene una gráfica donde se muestra en el eje X la vuelta y en el eje Y el tiempo realizado, en dicha grafica los puntos están representados por formas y colores para y contiene una leyenda para ser correctamente interpretada

Gráfico de Vueltas sospechosas

La opción de vueltas sospechosas también se muestra una gráfica con las características comentadas en el punto anterior pero los datos mostrados los forman las vueltas que han sido anormalmente lentas comparadas con el resto de vueltas.

Gráfico de Repostajes

En esta sección se muestra al usuario una gráfica un tanto diferente en ella se muestran líneas verticales donde el eje Y muestra la cantidad de combustible repostado por el piloto en su parada en boxes, y el eje X nos indica la vuelta en la que se ha entrado al box, esta gráfica también contiene una leyenda con el código de colores para ser correctamente interpretada.

Gráfico de Posiciones

En el apartado de posiciones se muestra un gráfico formado por líneas donde el eje Y nos indica la posición del piloto y en el Eje X se muestran las vueltas de la carrera de este modo obtenemos un gráfico con la variación de las posiciones en el transcurso de semifinal o final.

Visor carrera

En esta última opción se genera un visor donde se puede ver el transcurso de una carrera seleccionada permitiendo al usuario controlar la reproducción, insertado valores como el refresco (velocidad en ms que se refresca el visor), el multiplicador (controla la velocidad de reproducción de la carrera) y por último el pausado o la reproducción del visor.

2.3.4. Requisitos de Rendimiento

La aplicación debe funcionar de una manera fluida sin excesivos periodos de carga permitiendo un uso de la aplicación agradable en el caso de la carga de las bases de datos dependerá del tamaño de dicha base de datos y la velocidad de la conexión de red.

2.3.5. Restricciones de Diseño

Para el diseño de la aplicación se parte de la base del programa CarreraSlot por lo que esta condicionada a los formatos y estructuras ya utilizadas por dicha aplicación siendo está archivos XML con los datos de la metacompetición y competiciones, y las bases de datos procedentes de PCLapCounter.

3. Análisis

En Este apartado se va a proceder a mostrar el diagrama de clases perteneciente a la aplicación, así como a realizar un análisis por casos de uso, mostrando detalladamente el comportamiento a cada una de las acciones que puede realizar el usuario y explicar cómo está estructurada la aplicación.

3.1. Diagrama de clases

Partiendo de toda la infraestructura que se ha analizado para entender el comportamiento de las otras aplicaciones en las que se basa la aplicación, se ha desarrollado el siguiente diagrama de clases:

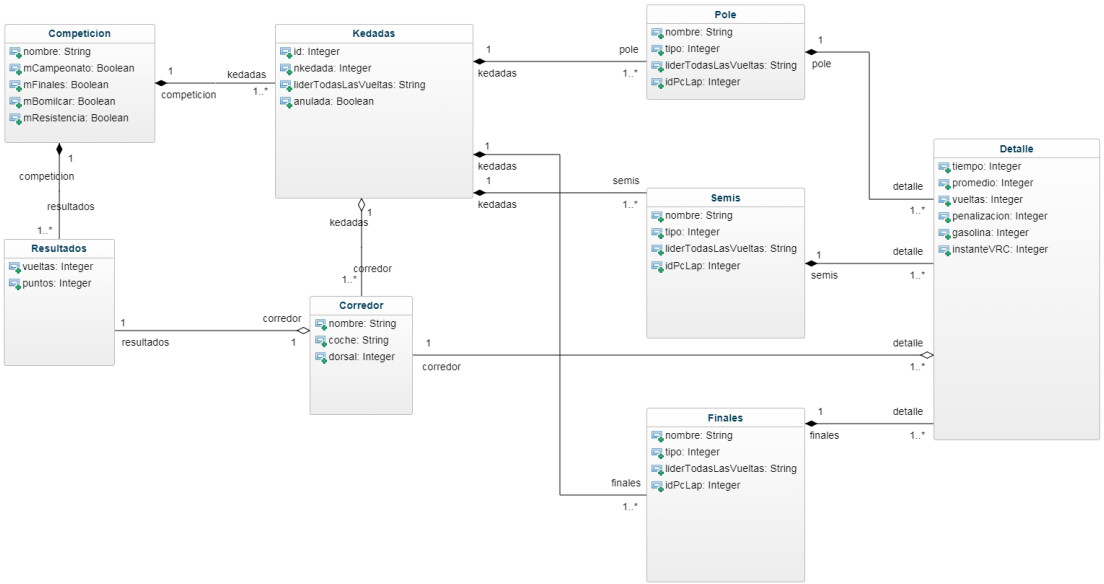

Ilustración 5. Diagrama de clases

Con el anterior diagrama de clases se pretende satisfacer todas las necesidades estructurales para procesar la información, permitiendo mostrar y almacenar toda la información procedente de los archivos XML y las bases de datos, permitiendo realizar las consultas y cálculos necesarios para generar datos que no se encuentran definidos y se obtienen mediante el procesamiento de la información obtenida, también están las relaciones que existen entre las diferentes clases permitiéndonos hacernos una idea de la estructura que tiene la aplicación y la información.

3.2. Casos de uso

Para desarrollar la funcionalidad de la aplicación se realiza un análisis por casos de uso para entender las diferentes acciones que puede realizar el usuario en el sistema, al tratarse de una aplicación de consulta de datos el diagrama de casos de uso también resulta sencillo.

Caso de uso 1	Establecer enlace de competición
Actor	Usuario de la aplicación
Descripción	El usuario establece un enlace a una competición de slot para ser mostrada en la aplicación
Flujos alternativos	
Comentario	

Caso de uso 2	Lectura de una metacompetición
Actor	Usuario de la aplicación
Descripción	El usuario accede a una meta competición establecida en el programa para ver la clasificación de las competiciones que integran la meta competición pudiendo observar el nombre de la competición, el pódium de cada competición con los puntos obtenidos y el número de kedadas que forman la competición
Flujos alternativos	En el caso de que la metacompetición este formada por competiciones y metacompeticiones el usuario será dirigido a una nueva vista donde tendrá una lista con el nombre de las competiciones y metacompeticiones que forman la meta competición que desea observar.
Comentario	

Caso de uso 3	Lectura de una competición
Actor	Usuario de la aplicación
Descripción	El usuario accede a una competición establecida en el programa para ver la clasificación de las kedadas que forman dicha competición, obteniendo una clasificación por kedada con los puntos obtenidos por los pilotos y la diferencia de puntos al primer clasificado, así como resaltado en colores el pole-man de la kedada, la vuelta rápida.
Flujos alternativos	
Comentario	

Caso de uso 4	Lectura de una Kedada
Actor	Usuario de la aplicación
Descripción	El usuario accede a una kedada para observar la clasificación los resultados de la kedada en detalle pudiendo observar resultados de pole, semifinales, finales y general de la kedada.
Flujos alternativos	
Comentario	

Caso de uso 5	Lectura de una pole
Actor	Usuario de la aplicación
Descripción	El usuario al acceder a la pole position puede observar el número máximo de vueltas y el tiempo con el que se ha obtenido la pole además de otras opciones para visualizar las vueltas rápidas, vueltas sospechosas, y un visor para reproducir la pole position.
Flujos alternativos	
Comentario	

Caso de uso 6	Lectura de una semifinal
Actor	Usuario de la aplicación
Descripción	El usuario al acceder a la semifinal seleccionada y puede observar el número máximo de vueltas y la vuelta rápida de la semifinal además de otras opciones para visualizar las vueltas rápidas, vueltas sospechosas, posiciones, repostajes y el visor
Flujos alternativos	
Comentario	

Caso de uso 7	Lectura de una final
Actor	Usuario de la aplicación
Descripción	El usuario al acceder a la final seleccionada y puede observar el número máximo de vueltas y la vuelta rápida de la final además de otras opciones para visualizar las vueltas rápidas, vueltas sospechosas, posiciones, repostajes y el visor
Flujos alternativos	
Comentario	

Caso de uso 8	Lectura de vueltas rápidas
Actor	Usuario de la aplicación
Descripción	El usuario obtiene un gráfico con las vueltas rápidas
Flujos alternativos	
Comentario	Para todos, algunos o sólo un piloto busca y gráfica las vueltas rápidas. Estás las calcula el PCLAP y las marca como BestLap=1

Caso de uso 9	Lectura de vueltas sospechosas
Actor	Usuario de la aplicación
Descripción	El usuario obtiene un gráfico con las vueltas sospechosas
Flujos alternativos	
Comentario	Para todos, algunos o sólo un piloto busca y grafica las vueltas sospechosas. Estás las calcula el PCLAP y las marca como BestLap=3.

Caso de uso 10	Lectura de posiciones
Actor	Usuario de la aplicación
Descripción	El usuario obtiene un gráfico que muestra las veces que entró en el pitlane, mostrando la gasolina que tenía en el momento de entrar y la que tenía cuando salió.
Flujos alternativos	
Comentario	

Caso de uso 11	Lectura de repostajes
Actor	Usuario de la aplicación
Descripción	El usuario obtiene un gráfico con los repostajes
Flujos alternativos	
Comentario	

Caso de uso 10	visor
Actor	Usuario de la aplicación
Descripción	El usuario reproduce la semifinal final o pole seleccionada para ver con detalle lo sucedido.
Flujos alternativos	
Comentario	

4. Diseño

Una vez establecido el comportamiento de la aplicación en la fase de diseño se procede a mostrar la arquitectura de la aplicación, a establecer la estructura interna, explicando en detalle las capas que componen el proyecto, así como a definir las transiciones entre diferentes partes de la aplicación.

4.1. Arquitectura.

La aplicación se encuentra distribuida en distintas secciones bien diferenciadas, la aplicación que consulta la información almacenada en un servidor donde se encuentran almacenados todos los archivos XML y bases de datos con la información de las competiciones.

Y un servidor donde se encuentra la estructura de archivos (apéndice 8.1), donde están organizados en directorios todos los documentos XML de las competiciones permitiendo la navegación a través de las competiciones, también contiene un directorio con las bases de datos.

4.2. Estructura.

La aplicación está desarrollada con una estructura de tres capas presentación, lógica, persistencia donde cada capa es totalmente independiente de las demás.

- Capa de presentación: Es la primera capa de la aplicación con la que el usuario interacciona, en ella se obtiene la información procedente del usuario y muestra la interfaz diseñada, además mantiene la comunicación con la capa de lógica.
- Capa de lógica: En esta capa se encuentran todos los procesos a realizar con la información proveniente de la capa de presentación y es la encargada de enviar la información a correcta a la capa de presentación. Esta capa está situada entre la presentación y la persistencia manteniendo una comunicación constante entre ambas capas ya que como hemos dicho se encarga de proporcionar la información al usuario a través de la capa de presentación y a la vez es la encargada de solicitar esta información a la capa de persistencia.
- Capa de persistencia: Esta capa es la última capa y es la encargada de almacenar toda la información, esta capa accede, actualiza, borra y almacena los datos. A su vez está en comunicación con la capa de negocio para proporcionarle toda la información solicitada.

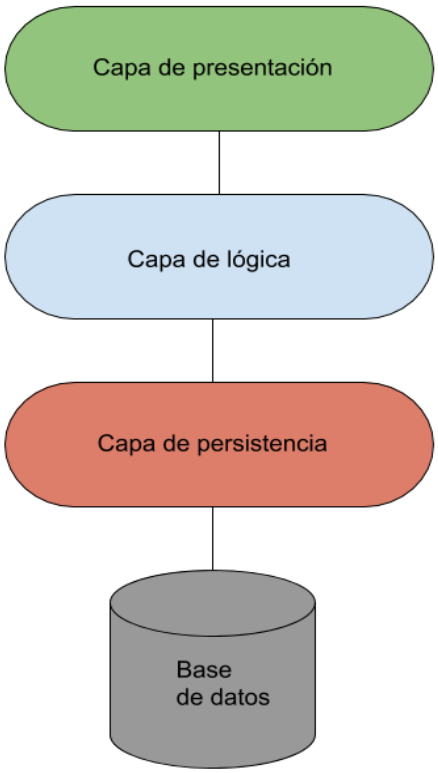

Ilustración 6. Estructura aplicación

La principal ventaja de estructurar el proyecto de esta forma es que en caso de tener que realizar alguna modificación solo con modificar la capa afectada es suficiente, además de al estar dividido el código en capas esta todo más organizando mejorando la comprensión del mismo y permitiendo el desarrollo por diferentes partes.

4.2.1. Capa de presentación

En esta capa se define la apariencia de la aplicación, es la parte visible para el usuario, en esta capa el usuario interactúa con la aplicación introduciendo datos, realizando acciones y observando los datos solicitados.

Para comprender como está formada esta capa se presenta el siguiente diagrama que muestra el nombre de las clases y las interacciones entre las clases que forman la presentación, en la siguiente ilustración.

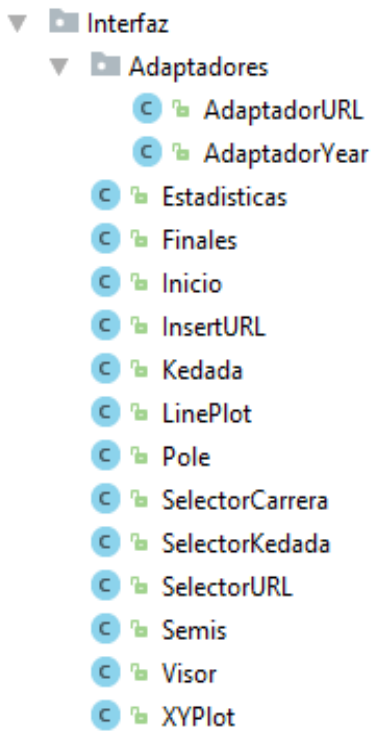

Ilustración 7. Capa de presentación

Una vez vistas las clases que componen esta capa se van a mostrar de manera gráfica las diferentes interacciones con la aplicación y poder observar su funcionalidad.

Inicio de la aplicación

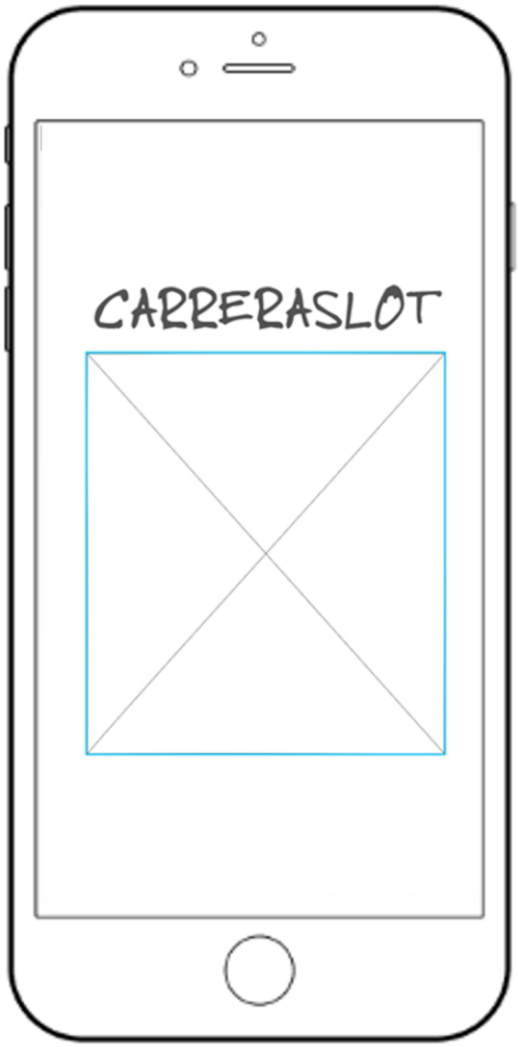

Ilustración 8. Pantalla inicio

Al iniciar la aplicación se puede observar el logo de CarreraSlot donde se produce un proceso de carga para posteriormente mostrar las competiciones o metacompeticiones introducidas anteriormente o solicitar la introducción de una o varias URL para la carga de una o varias competiciones o mentacompeticiones.

Añadir una URL

Si es la primera vez que se accede a la aplicación se solicitara que se inserte una URL para mostrar los datos de las competiciones o metacompeticiones que se encuentran en dicha dirección.

Ilustración 9. Añadir URL

Como se puede observar en la imagen se dispone de un campo para escribir la dirección de la competición a añadir junto con un botón para añadir la dirección introducida en el campo, También se observa una lista de direcciones donde aparecen las nuevas direcciones añadidas y las direcciones ya añadidas anteriormente estando está vacía si es la primera vez que se inicia la aplicación.

Lista de competiciones

Una vez añadidas las diferentes direcciones introducidas se puede observar la siguiente escena.

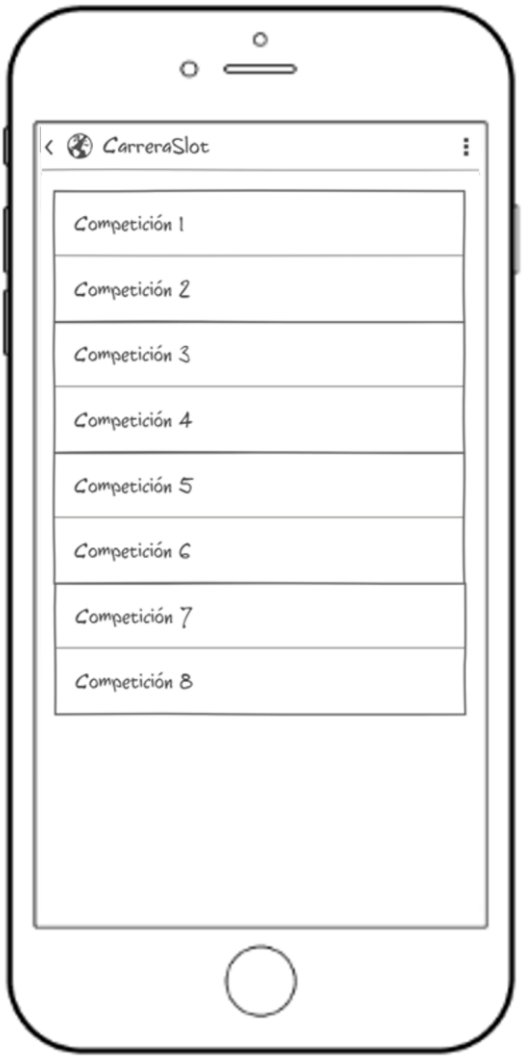

Ilustración 10. Menú principal

Aquí se han procesado y se puede observar una lista con las diferentes competiciones o metacompeticiones que hemos introducido pudiendo seleccionar cualquier elemento de esta lista y acceder a él para ver más detalles sobre la competición.

