Facultad
de
Ciencias

OPEN TRIVIA APP: APLICACIÓN MÓVIL MULTIPLATAFORMA PARA JUEGOS BASADOS EN PREGUNTAS Y RESPUESTAS
(OPEN TRIVIA APP: MOBILE APP FOR QUESTION & ANSWER GAMES)

Trabajo de Fin de Grado
para acceder al

GRADO EN INGRANIERÍA INFORMÁTICA

Autor: Daniel Sánchez Díez

Director: Carlos Blanco

Noviembre – 2021

Resumen

Uno de los juegos de mesa más conocidos es el clásico Trivial, en el que se plantean preguntas de todo tipo y dificultad con el objetivo de aprender nuevos conceptos o información o poner a prueba los que ya se poseen. Sin embargo, en la actualidad prácticamente nadie tiene tiempo para jugar a estos juegos de mesa, y mucho menos con la situación actual provocada por la pandemia que nos asola a todos, que no permite a la gente reunirse como se podía hace un tiempo. Este proyecto tiene como objetivo ofrecer una forma de jugar a este clásico donde y cuando quieras, en partidas rápidas y dinámicas.

Por otro lado, llamamos Open Data a toda información o repositorio de datos de dominio público que pueda servir para generar ideas de negocio o para aportar recursos de valor alrededor de los que se pueda crear una utilidad. Gracias a APIs como OpenTriviaDB, que ofrece preguntas de Trivial que los usuarios pueden actualizar, y con la tecnología de React Native se podrá crear una aplicación móvil multiplataforma, es decir, que permita jugar a este juego en cualquier dispositivo móvil. Además se hará uso de otra API que ofrece una tabla de clasificaciones online para almacenar las puntuaciones más altas.

Por lo tanto, podemos concluir que el objetivo de este proyecto es el aprendizaje y manejo de fuentes de Open Data, al mismo tiempo que se adquieren conocimientos del lenguaje de React Native y su integración con el objetivo principal, ya que es un framework de interés en la industria actual.

Palabras Clave: Open Data, React Native, JavaScript, Juego, Trivial, Aplicación Móvil, APIs Externas.

Abstract

One of the best known board games is the classic Trivial, in which questions of all kinds and difficulties are asked with the aim of learning new facts or testing those you already have. However, nowadays hardly anyone has time to play these board games, and even less so with the current situation caused by the pandemic that is ravaging us all, which does not allow people to get together as they could a while ago. This project aims to offer a way to play this classic game wherever and whenever you want, in quick and dynamic games.

The main goal to achieve with this project is to build a multi-platform app that exploits Open Data sources, which in this case will be "Trivial"questions. We call Open Data any information or repository of public domain data that can be used to generate business ideas or to provide valuable resources around which a utility can be created. Thanks to APIs such as Open Trivia DB, which offers the "Trivial"questions mentioned before, and using React Native technology, it will be possible to create a multi-platform mobile app, that is, one that allows this game to be played on any mobile device. In addition, we will make use of another API that offers an online leaderboard to upload high scores.

Therefore we can conclude that the objective of this project is to learn and manage Open Data sources, while acquiring knowledge of the React Native language and its integration with the main objective, as it is a framework of interest in the current industry.

Key Words: Open Data, React Native, JavaScript, Game, Trivial, Mobile App, External APIs.

Índice

Parte I
Introducción

El juego del Trivial es mundialmente conocido por poner a prueba los conocimientos de los jugadores que lo juegan, normalmente separando las preguntas por categorías, que pueden variar desde datos históricos o de anatomía, hasta datos curiosos sobre medios de entretenimiento o sucesos de actualidad, lo que hace que se pueda adaptar a cualquier época, entorno cultural y lugar. El hecho de querer llevar este juego al mundo digital quiere decir que se va a necesitar un conjunto muy amplio de preguntas que utilizar, ya que si no, se tornará repetitivo en poco tiempo. Para poder acceder a grandes conjuntos de datos disponemos de fuentes de Open Data.

Las fuentes de Open Data son bases de datos que son accesibles por todo el mundo que tenga conexión a Internet. Pueden abarcar cualquier ámbito, desde datos de ayuntamientos hasta recetas de cocina. Como son de libre acceso tienen la ventaja de, por ejemplo, ofrecer transparencia en las instituciones públicas, como puede ser el ayuntamiento de Santander. Si existe una fuente pública de preguntas de Trivial, puede ser explotada para crear un juego sin tener que recurrir a la búsqueda y almacenamiento de preguntas individuales, que sería poco eficiente y muy costoso.

La industria del videojuego ha llegado a crecer de forma exponencial en los últimos años, alcanzando la impresionante cifra de 3000 millones de jugadores alrededor del mundo. De esta enorme cifra, un 88.9 % han jugado o juegan actualmente en dispositivos móviles. Esto se debe en su mayor parte a que esta tecnología permite con cada vez más facilidad mover juegos y software que no hace mucho únicamente se podían encontrar en PC o consolas. Esto hace que gran parte de esa base de jugadores migre a una plataforma más accesible desde cualquier lugar y en cualquier momento, como es el teléfono móvil. Este hecho lleva a pensar que la creación de aplicaciones móviles hoy en día es una actividad de gran importancia y utilidad.

Sin embargo, entre todo este avance y complejidad a menudo se olvida que algo más simple no lo hace más aburrido. Por ello, y teniendo en cuenta lo anteriormente discutido, este proyecto pretende llevar al mercado actual este juego clásico y sencillo que ha entretenido a miles de personas durante años y que nunca envejece. Esto es, crear una aplicación móvil que imite el juego de mesa clásico.

1. Objetivos

Como se ha concluido en la introducción, el proyecto que se va a realizar tratará de llevar al dispositivo móvil una versión del Trivial que se pueda jugar con facilidad y en partidas cortas de un jugador, con el objetivo de superar la anterior puntuación más alta y poner a prueba los conocimientos que se poseen.

Para alcanzar dicho objetivo se plantean los siguientes sub-objetivos:

- **Explotar fuentes de Open Data:**
  Este apartado pretende abarcar el aprendizaje relativo a la utilización de las conocidas como Open Data Sources, es decir, obtener datos de las mismas y procesar dichos datos para utilizar la información relevante. Esto implica realizar las peticiones, parsear los datos y mostrarlos en la interfaz cuando sea deseado.

- **Crear la aplicación multiplataforma:**
  Otro aspecto a tener en cuenta es el hecho de que se desea que la aplicación pueda ser utilizada por un rango de usuarios más amplio, lo que quiere decir que hay que buscar una forma de hacer que funcione en plataformas con distintas características, como pueden ser los sistemas operativos. Para ello se puede, o bien adaptar el código fuente a cada sistema, o usar frameworks que puedan empaquetar una sola versión del mismo para que pueda ser ejecutado de forma nativa en cada entorno.

- **Utilizar servicios externos basados en APIs:**
  El concepto de una API externa (gratuita) es parecido al de una fuente de Open Data, solo que en vez de proporcionar información pura, pone a disposición del usuario una funcionalidad o conjunto de operaciones

que pueda utilizar. De esta forma, podemos acelerar o apoyar el proceso de desarrollo evitando tener que crear desde cero funcionalidades que ya están disponibles en Internet. La principal utilidad que se busca para este proyecto es un servicio de almacenamiento de puntuaciones en una tabla con clasificaciones.

Aparte de aprender a explotar la funcionalidad de las fuentes de Open Data, una de los beneficios que proporciona este proyecto es aprender a utilizar el framework de React Native, ya que a día de hoy React es el más usado a la hora de desarrollar frontend para productos software en todo el mundo. Específicamente, React Native permite hacer que ese frontend pueda ser ejecutado de forma nativa tanto en dispositivos con el sistema operativo Android como aquellos que usen IOS sin tener que programar dos versiones separadas del código.

Además, se pretende adquirir habilidades de aprendizaje y planificación individual en el contexto de un proyecto completo, ya que hasta el momento todos los que he realizado han sido en equipo, lo que cambia completamente la dinámica. Esto incluye la búsqueda de información y capacidad de resolución de problemas.

Para acabar, también se quiere poner a prueba la capacidad de aplicar el conocimiento adquirido durante la carrera, tanto teórico como práctico, intentando tener en cuenta todo lo posible en el proceso.

Parte II
Material y Metodología

2. Material utilizado

En esta sección se discutirán las herramientas usadas en el desarrollo del proyecto en todas sus fases, ya sea la planificación, que incluiría diagramas o mockups, tanto como en la implementación, que usaría editores de código, librerías y APIs.

Antes de comenzar el proyecto se dedicó un tiempo a discutir y establecer cuáles serían estas herramientas y por qué se escogieron, ya sea por las características adicionales que ofrecen, como la integración de GitHub en VS Code, o por la sencillez de uso, como podría ser Microsoft Project.

2.1. React Native

React Native es un framework de desarrollo de aplicaciones móviles basado en Javascript que permite construir aplicaciones que se empaquetan, compilan y renderizan de forma nativa en Android e IOS. Esto quiere decir que con un código único en Javascript se pueden generar aplicaciones que corran en cualquiera de esos dos sistemas operativos. Esto supone una enorme ventaja porque así se evita tener que crear dos versiones de una aplicación, una por cada sistema operativo, reduciendo el tiempo de desarrollo en gran magnitud.

Cabe destacar que este framework es una extensión de React, lenguaje creado por Facebook que se usa en el desarrollo de frontend, pero añade ciertos componentes que se pueden renderizar en múltiples plataformas.

Siguiendo el hilo de los componentes, se pueden definir como las diferentes unidades que se pueden combinar y reutilizar para formar una interfaz. Estos componentes pueden variar desde campos de texto hasta listas desplazables, dando lugar a cualquier tipo de combinación. De hecho, una de las mayores ventajas que proporciona este lenguaje es que se pueden crear componentes personalizados, que se pueden usar desde otros componentes, proporcionando

un alto grado flexibilidad a la hora de desarrollar las interfaces.[1]

A cada componente se le puede pasar una variable llamada 'props', que es la abreviatura de 'properties', la cual nos permite pasar datos como argumento y así personalizar aspectos del mismo. Además cada componente tiene un estado que puede ir cambiando con el tiempo, es decir, posee una variable llamada 'state'[2] que se puede usar para ir almacenando y actualizando datos, con el objetivo de cambiar lo que se renderiza en tiempo de ejecución.

2.2. Javascript

Javascript es un lenguaje que permite definir funciones y lógica operacional en entornos principalmente web para así darles elementos interactivos y dinámicos. Estas funcionalidades pueden ser almacenar datos en variables, operar con ellos y reaccionar a eventos o cambios en la interfaz, generando una respuesta adecuada. Además de esto se permite utilizar APIs externas para ampliar la funcionalidad del código o facilitar el desarrollo del mismo.

En el contexto de mi aplicación se ha usado Javascript para definir toda la parte de la lógica de negocio, en forma de funciones que se llamarán desde la interfaz (explicado con más detalle en el apartado de diseño de la aplicación).

2.3. Visual Studio Code

Visual Studio Code es un editor de código que además tiene soporte para otras herramientas de apoyo al programador, como pueden ser un debugger, terminales e integración con GitHub (explicado en el apartado de dicha herramienta).

Además de las herramientas base que incluye este editor, se han usado una serie de plugins con utilidades varias:

- **Auto Close Tag y Auto Rename Tag**
  Estos dos plugins funcionan prácticamente en conjunto a la hora de gestionar las etiquetas de los componentes. El primero de ellos funciona de tal forma que cuando se abre una etiqueta como por ejemplo <View>, automáticamente se crea su cierre, es decir, </View>. El segundo, por

otro lado, hace que al modificar la apertura de una etiqueta, cambia también el cierre y viceversa. Así sólo hay que cambiar una de ellas.

- **Babel Javascript**
  Babel es una herramienta que compila código de Javascript de cualquier tipo y versión, incluso con extensiones, y lo transforma a otro estándar más antiguo que cualquier navegador o versión de Node.js puede entender. Así se puede programar en cualquier estándar reciente sin miedo a perder compatibilidad con navegadores o compiladores.

- **ESLint y Prettier** Ambas extensiones sirven para comprobar la funcionalidad y formato del código, pero con ciertas diferencias. ESLint se encarga de detectar fallos de sintaxis en el código y da la opción de arreglarlos de forma rápida. Por otro lado Prettier se encarga de dar un formato consistente al código con el objetivo de darle claridad y orden. Por ejemplo cómo se establecen los sangrados, los espacios entre operandos, lineas en blanco, etc...

- **React Native Tools** Esta extensión proporciona un conjunto de herramientas para facilitar el desarrollo de proyectos de React Native, como ejecutar comandos de dicho lenguaje desde la 'command palette', por ejemplo.

Por último, se han usado un conjunto de librerías como utilidad para manejar ciertos aspectos de la aplicación de forma más sencilla:

- **React Navigation** Para poder configurar la navegación entre componentes de la app hace falta esta librería. Lo que permite hacer principalmente es definir un 'stack' de pantallas entre las que se puede navegar. Adicionalmente cada componente dispondrá de una propiedad llamada 'navigation' que contiene las operaciones necesarias para llevar a cabo la navegación, como la operación navigate(), que es la principalmente usada en la aplicación. Permite además enviar datos del componente actual al siguiente.[3]

- **Axios** Esta librería sirve para simplificar el proceso de realizar peticiones HTTP y permite usar la API de Promise para tratar las respuestas. Además permite recibir las respuestas en formato JSON por defecto, facilitando el proceso de tratamiento de dichos datos.[6]

- **Countown Component** Esta librería simplemente permite utilizar un componente de React que consiste en un reloj con cuenta atrás. Cuando esta acaba se puede establecer cualquier evento, que puede ser actualizar datos en pantalla, lanzar alertas o navegar a otras pantallas, por ejemplo.[7]

2.4. Microsoft Project

A la hora de realizar la planificación del proyecto, tanto temporal como funcional se uso este software, ya que ofrece una forma intuitiva y sencilla de calcular dichas estadísticas. Este programa ofrece una serie de herramientas que permite administrar tareas en el tiempo junto con recursos asociados, que pueden ser humanos, materiales, etc... Además permite redistribuir los tiempos de dichas tareas si se da el caso de sobrecargar cualquier aspecto que influya al desarrollo de las mismas, como puede ser escasez de mano de obra o falta de material.

2.5. MagicDraw y NinjaMock

En primer lugar, MagicDraw se utiliza para las tareas de diseño de diagramas UML a la hora de desarrollar productos software. Se utilizó principalmente también para planificar y representar de forma más concisa las estructuras y comportamientos del proyecto, como puede ser el diagrama de componentes.

Por otra parte, NinjaMock se usa para crear los mockups iniciales de las interfaces de la aplicación. Este software ofrece una serie de utilidades que permiten diseñar de forma rápida bocetos simples arrastrando elementos a un cuadro central y cambiando sus parámetros. Es una herramienta relativamente básica pero cumple la funcionalidad pertinente.

2.6. APIs y servicios Externos

Para llevar a cabo la funcionalidad de la aplicación se ha hecho uso de dos APIs externas con la siguientes funcionalidades:

- **Open Trivia DB** Se trata de una fuente de Open Data en la que los usuarios pueden subir preguntas de Trivial con sus respectivas respues-

tas. Estas preguntas se clasifican de diferentes formas, como puede ser su categoría, dificultad y tipo (verdadero / falso o elección múltiple). Dependiendo de las características que se quieran se modificará la URL de forma correspondiente. Es un servicio muy intuitivo y fácil de utilizar. [4]

- **Dreamlo** Esta API proporciona acceso a una tabla de clasificaciones personal y operaciones para operar sobre ella. Cada tabla tiene un código de escritura y uno de lectura que se incluirán en la URL de la petición en las ocasiones respectivas.[5]

2.7. GitHub (integrado en vscode)

Git es una herramienta que permite llevar un control de versiones eficiente para mantener el contenido del proyecto estructurado y controlado. Para ello se dispone de un repositorio remoto (es decir en Internet, en este caso GitHub) en el que se aloja el proyecto, y dicho repositorio se puede clonar de forma local, con todos los contenidos. Una vez se tiene el contenido del proyecto, se hacen los cambios que se desee y se guarda, para posteriormente volver a subirlo a remoto.

Adicionalmente Git ofrece el sistema de ramas, que consiste en poder gestionar varias versiones diferentes del proyecto en paralelo, cada una con sus cambios. Estas ramas se pueden separar, volver a juntar y eliminar cuando se desee.