Metacompetición

Si la opción seleccionada es una meta competición se puede observar los datos que la forman presentando la siguiente estructura.

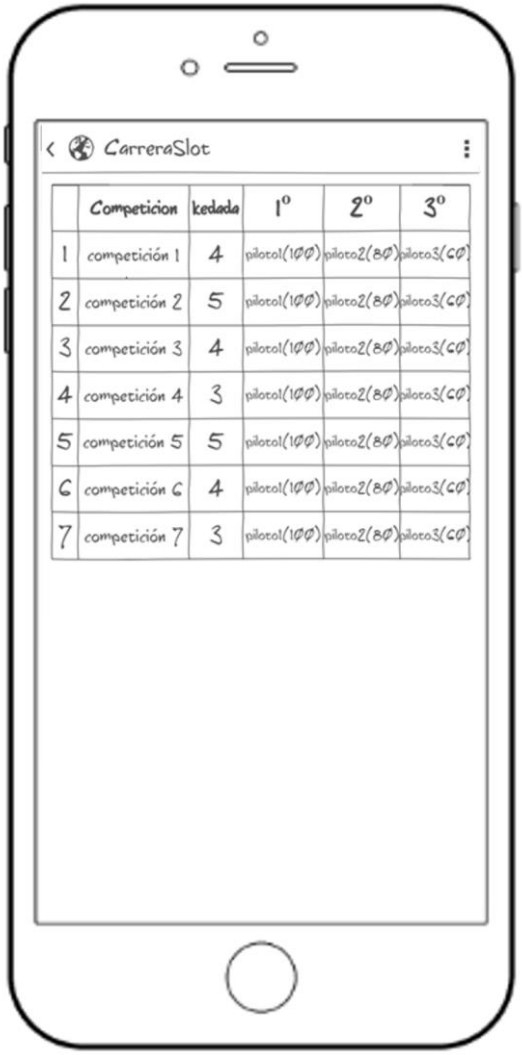

Ilustración 11. MetaCompetición

El usuario puede observar las competiciones que forman la metacompetición así como el número de kedadas que forman cada competición y el pódium de cada competición, además puede seleccionar cualquier competición para ver más detalles sobre esta.

Competiciones

En las siguientes imágenes se puede ver como aparecen los datos de una competición, el usuario puede llegar a esta vista si ha seleccionado una competición en la pantalla de una metacompetición o en la lista de direcciones el elemento seleccionado es una competición.

Ilustración 12. Competición 1

En primer lugar, se puede observar una tabla con la general de la competición, así como el reparto de puntos, penalizaciones, pole position, y vueltas rápidas por cada uno de los pilotos participantes en alguna kedada de la competición.

Continuando desplazándonos por la vista de competición podemos observar más datos como son el número de victorias de los pilotos que han conseguido al menos una, el piloto que ha realizado la vuelta rápida de carrera en alguna kedada de la competición y una tabla con una relación de pilotos con la vuelta rápida en la pole que ha marcado cada uno, así como el coche que utilizaron para realizar la vuelta rápida.

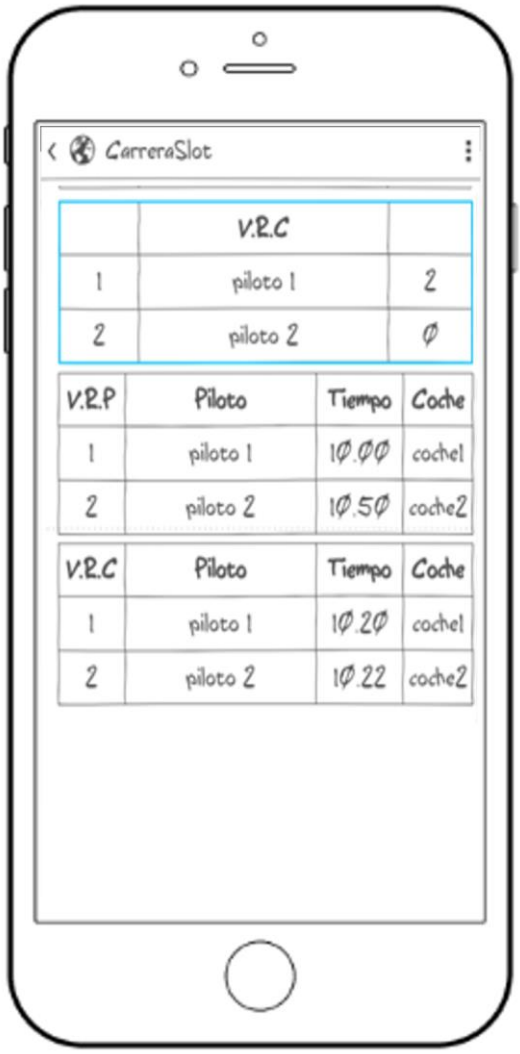

Ilustración 13. competición 2

Para finalizar el usuario puede observar una tabla donde se muestran los pilotos y su vuelta rápida en carrera, realizada en alguna de las kedadas que forman la competición.

El usuario puede seleccionar en esta vista cualquier kedada para seguir observando más datos sobre la misma.

Kedada

En las siguientes imágenes se puede observar cómo se presenta la información de una kedada, Aquí el usuario puede observar diferentes datos que procedemos a explicar a continuación.

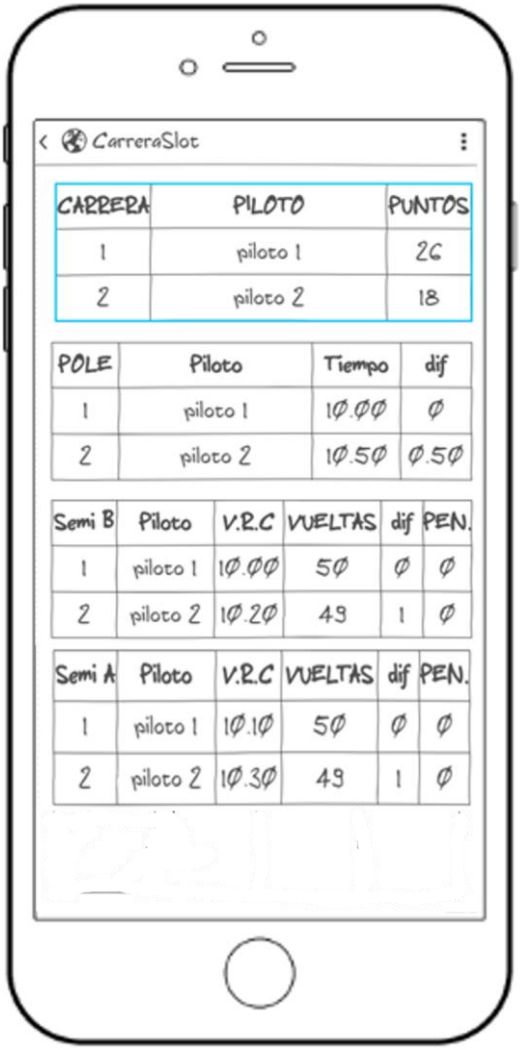

Ilustración 14. Kedada 1

Al inicio de la vista el usuario puede observar el resultado de la quedada con el reparto de puntos según la posición en la que ha terminado, si ha realizado la pole position o es poseedor de la vuelta rápida de carrera, más abajo se puede ver el resultado de la pole position con el tiempo realizado en ella por cada piloto y la diferencia de cada piloto con el tiempo de la pole.

Si se continua observando se ve una serie de tablas que pertenecen a cada una de las semifinales y finales disputadas en la kedada mostrando el resultado de la final o semifinal la vuelta rápida que ha realizado cada piloto, el número de vueltas, la diferencia expresada en vueltas con el vencedor y las penalizaciones si algún piloto ha sido sancionado.

En esta vista el usuario puede acceder a los datos más detallados de la pole, semifinales y finales que forman la kedada.

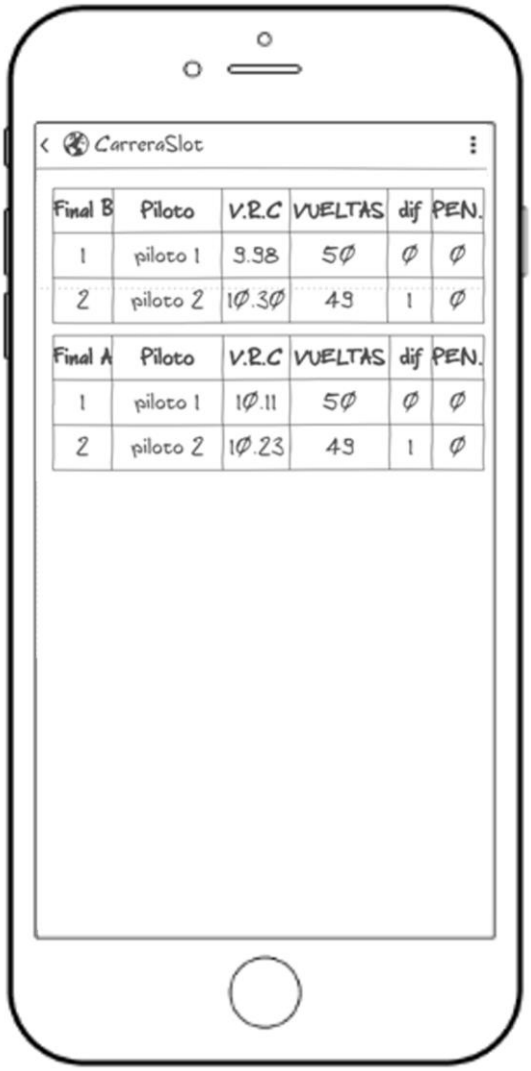

Ilustración 15. Kedada 2

Pole position

Al acceder a los detalles de la pole el usuario puede ver un menú como el que se muestra permitiéndole acceder a las vueltas rápidas y a las vueltas sospechosas mostrando gráficos con los datos. Estas vistas serán explicadas a continuación en la sección de semifinales y finales ya que son comunes a todas.

Ilustración 16. Pole

Semifinales y finales

El usuario puede observar en detalle los datos de las semifinales y finales disputadas accediendo desde la vista de la kedada, al acceder a una semifinal o final observa un menú con diferentes acciones que se pueden realizar.

Ilustración 17. Menú semifinal y final

Vueltas rápidas, Vueltas sospechosas, Posiciones y Repostajes

En cada una de estas acciones el usuario podrá observar diferentes gráficas, con la información de cada piloto participante, en esta vista además encontrará una leyenda informativa para poder interpretar la información correctamente.

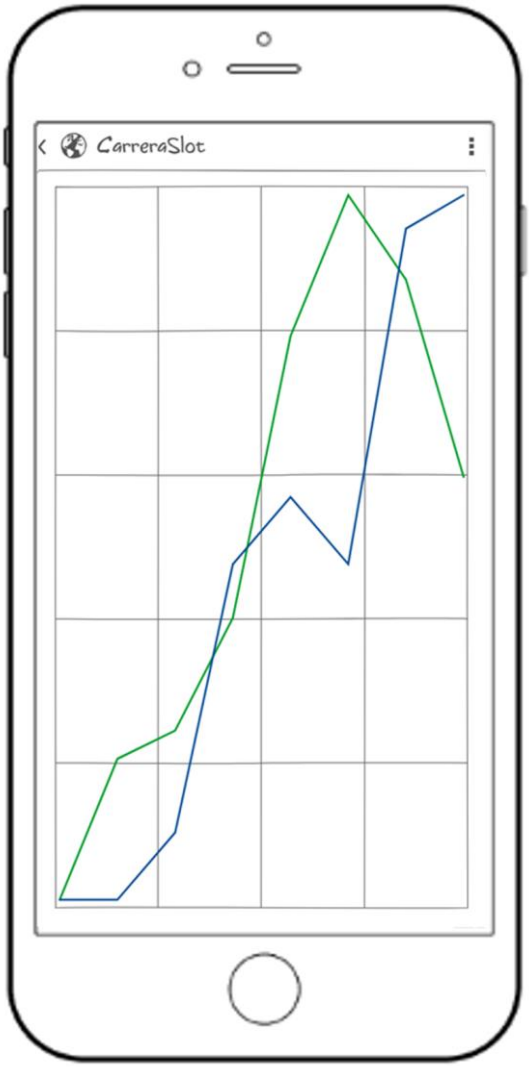

Ilustración 18. Gráficos

Visor

Con el visor el usuario puede reproducir la carrera mostrando los datos de cada piloto en el trascurso de la carrera, también dispone de opciones de reproducción donde puede variar la velocidad de reproducción, variando la tasa de refresco e incrementando la velocidad de reproducción.

Ilustración 19. Visor

4.2.2. Capa de lógica

Esta capa contiene las clases y objetos que forman la aplicación, permitiendo definir los modelos de datos que sean necesarios para el funcionamiento. Así como las acciones a realizar con dichos datos, esta capa es la encargada de dar forma a la información que obtenemos de la capa de persistencia.

El diseño de esta capa está basado el diagrama de clases obtenido en el análisis de la aplicación, Con ligeras modificaciones surgidas durante el desarrollo.

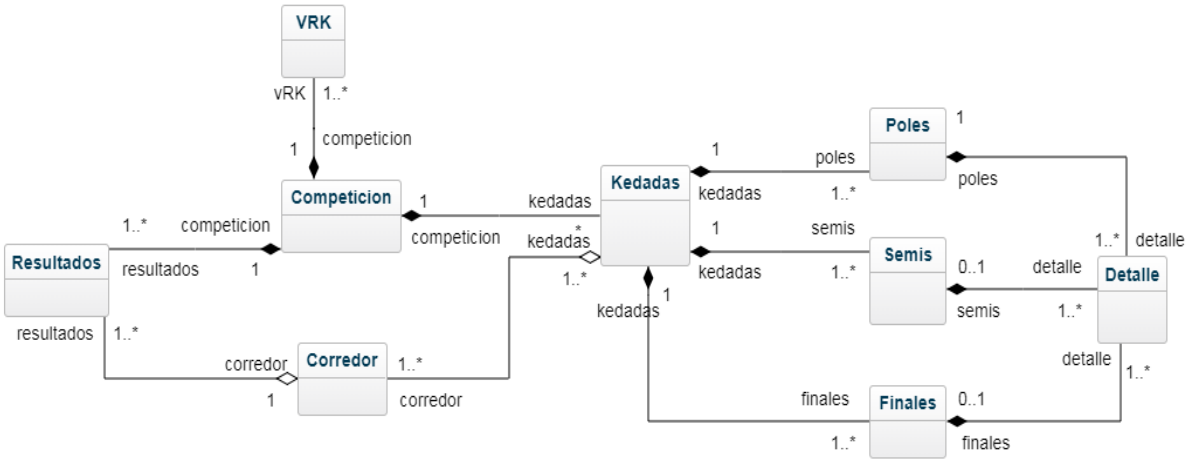

Ilustración 19. Diagrama clases capa de negocio

4.2.3. Capa de persistencia

Esta capa es la encargada de manejar los datos, es la que se encarga de guardar y leer toda la información necesaria. En nuestro caso es la encargada de leer los enlaces introducidos por el usuario donde se encuentran los archivos XML con los datos de la competición, obtener toda la información necesaria de dichos enlaces, descargar las bases de datos con los datos de las competiciones y acceder a ellas.

Esta capa está formada por las clases que podemos observar en la siguiente imagen y son las encargadas de procesar los archivos XML y lectura de la base de datos de PCLapCounter.

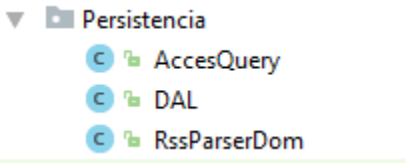

Ilustración 20. Capa de persistencia

La clase encargada de leer los Archivos XML generados por el programa CarreraSlot es RssParserDom.class, Estos archivos se pueden diferenciar dos tipos archivos unos con extensión .comp y otros .metaComp,

A continuación, se expone de manera superficial cada uno de los archivos que se puede encontrar la aplicación al procesar los enlaces.

.metaComp
Estos archivos contienen las rutas relativas donde se encuentran los archivos competición que forman la metacompetición

Para el procesamiento en la aplicación se encuentran dos casos en estos archivos:

1. Todas las rutas que contienen son a archivos de competiciones (.comp) por lo que la aplicación procesa las rutas mostrando los datos de cada una de las competiciones que contiene como se muestra en la ilustración.

2. El archivo contiene rutas a archivos de competición y metacompetición juntos por lo que la aplicación los procesa y muestra una lista con el nombre de las competiciones o metacompeticiones que contiene.

Pese a la existencia de estas dos variaciones para el procesamiento en la aplicación comparte en mismo XMLSchema[3] que se muestra a continuación.

[3] XMLschema. Lenguaje utilizado para describir la estructura y restricciones de los documentos xml, World Wide Web Consortium (1998-act.)