Visual Studio Code ofrece una característica de alta utilidad, y es que permite utilizar repositorios de Github de forma integrada en el propio editor. Estas características incluyen prácticamente todas las presentes en Git, pero acompañadas de una interfaz que facilita el flujo de trabajo.

3. Metodologías seguidas

3.1. Metodología

La metodología que se ha seguido es la iterativa incremental por bloques de desarrollo. Lo que esto quiere decir es que el proyecto avanzará a base de implementar funcionalidades de una forma secuencial de forma que se tiene claro en todo momento qué desarrollar, dónde y cómo. En caso contrario se podrían dejar características a medias o producir bloqueos que impidan la progresión fluida del desarrollo.

Cada nueva característica importante que se quisiera introducir se aislaría en su propia rama de Git hasta finalizarla. Si existiera algún cambio menor que se quisiera añadir se haría en dicha etapa del desarrollo también. Al finalizar el bloque, éste se fusiona con la rama principal y se comienza un nuevo bloque de desarrollo, no sin antes asegurarse de que la nueva versión del proyecto no haya dejado de funcionar por culpa de los añadidos.

Las ventajas de utilizar esta metodología se pueden resumir principalmente a una sola: flexibilidad. Lo que esto quiere decir es que se puede ir modificando el desarrollo de la aplicación según convenga al mismo o si la situación lo requiere. Por ejemplo, si al estar desarrollando un bloque de trabajo se observa que hay características que pueden ser implementadas como apoyo antes de lo planificado, porque pueden reducir la carga de trabajo o complejidad que puedan surgir posteriormente, se puede incluir en dicho bloque. Si por el contrario hay alguna característica que puede ser desplazada a un punto posterior en el desarrollo también se puede organizar, siempre y cuando haya una razón, así se evita mezclar los bloques entre sí, problema que iría en contra de los principios de la metodología.

Un detalle del flujo de trabajo a resaltar es que se ha de tener dos terminales disponibles en todo momento. El primero servirá para tener corriendo Metro, que es un empaquetador Javascript que combina todos los archivos individuales que se tienen en uno solo, y otro en el que ejecutemos los comandos para construir la build de la aplicación que se quiere emular.

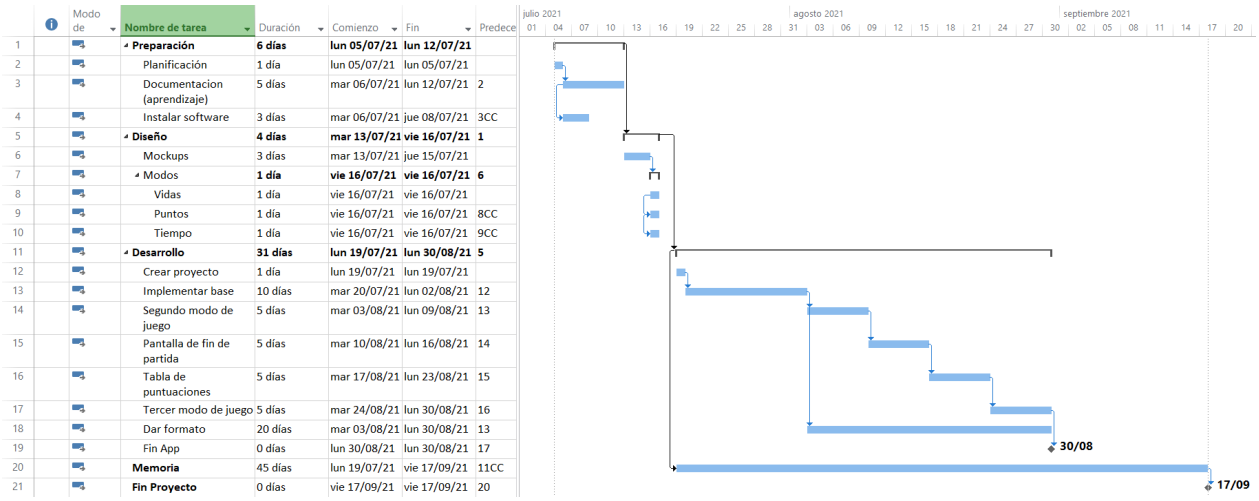
En primer lugar se inicia Metro y una vez esta activo, ejecutamos el comando de construcción. Ambos comandos[8], por orden, son:

```
$ npx react-native start

$ npx react-native run-android
```

3.2. Planificación

Con la intención de guiar el proyecto se creó un diagrama de Gantt para poder tener una idea general de la duración del proyecto, y si fuera necesario, acortar o alargar diferentes fases para acomodar el desarrollo a las necesidades del mismo. Dicho diagrama presenta la siguiente estructura general:



	Modo de	Nombre de tarea	Duración	Comienzo	Fin	Predece
1		**Preparación**	**6 días**	**lun 05/07/21**	**lun 12/07/21**	
2		Planificación	1 día	lun 05/07/21	lun 05/07/21	
3		Documentacion (aprendizaje)	5 días	mar 06/07/21	lun 12/07/21	2
4		Instalar software	3 días	mar 06/07/21	jue 08/07/21	3CC
5		**Diseño**	**4 días**	**mar 13/07/21**	**vie 16/07/21**	**1**
6		Mockups	3 días	mar 13/07/21	jue 15/07/21	
7		Modos	**1 día**	**vie 16/07/21**	**vie 16/07/21**	**6**
8		Vidas	1 día	vie 16/07/21	vie 16/07/21	
9		Puntos	1 día	vie 16/07/21	vie 16/07/21	8CC
10		Tiempo	1 día	vie 16/07/21	vie 16/07/21	9CC
11		**Desarrollo**	**31 días**	**lun 19/07/21**	**lun 30/08/21**	**5**
12		Crear proyecto	1 día	lun 19/07/21	lun 19/07/21	
13		Implementar base	10 días	mar 20/07/21	lun 02/08/21	12
14		Segundo modo de juego	5 días	mar 03/08/21	lun 09/08/21	13
15		Pantalla de fin de partida	5 días	mar 10/08/21	lun 16/08/21	14
16		Tabla de puntuaciones	5 días	mar 17/08/21	lun 23/08/21	15
17		Tercer modo de juego	5 días	mar 24/08/21	lun 30/08/21	16
18		Dar formato	20 días	mar 03/08/21	lun 30/08/21	13
19		Fin App	0 días	lun 30/08/21	lun 30/08/21	17
20		**Memoria**	45 días	lun 19/07/21	vie 17/09/21	11CC
21		**Fin Proyecto**	0 días	vie 17/09/21	vie 17/09/21	20

Figura 1: Diagrama de Gantt de planificación

La planificación inicial establece que las primeras semanas se usarán para establecer qué herramientas se van a usar, así como para adquirir una visión básica del funcionamiento de las mismas. Esto incluye instalar software, aprender las bases de los nuevos lenguajes y probar que todo funcione como es esperado, sobre todo las APIs. Posteriormente se realizarán los mockups correspondientes a la estructura general que seguirá la aplicación, y se solidificará la idea detrás del flujo de funcionamiento de cada modo de juego.

Una vez se entra en el desarrollo el primer paso es realizar la base de la aplicación, con una funcionalidad inicial que implemente un modo de juego y los sistemas básicos de vidas y puntos. Este paso es vital para aprender cómo funciona React Native y comenzar a adquirir fluidez a la hora de programar en dicho framework.

Una vez se tiene eso, se pasa a la implementación del resto de funcionalidades, como los modos de juego junto a la pantalla de selección de los mismos, tabla de puntuaciones y la parte de formato o aspecto de la aplicación, para mejorar la experiencia del usuario con la interfaz.

Parte III
Desarrollo

En esta sección se va a discutir todo lo referente al proceso de creación de la aplicación, desde la captura de requisitos a la finalización de la misma. El objetivo de esta parte es documentar toda la información posible sobre el proceso que se ha llevado a cabo y cómo se ha hecho, exponiendo detalles y complejidades del proyecto.

4. Presentación del problema y análisis de requisitos

El objetivo central de esta aplicación es, como se ha establecido anteriormente, una aplicación de juego basado en Trivial. Para poder comenzar a diseñar dicha aplicación el primer paso es establecer el alcance del proyecto, para lo que usaremos la ingeniería de requisitos.

El método que se ha usado principalmente para extraer estos requisitos ha sido la tormenta de ideas, en la que se han ido lanzando conceptos según se le ocurre al participante y se anotan para posteriormente realizar un filtrado y dejar solamente las mejores. Adicionalmente. se han usado otros métodos como el análisis de otros productos ya existentes, como otros juegos de Trivial u otros juegos móviles.

4.1. Requisitos funcionales

Para comenzar el diseño es necesario tener definidos los requisitos funcionales de la aplicación, es decir, aquellas funcionalidades que son indispensables para poder considerar el proyecto como satisfactorio. Se concluyó que los requisitos indispensables son los siguientes.

Referencia	Descripción
RF01	El usuario podrá elegir que modo de juego desea utilizar.
RF02	Las preguntas y respuestas se obtendrán de la API Open Trivia DB, y estos datos deberán ser procesados para poder representarlos en la pantalla y actualizarlos cuando rea requerido.
RF03	La aplicación deberá implementar un sistema de puntos, de forma que cuando se acierta una pregunta se suma un numero de puntos. Cada modo de juego aplicará un criterio para calcularlos.
RF04	Al finalizar una partida se dará la opción al usuario de subir su puntuación a una tabla clasificatoria a modo de competición con otros usuarios. Esta tabla será proporcionada por la API Dreamlo.
RF05	Existirá un modo de juego por vidas. Cuando se falla una respuesta se pierde una vida y si se llega a cero, se acaba la partida.
RF06	Existirá un modo de juego que use un temporizador, sin límite de fallos. Cuando el temporizador llega a cero se acaba la partida.
RF07	Se ofrecerán dos subdivisiones de los modos de juego: preguntas de verdadero o falso y preguntas de respuesta múltiple.

4.2. Requisitos no funcionales

Aparte de todos estos requisitos que establecen la funcionalidad de la aplicación, existen otros que se implementan para añadir aspectos de calidad al proyecto. Estos se llaman requisitos no funcionales.

Al igual que los requisitos funcionales, estos se establecen al inicio del proyecto y se tienen en cuenta durante la etapa de diseño para decidir aspectos como el tipo de arquitectura o, en el caso de necesitarla, la tecnología de bases de datos, por ejemplo. Estos consisten en:

Referencia	Descripción
RNF01	La aplicación ha de funcionar en Android e IOS por igual.
RNF02	Los tiempos de carga y actualización de las preguntas han de durar menos de 1 segundo.
RNF03	La aplicación ha de pesar menos de 300 MB, para minimizar el uso de almacenamiento.

4.3. Funcionamiento deseado de la aplicación

El proyecto deberá comportarse de una forma determinada, que es necesario planear con anterioridad, para así diseñar el flujo de pantallas adecuadamente. En el caso de este proyecto se desea comenzar con un menú principal que permita al usuario dos opciones: jugar una partida o ver la tabla de puntuaciones, por lo que habrá que mostrar dos botones claramente etiquetados y diferenciados. La pantalla de las puntuaciones mostrará todos los jugadores junto con sus puntos y la posición que ocupan en el ranking. Por otro lado, para iniciar la partida se mostrará al usuario una pantalla que le permita ajustar las opciones del juego antes de comenzar, y una vez esté satisfecho con dichos ajustes podrá avanzar a la siguiente pantalla, que contiene el juego como tal. Esta pantalla por lo tanto deberá ir actualizándose a medida que

avance la partida, cambiando la información mostrada y las respuestas que ofrece. Para acabar, después de finalizar el juego se mostrará una pantalla que permita al usuario revisar su puntuación, además de registrarla en el ranking si así lo desease.

4.4. Modos de juego

En esta sección se van a discutir en más profundidad los modos de juego de la aplicación y en que se diferencia cada uno.

Para empezar, podemos hacer una división entre los formatos que los separe en dos grandes categorías: partidas por vidas y por tiempo. En la primera, el jugador comenzará con un número de vidas, que representa la cantidad de respuestas erróneas que se puede permitir antes de acabar la partida. Esto quiere decir que cada vez que el usuario falla, se resta una vida, y al llegar a cero pierde. Por cada respuesta correcta el usuario recibirá una cantidad fija de puntos, dependiendo de la dificultad de las mismas. Por otro lado se dispone del modo de tiempo, en el que se establece un temporizador que cuenta hacia atrás, y la partida acaba cuando se llega a cero. Ya que los fallos no tienen penalización directa, el sistema de puntos variará con respecto al modo de vidas. Cuantas más preguntas seguidas acierte el jugador más puntos se le otorgarán, y si falla, se volverá a sumar la cantidad inicial, teniendo que volver a iniciar el 'combo'. De esta forma se promueve el hecho de contestar pensando y no abusar del sistema de puntos.

Adicionalmente existirá otra división para aportar variedad a las partidas, pero esta diferencia radica en las preguntas como tal. Se podrá elegir entre preguntas en las que se ha de responder si la afirmación propuesta es verdadera o falsa, y otro tipo de preguntas en las que se ofrecen cuatro opciones entre las que el usuario tendrá que elegir.

Todas estas opciones serán elegidas por el usuario al comienzo de cada partida, para así ofrecer variedad y flexibilidad, manteniendo el interés del jugador.

4.5. Apartado gráfico y Mockups

Para apoyar el desarrollo de la aplicación se crearon una serie de mockups con la idea de tener una estructura básica de las pantallas. Ya que se tratan de una versión prototipo, la aplicación final presenta algunas diferencias con los mismos, lo que es de esperar, ya que ciertos elementos pueden necesitar más o menos espacio en la interfaz.

Figura 2: Mockups iniciales de las pantallas de la app

5. Diseño

Tras haber establecido los requisitos que ha de satisfacer el proyecto se puede pasar a realizar el diseño del mismo a todos los niveles, desde cómo se va a ver hasta cómo se va a estructurar el código como tal. Este paso es extremadamente importante ya que el éxito del proceso de desarrollo de la aplicación depende directamente del diseño. Si se realiza un trabajo pobre en este aspecto, los futuros pasos sufrirán retrasos o bloqueos, cosa que influye negativamente en el resultado final.

5.1. Diseño arquitectónico y detallado

La aplicación se estructurará conforme a una arquitectura por capas que separe la parte de las vistas claramente de la parte que contenga la lógica de negocio. Se difiere del clásico diseño a tres capas (presentación, negocio y persistencia) ya que en realidad no se requiere ninguna clase de persistencia aparte de la recogida de puntos para las puntuaciones, que puede ser obviado ya que la API 'Dreamlo' ofrece dichas características por defecto [5]. Por lo tanto, las dos capas restantes quedarían definidas de la siguiente forma:

- **Capa de presentación:**

  En esta capa se incluyen todos los elementos de la aplicación que se encargan de renderizar componentes en la interfaz y mostrar la información relevante para el usuario en cada momento.

  Las pantallas o componentes deberán estar diseñados de tal forma que el flujo de la aplicación permita enviar y recibir información de forma coherente y clara entre ellas. Además se deberá procesar esa información para que se pueda utilizar y mostrar en los momentos oportunos.

  Los elementos que incluye esta capa son:

  - **Home**: Este componente abarca la pantalla inicial, que da la opción de comenzar una partida o ir a la tabla de clasificaciones.
  - **LeaderBoard**: Proporciona la pantalla que mostrará la tabla de clasificaciones en una lista desplazable.

- **ModeSelect**: En esta pantalla previa al juego se permite seleccionar las opciones de la partida al gusto del usuario.
- **Game**: Componente que corresponde al juego. Contendrá datos de la partida como las vidas o el tiempo, la puntuación y las preguntas y respuestas.
- **EndScreen**: Pantalla que se muestra al final de la partida. Contendrá la puntuación final y las opciones relativas a subir dichos puntos a la tabla.

- **Capa de negocio:**

  Esta capa contiene todo el código que representa la lógica de negocio y funcionalidad de la aplicación, desde calcular puntuaciones hasta las llamadas a las APIs externas. Se encuentra separada a su vez en 'controladores' aislados que agrupan las diferentes funciones para así poder localizar diferentes tipos de operaciones de manera más sencilla, por ejemplo, separar la administración de preguntas y respuestas y la de las puntuaciones o vidas.

  Todas estas funciones, programadas en javascript, podrán ser importadas y llamadas desde la capa de presentación, evitando así tener que mezclar funciones y componentes en la medida de lo posible.