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="MetaCompeticion">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="competiciones">
          <xs:complexType>
            <xs:sequence>
              <xs:element type="xs:string" name="string" maxOccurs="unbounded" minOccurs="0"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element type="xs:short" name="nombre"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ilustración 21. Archivo metaComp

.Comp

En estos archivos se encuentran los datos que forman una competición, pudiendo observar los datos de cada una de las kedadas, entre los datos que se pueden observar a rasgos más generales la fecha de la kedada, el número de la kedada, los corredores que han participado en la kedada, las poles que se han realizado, las semifinales y finales que tiene la kedada. Los archivos tienen el siguiente XML Schema:

```
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified" xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Competicion">
    <xs:complexType>
      <xs:sequence>
        <xs:element type="xs:string" name="nombre"/>
        <xs:element type="xs:string" name="modoCampeonato"/>
        <xs:element type="xs:string" name="modofinales"/>
        <xs:element type="xs:string" name="modoBomilcar"/>
        <xs:element type="xs:string" name="modoResitenciaDiscont"/>
        <xs:element name="dias">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="Kedada" maxOccurs="unbounded" minOccurs="0">
                <xs:complexType>
                  <xs:sequence>
                    <xs:element type="xs:dateTime" name="fecha"/>
                    <xs:element type="xs:byte" name="nKedada"/>
                    <xs:element name="corredores">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="CorreCon" maxOccurs="unbounded" minOccurs="0">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element type="xs:string" name="piloto"/>
                                <xs:element type="xs:string" name="coche"/>
                                <xs:element type="xs:byte" name="dorsal"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
```

```
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="poles">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="Carrera" maxOccurs="unbounded" minOccurs="0">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element type="xs:byte" name="tipo"/>
                              <xs:element type="xs:string" name="nombre"/>
                              <xs:element name="detalles">
                                <xs:complexType>
                                  <xs:choice maxOccurs="unbounded" minOccurs="0">
                                    <xs:element name="Detalle">
                                      <xs:complexType>
                                        <xs:sequence>
                                          <xs:element type="xs:float" name="tiempo"/>
                                          <xs:element type="xs:float" name="promedio"/>
                                          <xs:element type="xs:byte" name="vueltas"/>
                                          <xs:element type="xs:byte" name="penalizacion"/>
                                          <xs:element name="corredor">
                                            <xs:complexType>
                                              <xs:sequence>
                                                <xs:element type="xs:string" name="piloto"/>
                                                <xs:element type="xs:string" name="coche"/>
                                                <xs:element type="xs:byte" name="dorsal"/>
                                              </xs:sequence>
                                            </xs:complexType>
                                          </xs:element>
                                          <xs:element type="xs:float" name="gasolina"/>
                                          <xs:element type="xs:float" name="instanteVRC"/>
                                        </xs:sequence>
                                      </xs:complexType>
                                    </xs:element>
                                  </xs:choice>
                                </xs:complexType>
                              </xs:element>
                              <xs:element type="xs:short" name="IDCarreraDelPCLap"/>
                              <xs:element type="xs:string" name="lideroTodasLasVueltas"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="semis">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element name="Carrera" maxOccurs="unbounded" minOccurs="0">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element type="xs:byte" name="tipo"/>
                              <xs:element type="xs:string" name="nombre"/>
                              <xs:element name="detalles">
                                <xs:complexType>
                                  <xs:sequence>
                                    <xs:element name="Detalle" maxOccurs="unbounded" minOccurs=
    "0">
                                      <xs:complexType>
                                        <xs:sequence>
                                          <xs:element type="xs:float" name="tiempo"/>
                                          <xs:element type="xs:float" name="promedio"/>
                                          <xs:element type="xs:byte" name="vueltas"/>
                                          <xs:element type="xs:byte" name="penalizacion"/>
                                          <xs:element name="corredor">
                                            <xs:complexType>
```

```
                                                <xs:sequence>
                                                  <xs:element type="xs:string" name="piloto"/>
                                                  <xs:element type="xs:string" name="coche"/>
                                                  <xs:element type="xs:byte" name="dorsal"/>
                                                </xs:sequence>
                                              </xs:complexType>
                                            </xs:element>
                                            <xs:element type="xs:float" name="gasolina"/>
                                            <xs:element type="xs:float" name="instanteVRC"/>
                                          </xs:sequence>
                                        </xs:complexType>
                                      </xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                                <xs:element type="xs:short" name="IDCarreraDelPCLap"/>
                                <xs:element type="xs:string" name="lideroTodasLasVueltas"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="finales">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="Carrera" maxOccurs="unbounded" minOccurs="0">
                            <xs:complexType>
                              <xs:sequence>
                                <xs:element type="xs:byte" name="tipo"/>
                                <xs:element type="xs:string" name="nombre"/>
                                <xs:element name="detalles">
                                  <xs:complexType>
                                    <xs:sequence>
                                      <xs:element name="Detalle" maxOccurs="unbounded" minOccurs=
    "0">
                                        <xs:complexType>
                                          <xs:sequence>
                                            <xs:element type="xs:float" name="tiempo"/>
                                            <xs:element type="xs:float" name="promedio"/>
                                            <xs:element type="xs:byte" name="vueltas"/>
                                            <xs:element type="xs:byte" name="penalizacion"/>
                                            <xs:element name="corredor">
                                              <xs:complexType>
                                                <xs:sequence>
                                                  <xs:element type="xs:string" name="piloto"/>
                                                  <xs:element type="xs:string" name="coche"/>
                                                  <xs:element type="xs:byte" name="dorsal"/>
                                                </xs:sequence>
                                              </xs:complexType>
                                            </xs:element>
                                            <xs:element type="xs:float" name="gasolina"/>
                                            <xs:element type="xs:float" name="instanteVRC"/>
                                          </xs:sequence>
                                        </xs:complexType>
                                      </xs:element>
                                    </xs:sequence>
                                  </xs:complexType>
                                </xs:element>
                                <xs:element type="xs:short" name="IDCarreraDelPCLap"/>
                                <xs:element type="xs:string" name="lideroTodasLasVueltas"/>
                              </xs:sequence>
                            </xs:complexType>
                          </xs:element>
                        </xs:sequence>
```

```
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="detallePole">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element type="xs:float" name="tiempo"/>
                        <xs:element type="xs:float" name="promedio"/>
                        <xs:element type="xs:byte" name="vueltas"/>
                        <xs:element type="xs:byte" name="penalizacion"/>
                        <xs:element name="corredor">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element type="xs:string" name="piloto"/>
                              <xs:element type="xs:string" name="coche"/>
                              <xs:element type="xs:byte" name="dorsal"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <xs:element type="xs:float" name="gasolina"/>
                        <xs:element type="xs:float" name="instanteVRC"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="detalleCarrera">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element type="xs:float" name="tiempo"/>
                        <xs:element type="xs:float" name="promedio"/>
                        <xs:element type="xs:byte" name="vueltas"/>
                        <xs:element type="xs:byte" name="penalizacion"/>
                        <xs:element name="corredor">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element type="xs:string" name="piloto"/>
                              <xs:element type="xs:string" name="coche"/>
                              <xs:element type="xs:byte" name="dorsal"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <xs:element type="xs:float" name="gasolina"/>
                        <xs:element type="xs:float" name="instanteVRC"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element name="detalleVRKedada">
                    <xs:complexType>
                      <xs:sequence>
                        <xs:element type="xs:float" name="tiempo"/>
                        <xs:element type="xs:float" name="promedio"/>
                        <xs:element type="xs:byte" name="vueltas"/>
                        <xs:element type="xs:byte" name="penalizacion"/>
                        <xs:element name="corredor">
                          <xs:complexType>
                            <xs:sequence>
                              <xs:element type="xs:string" name="piloto"/>
                              <xs:element type="xs:string" name="coche"/>
                              <xs:element type="xs:byte" name="dorsal"/>
                            </xs:sequence>
                          </xs:complexType>
                        </xs:element>
                        <xs:element type="xs:float" name="gasolina"/>
                        <xs:element type="xs:float" name="instanteVRC"/>
                      </xs:sequence>
                    </xs:complexType>
                  </xs:element>
                  <xs:element type="xs:string" name="anulada"/>
```

```
                    <xs:element type="xs:string" name="lideroTodasLasVueltasDeKedada"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Ilustración 22. Archivo .comp

La clase encarada de la realización de las consultas sobre la base de datos de PCLapCounter es la clase AccesQuery, que contiene las funciones necesarias para realizar las consultas para necesarias para obtener la información que se enviara a la capa de presentación para mostrar.

Para una mejor comprensión de las funciones de esta clase se va a explicar la estructura de la base de datos y la información que contiene en cada una de sus tablas.

Base de datos PCLapCounter

Como se vio anteriormente los datos de la aplicación se encuentra divididos entre los archivos XML generados por el programa CarreraSlot y las bases de datos procedentes el PCLapCounter, a continuación, se va explicar estas últimas mostrando a rasgos generales su estructura y los datos que son relevantes para el desarrollo de nuestra aplicación.

En ella se encuentran las siguientes tablas relevantes para obtener los datos necesarios para la aplicación:

- RaceHistory: En esta tabla se observa la configuración de cada carrera como por ejemplo la longitud, si es una carrera programada a vueltas o a tiempo, si la carrera tiene clasificación, entrenamientos, si se consume gasolina, etc...
- RaceHistoryClas: Aquí se puede ver los datos generales de cada una de las poles, semifinales y finales que forman las kedadas observando el id de la sesión, el piloto, la posición, el número total de vueltas, y el tiempo empleado en realizarlas.
- RaceHistoryLap: En esta tabla es donde está la mayoría de la información ya que contiene un registro vuelta a vuelta de cada una de las sesiones que forman la kedada permitiéndo obtener todos los datos necesarios para posteriormente generar los gráficos que se han visto anteriormente en el análisis.
- RaceHistorySum: Para finalizar en aquí se observa un resumen por cada sesión por piloto, donde están el tiempo más rápido en dar una vuelta, el tiempo medio por vuelta de la sesión, el tiempo más lento en una vuelta, las vueltas totales, así como otros datos.

Una vez visto las tablas y el contenido de cada una de estas ya se puede entender las relaciones existentes entre ellas. A continuación, se muestra un esquema de la base de datos con las relaciones existentes entre las tablas.

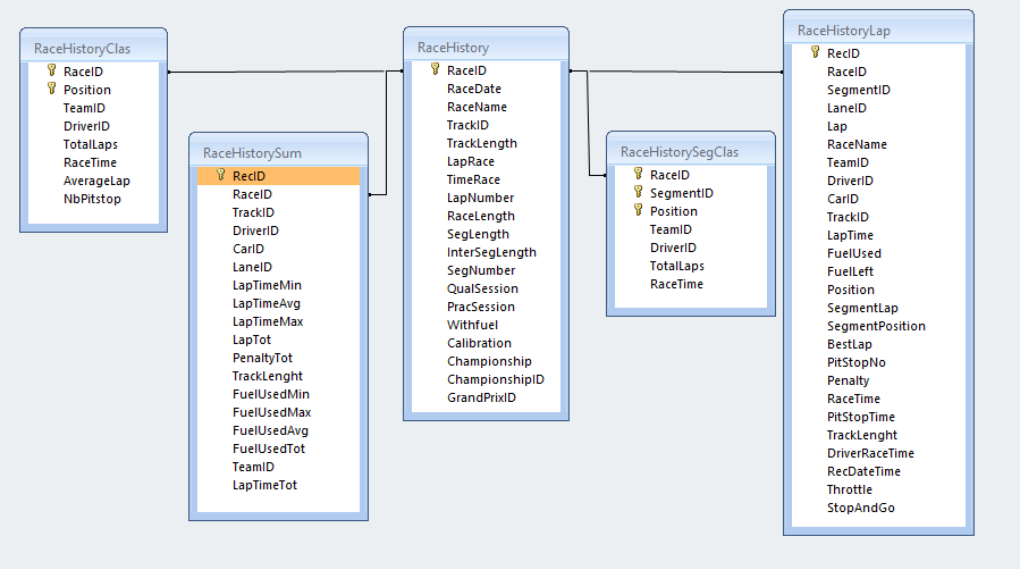

Ilustración 23. Esquema Base de datos

5. Implementación

Una vez realizado el análisis y conociendo todos los requisitos que debe cumplir la aplicación, así como el comportamiento de la misma. Se procede a la implementación de la aplicación como se verá en este apartado vamos a ver todas las herramientas utilizadas, los problemas que aparecen en el desarrollo y las soluciones a dichos problemas para acabar obteniendo nuestra aplicación.

En primer lugar, se presenta el entorno de desarrollo y todas las herramientas que necesarias para el desarrollo. Una vez presentadas todas las herramientas necesarias se va a explicar cómo hemos desarrollado nuestra aplicación mostrando las soluciones implementadas.

5.1. Lenguajes empleados

Para el desarrollo de nuestro proyecto se ha hecho uso de diversos lenguajes de programación:

- Java
  Java es un lenguaje de programación utilizado para la implementación del proyecto, Java es un lenguaje de propósito general, concurrente y orientado a objetos, fue desarrollado por Sun Microsistems que fue adquirida por Oracle.
- XML
  XML se define como las siglas de eXtensible Markup Language, se trata de un meta-lenguaje de marcas extensible. Fue desarrollado por el World Wide Web Consortium (W3C) y se emplea para almacenar datos de forma legible.
- SQL
  SQL es un lenguaje declarativo estándar internacional de comunicación dentro de las bases de datos que nos permite a todos el acceso y manipulación de datos en una base de datos, y además se puede integrar a lenguajes de programación

5.2. Tecnologías utilizadas

Para desarrollar la aplicación al ser una aplicación móvil para teléfonos con Android, se dispone del entorno de desarrollo “Android Studio”, este entorno de desarrollo está basado en intelliJ IDEA de JetBrains y está publicado bajo una licencia apache 2.0. Android Studio es el IDE oficial de Android.

Debido a que las bases de datos procedentes de PCLapCounter están en Access también se necesita utilizar Microsoft Access para poder ver las bases de datos y de esta forma analizarlas para entender como están estructuradas y como localizar los datos necesarios para nuestra aplicación.

También se ha necesitado el programa CarreraSlot para poder ver los datos de los XML y de esta forma entender su estructura, permitiendo generar las gráficas y las tablas para comprobar los datos con nuestra aplicación.

5.3. Aplicación Android

A la hora de implementar una aplicación Android el entorno de desarrollo Android Studio impone una estructura general para las aplicaciones permitiendo poca variación dicha estructura es la siguiente:

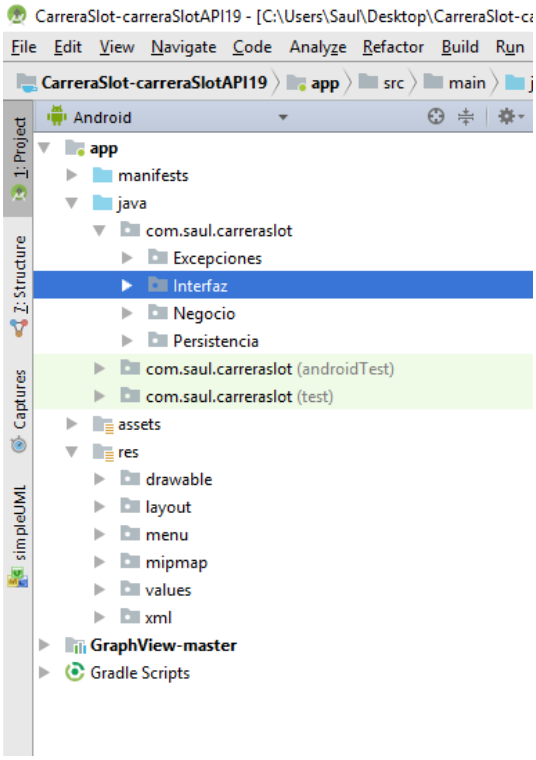

Ilustración 24. Proyecto CarreraSlot

Como se puede observar los proyectos Android están estructurados en diferentes carpetas, la primera carpeta manifests contiene un documento donde se declaran las actividades de nuestra aplicación para posteriormente poder ser ejecutadas por el terminal, también se establece los permisos de la aplicación y a que funciones del teléfono tiene acceso, dentro de java se encuentra un paquete donde van ubicadas las clases java, en las cuales se establece la funcionalidad de la aplicación.
El directorio Assets es donde se almacenan las bases de datos de nuestro proyecto, así como cualquier añadido que queramos como tipografías y res donde se encuentran diferentes carpetas que contienen diferente contenido grafico de la aplicación como son:

- /res/drawable/. Contienen las imágenes de la aplicación y otros elementos gráficos.
- /res/layout/. Contienen los ficheros XML que definen las pantallas de la interfaz gráfica.
- /res/menu/. Contiene los ficheros XML que definen los menús de la aplicación.
- /res/values/. Contiene otros ficheros XML de recursos de la aplicación como por ejemplo cadenas de texto (strings.xml), estilos (styles.xml), colores (colors.xml), etc.
- /res/xml/. Contiene otros ficheros de datos XML utilizados por la aplicación.

5.4. Programación de la aplicación

Para la implementación de la aplicación como se ha visto anteriormente se ha realizado una implementación por capas definiendo tres capas que pasaremos a explicar más detalladamente a continuación.

5.4.1. Capa de presentación (Interfaz):

Aquí se implementan todas las clases con las que tiene interacción el usuario mostrando la información y ejecutando las acciones del usuario, a continuación, se muestran las clases contenidas en esta capa.

inicio

Aquí se comprueban las URL introducidas por el usuario y la lectura los ficheros XML ubicados en dichas direcciones.

En esta actividad además de leer los ficheros XML se crea una pequeña base de datos que contiene las URL introducidas por el usuario.

```
1.  //Buscar url introducidas
2.  //Abrimos la base de datos 'DBUsuarios' en modo escritura
3.  SQLiteHelper helper =
4.          new SQLiteHelper(context, "DBURL", null, 1);
5.  SQLiteDatabase db = helper.getWritableDatabase();
6.  if(db != null)
7.  {
8.      //Leemos los datos en la tabla Usuarios
9.      Cursor cursor = db.rawQuery("SELECT url FROM URLS",null);
10.     if (cursor.moveToFirst()) {
11.         do{
12.             String url= cursor.getString(0);
13.             Log.i("URL INICIO",url);
14.             urls.add(url);
15.         }while (cursor.moveToNext());
16.     }
17.     //Cerramos la base de datos
18.     db.close();
19. }
20. if (urls.isEmpty()) {
21.     Intent intent = new Intent().setClass(
22.             Inicio.this, InsertURL.class);
23.     intent.putExtra("URLS",urls);
24.     startActivity(intent);
25. }else{
26.     CargarXmlTask tarea = new CargarXmlTask();
27.     try {
28.         /*Leeremos URL introducidas en la configuración*/
29.         String[] links = urls.toArray(new String[0]);
30.         carreras = tarea.execute(links).get();
31.     } catch (InterruptedException e) {
32.         e.printStackTrace();
33.     } catch (ExecutionException e) {
34.         e.printStackTrace();
35.     }
36. }
```

Ilustración 25. Inicio.class

InsertURL

El usuario introduce las URL que quiere cargar para mostrar los datos, estas URL introducidas son almacenadas en la base de datos que se ha creado al iniciar la aplicación por primera vez permitiendo al usuario guardar las URL's y no tener que introducirlas cada vez que inicia la aplicación.