  En esta capa se contienen los siguientes elementos concretos:

  - **PreguntasController**: Archivo que contiene las funciones relativas a manejar todos los datos de las preguntas, como puede ser pedirlas a la API, procesar los datos recibidos, devolverlos a la capa de presentación, comprobar la validez de las respuestas, etc...
  - **PuntosController**: En este archivo se encontrarán las funciones que manejan la puntuación durante la partida, ya sea calcularla, subirla a la API o volver a pedirla.
  - **VidasController**: Aunque solo tiene una función, está estructurada así por claridad. Dicha función se encarga de actualizar las vidas al fallar y comprobar que no se hayan perdido todas.

La comunicación entre capas se lleva a cabo mediante las conocidas como 'promesas' de Javascript ya que las peticiones de las APIs tardan más en devolver datos que la aplicación en realizar otras operaciones, lo que lleva a tener que gestionarlas en ambas capas.

Teniendo esto en cuenta, el flujo general de los componentes de la aplicación seguirá una estructura como la descrita en la figura siguiente:

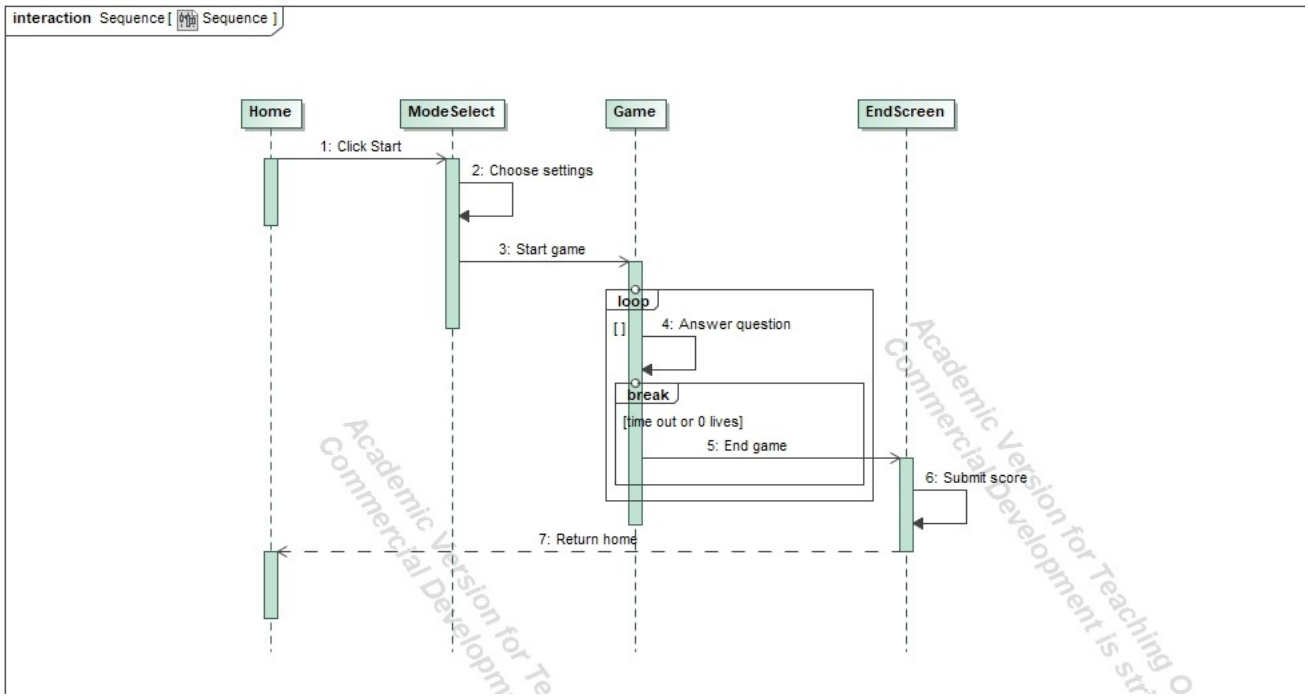


Figura 3: Diagrama de secuencia [9] de una partida normal

6. Implementación

6.1. Consideraciones previas

React funciona en torno al concepto de Componentes, elemento explicado en el apartado de herramientas, y en el caso de este proyecto, cada componente contiene toda la información referente a cada pantalla de la aplicación, y el código relativo a cada componente estará aislado en su propio archivo (en formato .js).

Cabe destacar que para poder navegar entre componentes se ha usado una librería añadida, para la cual es necesario incluir más código en el componente 'principal' de la aplicación, que es App.js. Dicho código establece un 'Stack' o pila de pantallas, que ha de incluir todas aquellas que se quieran establecer como navegables, tanto como para llegar a ellas como para ir de estas a la siguiente. Ya que dicha pila contiene todos los demás componentes y no posee ninguna utilidad definida en términos de interfaz, se ha decidido dejar este archivo fuera de las capas.

6.2. Creación del proyecto

Para comenzar con el desarrollo de la aplicación lo primero que se hizo fue seguir la documentación de React Native para generar un nuevo proyecto con el comando [8]:

```
$ npx react-native init Trivia
```

Una vez se ha realizado el paso anterior lo primero que se hizo fue instalar la librería de navegación, para lo que hay que utilizar este comando:

```
$ npm install
  @react-navigation/native @react-navigation/stack
```

Ahora que se tienen las herramientas básicas para empezar a experimentar, se comienza a experimentar con el framework, para lo que se modificará el archivo App.js para que uno de los botones que vienen en la plantilla por defecto lleve a otra en el que simplemente se muestra un 'Hola Mundo'.

6.3. Desarrollo de las bases de la aplicación

Ya que hasta ahora se tiene conocimiento sobre la navegación, el siguiente paso será instalar la librería que facilita el proceso de realizar las peticiones a las APIs, Axios. Tras esto, se crea una carpeta en el proyecto en la que guardaremos todos y cada uno de los componentes personalizados que se irán utilizando. De momento no se hace separación entre Presentación y Negocio ya que se está probando aún la funcionalidad de las librerías en un sólo archivo:

```
6  export default class Preguntas extends React.Component {
7    getPreguntas = () => {
8      axios
9        .get('https://opentdb.com/api.php?amount=10')
10       .then(function (response) {
11         console.log(response);
12       })
13       .catch(function (error) {
14         console.log(error);
15       });
16   };
17   render() {
18     return (
19       <View style={styles.container}>
20         <Text style={styles.texto}>Clicka para parsear preguntas</Text>
21         <Button onPress={this.getPreguntas} title="Obtener preguntas" />
22       </View>
23     );
24   }
25 }
```

Figura 4: Función de prueba de llamadas a la API OpenTriviaDB

Como se puede observar en la figura, el componente consta de una simple función que realiza una petición de tipo GET a la API, de tal forma que devuelva 10 preguntas de cualquier tipo y dificultad, e imprime los datos recibidos por consola de debugging que se puede abrir en el navegador. Ya que se usa Axios, dicha respuesta se da por defecto en formato JSON, lo cual facilita en gran cantidad el manejo de los datos recibidos.

Tras esto se efectuaron modificaciones al código para que en vez de imprimir la respuesta de la petición por la consola, se guardaran los elementos como el enunciado, las respuestas o el tipo en variables dentro del estado del componente. El estado o 'State' de un componente es un 'Hook' de React (significa que es una funcionalidad muy común en muchas aplicaciones, por lo que se ha incluido como una función predefinida de React para mejor acce-

sibilidad y uso) que se puede definir como una variable personalizada y que se irá actualizando durante la interacción con dicho componente en tiempo real. En el caso de esta aplicación se ha usado para actualizar la información que se renderiza, principalmente los datos de la pregunta, por ejemplo, evitando así implementaciones más engorrosas como podría ser una pantalla nueva por pregunta, que además de consumir memoria, ralentizaría la aplicación. [2]

```
12    state = {
13      enunciado: '',
14      correcta: '',
15      incorrecta: '',
16      categoria: '',
17      dificultad: '',
18      tipo: '',
19    };
20
21    getPregunta = () => {
22      axios
23        .get('https://opentdb.com/api.php?amount=1&type=boolean')
24        .then(response => {
25          this.setState({
26            enunciado: response.data.results[0].question,
27            correcta: response.data.results[0].correct_answer,
28            incorrecta: response.data.results[0].incorrect_answers[0],
29            categoria: response.data.results[0].category,
30          });
31        })
32        .catch(function (error) {
33          console.log(error);
34        });
35    };
```

Figura 5: Ejemplo de uso del 'Hook' State

Como es aparente en la figura, se guarda cada elemento de la respuesta a la petición en su correspondiente variable del estado.