```
1.  save.setOnClickListener(new View.OnClickListener() {
2.      public void onClick(View v) {
3.          SQLiteHelper helper =
4.                  new SQLiteHelper(context, "DBURL", null, 1);
5.          SQLiteDatabase db = helper.getWritableDatabase();
6.          if(db != null)
7.          {
8.              //Insertamos los datos en la tabla Usuarios
9.              db.execSQL("INSERT INTO URLS (url) " +
10.                     "VALUES ('" + url.getText()+"')");
11.             //Cerramos la base de datos
12.             db.close();
13.         }
14.         if (urls!=null){
15.             urls.add(url.getText()+"");
16.         }else{
17.             urls= new ArrayList<String>();
18.             urls.add(url.getText()+"");
19.         }
20.         url.setText("http://www.pep0n.com/MORATROS/DATOS/MORATROS/");
21.         String[] items = urls.toArray(new String[0]);
22.         Log.i("insert",items.toString());
23.         ArrayAdapter<String> adapter = new ArrayAdapter<String>(context,
24.                 android.R.layout.simple_list_item_1, items);
25.         list.setAdapter(adapter);
26.
27.     }
28. });
```

Ilustración 26. Función guardado URL

SelectorURL

Se muestran las competiciones o metacompeticiones cargadas al introducir una URL, y se actualizan las bases de datos.

En esta actividad dependiendo de la competición seleccionada se ejecuta una actividad diferente, si la competición seleccionada es una metacompetición donde todas las competiciones que la forman son competiciones se muestra una clasificación general por competición como hemos visto en la ilustración 5, la actividad lanzada es la SelectorCarrera, si por el contrario la metacompetición lanzada es una mezcla de competiciones y metacompeticiones se vuelve a cargar la misma actividad, pero esta vez con las competiciones y metacompeticiones que se encuentran en el interior de la seleccionada.

Para finalizar si la URL seleccionada es una competición, se carga la actividad SelectorKedada donde se muestra toda la información de la competición seleccionada y se permite al usuario seleccionar las kedadas que forman la competición.
Para ello al procesar el archivo se realiza una comprobación de las etiquetas y dependiendo del resultado se realiza cada una de las acciones anteriormente descritas. Todo este proceso se realiza en una tarea asíncrona para no bloquear el dispositivo dicha tarea la se muestra a continuación:

```
1.  private class CargarXmlTask extends AsyncTask<String, Integer, Integer> {
2.
3.          String url;
4.          protected Integer doInBackground(String... params) {
5.              Log.i("PARAMS",params[0]);
6.              url=params[0];
7.              RssParserDom saxparser =
8.                      new RssParserDom(params[0]);
9.              int accion = saxparser.selectorNew();
10.
11.             if(accion==1){
12.                 competicion = saxparser.parseMetacompeticiones();
13.             }else if(accion==2){
14.                 competicion = saxparser.parseMetacompeticiones();
15.             } else if(accion==0){
16.                 kedadas = saxparser.parseKedadas();
17.             }
18.                 return accion ;
19.         }
20.         @Override
21.         protected void onPreExecute() {
22.             pDialog = new ProgressDialog(SelectorURL.this);
23.             pDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
24.             pDialog.setMessage("Procesando...");
25.             pDialog.setCancelable(false);
26.             pDialog.setIndeterminate(true);
27.             pDialog.show();
28.         }
29.         @Override
30.         protected void onPostExecute(Integer result) {
31.             pDialog.dismiss();
32.             if(result==1){
33.                 intent = new Intent(SelectorURL.this, SelectorURL.class);
34.                 intent.putExtra("URL",urlBase);
35.                 intent.putExtra("CARRERAS",competicion);
36.             }else if(result==2){
37.                 intent = new Intent(SelectorURL.this, SelectorCarrera.class)
    ;
38.                 intent.putExtra("CARRERAS",competicion);
39.                 String newUrl = url.substring(0,url.lastIndexOf("/")+1);
40.                 intent.putExtra("URL",newUrl);
41.                 intent.putExtra("TITULO",titulo);
42.             }
43.             else if(result==0){
44.                 intent = new Intent(SelectorURL.this, SelectorKedada.class);

45.                 intent.putExtra("URL",url);
46.                 ArrayList<Resultados> resultados = CalcularPodium((ArrayList
    <Kedadas>) kedadas);
47.                 intent.putExtra("GENERAL", resultados);
48.                 intent.putExtra("KEDADAS", (Serializable) kedadas);
49.                 intent.putExtra("TITULO",titulo);
50.             }
51.             startActivity(intent);
52.         }
53.
54.         @Override
55.         protected void onCancelled() {
56.             Toast.makeText(SelectorURL.this, "Tarea cancelada!",
57.                     Toast.LENGTH_SHORT).show();
58.         }
59.     }
```

Ilustración 27. Función CargarXML

Como se ve en el código primero llamando a la función selectroNew que procesa el archivo y devuelve 0,1,2 dependiendo del tipo de archivo, según este resultado el archivo se procesa como una meta competición o una competición para después realizar las acciones anteriormente descritas, en la función onPostExecute de la tarea asíncrona se lanza la actividad según se ha visto al inicio de esta sección.

SelectorCarrera

Esta actividad quizás sea la que más cálculos realiza ya que de cada competición albergada en la metacompetición se tiene que leer las kedadas que la forman y calcular el pódium de dicha kedada, para ello se realiza a través de una tarea asíncrona que lee las kedadas de la competición y calcula el pódium de cada competición, dicha tarea es la siguiente:

```
1.  private class CalcularPodium extends AsyncTask<String, Integer, Boolean> {
2.          Context context;
3.          public CargarXmlTask(Context context) {
4.              this.context = context;
5.          }
6.          @Override
7.          protected Boolean doInBackground(String... params) {
8.               /*Leo todas las kedadas para generar los datos*/
9.              meta=new Competicion(titulo);
10.             List<Kedadas> kedadaToPases;
11.             for (int i=0;i<competicion.size();i++){
12.                 String url=procesarUrl(urlBase,competicion.get(i));
13.                 RssParserDom saxparserKedada =
14.                         new RssParserDom(url);
15.                 kedadaToPases = saxparserKedada.parseKedadas();
16.                 meta.addKedadas((ArrayList<Kedadas>) kedadaToPases);
17.             }
18.             return true;
19.         }
20.         @Override
21.         protected void onPreExecute() {
22.             pDialog = new ProgressDialog(SelectorCarrera.this);
23.             pDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);
24.             pDialog.setMessage("Procesando...");
25.             pDialog.setCancelable(false);
26.             pDialog.setIndeterminate(true);
27.             pDialog.show();
28.         }
29.         @Override
30.         protected void onPostExecute(Boolean result) {
31.             pDialog.dismiss();
32.             for (int i = 0; i < competicion.size(); i++) {
33.                 //Log.i("SELECTORCARRERA",meta.getKedadas().get(i).toString(
    ));
34.                 podium = CalcularPodium(meta.getKedadas().get(i));
35.                 res.add(podium);
36.                 for (int j = 0; j < 6; j++) {
37.                     datos[i][0] = String.valueOf(i+1);
38.                     datos[i][1] = competicion.get(i).substring(competicion.g
    et(i).indexOf("-") + 2, competicion.get(i).lastIndexOf("\\"));
39.                     datos[i][2] = String.valueOf(podium.get(0).getMaxKedadas
    ());
40.                     datos[i][3] = podium.get(0).getNombre()+"("+podium.get(0
    ).getSumaPuntos()+")";
41.                     datos[i][4] = podium.get(1).getNombre()+"("+podium.get(1
    ).getSumaPuntos()+")";
42.                     datos[i][5] = podium.get(2).getNombre()+"("+podium.get(2
    ).getSumaPuntos()+")";
43.                 }
44.             }
45.             rellenarTabla(context);
46.         }
47.
48.         @Override
49.         protected void onCancelled() {
50.             Toast.makeText(SelectorCarrera.this, "Tarea cancelada!",
51.                     Toast.LENGTH_SHORT).show();
52.         }
53.     }
```

Ilustración 28. Función CalcularPodium

Como se ve en la función doInBackground de la tarea asíncrona se recorre cada una de las competiciones que forman nuestra metacompetición obteniendo los datos de las kedadas con la función parseKedadas.

Finalmente, una vez obtenidas todas las kedadas en la sección OnPostExecute se calcula el pódium de cada una de las competiciones y se realiza una carga de datos en un array bidimensional con los datos a mostrar para posteriormente en la función rellenarTabla montar la vista final para el usuario.

SelectorKedada

En esta actividad se muestra al usuario todos los datos pertenecientes a la competición seleccionada, así que es la encargada de mostrar la información y cargarla correctamente esta actividad recibe de su predecesora la información de la competición por lo que ya no tiene que leer los archivos XML, para ello se implementan las funciones, cargarClasificacion, cargarPole, cargarvictorias, CargarVRC, cargarVRP y cargarVRK.

Kedada

Al igual que la actividad explicada anteriormente esta actividad recibe toda la información de su predecesora agilizando la muestra de resultados y la navegación, en esta vista se cargan las tablas con el resultado de la carrera, la pole y las diferentes semifinales y finales para ello se implementan las siguientes funciones: TablaCarrera, TablaPole, Tablasemi, TablaFinal.

A continuación, se explica el funcionamiento de una de estas funciones, que es similar al de las funciones de la actividad anterior, para este ejemplo se ha elegido la función TablaCarrera que se encarga de generar una tabla con el reparto de puntos según la posición de la kedada quedando la función de la siguiente manera:

```
1.  public void TablaCarrera(LinearLayout container, Context context, Kedadas kedada)
    {
2.
3.          String cabeceras[]={"CARRERA","PILOTO","PUNTOS"};
4.          //Log.i("POLES KEDADA",kedada.getPoles().toString());
5.          ArrayList<Resultados> puntos = kedada.CalcularPuntos();
6.          String datos[][] = new String[puntos.size()][cabeceras.length];
7.          /*Preparar los datos de la tabla*/
8.          for (int i = 0; i < puntos.size(); i++) {
9.              datos[i][0] = String.valueOf(i+1);
10.             datos[i][1] = puntos.get(i).getNombre();
11.             datos[i][2] = String.valueOf(puntos.get(i).getPuntos());
12.         }
13.         LinearLayout tablalayout = new LinearLayout(this);//layout con todas las t
    ablas
14.         tablalayout.setBackgroundDrawable(this.getResources().getDrawable(android.
    R.drawable.dialog_holo_light_frame));
15.         LinearLayout tabla = new LinearLayout(this);
16.         tabla.setLayoutParams(new TableLayout.LayoutParams(
17.                 TableLayout.LayoutParams.MATCH_PARENT, TableLayout.LayoutParams.MA
    TCH_PARENT));
18.         //tabla datos
19.         TableLayout tableLayout = new TableLayout(this);
20.         LinearLayout.LayoutParams params = new LinearLayout.LayoutParams(ViewGroup
    .LayoutParams.MATCH_PARENT, ViewGroup.LayoutParams.WRAP_CONTENT);
21.         params.setMargins(30,30,30,30);
22.         tableLayout.setLayoutParams(params);
23.
24.         TableRow cabecera = new TableRow(this);
25.         cabecera.setLayoutParams(new TableLayout.LayoutParams(
26.                 TableLayout.LayoutParams.MATCH_PARENT, TableLayout.LayoutParams.WR
    AP_CONTENT));
27.         for (int i = 0; i < cabeceras.length; i++)
28.         {
29.             final TextView columna = new TextView(context);
30.             TableRow.LayoutParams layoutParams = new TableRow.LayoutParams(
31.                     0, TableLayout.LayoutParams.WRAP_CONTENT);
```

```
32.             switch (i){
33.                 case 0:
34.                     layoutParams.weight = (float) 0.5;
35.                     break;
36.                 case 1:
37.                     layoutParams.weight = (float) 1;
38.                     break;
39.                 case 2:
40.                     layoutParams.weight = (float) 0.8;
41.                     break;
42.             }
43.             columna.setLayoutParams(layoutParams);
44.             columna.setText(cabeceras[i]);
45.             columna.setTextColor(Color.parseColor("#FFFFFF"));
46.             columna.setTextSize(TypedValue.COMPLEX_UNIT_SP, 14);
47.             columna.setGravity(Gravity.CENTER_HORIZONTAL);
48.             columna.setPadding(5, 5, 5, 5);
49.             columna.setBackgroundColor(Color.parseColor("#3E50B4"));
50.             cabecera.addView(columna);
51.         }
52.         tableLayout.addView(cabecera);
53.         //filas de carreras
54.         for (int i = 0; i < puntos.size(); i++){//recorremos todas las filas
55.             final TableRow fila = new TableRow(this);
56.             for(int j = 0; j < cabeceras.length; j++)//añadimos todos las columnas

57.             {
58.                 TextView columna = new TextView(this);
59.                 //primero numeramos las carreras
60.                 TableRow.LayoutParams layoutParams = new TableRow.LayoutParams(
61.                         0, TableLayout.LayoutParams.WRAP_CONTENT);
62.                 columna.setLayoutParams(layoutParams);
63.                 switch (j){
64.                     case 0:
65.                         layoutParams.weight = (float) 0.5;
66.                         break;
67.
68.                     case 1:
69.                         layoutParams.weight = (float) 1;
70.                         break;
71.
72.                     case 2:
73.                         layoutParams.weight = (float) 0.8;
74.                         break;
75.                 }
76.                 columna.setText(datos[i][j]);
77.                 columna.setTextColor(Color.parseColor("#000000"));
78.                 columna.setTextSize(TypedValue.COMPLEX_UNIT_SP, 14);
79.                 columna.setGravity(Gravity.CENTER_HORIZONTAL);
80.                 columna.setPadding(5, 5, 5, 5);
81.                 fila.addView(columna);
82.             }
83.             if(i%2==0) {
84.                 fila.setBackgroundColor(Color.parseColor("#F5F5F5"));
85.             }
86.             tableLayout.addView(fila);
87.         }
88.         // Línea que separa los datos de la fila de totales
89.         //pCon.addView(tableLayout);
90.         tabla.addView(tableLayout);
91.         tablalayout.addView(tabla);
92.         container.addView(tablalayout);
93.     }
```

Ilustración 29. Función TablaCarrera

En estas funciones se observan dos comportamientos, en una primera instancia se preparan los datos necesarios para rellenar la tabla y posteriormente con los datos calculados se rellena las tablas que vera el usuario.

Para finalizar se encuentran las actividades Pole, Semis y Finales, estas actividades son las encargadas de mostrar toda la información detallada de la pole, semi o final seleccionada.
La información a mostrar en estas actividades se encuentra en la base de datos procedentes de PCLapCounter.

5.4.2. Capa de lógica (Negocio):

En esta capa están todas las clases que definen el modelo de datos permitiendo trabajar con ellos mediante funciones, para entender las relaciones que existen entre las diferentes clases se ha confeccionado el siguiente diagrama con el nombre de las clases y las relaciones existentes entre ellas.

La capa de negocio tiene la estructura del diagrama de clases visto en el análisis con el añadido de la clase VRK que surge de la necesidad de mostrar el número de vueltas rápidas realizadas en la competición y que piloto las ha realizado.

A continuación, se procede a explicar cada una de las clases que lo forman.

Competición:

Esta clase contiene una lista de kedadas que forman la competición, así como el nombre de la competición para poder identificarla, también se han implementado funciones que nos permiten añadir kedadas a la competición y consultar las kedadas de esta.

```
1.  public class Competicion  {
2.      /*datos competicion*/
3.      String nombre;
4.      ArrayList<ArrayList<Kedadas>> kedadas= new ArrayList<ArrayList<Kedadas>>()
    ;
5.
6.      public Competicion(String nombre) {
7.          this.nombre = nombre;
8.      }
9.
10.     @Override
11.     public String toString() {
12.         return "Competicion{" +
13.                 "nombre='" + nombre + '\'' +
14.                 '}';
15.     }
16.
17.     public String getNombre() {
18.         return nombre;
19.     }
20.
21.     public void setNombre(String nombre) {
22.         this.nombre = nombre;
23.     }
24.
25.     public ArrayList<ArrayList<Kedadas>> getKedadas() {
26.         return kedadas;
27.     }
28.
29.     public void addKedadas(ArrayList<Kedadas> kedadas) {
30.         this.kedadas.add(kedadas);
31.     }
```

Ilustración 30. Competicion.class

VRK

En la clase VRK encontramos todo lo necesario para la ordenación del número de vueltas rápidas de la kedada realizadas por cada piloto.

```
1.  public class VRK {
2.      String piloto;
3.      int numVueltas;
```

Ilustración 31. VRK.class

Kedada

Aquí se encuentra más información puesto que en esta clase está contenida toda la información como son las poles, semis y finales, así como los datos generales de la kedada como tiempo de la pole, pole-man, ganador, etc…

A continuación, se puede observar la definición de todas las variables necesarias para esta clase.

```
1.  public class Kedadas implements Parcelable{
2.      /*datos kedada*/
3.      int id,nkedada;
4.      String date;
5.      /*detalle pole*/
6.      String tiempoPole,poleMan,vueltasPole,instantePole,penalizacionPole,prom
    edioPole,gasolinaPole;
7.      /*detalle carrera*/
8.      String winner,vueltasCarrera,vrc,promedioCarrera,penalizacionCarrera,gas
    olinaCarrera;
9.      /*detalles fasterLap kedada*/
10.     String fastterLap,promedioKedada,vueltasKedada,penalizacionKedada,faster
    lapPilot,gasolinakedada,instantekedada,anulada;
11.     /*Datos poles semis finales*/
12.     ArrayList<Poles> poles = new ArrayList<Poles>();
13.     ArrayList<Semis> semis = new ArrayList<Semis>();
14.     ArrayList<Finales> finales = new ArrayList<Finales>();
15.     ArrayList<Corredor>corredores = new ArrayList<Corredor>();
16.
17.     private String instanteVrc;
18.     /*datalles puntos competicion*/
19.     int [] puntos;
20.     int puntosPorPole;
21.     int puntosPorVRC;
22.     int kedadasAntesDeAplicarPeores;
23.     int resultadosAQuitar;
```

Ilustración 32. Kedada.class

En esta clase además también se encuentran numerosas funciones para calcular los datos necesarios, además de las correspondientes funciones get y set de cada campo, estas funciones permiten calcular los puntos de la kedada y calcular los resultados de las poles, semis y finales de la kedada.

Pole

En esta clase se define los datos sobre la pole que no es más que los DetallePole de cada piloto, el nombre de la pole, el IDCarreraDelPCLap.

```
1.  public class Poles implements Parcelable{
2.      int tipo;
3.      int IDCarreraDelPCLap;
4.      String nombre;
5.      ArrayList<DetallePole> detallePoles;
```

Ilustración 33. Poles.class

Semis y Finales

Al igual que en la pole está el DetalleCarrera de cada piloto que compitió en la semifinal o final, el IDCarreraDelPCLap, para poder realizar las búsquedas en la base de datos y el nombre de la semifinal

```
1.  public class Semis implements Parcelable {
2.      int tipo;
3.      int IDCarreraDelPCLap;
4.      String nombre,lideroTodasLasVueltas;
5.      ArrayList<DetalleCarrera> detalleSemis;
6.  }
7.  public class Finales implements Parcelable{
8.      int tipo;
9.      int IDCarreraDelPCLap;
10.     String nombre;
11.     ArrayList<DetalleCarrera> detalleFinal;
12.     String lideroTodasLasVueltas;
```

Ilustración 34. Semis.class

DetalleCarrera

Como en DetallePole se encuentra toda la información leída de los XML pero esta vez relacionada con las semifinales o finales disputadas.

```
1.  public class DetalleCarrera implements Parcelable {
2.      String piloto,coche,vueltas;
3.      String tiempo,promedio,dorsal,gasolina,instante,penalizacion;
```

Ilustración 35. DetalleCarrera.class

Resultados:

Esta clase es de ayuda para calcular toda la información relevante por piloto de una competición a continuación la cabecera de la clase donde definimos las variables necesarias.

```
1.  public class Resultados implements Comparable<Resultados>, Parcelable{
2.      String nombre;
3.      float promedio;
4.      float vueltas;
5.      int puntos;
6.      int minPuntos=25;
7.      int nKedadas=0;
8.      int npoles=0;
9.      boolean pole;
10.     int totalPuntos=puntos;
11.     int peores;
12.     int AntesDePeores;
13.     int MaxKedadas;
14.
15.     ArrayList<Integer>general=new ArrayList<Integer>();
16.     ArrayList<Integer>corridas=new ArrayList<Integer>();
17.     ArrayList<Integer>poles=new ArrayList<Integer>();
18.     ArrayList<Integer>anuladas=new ArrayList<Integer>();
19.     private int sumaPuntos;
```

Ilustración 36. Resultados.class

Como se ve con esta cabecera tenemos toda la información de un piloto sobre una competición, ahora se detalla cada una de estas variables y el uso que tienen para una mejor comprensión del proyecto.

- General, aquí se almacena una lista con los puntos obtenidos en cada una de las kedadas que ha disputado el piloto
- Corridas, en esta variable tiene una lista donde está el número de las kedadas que ha disputado
- Poles, en las poles hay una lista de enteros donde almacenamos el número de kedada donde el piloto ha obtenido una pole
- Anuladas aquí se almacena una lista con el número de las kedadas que han sido anuladas.

Estas variables son de utilidad a la hora de confeccionar la clasificación de la competición, también hay más variables que son de utilidad para controlar diversos aspectos de la competición como pueden ser:

- AntesDePeores que son el número de kedadas que tiene que participar un piloto antes de eliminar peores resultados.
- Peores son el número de peores resultados a eliminar.
- minPuntos son la cantidad más baja de puntos que ha conseguido un piloto en un campeonato lo que viene siendo su peor resultado.
- nKedadas aquí tenemos almacenado el número de kedadas que ha participado que junto a AntesDePeores ayuda a controlar cuando hay que eliminar los peores resultados
- nPoles aquí esta el número de poles obtenidas por el piloto en la competición.
- Puntos aquí se almacena la cantidad de puntos totales obtenidos por el piloto.
- sumaPuntos es una variable auxiliar utilizada para calcular los putos obtenidos por el piloto eliminando los peores resultados.

5.4.3. Capa de datos (Persistencia):

En la capa de persistencia se encuentra las clases que permite leer los archivos XML procesándolos y creando todos los modelos según el archivo leído.

Dado la existencia de dos tipos de archivos claramente diferenciados como se ha visto en el transcurso de este documento, los .comp y .metaComp, se han implementado dos funciones para el procesado de cada uno de estos archivos en el apéndice se entra en más en profundidad sobre cada una de las funciones y la estructura de estos archivos.

Función lectura archivos .comp: Esta función es la encargada de leer los archivos .comp y procesarlo para obtener como se vio en la capa de negocio una serie de kedadas con todos los datos que contiene este archivo.

Función de lectura archivos .metaComp: Esta función es la encargada de procesar los archivos .metaComp y devuelve una lista con las competiciones que comprenden dicha metacompetición para posteriormente mostrar la lista de competiciones o metacompeticiones al usuario para la navegación a través de estas.

Así mismo también se encuentran en esta capa una función encargada de leer el archivo de configuración de cada kedada para completar la información necesaria de las kedadas para el correcto procesamiento de los datos.

En esta capa también está la clase AccessQuery, la encargada de leer la información de las bases de datos procedentes de PCLapCounter, esta es una base de datos de Microsoft Access, como Android no tiene soporte para bases de datos Access tenemos que utilizar librerías externas, en este caso se ha utilizado ucanaccess [4] que se basa en una implementación del controlador java JDBC que nos permite leer y escribir en bases de datos.

Para la lectura de estos datos se han implementado diversas funciones según la opción seleccionada en las actividades Poles, Semis, Finales. Las funciones son las siguientes:

queryRapidas

```
1.  public ArrayList<ArrayList<Vuelta>> queryRapidas(int idRace , String year){
2.      ArrayList<Vuelta> rapida = new ArrayList<Vuelta>();
3.      ArrayList<ArrayList<Vuelta>> tiempospilotosvueltas = new ArrayList<ArrayList<Vuelta>>();
4.      ArrayList<String> nombrePilotos = new ArrayList<String>();
5.
6.      Connection conn = null;
7.      String extStore = System.getenv("EXTERNAL_STORAGE");
8.      File f_exts = new File(extStore + "/BD/pclc_505_MORATROS"+
9.              year.substring(2,year.length())+".mdb");
10.     //File f_exts = new File(extStore + "/BD/pclc_505_MORATROS15.mdb");
11.     try {
12.         Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
13.
14.         //coger el número maximo de vueltas
15.         conn = DriverManager.getConnection(
16.                 "jdbc:ucanaccess://" + f_exts.getAbsolutePath()+
17.                         ";keepMirror="+extStore + "/BD/"+year);
18.         Statement s = conn.createStatement();
19.         //RAPIDAS
20.         ResultSet rs = s.executeQuery("SELECT Lap,DriverID,LapTime " +
21.                 "FROM RaceHistoryLap " +
22.                 "WHERE RaceID ="+idRace+" " +
23.                 "AND BestLap = 1 ORDER BY DriverID,Lap;");
24.         while (rs.next()) {
```

[4] Ucanaccess (http://ucanaccess.sourceforge.net/site.html), Marco Amadei

```
            //Procesar Resultados
            //SI no esta el nombre del piloto agrego
            if(!nombrePilotos.contains(rs.getString(2))){
                //cambio de piloto y ya tengo uno en la lista, agrego tiempos y borro anteriores
                if(nombrePilotos.size()>0){
                    tiempospilotosvueltas.add(rapida);
                    rapida=new ArrayList<Vuelta>();
                }
                nombrePilotos.add(rs.getString(2));
            }
            //AGREGO TIEMPOS
            Vuelta v = new Vuelta(Double.parseDouble(rs.getString(1)),
                    rs.getString(2),
                    Double.parseDouble(rs.getString(3)));
            rapida.add(v);
        }
        //agrego ultimo piloto
        tiempospilotosvueltas.add(rapida);
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    return tiempospilotosvueltas;
}
```

Ilustración 37. Función queryRapidas

En esta función se obtiene una lista con los datos de las vueltas rápidas de cada piloto para después poder generar el grafico para ello realizamos una consulta SQL sobre la base de datos buscando aquellas vueltas que están marcadas con el indicador BestLap = 1.

querySospechosas

```
public ArrayList<ArrayList<Vuelta>> querySospechosas(int idRace , String year){
        ArrayList<Vuelta> sospechosa = new ArrayList<Vuelta>();
        ArrayList<ArrayList<Vuelta>> vueltasSospechosas = new ArrayList<ArrayList<Vuelta>>();
        ArrayList<String> nombrePilotos = new ArrayList<String>();
        Connection conn = null;
        String extStore = System.getenv("EXTERNAL_STORAGE");
        File f_exts = new File(extStore + "/BD/pclc_505_MORATROS"+
                year.substring(2,year.length())+".mdb");
        try {
            Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
            //coger el número maximo de vueltas
            conn = DriverManager.getConnection(
                    "jdbc:ucanaccess://" + f_exts.getAbsolutePath()+
                            ";keepMirror="+extStore + "/BD/"+year);

            Statement s = conn.createStatement();
            //SOSECHOSAS
            ResultSet rs = s.executeQuery("SELECT Lap,DriverID,LapTime" +
                    " FROM RaceHistoryLap " +
                    "WHERE RaceID =" + idRace + " "+
                    "AND BestLap = 3 ORDER BY DriverID,Lap;");
            while (rs.next()) {
                if(! nombrePilotos.contains(rs.getString(2))){
                    //cambio de piloto y ya tengo uno en la lista, agrego tiempos y borro anteriores

                    if(nombrePilotos.size()>0){
                        vueltasSospechosas.add(sospechosa);
                        sospechosa=new ArrayList<Vuelta>();
                    }
                    nombrePilotos.add(rs.getString(2));
                }
                //AGREGO TIEMPOS
                Vuelta v = new Vuelta(Double.parseDouble(rs.getString(1)),
                        rs.getString(2),
                        Double.parseDouble(rs.getString(3)));
                sospechosa.add(v);
            }
            vueltasSospechosas.add(sospechosa);
            conn.close();
        } catch (SQLException e) {
```

```
            e.printStackTrace();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }
        return vueltasSospechosas;
    }
```

Ilustración 38. Función querySospechosas

Aquí lo que se está buscando es totalmente lo contrario que son las vueltas extrañamente lentas que ha realizado el piloto, para ello ejecutamos una consulta SQL donde buscamos las vueltas marcadas con el indicador BestLap = 3 y se forma al igual que en la anterior una lista con las vueltas.

queryPos

```
public ArrayList<ArrayList<Vuelta>> queryPos(int idRace , String year){
    ArrayList<Vuelta> posicion = new ArrayList<Vuelta>();
    ArrayList<ArrayList<Vuelta>> posiciones = new ArrayList<ArrayList<Vuelta>>();
    ArrayList<String> nombrePilotos = new ArrayList<String>();

    Connection conn = null;
    String extStore = System.getenv("EXTERNAL_STORAGE");
    File f_exts = new File(extStore + "/BD/pclc_505_MORATROS"+
            year.substring(2,year.length())+".mdb");
    try {
        Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
        //coger el número maximo de vueltas
        conn = DriverManager.getConnection(
                "jdbc:ucanaccess://" + f_exts.getAbsolutePath()+
                        ";keepMirror="+extStore + "/BD/"+year);
        Statement s = conn.createStatement();
        //POSICIONES
        ResultSet rs = s.executeQuery("SELECT Lap,DriverID,Position " +
                "FROM RaceHistoryLap " +
                "WHERE RaceID = " + idRace + " ORDER BY DriverID,Lap;");
        while (rs.next()) {
            if(! nombrePilotos.contains(rs.getString(2))){
                //cambio de piloto y ya tengo uno en la lista, agrego tiempos y borro anteriores
                if(nombrePilotos.size()>0){
                    posiciones.add(posicion);
                    posicion=new ArrayList<Vuelta>();
                }
                nombrePilotos.add(rs.getString(2));
            }
            //AGREGO TIEMPOS
            Vuelta v = new Vuelta(Double.parseDouble(rs.getString(1)),
                    rs.getString(2),
                    Double.parseDouble(rs.getString(3)));
            posicion.add(v);
        }
        posiciones.add(posicion);
        conn.close();
    } catch (SQLException e) {
        e.printStackTrace();
    } catch (ClassNotFoundException e) {
        e.printStackTrace();
    }
    return posiciones;
}
```

Ilustración 39. función queryPos

Con esta función lo que se quiere obtener es la posición de cada piloto en cada vuelta para de esta forma poder formar el grafico de las posiciones para ello se consulta la información de la carrera ordenada por vueltas para posteriormente ir obteniendo la información de cada piloto en cada vuelta de la carrera.

queryGas

```
1.  public ArrayList<ArrayList<Vuelta>> queryGas(int idRace , String year){
2.      ArrayList<Vuelta> repos = new ArrayList<Vuelta>();
3.      ArrayList<ArrayList<Vuelta>> repostajes = new ArrayList<ArrayList<Vuelta>>();
4.      ArrayList<String> nombrePilotos = new ArrayList<String>();
5.      Connection conn = null;
6.      String extStore = System.getenv("EXTERNAL_STORAGE");
7.      File f_exts = new File(extStore + "/BD/pclc_505_MORATROS"+
8.              year.substring(2,year.length())+".mdb");
9.      try {
10.         Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
11.         conn = DriverManager.getConnection(
12.                 "jdbc:ucanaccess://" + f_exts.getAbsolutePath()+
13.                         ";keepMirror="+extStore + "/BD/"+year);
14.
15.         Statement s = conn.createStatement();
16.         ResultSet rs = s.executeQuery("SELECT DISTINCT DriverID " +
17.                 "FROM RaceHistoryLap " +
18.                 "WHERE  RaceId=" + idRace +";");
19.         while (rs.next()) {
20.             nombrePilotos.add(rs.getString(1));
21.         }
22.         for(String piloto : nombrePilotos){
23.             //número de paradas
24.             repos=new ArrayList<Vuelta>();
25.             int stops=0;
26.             rs = s.executeQuery("SELECT MAX( PitStopNo ) " +
27.                     "FROM RaceHistoryLap " +
28.                     "WHERE  RaceId=" + idRace + " AND DriverID = '" + piloto + "';");
29.             while (rs.next()) {
30.                 stops = rs.getInt(1);
31.             }
32.             //registramos paradas
33.             for(int i=1;i<=stops;i++){
34.                 int lapStop=0;
35.                 double gasStop=0;
36.                 double gasExit=0;
37.                 rs = s.executeQuery("SELECT MIN( Lap ) "+
38.                         "FROM RaceHistoryLap "+
39.                         "WHERE  RaceId="+ idRace+" AND DriverID = '"+piloto
40.                         +"' AND PitStopNo ="+i+";");
41.                 while (rs.next()) {
42.                     lapStop = rs.getInt(1);
43.                 }
44.                 rs = s.executeQuery("SELECT FuelUsed,FuelLeft "+
45.                         "FROM RaceHistoryLap "+
46.                         "WHERE  RaceId="+ idRace+" AND DriverID = '"+piloto
47.                         +"' AND Lap ="+lapStop+";");
48.                 while (rs.next()) {
49.                     gasExit = rs.getDouble(1)+rs.getDouble(2);
50.                 }
51.                 Vuelta salida = new Vuelta((double) lapStop,piloto,gasExit);
52.                 rs = s.executeQuery("SELECT FuelUsed,FuelLeft "+
53.                         "FROM RaceHistoryLap "+
54.                         "WHERE  RaceId="+ idRace+" AND DriverID = '"+piloto
55.                         +"' AND Lap ="+(lapStop-1)+";");
56.                 while (rs.next()) {
57.                     gasStop = rs.getDouble(1)+rs.getDouble(2);
58.                 }
59.                 Vuelta entrada = new Vuelta((double) lapStop-1,piloto,gasStop);
60.                 repos.add(salida);
61.                 repos.add(entrada);
62.             }
63.             repostajes.add(repos);
64.             repos=new ArrayList<Vuelta>();
65.         }
66.         conn.close();
67.     } catch (SQLException e) {
68.         e.printStackTrace();
69.     } catch (ClassNotFoundException e) {
70.         e.printStackTrace();
71.     }
72.     return repostajes;
73. }
```

Ilustración 40. Función queryGas

Esta función es utilizada para rellenar la gráfica de repostajes para ello se tiene que conocer el número de repostajes que ha realizado cada piloto para ello se busca el número máximo que posee cada piloto en el campo pitStopNo posteriormente sabiendo el número de pit stop que ha realizado se busca la vuelta de salida de cada pit stop que estará marcada en el campo pitStopNo con el número de la parada de esta forma se conoce la vuelta en la que salió de boxes con el depósito recién cargado, ahora solo queda buscar la vuelta anterior a la parada para conocer el nivel de gasolina que tenía antes de realizar la parada y con esto ya se tiene toda la información necesaria para mostrar la gráfica de repostajes.

queryPilotos