Para acabar se creó una función que comprueba que la respuesta seleccionada es la correcta utilizando el estado, que se llama al pulsar el botón de cualquier respuesta. Ahora que se tienen los datos de la pregunta guardados en el estado, se pueden usar para ser renderizados. La versión inicial quedó como en la figura siguiente:

```
53    render() {
54
55      return (
56        <View style={styles.container}>
57          <Text style={styles.texto}> {this.state.categoria} </Text>
58          <Text style={styles.texto}> {this.state.enunciado} </Text>
59          <Text style={styles.texto}> {this.state.dificultad} </Text>
60          <Text style={styles.texto}> Correcta: {this.state.correcta} </Text>
61          <Button onPress={() => this.comprueba(true)} title="True" />
62          <Button onPress={() => this.comprueba(false)} title="False" />
63        </View>
64      );
65    }
66  }
```

Figura 6: Ejemplo de uso del State para mostrar información en la interfaz

Cabe resaltar que el archivo App.js ha quedado prácticamente libre del material proporcionado por la plantilla y ahora sirve como contenedor para el Stack de pantallas, así como el componente de la pantalla principal.

Viendo como el código que se tiene hasta ahora funciona, se decide modularizarlo y separarlo en las capas pertinentes. Todas las funciones de manejo de datos se mueven a la capa de negocio, en la carpeta 'controllers', mientras que el código que corresponde a todos los componentes como tal se deja aparte en la carpeta 'views'. Esto implica que el componente de la pantalla principal queda separado de App.js, haciendo que este archivo ahora sólo incluya el Stack de navegación. Además, fuera de estas dos capas se ubica el archivo styles.js que servirá para guardar los estilos de los componentes, como el formato de los textos, colores de fondo, etc... Así se elimina un bloque de código que se repetiría en cada archivo y simplemente basta con importarlo.

Todo el desarrollo hasta este punto se ha realizado en la rama principal, ya que no se corre el riesgo de añadir características que vayan a aplicar características fuera de la funcionalidad más básica. A partir de este punto se creará una rama por cada bloque importante de características que se deseen incorporar, ya que se corre peligro de provocar que operaciones que antes funcionaban dejen de hacerlo.

6.4. Puntos y Vidas

En primer lugar se implementó el sistema de cálculo de vidas y puntos, para lo cual se crearon dos 'controllers' separados adicionales: VidasController y PuntosController. El primero contiene una función que se llama cada vez que se falla una pregunta, que simplemente resta una vida y comprueba si el número está por encima de cero. En caso de que este por debajo se lleva al usuario de vuelta a la pantalla de fin de partida. Por otro lado, PuntosController tiene una función que dependiendo de la racha de preguntas seguidas que se contesten correctamente, suma un numero creciente de puntos. Tanto los puntos como las vidas y la racha de preguntas se guardan como nuevos parámetros en el estado del componente del juego.

La anteriormente mencionada pantalla de fin de partida, consiste en un componente que simplemente muestra la puntuación final de la partida y de un botón que devuelve al usuario a la pantalla de inicio. Para poder recibir datos entre componentes se usa el parámetro de 'route', que recoge todo aquello que haya sido enviado con 'navigation' en el componente anterior. En la siguiente figura se observa como usar el parámetro 'route', así como el resto del código del prototipo de pantalla final.

```
6   const End = ({ navigation, route }) => {
7     return (
8       <SafeAreaView style={styles.container}>
9         <View style={styles.containerPregunta}>
10          <Text style={styles.texto}> Final Score: {route.params.puntos} </Text>
11        </View>
12        {/* Comprobar record*/}
13        <TouchableOpacity
14          onPress={() => navigation.navigate('Home')}
15          style={styles.button}>
16          <Text style={styles.texto}>Back to Menu</Text>
17        </TouchableOpacity>
18      </SafeAreaView>
19    );
20  };
```

Figura 7: Ejemplo de uso del parámetro 'route' para recibir datos

6.5. Modo de respuesta múltiple

Para comenzar se hizo una copia del componente que conforma la pantalla de juego en modo 'Verdadero o Falso', que posteriormente será modificado en conjunto con el controlador para implementar este nuevo modo de juego. Se crearán una serie de funciones que sean similares a las ya existentes en el 'controller' de las preguntas, pero adaptadas de tal forma que funcionen para el modo de preguntas de múltiple elección. Los cambios más relevantes se pueden resumir en la modificación de la URL en la petición a la API, se cambia el estado del componente para guardar cuatro opciones en vez de dos y se necesitan cuatro botones en la interfaz. Además, para que la respuesta correcta no estuviese siempre en el mismo botón se creó una función para que se distribuyera de forma aleatoria.

En este punto del desarrollo este paso se llevó a cabo con bastante agilidad ya que es prácticamente igual al modo básico, excepto con el añadido de mezclar las respuestas y cambiar el método con el que se comprueba la corrección de dichas respuestas, que varía ligeramente.

6.6. Tabla de puntuaciones

Este paso es más complejo ya que hay que implementar la otra API que se iba a utilizar y además aprender a utilizar las FlatList, que son un componente que permite crear listas desplazables.

Para empezar se añaden tres funciones al 'controller' de preguntas que implementan las funciones necesarias para cubrir las necesidades de la aplicación. La primera, dados un nombre y una puntuación, las registra en la tabla de puntuaciones si ese nombre no estaba registrado ya. Adicionalmente, si ya existía ese usuario y la puntuación es más alta que la anterior notifica dicho récord. La siguiente, como apoyo a la anterior, devuelve la puntuación de un usuario.

Para acabar, la última función, mostrada a continuación, devuelve una lista completa de todos los usuarios con sus puntuaciones, junto con el índice que ocupan en la tabla.

```
export function getUserData() {
  return axios
    .get('http://dreamlo.com/lb/610571cd8f40bb8ea051de90/json')
    .then(response => {
      let respuesta = [];
      type Elem = { name: string, score: number, index: number };
      let elem: Elem = {};
      let indice = 1;

      if (response.data.dreamlo.leaderboard === null) {
        elem = { name: 'Empty', score: 'Empty', index: 0 };
        respuesta.push(elem);
      } else if (
        Object.keys(response.data.dreamlo.leaderboard.entry).length === 1
      ) {
        elem = {
          name: response.data.dreamlo.leaderboard.entry.name,
          score: response.data.dreamlo.leaderboard.entry.score,
          index: 1,
        };
        respuesta.push(elem);
      } else {
        for (let position of response.data.dreamlo.leaderboard.entry) {
          elem = { name: position.name, score: position.score, index: indice };
          indice++;
          respuesta.push(elem);
        }
      }
      return respuesta;
    })
    .catch(function (error) {
      console.log(error);
      throw error;
    });
}
```

Figura 8: Función que retorna la lista de usuarios con sus puntuaciones

Como se puede ver tenemos un array llamado 'respuesta' que contendrá objetos de tipo Elem, cuyos parámetros son los que se mostrarán por pantalla al renderizar. Dependiendo de la respuesta a la petición se tomarán diferentes acciones, ya que si no la interfaz lanzará errores. Si la tabla esta vacía, se devolverla una lista con un solo elemento, de índice 0 y nombre y puntuación 'Empty'; si sólo hay un elemento la respuesta no será un array, por lo que la estructura del JSON será diferente a si fueran varios objetos, y por eso están en casos diferentes.

Estas funciones se llamarán desde el componente del fin de la partida (EndScreen), que también cambiará desde su versión anterior para adaptarse a esta funcionalidad. Se añadirá un campo de input de texto para que el usuario introduzca su nombre, y un botón que llame a la función que sube las puntuaciones. Una vez se pulse el botón se usa el estado del componente para marcarlo como desactivado, evitando que el usuario envíe la misma puntuación varias veces.

Por otro lado, tenemos el componente que contiene la lista con las puntuaciones de todos los usuarios. Para llevar esto a cabo, necesitaremos primero un componente personalizado que represente una entrada en la lista, y una función que nos retorne una instancia de ese componente usando un elemento de la lista que devolvía la función de la figura 7, que se pasa como parámetro. Para acabar se renderiza una Flatlist a la que se le pasa como datos la lista de usuarios, se le asigna la función antes mencionada como método de renderizado de entradas individuales y establecemos que se use el nombre de usuario como clave.

```
14  const Item = ({ item }) => (
15    <View style={styles.row}>
16      <TouchableOpacity style={styles.button}>
17        <Text>{item.index}</Text>
18      </TouchableOpacity>
19      <TouchableOpacity style={styles.button}>
20        <Text>{item.name}</Text>
21      </TouchableOpacity>
22      <TouchableOpacity style={styles.button}>
23        <Text>{item.score}</Text>
24      </TouchableOpacity>
25    </View>
26  );
27
28  const renderItem = ({ item }) => {
29    return <Item item={item} />;
30  };
```

```
44    render() {
45      return (
46        <SafeAreaView style={styles.container}>
47          <Text style={styles.titulo}>TOP 25</Text>
48          <FlatList
49            style={styles.containerLista}
50            data={this.state.datos}
51            renderItem={renderItem}
52            keyExtractor={item => item.name}
53          />
54        </SafeAreaView>
55      );
56    }
57  }
```

Figura 9: Creación del formato de cada ítem de la FlatList y utilización de éste mismo, respectivamente

Para acabar esta parte, cabe destacar que se añadió una funcionalidad al botón de 'volver' presente en todos los teléfonos, ya que si se dejaba por defecto llevaba a diversos errores, el mas destacable siendo que desde la pantalla de fin de partida se podía volver al juego (con vidas negativas) hasta volver a

fallar. Ahora este botón hace aparecer una ventana de alerta que pregunta si de verdad se desea volver, y si el usuario selecciona que sí se le devuelve al inicio. En la figura siguiente se puede observar el código relativo a dicha alerta:

```
32    backAction = () => {
33      Alert.alert('Hold on!', 'Are you sure you want to go back?', [
34        {
35          text: 'Cancel',
36          onPress: () => null,
37        },
38        { text: 'YES', onPress: () => this.props.navigation.navigate('Home') },
39      ]);
40      return true;
41    };
42
43    componentDidMount() {
44      this.backHandler = BackHandler.addEventListener(
45        'hardwareBackPress',
46        this.backAction,
47      );
48    }
49
50    componentWillUnmount() {
51      this.backHandler.remove();
52    }
```

Figura 10: Código que añade la alerta al pulsar el botón de 'volver'

6.7. Modo de tiempo y estructura

Este bloque en un principio pretendía introducir únicamente el modo de tiempo limitado al juego, pero esto llevo a la necesidad de crear la pantalla de elección de modos de partida antes de lo planificado, ya que facilita llevar a cabo las pruebas de aceptación, por lo que se incluye en este mismo apartado.

Para empezar, se instaló el componente del contador que simplemente cuenta hasta cero y cuando llega envía al usuario a la pantalla de fin de partida.

Antes de empezar a programar el modo de tiempo se creó la selección de modos. Esta pantalla dará la posibilidad de elegir tres parámetros para la partida que se va a jugar: modo tiempo o vidas, tipo de pregunta y dificultad. Cada una de estas elecciones se representan con un grupo de botones que sólo permite seleccionar uno, que queda resaltado indicando el estado, y al

finalizar se pulsa el botón de comenzar partida. En ese momento se envían dichos parámetros al siguiente componente, que será el de la partida como tal.

Hasta ahora, como ya se sabe, la funcionalidad del juego como tal se encontraba separado en dos archivos, uno para verdadero y falso y otro para la elección múltiple, pero se cambió en concordancia con la selección de modos de juego, con el objetivo de evitar tener que introducir duplicados del código en archivos diferentes. Lo que se llevó a cabo fue la unión de ambos tipos de juego en un sólo archivo que, dependiendo de los datos que se reciban de la selección de modo, renderizará ciertos componentes u otros.

Esto quiere decir que habrá que crear funciones que recojan los parámetros de la pantalla de modos y devuelvan un componente u otro en cada caso. Esto se aplica tanto para los tipos de respuesta como para el modo de tiempo o vidas. Para este último, se implementó simultáneamente el contador y la alerta que lanza, ya que es sencillo de realizar. Por lo tanto, la función que devuelve el contador o vidas, por ejemplo, quedaría tal que así:

```
timeOrLives() {
  if (this.state.timeOrLives === 'time') {
    return (
      <View style={styles.containerPuntos}>
        <CountDown
          until={300}
          onFinish={() =>
            this.props.navigation.navigate('EndScreen', {
              puntos: this.state.puntos,
            })
          }
          size={15}
          timeToShow={['M', 'S']}
        />
      </View>
    );
  } else if (this.state.timeOrLives === 'lives') {
    return (
      <View style={styles.containerPuntos}>
        <Text style={styles.textoP}>Lives: {this.state.vidas}</Text>
      </View>
    );
  }
}
```

Figura 11: Código que retorna un componente u otro dependiendo del 'State'

6.8. Formato

Lo último que se implementó fue un cambio de formato en la aplicación, desde las propiedades de los componentes, como formas y color hasta el icono y nombre de la aplicación.

En general, la apariencia de la aplicación no tiene grandes complicaciones, simplemente consiste en organizar todo de una manera limpia y simple creando estilos en el archivo styles.js y utilizándolos en los componentes. Estos cambios incluyen formato de texto, contenedores, tamaños y formas de botones, colores de fondo y principales, y colocación de todos los elementos en la interfaz de forma que sea claro y no se superponga nada.

Por otro lado, el nombre de la aplicación no se puede cambiar desde el proyecto de React Native, sino que se tiene que hacer desde ambos módulos individuales de Android e IOS. Para el primer caso hay que ir al archivo que contiene los strings comunes de la app (strings.xml) y cambiarlo ahí, mientras que para IOS hay que ir al archivo Info.plist y cambiarlo ahí. Para los iconos ocurre algo similar, en cada módulo hay que introducir los iconos con los formaros específicos de manera manual. Para Android existen carpetas que contendrán las versiones de los iconos en distintos tamaños o resoluciones (bajo la carpeta de recursos), al igual que en IOS en su propio sistema de archivos, aunque cada entorno requiere una serie de tamaños diferentes. Una vez se empaqueta el proyecto de React Native, la aplicación generada tendrá estas propiedades incluidas.

6.9. Resultado final visual

Al final del desarrollo la aplicación quedó de la siguiente forma:

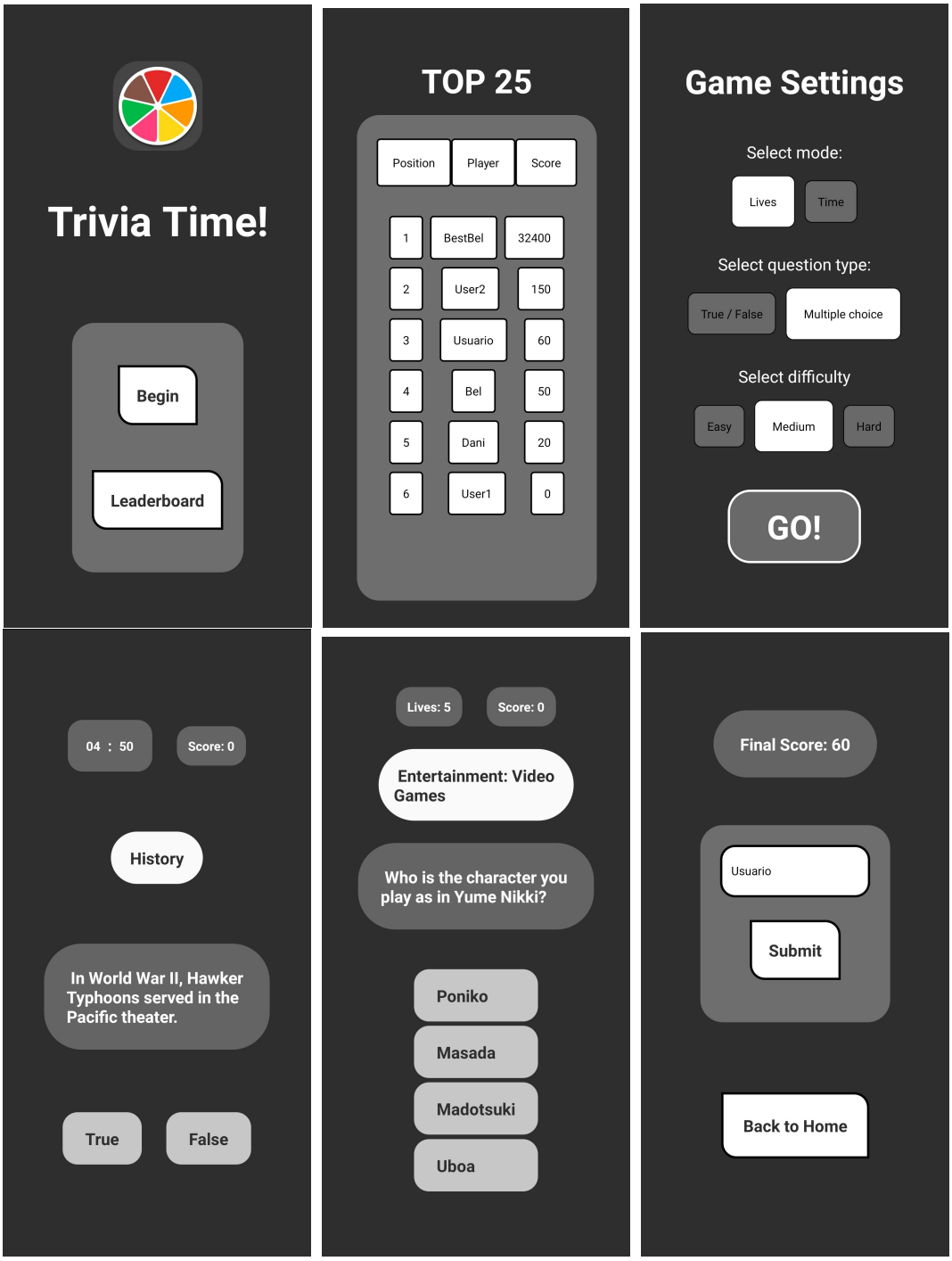


Figura 12: Resultado final de las pantallas

Como se puede observar se han seguido los mockups salvo algunos cambios menores para conseguir mostrar la información u opciones. Por ejemplo, la pantalla de selección de modos comenzó con el propósito de incluir menús 'drop-down' pero acabó siendo una serie de botones que resaltan las opciones elegidas. También se ha cambiado la disposición de los botones de respuesta en el modo de múltiple elección.

En general se ha seguido un esquema común y coherente de colores y formas para generar una sensación de uniformidad.

7. Pruebas

En este apartado se pretende discutir el tipo de pruebas que se han ido aplicando a medida que se ha producido el desarrollo del proyecto y cómo han conseguido detectar fallos y bugs en la aplicación, pudiendo así eliminarlos.

7.1. Pruebas unitarias y de integración

Se incluyen estos dos tipos de pruebas en una sola categoría porque, aunque no se han programado casos de prueba como tal de la funcionalidad, durante el desarrollo se han ido debuggeando las funciones usando la consola de debug y se han creado componentes de prueba con botones que llamen a dichas funciones en el caso de que fuera necesario. Gracias al output proporcionado por la consola se pueden trazar los orígenes de los errores y arreglar los mismos. Este proceso vale tanto para las funciones individuales como la integración entre las mismas y la interfaz.

7.2. Pruebas de sistema

Para testear los Requisitos No Funcionales se aplican este tipo de pruebas. Por ejemplo, se miden los tiempos de carga para corroborar que no se supera el límite estipulado de un segundo. En el caso del proyecto se hicieron numerosas pruebas, haciendo peticiones al servidor de preguntas (OpenTriviaDB) y se corroboró que efectivamente los tiempos de carga eran notablemente menores a 1 segundo. Además se comprobó el almacenamiento del dispositivo para

comprobar que, efectivamente, la aplicación ocupa un total de 84,34 MB, que está por debajo de los 300 MB estipulados.

7.3. Pruebas de aceptación

La clase de pruebas que se han realizado han sido principalmente de aceptación, es decir, se ha probado manualmente la aplicación en un dispositivo móvil real, comprobando que todo funciona correctamente en cada escenario y que no hay posibilidad de fallos fatales o bugs inesperados. Para cubrir esta función se ha ofrecido la aplicación a familiares y amigos con el objetivo de que jueguen unas partidas y traten de descubrir fallos y bugs que haya que arreglar.

Parte IV
Conclusión y trabajos futuros

Habiendo concluido el proyecto, se puede echar la vista atrás a los objetivos propuestos al inicio, y evaluar si se han cumplido y de qué forma.

Para empezar, se puede llegar a la conclusión de que se han cumplido el objetivo de crear una aplicación multiplataforma capaz de utilizar y explotar las fuentes de Open Data, implementando varios modos de juego y una visualización de las puntuaciones de los jugadores.

Por un lado, se ha conseguido llevar a cabo el aprendizaje de la utilización de las fuentes de Open Data gracias a la librería Axios, que como se ha discutido anteriormente, facilita el proceso de petición y respuesta de datos a servicios externos mediante peticiones HTTP. El procesado de datos se ha solucionado parseando el JSON obtenido en funciones de JavaScript, y gracias a React se ha creado una interfaz capaz de actualizarse usando el estado dinámico y de mostrar los datos pertinentes de forma clara y ordenada.

Por otro lado, aprovechando las características que ofrece la librería de React Native se ha podido desarrollar el proyecto de tal forma que se pueda compilar de forma nativa tanto en Android como IOS con una única versión del código fuente, haciendo que éste se convierta así en una aplicación multiplataforma.

Para acabar, también se ha conseguido integrar la API de Dreamlo para manejar los récords de los jugadores, también usando Axios para enviar y recibir los datos en peticiones HTTP. Combinando las repuestas de la API con los componentes proporcionados por React, se ha construido una lista capaz de mostrar de forma ordenada las puntuaciones con sus respectivos usuarios.

Con respecto al apartado personal, al acabar el proyecto se puede llegar a la conclusión de que aprender a trabajar con nuevos lenguajes y frameworks es extremadamente útil, ya que estar al día con las nuevas tecnologías proporciona una gran ventaja a la hora de llevar a cabo trabajos, tanto por la

facilidad de acceso a los recursos como por la abundancia de documentación y ayuda que se puede encontrar. Sin embargo, si únicamente contamos con la documentación, aprender de forma individual se convierte en una tarea más ardua que si aprendes de forma estructurada, lo que lleva al siguiente punto.

El hecho de que sea individual tiene sus ventajas e inconvenientes. Por un lado, los conceptos que aprendes se fijan mejor y a un ritmo natural y de forma más 'cercana', pero por otro lado puede ser que el aprendizaje sea desordenado, incompleto o ineficiente. Lo que se puede extraer de esto es que con más práctica y probando nuevos conceptos se puede llegar a dominar una materia específica aunque lleve tiempo.

Adicionalmente he aprendido que la organización, planificación y constancia son incluso más esenciales de lo que se puede observar al principio. Principalmente me he dado cuenta de esto al final del proyecto, cuando al añadir características nuevas a la aplicación se complicaba el hecho de complementarlas de forma que funcionen con todo el resto de aspectos de la misma, dejando en evidencia un fallo en el diseño.

Con respecto a trabajos futuros, me gustaría seguir probando con el desarrollo de aplicaciones, tanto móviles como web, poniendo en práctica los conocimientos adquiridos en este trabajo, así como aprendiendo todavía más cosas por el camino. En un futuro también me gustaría desarrollar mi propio videojuego, que aunque no esté directamente relacionado con el desarrollo de aplicaciones, es un tema que me llama la atención, y las ideas surgen con frecuencia. Creo que podría ser una buena forma de poner a prueba mi capacidad creativa, de diseño y de aprendizaje, otro desafío más que superar.

Parte V
Bibliografía

Prácticamente se ha usado la documentación oficial de las herramientas pertinentes, así como conocimientos adquiridos en la carrera.

[1] Componentes de React native
https://reactnative.dev/docs/intro-react-native-components

[2] States en React native
https://reactnative.dev/docs/intro-react

[3] Navegación en react Native
https://reactnative.dev/docs/navigation

[4] Documentación de la API Open Trivia DataBase
https://opentdb.com/api_config.php

[5] Documentación de la API Dreamlo
http://dreamlo.com/developer

[6] Documentación de Axios (GitHub)
https://github.com/axios/axios

[7] Documentación del contador de React
https://www.npmjs.com/package/react-countdown

[8] Comandos para la creación y gestión del proyecto
https://reactnative.dev/docs/environment-setup

[9] Documentación sobre UML para realizar diagramas
https://www.uml-diagrams.org/