```
1.  public ArrayList<String> queryPilotos(int idRace, String year){
2.      ArrayList<String> nombrePilotos = new ArrayList<String>();
3.      Connection conn = null;
4.      String extStore = System.getenv("EXTERNAL_STORAGE");
5.      File f_exts = new File(extStore + "/BD/pclc_505_MORATROS"
6.              +year.substring(2,year.length())+".mdb");
7.      try {
8.          Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
9.          //coger el número maximo de vueltas
10.         conn = DriverManager.getConnection(
11.                 "jdbc:ucanaccess://" + f_exts.getAbsolutePath()+
12.                         ";keepMirror="+extStore + "/BD/"+year);
13.         Statement s = conn.createStatement();
14.         ResultSet rs = s.executeQuery("SELECT DISTINCT DriverID " +
15.                 "FROM RaceHistoryLap " +
16.                 "WHERE  RaceId=" + idRace +";");
17.         while (rs.next()) {
18.             nombrePilotos.add(rs.getString(1));
19.         }
20.         conn.close();
21.     } catch (SQLException e) {
22.         e.printStackTrace();
23.     } catch (ClassNotFoundException e) {
24.         e.printStackTrace();
25.     }
26.     return nombrePilotos;
27. }
```

Ilustración 41. Función queryPilotos

Las próximas dos funciones son las utilizadas para mostrar lo datos del visor, en la primera de ellas se obtiene el tiempo que dura la carrera mientras que en la segunda se busca la información de la carrera en un instante de tiempo determinado.

queryTime

```
1.  public double queryTime(int idRace,String year){
2.      double tiempo = 0;
3.      Connection conn = null;
4.      String extStore = System.getenv("EXTERNAL_STORAGE");
5.      File f_exts = new File(extStore + "/BD/pclc_505_MORATROS"+
6.              year.substring(2,year.length())+".mdb");
7.      try {
8.          Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
9.          //coger el número maximo de vueltas
10.         conn = DriverManager.getConnection(
11.                 "jdbc:ucanaccess://" + f_exts.getAbsolutePath()+
12.                         ";keepMirror="+extStore + "/BD/"+year);
13.         Statement s = conn.createStatement();
14.         ResultSet rs = s.executeQuery("SELECT MAX(DriverRaceTime) "+
15.                 "FROM RaceHistoryLap "+
16.                 "WHERE RaceID ="+idRace+";"
17.         );
18.         int i = 0;
19.         while (rs.next()) {
20.             tiempo=rs.getFloat(1);
21.         }
22.         conn.close();
```

```
23.      } catch (SQLException e) {
24.          e.printStackTrace();
25.      } catch (ClassNotFoundException e) {
26.          e.printStackTrace();
27.      }
28.      return tiempo;
29.  }
```

Ilustración 42. Función queryTime

queryRace

```
1.   public  ArrayList<Vuelta> queryRace(int idRace, String year,
2.                                       ArrayList<String> nombrePilotos,
3.                                       double instant){
4.
5.       ArrayList<Vuelta> vueltas = new ArrayList<Vuelta>();
6.       Connection conn = null;
7.       String extStore = System.getenv("EXTERNAL_STORAGE");
8.       File f_exts = new File(extStore + "/BD/pclc_505_MORATROS"
9.               +year.substring(2,year.length())+".mdb");
10.      try {
11.          Class.forName("net.ucanaccess.jdbc.UcanaccessDriver");
12.          //coger el número maximo de vueltas
13.          conn = DriverManager.getConnection(
14.                  "jdbc:ucanaccess://" + f_exts.getAbsolutePath()+
15.                          ";keepMirror="+extStore + "/BD/"+year);
16.          Statement s = conn.createStatement();
17.          ArrayList<DataPoint> series = new ArrayList<DataPoint>();
18.          ResultSet rs = s.executeQuery("SELECT Position,CarId,DriverID" +
19.                  ",Lap,LapTime,pitStopNo,FuelLeft,LaneID " +
20.                  "FROM RaceHistoryLap "+
21.                  "WHERE RaceID ="+idRace+" AND DriverRaceTime <+"+instant+
22.                  " ORDER BY DriverRaceTime DESC;");
23.          int i = 0;
24.          vueltas=new ArrayList<Vuelta>();
25.          while (rs.next() && i < nombrePilotos.size()) {
26.              //Procesar Resultados
27.              //AGREGO TIEMPOS
28.              Vuelta v = new Vuelta(rs.getInt(1),
29.                      rs.getString(2),
30.                      rs.getString(3),
31.                      rs.getDouble(4),
32.                      rs.getDouble(5),
33.                      0.0,rs.getInt(6),
34.                      rs.getDouble(7),
35.                      rs.getInt(8));
36.              vueltas.add(v);
37.              i++;
38.          }
39.          conn.close();
40.      } catch (SQLException e) {
41.          e.printStackTrace();
42.      } catch (ClassNotFoundException e) {
43.          e.printStackTrace();
44.      }
45.      Log.i("VUELTAS",vueltas.toString());
46.      //pDialog.dismiss();
47.      return vueltas;
48.  }
```

Ilustración 43. Función queryRace

6. Conclusiones y trabajos futuros

Con el desarrollo del presente proyecto hemos aprendido a integrar la información procedente de diversas fuentes en un solo sitio para posteriormente tratarla y mostrar aquella que nos es relevante, esta información es la procedente de las bases de datos del PcLapCounter, los archivos XML procedentes de CarreraSlot.

Hemos aplicado los conocimientos adquiridos durante la carrera para la realización del proyecto donde hemos realizado diferentes fases para finalmente obtener una aplicación funcional.

Empezando por el análisis de la información a tratar, para a continuación realizar un prototipo que durante varios ciclos de test y de comunicación con José Ángel Carsí Cubel, se fue modificando hasta obtener la aplicación que estamos hablando en este proyecto.

En conclusión, ha sido un proyecto que me ha gustado realizarlo, por la afición que presento hacia este hobby.

En un futuro se podría modificar la aplicación para crear otro proceso de carga de competiciones ya que la introducción de una URL puede ser larga y tediosa, así como difícil de memorizar, quizá se podría hace un asistente que mostrase todas las competiciones existentes y posteriormente el usuario solo tuviese que elegir aquellas que quiera agregar a la lista inicial para acceder a ellas de forma rápida.

7. Bibliografía

Documentación Ucanacces:

http://ucanaccess.sourceforge.net/site.html

Documentación Android:

https://developer.android.com/docs/

Documentacion GraphView

http://www.android-graphview.org/

Manual de usuario CarreraSlot

8. Apéndice

8.1. Archivos CarreraSlot

Se va a describir toda la información contenida en los archivos procedentes del programa CarreraSlot, con estos archivos se obtiene la información necesaria para crear las tablas que se muestran en la aplicación, así como la lista de competiciones que se muestran, estos documentos son procesados en nuestra capa de persistencia.

En este punto se encuentran diferentes tipos de archivos aquellos que tiene la extensión .metaComp, los que tiene la extensión .comp y archivos de configuración para la competición.

8.1.1. Localización de los archivos

Los datos procedentes de CarreraSlot tienen una estructura de directorios definida donde se encuentran organizados los archivos necesarios para la aplicación, la estructura de archivos es la siguiente.

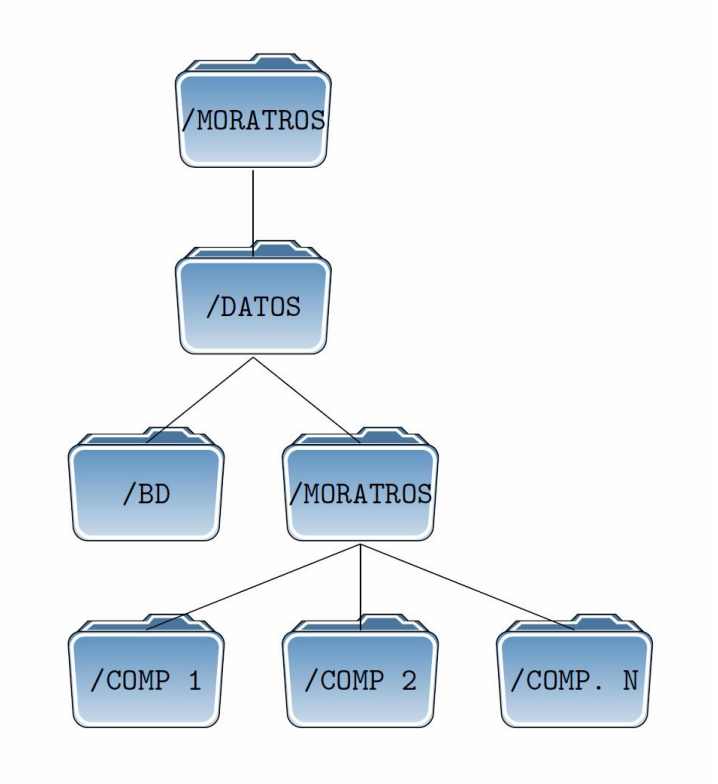

Ilustración 44. Estructura archivos CarreraSlot

A continuación, se explica de forma detallada los archivos que tiene cada directorio en su interior.

En el directorio BD se encuentran las bases de datos procedentes del programa PCLapCounter que son necesarias para realizar el visor de carreras de nuestra aplicación.

En el interior de la segunda carpeta MORATROS se encuentran una serie de directorios que se corresponde a cada una de las competiciones disputadas a si mismo también están diferentes archivos de metacompeticiones.

Ya en el interior de las competiciones se ve un archivo de la competición y los archivos de configuración para CarreraSlot.

Los diferentes archivos serán explicados a continuación de forma más detallada.

8.1.2. .metaComp

Estos archivos contienen la información que forma una metacompetición que como se definió al inicio de este documento es un agregado de competiciones y metacompeticiones.

Los archivos presentan la siguiente apariencia:

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <MetaCompeticion xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
3.    <competiciones>
4.      <string>.\2015 - COPA GR5\2015 - COPA GR5.comp</string>
5.      <string>.\2015 - COPA F1\2015 - COPA F1.comp</string>
6.      <string>.\2015 - COPA LMP\2015 - COPA LMP.comp</string>
7.      <string>.\2015 - COPA GPL\2015 - COPA GPL.comp</string>
8.      <string>.\2015 - COPA GT\2015 - COPA GT.comp</string>
9.      <string>.\2015 - COPA GRUPO C\2015 - COPA GRUPO C.comp</string>
10.   </competiciones>
11.   <nombre>MORATROS 2015</nombre>
12. </MetaCompeticion>
```

Ilustración 45. Archivo .metaComp

Como se observa los archivos están en formato XML y comienzan con la etiqueta MetaCompetición indicando el tipo de información que tienen , a continuación dentro de la etiqueta competiciones se encuentran las rutas para acceder a las competiciones que forman dicha metacompetición, estas rutas son rutas relativas ya que los archivos viene organizados y para terminar se encuentra la etiqueta nombre donde está el nombre que adquiere esta metacompetición en este caso es la metacompetición MORATROS 2015 que engloba todas las competiciones realizadas en el club en el año 2015.

8.1.3. .Comp

Estos archivos contienen toda la información relevante sobre una competición por lo que se encuentra una gran cantidad de datos para procesar y entender, como se explicó al inicio de este documento las competiciones están formadas por un número indeterminado de kedadas por lo que en este documento está la información agrupada por kedadas, una vez entendida esta jerarquía es fácil entender el contenido de estos archivos.

A continuación, se va mostrar el código de una competición, pero únicamente la primera kedada para proceder a su explicación.

```
<?xml version="1.0" encoding="utf-8"?>
<Competicion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <nombre>2015 - COPA GR5</nombre>
  <modoCampeonato>false</modoCampeonato>
  <modofinales>false</modofinales>
  <modoBomilcar>false</modoBomilcar>
  <modoResitenciaDiscont>false</modoResitenciaDiscont>
  <dias>
    <Kedada>
      <fecha>2015-01-09T00:00:00</fecha>
      <nKedada>43</nKedada>
```

Ilustración 46. Archivo .comp

Al igual que los archivos .metaComp los archivos .comp inician con la etiqueta competición, seguida del nombre donde está definido el nombre de la competición y cuatro etiquetas que con la configuración de la competición que son modoCampeonato, modofinales, modoBomilcar, modoResistenciaDiscont.

A continuación de la etiqueta días……..

Seguida de la etiqueta kedada donde apartir de aquí se dispone de toda la información relevante a la kedada.

```
    <Kedada>
      <fecha>2015-01-09T00:00:00</fecha>
      <nKedada>43</nKedada>
      <corredores>
        <CorreCon>
          <piloto>PEP0N</piloto>
          <coche>Ferrari</coche>
          <dorsal>8</dorsal>
        </CorreCon>
        <CorreCon>
          <piloto>ALBERT</piloto>
          <coche>Ferrari</coche>
          <dorsal>1</dorsal>
        </CorreCon>
        <CorreCon>
          <piloto>JORGE</piloto>
          <coche>.</coche>
          <dorsal>4</dorsal>
        </CorreCon>
        <CorreCon>
          <piloto>RUBENS</piloto>
          <coche>Capri</coche>
          <dorsal>9</dorsal>
        </CorreCon>
        <CorreCon>
          <piloto>VICENTE RC</piloto>
          <coche>BMW</coche>
          <dorsal>3</dorsal>
```

```
        </CorreCon>
        <CorreCon>
          <piloto>TONI</piloto>
          <coche>.</coche>
          <dorsal>6</dorsal>
        </CorreCon>
        <CorreCon>
          <piloto>SERITOBOM</piloto>
          <coche>.</coche>
          <dorsal>2</dorsal>
        </CorreCon>
      </corredores>
```

Ilustración 47 Archivo kedada 1

Las kedadas comienzan con la fecha de la kedada y el número de la kedada y posteriormente una lista de todos los corredores que participan en la kedada, los corredores están definidos entre las etiquetas correCon donde se muestra el nombre del piloto el coche utilizado en la kedada y el dorsal.

Una vez terminados los corredores se presenta la información de las diferentes etapas que forman la kedada, que suelen estar compuestas por una o varias poles, seguida de las semifinales y para terminar las finales.

Aquí se puede ver los datos sobre la pole.

```
  <poles>
        <Carrera>
          <tipo>1</tipo>
          <nombre>Pole A</nombre>
          <detalles>
            <Detalle>
              <tiempo>10.2</tiempo>
              <promedio>0</promedio>
              <vueltas>0</vueltas>
              <penalizacion>0</penalizacion>
              <corredor>
                <piloto>PEP0N</piloto>
                <coche>Ferrari</coche>
                <dorsal>8</dorsal>
              </corredor>
              <gasolina>51.16</gasolina>
              <instanteVRC>79.86</instanteVRC>
            </Detalle>
            <Detalle>
              <tiempo>10.63</tiempo>
              <promedio>0</promedio>
              <vueltas>0</vueltas>
              <penalizacion>0</penalizacion>
              <corredor>
                <piloto>ALBERT</piloto>
                <coche>Ferrari</coche>
                <dorsal>1</dorsal>
              </corredor>
              <gasolina>53.68</gasolina>
              <instanteVRC>145.7</instanteVRC>
            </Detalle>
            <Detalle>
              <tiempo>10.59</tiempo>
              <promedio>0</promedio>
              <vueltas>0</vueltas>
              <penalizacion>0</penalizacion>
              <corredor>
                <piloto>JORGE</piloto>
                <coche>.</coche>
                <dorsal>4</dorsal>
              </corredor>
              <gasolina>57.31</gasolina>
              <instanteVRC>63.04</instanteVRC>
            </Detalle>
            <Detalle>
              <tiempo>10.59</tiempo>
              <promedio>0</promedio>
```

```
            <vueltas>0</vueltas>
            <penalizacion>0</penalizacion>
            <corredor>
              <piloto>RUBENS</piloto>
              <coche>Capri</coche>
              <dorsal>9</dorsal>
            </corredor>
            <gasolina>45.94</gasolina>
            <instanteVRC>168.51</instanteVRC>
          </Detalle>
        </detalles>
        <IDCarreraDelPCLap>221</IDCarreraDelPCLap>
      </Carrera>
      <Carrera>
        <tipo>1</tipo>
        <nombre>Pole B</nombre>
        <detalles>
          <Detalle>
            <tiempo>10.44</tiempo>
            <promedio>0</promedio>
            <vueltas>0</vueltas>
            <penalizacion>0</penalizacion>
            <corredor>
              <piloto>VICENTE RC</piloto>
              <coche>BMW</coche>
              <dorsal>3</dorsal>
            </corredor>
            <gasolina>40.55</gasolina>
            <instanteVRC>182.82</instanteVRC>
          </Detalle>
          <Detalle>
            <tiempo>10.55</tiempo>
            <promedio>0</promedio>
            <vueltas>0</vueltas>
            <penalizacion>0</penalizacion>
            <corredor>
              <piloto>TONI</piloto>
              <coche>.</coche>
              <dorsal>6</dorsal>
            </corredor>
            <gasolina>39.94</gasolina>
            <instanteVRC>187.64</instanteVRC>
          </Detalle>
          <Detalle>
            <tiempo>10.99</tiempo>
            <promedio>0</promedio>
            <vueltas>0</vueltas>
            <penalizacion>0</penalizacion>
            <corredor>
              <piloto>SERITOBOM</piloto>
              <coche>.</coche>
              <dorsal>2</dorsal>
            </corredor>
            <gasolina>46.28</gasolina>
            <instanteVRC>190.58</instanteVRC>
          </Detalle>
        </detalles>
        <IDCarreraDelPCLap>220</IDCarreraDelPCLap>
      </Carrera>
    </poles>
```

Ilustración 48. Archivo kedada 2

Como se ve en las poles están comprendidas entre las etiquetas <Poles>, en este caso se han disputado dos poles , Pole A y Pole B, la información relevante a cada pole está comprendida entre las etiquetas <carrera> donde se puede ver el tipo y el nombre de la pole, seguidamente se muestran los detalles de la pole que no es más que la información de cada piloto que ha participado, esta información se encuentra entre las etiquetas <Detalle> donde se pueden ver datos como: tiempo, el tiempo promedio, el número de vueltas y si tiene alguna penalización.

En la última etiqueta se encuentra el IDCarreraDelPCLap que contiene el id de esta carrera en la base de datos que proviene del PCLapCounter.

Al igual que con las poles se presenta la información de las semifinales y finales que forman la kedada.

Una vez visto la información de cada una de las partes de la competición se puede observar un resumen de lo que ha sido la kedada

```
        <detallePole>
          <tiempo>10.2</tiempo>
          <promedio>0</promedio>
          <vueltas>0</vueltas>
          <penalizacion>0</penalizacion>
          <corredor>
            <piloto>PEP0N</piloto>
            <coche>Ferrari</coche>
            <dorsal>8</dorsal>
          </corredor>
          <gasolina>51.16</gasolina>
          <instanteVRC>79.86</instanteVRC>
        </detallePole>
        <detalleCarrera>
          <tiempo>10.29</tiempo>
          <promedio>11.417</promedio>
          <vueltas>50</vueltas>
          <penalizacion>0</penalizacion>
          <corredor>
            <piloto>PEP0N</piloto>
            <coche>Ferrari</coche>
            <dorsal>8</dorsal>
          </corredor>
          <gasolina>13.88</gasolina>
          <instanteVRC>572.59</instanteVRC>
        </detalleCarrera>
        <detalleVRKedada>
          <tiempo>10.23</tiempo>
          <promedio>11.36</promedio>
          <vueltas>50</vueltas>
          <penalizacion>0</penalizacion>
          <corredor>
            <piloto>PEP0N</piloto>
            <coche>Ferrari</coche>
            <dorsal>8</dorsal>
          </corredor>
          <gasolina>5.58</gasolina>
          <instanteVRC>323.89</instanteVRC>
        </detalleVRKedada>
        <anulada>false</anulada>
```

Ilustración 49. Archivo kedada 3

En este resumen tiene diferentes secciones, el detallePole donde está la información del poleman de la kedada, mostrando el tiempo de pole y la información del corredor que la ha realizado,

Al igual que el detallePole se encuentra el detalleCarrera donde se muestra la información del ganador de la kedada, pudiendo ver su vuelta rápida, su vuelta media, el número de vueltas que completo y si tiene alguna penalización, así como la información relevante del corredor ganador.

Para terminar, se muestra el detalle de la vuelta rápida de la kedada (detalleVRKedada) donde se puede ver la información sobre la vuelta rápida de la kedada, asi como el instante en el que se realizó.

Cerrando la kedada está la etiqueta anulada que indica si la kedada ha sido anulada o no para tenerla en cuenta a la hora de generar los datos.

Como se puede observar por el ejemplo la cantidad de información a procesar es grande por lo que se tiene que tener claro cómo se estructura una competición, al final de este documento podremos encontrar un archivo de una competición al completo.

8.1.4. Archivos de configuración

Para el correcto funcionamiento del programa CarreraSlot también existen diferentes archivos de configuración donde se configura la información de las competiciones como pueden ser los puntos a repartir o si se elimina el peor resultado, etc…

configCarreraSlot

```
<?xml version="1.0" encoding="utf-8"?>
<Configuracion xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <puntosPorPole>1</puntosPorPole>
  <puntosPorVRC>0</puntosPorVRC>
  <pilotosPorCarrera>8</pilotosPorCarrera>
  <pasarPorVueltas>false</pasarPorVueltas>
  <ordenarFinalesPorVueltasEnSemi xsi:type="xsd:boolean">false</ordenarFinalesPorVueltasEnSemi>
  <kedadasAntesDeAplicarPeores>1</kedadasAntesDeAplicarPeores>
  <resultadosAQuitar>1</resultadosAQuitar>
  <rutaPiloteIniPCLAP />
  <rutaCarIniPCLAP />
  <ultimosAlParedonSemis>true</ultimosAlParedonSemis>
  <IGUBoloko>false</IGUBoloko>
  <metaCompComoCompCombinada>true</metaCompComoCompCombinada>
  <rutaPCLAP>.\..\..\..\..\Moratros\DATOS\BD\\pclc_505_MORATROS16.mdb</rutaPCLAP>
  <rutaMAARDS />
  <rutaUR30 />
  <rutaExportPCLAP>C:\Pc Lap Counter\AAAAAAAAA.txt</rutaExportPCLAP>
  <promedioPCLAP>false</promedioPCLAP>
</Configuracion>
```

Ilustración 50. Archivo configCarreraSlot

Aquí se observa el contenido del archivo configCarreraSlot donde se muestra información sobre la configuración de la competición, la información más importante es:

- puntosPorPole donde se establecen los puntos que se dan al piloto que realiza la pole.
- puntosPorVRC aqui se definen los puntos que gana el piloto que hace la vuelta rápida de carrera
- pilotosPorCarrera es el número de pilotos que puede participar en cada carrera
- resultadosAquitar son el número de resultados que se puede quitar cada piloto para generar la general en este caso solo se elimina el peor resultado.

Además del archivo configCarreraSlot existe un archivo de texto con las puntuaciones a repartir en orden ascendente según el resultado.

8.2. Base de datos de PCLapCounter

Las bases de datos con la información de las carreras proceden del programa PCLapCounter que es un gestor de carreras de SLOT que nos permite configurar y tomar diferentes datos de una carrera.

La base de datos presenta la siguiente estructura donde se observa que toda la información está relacionada con la tabla RaceHistory que contiene las carreras disputadas.

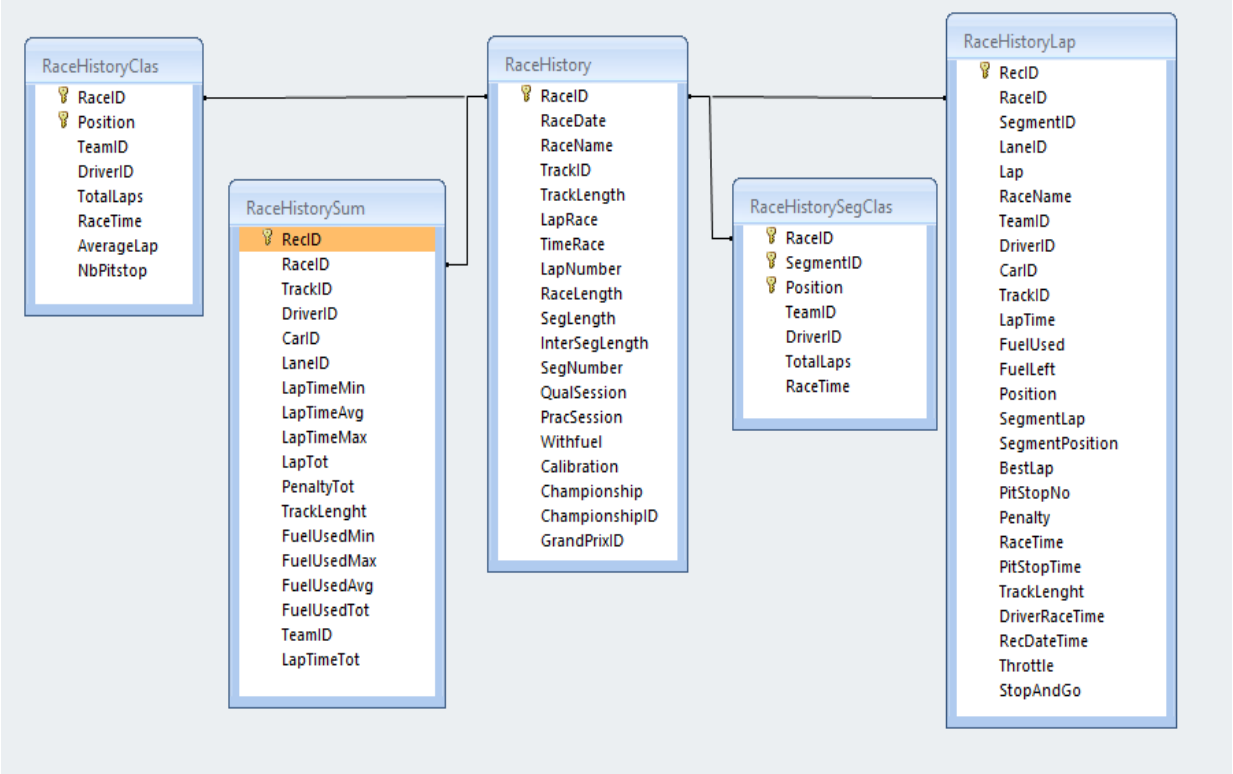

Ilustración 51. Esquema Base de datos PcLapCounter

A continuación, se muestra tabla por tabla una breve explicación ya vista en la memoria y una relación de los campos contenidos en cada tabla, el tipo de campo y una descripción del campo esta información se ha sido extraída de Microsoft Access.

8.1.1. RaceHistory:

En esta tabla se observa la configuración de cada carrera como por ejemplo la longitud, si es una carrera programada a vueltas o a tiempo, si la carrera tiene clasificación, entrenamientos, si se consume gasolina, etc...

Detalle de los campos:

Nombre del campo	Tipo de datos	Descripción
RaceID	Número	Identificación de la carrera
RaceDate	Fecha/hora	Fecha y hora de la carrera
RaceName	Texto	Nombre de la carrera
TrackID	Texto	Nombre del circuito
TrackLength	Número	Longitud de la carrera
LapRace	Si/no	Vueltas de la carrera
TimeRace	Si/no	Duración de la carrera
LapNumber	Número	En caso de establecer la carrera por vueltas, el número de vueltas de duración de la carrera
RaceLenght	Texto	En caso de establecer la carrera por tiempo, el tiempo de duración de la carrera en segundos
SegLenght	Texto	En caso de carrera por tiempo, duración de cada segmento
InterSegLenght	Texto	En caso de carrera por tiempo, tiempo entre segmentos
SegNumber	Número	Numero de segmentos
QualSession	Si/no	Sesión de clasificación
PracSession	Si/no	Sesión de practicas
Withfuel	Si/no	Carrera con gestión de combustible
Calibration	Si/no	Configuraión para el consumo de combustible
Championship	Si/no	Carrera puntuable para el campeonato
ChampionshipID	Texto	Nombre del campeonato
GrandPrixID	Texto	Nombre de la carrera

8.1.2. RaceHistoryClas:

Aquí se ve los datos generales de cada una de las poles, semifinales y finales que forman las kedadas observando el id de la sesión, el piloto, la posición, el número total de vueltas, y el tiempo empleado en realizarlas.

Detalle de los campos:

Nombre del campo	Tipo de datos	Descripcion
RaceID	Número	Identificación de la carrera
Position	Número	Posición del piloto
TeamID	Texto	Nombre del equipo
DriverID	Texto	Nombre del piloto

TotalLaps	Número	Vueltas totales
RaceTime	Número	Tiempo total
AverageLap	Número	Media tiempo por vuelta
NbPitStops	Número	Número de pitstop

8.1.3. RaceHistoryLap:

En esta tabla es donde está la mayoría de la información ya que contiene un registro vuelta a vuelta de cada una de las sesiones que forman la kedada permitiendo obtener todos los datos necesarios para posteriormente generar los gráficos que se han visto anteriormente en el análisis.

Detalle de los campos:

Nombre del campo	Tipo de datos	Descripción
RaceID	Auto numérico	Identificación de la carrera
SegmentID	Número	Número de segmento
LaneId	Número	Carril para analógico, id de tarjeta para digital
Lap	Número	Número de vuelta
RaceName	Número	Nombre de la carrera
TeamID	Texto	Nombre del equipo
DriverID	Texto	Nombre del piloto
CarID	Texto	Nombre del coche
TrackID	Texto	Nombre del circuito
LapTime	Número	Tiempo por vuelta
FuelUsed	Número	Gasolina usa por vuelta
FuelLeft	Número	Gasolina sobrante
Position	Número	Posición del piloto en la carrera
SegmentLap	Número	Vuelta en el segmento de la carrera
SegmentPosition	Número	Posición del piloto en el segmento de la carrera
BestLap	Número	1=Yes , 0=No, 3=suspect lap time
PitStopNo	Número	Número de pitstop
Penality	Número	Número de vueltas de penalización
RaceTime	Número	Tiempo desde el inicio de la carrera
PitStopTime	Número	Duración del pitstop
TrackLenght	Número	Longitud de la carrera
DriverRaceTime	Número	Tiempo del piloto desde el inicio del segmento
RecDateTime	Fecha/Hora	Fecha y hora de la carrera
Throttle	Número	
StopAndGo	Número	

8.1.4. RaceHistorySegClas:

En esta tabla se observan los resultados de cada una de las carreras mostrando los pilotos que participaron, la posición en la que terminaron, el número de vueltas que dieron y la duración de la carrera para cada piloto

Detalle de los campos:

Nombre del campo	Tipo de datos	Descripcion
RaceID	Número	Identificación de la carrera
SegmentID	Número	Número de segmento
Position	Número	Posición del piloto
TeamID	Texto	Nombre del equipo
DriverID	Texto	Nombre del piloto
TotalLaps	Número	Vueltas totales de la carrera
RaceTime	Número	Tiempo de la carrera

8.1.5. RaceHistorySum:

Para finalizar en aquí se tiene un resumen por cada sesión por piloto, donde están el tiempo más rápido en dar una vuelta, el tiempo medio por vuelta de la sesión, el tiempo más lento en una vuelta, las vueltas totales, así como otros datos.

Detalle de los campos:

Nombre del campo	Tipo de datos	Descripcion
RecID	Auto numérico	Identificación de registro
RaceID	Número	Identificación de la carrera
TrackID	Texto	Nombre del circuito
DriverID	Texto	Nombre del piloto
CarID	Texto	Nombre del coche
LaneID	Número	Carril para analógico, id de tarjeta para digital
LapTimeMin	Número	Tiempo mínimo de vuelta
LapTimeAvg	Número	Tiempo medio de vuelta
LapTimeMax	Número	Tiempo máximo de vuelta
LapTot	Número	Número total de vueltas
PenalityTot	Número	Total de penalizaciones
TrackLenght	Número	Longitud total de la carrera
FuelUsedMin	Número	Cantidad mínima de gasolina gastada
FuelUsedMax	Número	Cantidad máxima de gasolina gastada
FuelUsedAvg	Número	Cantidad media de gasolina gastada
FuelUsedTot	Número	Cantidad total de gasolina gastada
TeamID	Texto	Nombre del equipo
LapTimeTot	Número	Tiempo total de carrera

8.3. Manual de usuario

En la siguiente sección se procede a ver el funcionamiento de la aplicación sobre un entorno de trabajo real explicando el funcionamiento de las diferentes opciones que se pueden realizar en la aplicación, así como el significado de la información que aparece en las diferentes pantallas.

Para comenzar se habla sobre la instalación de la aplicación, esta aplicación no se encuentra alojada en Play Store, que es la plataforma que tiene Google para la distribución y control de aplicaciones, por lo que para su instalación hay que descargar el instalador de la aplicación e instalarla en el terminal.

Una vez se tiene instalada la aplicación procedemos a su ejecución en el terminal donde podemos observar la siguiente escena.

Ilustración 52. Inicio aplicación

En esta pantalla se produce el proceso de carga de la aplicación cargando los datos introducidos y según los datos nos dirigirá al menú principal donde se puede seleccionar las competiciones ya introducidas o a una ventana de configuración para insertar URL's de competiciones para su posterior carga.

Como se parte de una instalación nueva la aplicación nos dirige a actividad de configuración donde se le solicita al usuario la introducción de direcciones URL que contengan archivos de competiciones válidos. La actividad para introducir las direcciones presenta la siguiente apariencia.

Ilustración 53. Inserción URL

Como se observa se dispone de un cuadro de texto donde hay que escribir las direcciones de los archivos de las competiciones deseadas, una vez escrita la direcciones presionando sobre el botón "Añadir" se añadirá nuestra dirección a la lista inferior, en este caso ya tiene cargadas tres

metacompeticiones, se puede interaccionar con esta lista seleccionando elementos que nos rellenaran el cuadro de texto con la URL seleccionada permitiéndonos corregirla o deslizar elementos de la lista para su eliminación. Para terminar, hay un botón para aceptar los cambios y proceder a la carga de estas direcciones redirigiendo a la siguiente actividad.

Ilustración 54. Selección URL

Aquí se observa una lista con todas las competiciones o metacompeticiones que se han introducido en el apartado de configuración, en esta actividad se puede seleccionar cualquier elemento de la lista cargando los datos de la competición o añadir más direcciones a las ya introducidas, para la introducción de más direcciones en la parte superior derecha se dispone de un menú en el que se encuentra la opción Añadir que nos llevara de vuelta a la actividad anterior.
Si por el contrario se selecciona una metacompetición se dirigirá a una actividad donde se mostrarán las competiciones que la forman, el número de kedadas que componen cada

competición y los tres primeros pilotos de cada competición con los puntos que han obtenido en ella, esto se ve en la siguiente imagen.

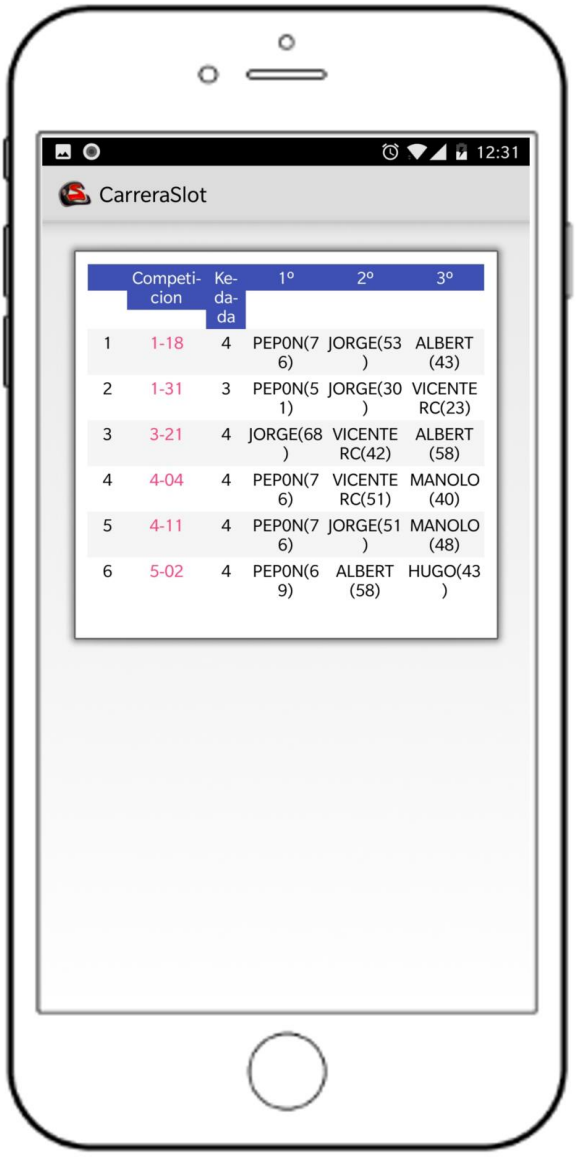

Ilustración 55. Vista metacompetición

Una vez aquí el usuario puede volver a la actividad anterior para seleccionar otra competición o mentacompetición o continuar profundizado en la metacompetición seleccionada viendo la información de cada una de las competiciones que la forman, para continuar profundizando y ver la información de las competiciones solamente hay que tocar sobre el nombre de cada competición (en color rosa), esta acción le llevara a una nueva actividad donde podrá ver los resultados de forma detallada de la competición seleccionada.

Una vez se ha accedido a la competición, se muestran diferentes tablas con información relevante de la competición se explican a continuación.

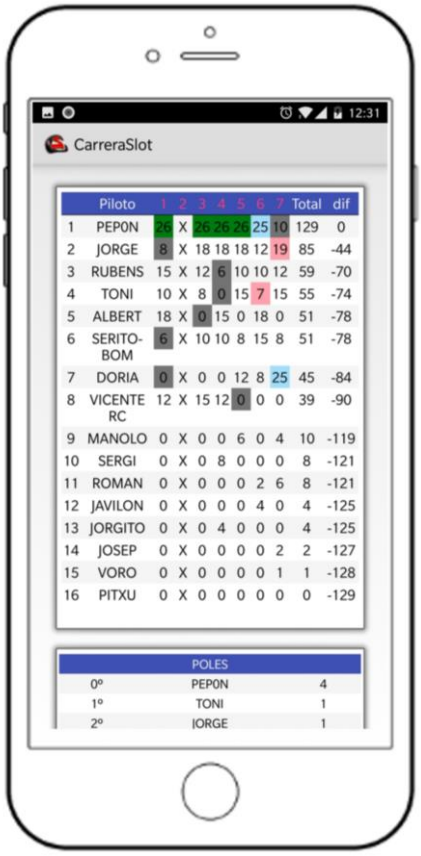

Ilustración 56. Vista competición 1

Nada más acceder a la información de la competición se encuentra una tabla con la clasificación de la competición donde se observa la posición de cada piloto en la general, el nombre del piloto, los puntos obtenidos en cada una de las kedadas que forman la competición, el total de puntos obtenido y la diferencia de puntos entre los demás pilotos y el primer clasificado, además se puede observar según un código de colores información extra sobre la clasificación.

Seguidamente se observa una tabla con información sobre las poles obtenidas en la competición donde se muestra el nombre de cada piloto y la cantidad de poles que ha obtenido en la competición.

Gran Chelem
Victoria y Superpole
Victoria
Superpole
Resultado descontado

Aquí está el significado de cada uno de los colores que aparecen en la tabla de la clasificación general de la competición.

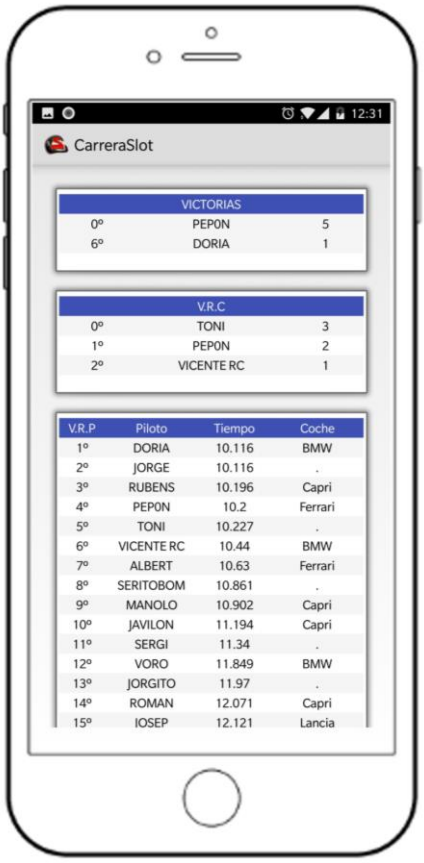

Ilustración 57. Vista competición 2

Deslizando la pantalla hay más tablas donde se tiene la información de victorias obtenidas en la competición y las vueltas rápidas de carrera.

Bajo estas tablas está una tabla donde se muestran las vueltas rápidas de la pole donde se ve la vuelta más rápida de todas las poles disputadas por cada piloto.

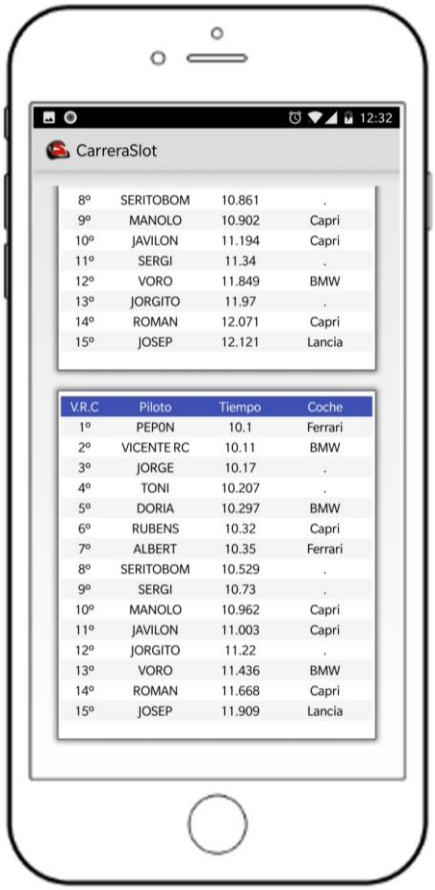

Ilustración 58. Vista competición 3

Para finalizar, una última tabla donde al igual que las vueltas rápidas de la pole se muestra la información sobre las vueltas rápidas de carrera, donde aparece la vuelta más rápida de cada piloto realizada durante las kedadas que forman la competición.

Una vez vista toda la información contenida en esta actividad se puede continuar profundizando y ver el detalle de cada kedada, para ello en la clasificación general se selecciona la kedada de la cabecera de la tabla simplemente tocando el número de la kedada que se quiere observar, al igual que las competiciones los números de las kedadas que se puede interaccionar están pintados de color rosa.

Al acceder a la kedada se vuelve a presentar una actividad con diversas tablas donde está la información de la kedada, mostrando la siguiente situación.

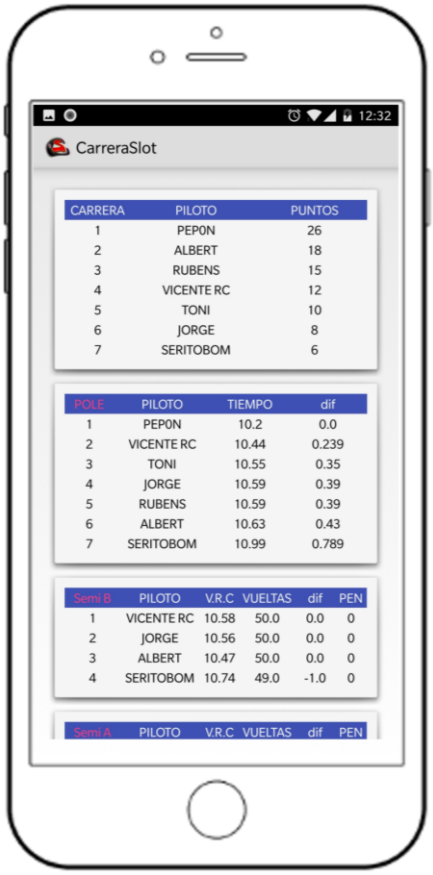

Ilustración 59. Vista kedada 1

En primer lugar, una tabla con el resultado de la kedada donde se muestra la posición que ocupa cada uno de los pilotos y los puntos obtenido.

A continuación, otra tabla con el resultado de la pole de la kedada en la cual están organizados los pilotos por vuelta rápida y se ve la posición que obtuvo cada uno así como el tiempo obtenido en la pole y la diferencia respecto al primero.

Una vez vistas estas dos tablas se muestran las tablas de las semifinales donde se puede ver el resultado obtenido en cada una de las semifinales

Ilustración 60. Vista kedada 2

Para finalizar tenemos las tablas de las finales se ve el resultado de cada una de las finales que se han disputado en la kedada.

Al igual que en las anteriores actividades se puede seleccionar la pole o cada una de las semifinales y finales para continuar viendo información más detalladas sobre estas siempre tocando sobre el nombre de lo que se quiera ver, según se seleccione una pole, una semifinal o una final se mostrara una ventana con diferentes opciones para continuar profundizando en los datos.

Ilustración 61. Menú poles

Al seleccionar la pole se muestra el siguiente menú donde se puede consultar una gráfica con las vueltas rápidas y el instante en el que se realizaron o una gráfica con las vueltas sospechosas

Ilustración 62. Menú semifinales y finales

Si por el contrario hemos elegido una semifinal o final se dispone de más opciones para visualizar datos además de los comentados anteriormente hay una gráfica donde se muestra como van variando las posiciones de los pilotos en el transcurso de la carrera, una gráfica con los repostajes mostrando la vuelta y la cantidad de gasolina que tenía al para y salir del box y para finalizar un visor donde puede ver la carrera con la información de cada piloto en cada instante de esta.

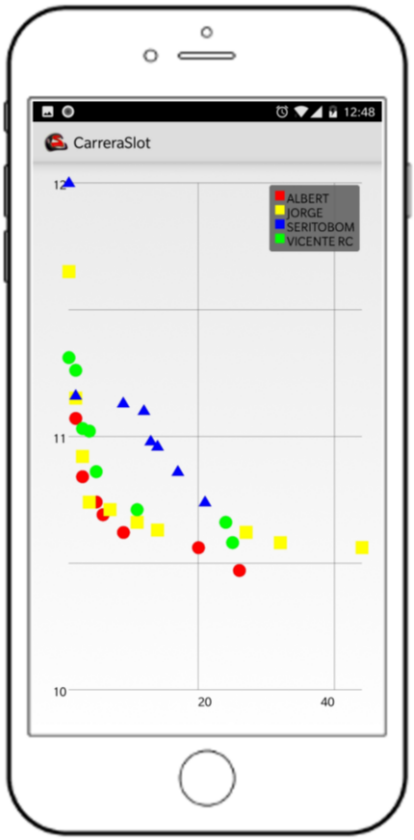

Ilustración 63. Visor vueltas rápidas

Vueltas rápidas:

En este grafico el usuario puede observar una gráfica donde se encuentran las vueltas rápidas realizadas por los pilotos que han participado en la pole, semifinal o final seleccionada, en la vista se puede observar una leyenda para ayudar a la comprensión de los datos, estos datos están distribuidos en el eje X la vuelta cuando se realizó la vuelta, en el eje Y el tiempo realizado en dicha vuelta.

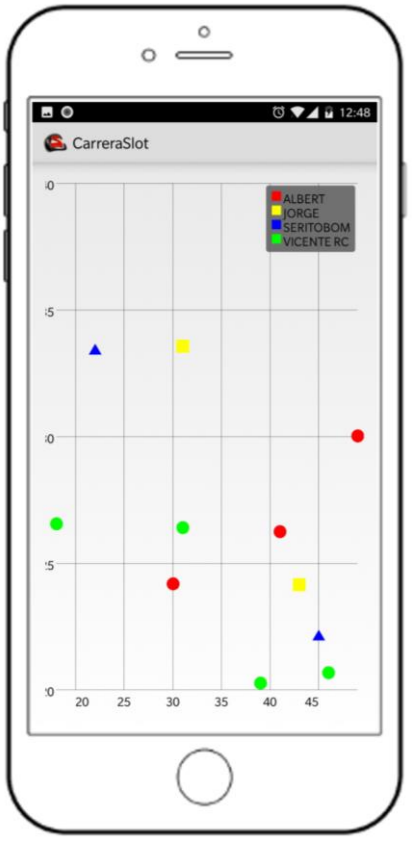

Ilustración 64. Visor vueltas sospechosas

Vueltas sospechosas:

Al igual que las vueltas rápidas al acceder a la sección de vueltas sospechosas se genera una tabla como la anterior pero esta vez se muestran las vueltas que resultan sospechosas es decir vueltas mucho más lentas de lo habitual que pueden ser provocadas por alguna salida o parada.

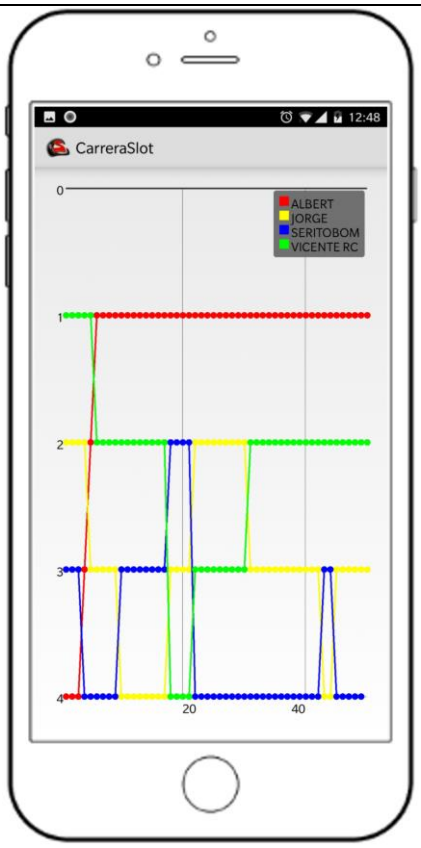

Ilustración 65. Visor posiciones

Posiciones:

Como se observa en la imagen en esta vista se muestra una gráfica lineal donde se ve que posición ha ocupado cada piloto en cada vuelta de la carrera mostrando una traza del trascurso de la misma.

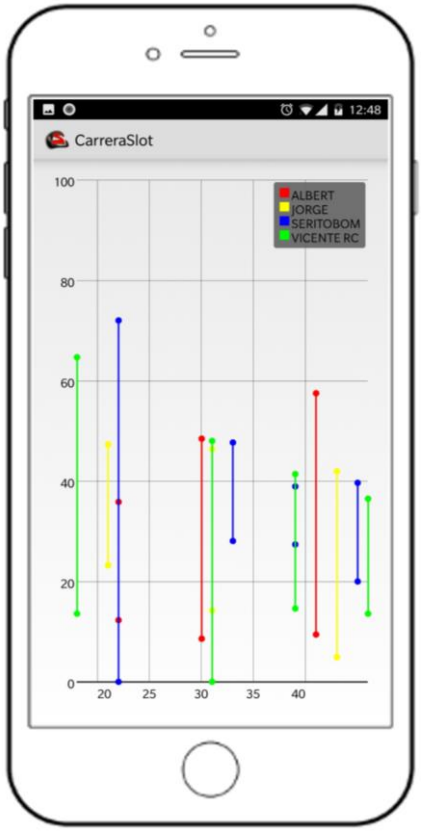

Ilustración 66. Visor repostajes

Repostajes:

Aquí usuario puede ver las paradas en boxes realizadas por cada piloto, así como la carga de combustible que disponían al iniciar la parada, la cantidad con la que salen del box y la vuelta en la que realizan dicha parada.

Por último, nos encontramos la opción de visor en las semifinales y finales, esta opción es la más completa puesto que permite reproducir la carrera para ello se muestra una pantalla como la siguiente:

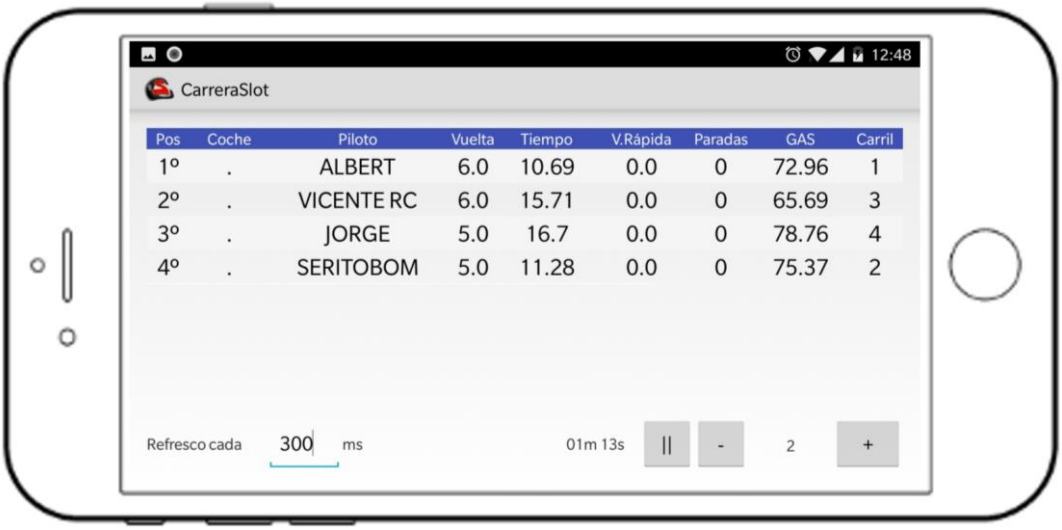

Ilustración 67. Visor carrera

En esta pantalla se muestra una tabla donde se va modificando la información según vaya pasando el transcurso de la carrera, para el visionado de los datos se pueden realizar una serie de configuraciones modificando el refresco hará que la actividad varié la velocidad con la que actualiza los datos de la tabla y modificando el multiplicador se aumentara o reducirá el intervalo de tiempo de obtención de los datos de la carrera siendo más precisos en el tiempo o provocando saltos más grandes en el tiempo.