Alumno: Javier Martínez Arias Máster MIMO – UPSA

Nombre del proyecto: React Native Rickpedia

Documentación - Proyecto final Desarrollo de aplicaciones cross-platform

1. Introducción

En este documento se va a presentar la aplicación desarrollada como proyecto final de la asignatura Desarrollo de aplicaciones cross-platform.

Se trata de una aplicación cuya temática está basada en la serie de animación “Rick y Morty”. En ella, se muestra un listado de contenidos diferenciados en 3 categorías: personajes, ubicaciones y episodios de la serie. A partir de los elementos del listado, podemos acceder al detalle de cada uno de ellos y añadirlos a listas de contenidos favoritos, según la categoría a la que pertenecen. Toda esta información estará asociada al perfil del usuario identificado, ya que la aplicación permite crear hasta 5 perfiles de usuario por dispositivo, cada uno de ellos con sus preferencias de contenido.
Por último, la aplicación incluye un juego que consiste en adivinar los personajes de la serie, dado el nombre y 4 imágenes, siendo solo una de ellas la opción correcta.

La aplicación consta de un flujo de pantallas que se van sucediendo en un orden coherente. En primer lugar, se nos da la bienvenida a la app, dándonos la opción de identificarnos si ya tenemos un perfil, o bien crear uno nuevo. Una vez identificados, pasaremos a la pantalla de home, que nos permite acceder a todas las secciones disponibles. Por defecto, se muestra la pantalla de contenidos sugeridos. Podremos navegar principalmente hacia el listado por categorías, juego y la sección del perfil.

Con esta aplicación se pretende poner de manifiesto las capacidades obtenidas durante el curso, según el temario visto durante la asignatura.

2. Librerías utilizadas

Para la realización de este proyecto, además de la librería principal de react native (y todas aquellas que vienen durante la instalación por defecto), se han utilizado otros módulos, entre los que identificamos:

- **@react-native-community/async-storage:** Librería que permite almacenar a nivel de fichero, las sesiones de usuario abiertas desde el último uso de la aplicación. De esta forma, evitamos al usuario el paso de identificarse cada vez, si lo había hecho ya con anterioridad. Al cambiar de perfil, se elimina la información de la sesión, para que no vuelva a iniciarse con su usuario por defecto. La información guardada para la sesión es el nickname del usuario, y un token aleatorio de acceso de la última sesión, que se irá actualizando en cada sesión por seguridad.
- **@react-navigation/bottom-tabs:** Librería encargada de gestionar la navegación a nivel de pestañas en la parte inferior de la pantalla. Se ha utilizado para navegar entre los distintos apartados de la aplicación, una vez el usuario se ha identificado.
- **@react-navigation/native:** Módulo de React Navigation que permite obtener el componente *"NavigationContainer", necesario para encapsular* la pila de pantallas de navegación en un "*Wrapper*" interpretable por los sistemas de navegación nativos de los dispositivos móviles.
- **@react-navigation/stack:** Librería encargada de gestionar la pila de navegaciones disponible para cada sección o pestaña de la aplicación. Se ha utilizado para navegar entre los distintos apartados de la aplicación, una vez el usuario se ha identificado.
- **axios:** Librería que permite realizar peticiones HTTP desde la aplicación hacia las distintas APIs utilizadas y gestionar las respuestas recibidas. En este caso, las peticiones realizadas han sido de tipo GET, pues solo se quería obtener la información que tienen estas APIs. Sin embargo, la librería permite realizar todo tipo de peticiones: POST, PUT, etc.
- **react:** Se trata del Core de React, que permite hacer uso de todas las funcionalidades propias del framework: componentes, hooks, ciclo de vida, etc.
- **react-native:** Se trata del Core de React-Native, que permite combinar las funcionalidades de React con las funcionalidades y componentes propios, compatibles con los Sistemas operativos Android e IOS.
- **react-native-gesture-handler:** Modulo que permite manejar los gestos del usuario en los dispositivos, y asociarles una funcionalidad. En este caso, se ha incluido por ser requerida por la librería *@react-navigation/stack.*

- **react-native-get-random-values:** Librería que permite la generación aleatoria de valores. Se ha instalado por ser requerida en el módulo uuid, que veremos a continuación.
- **react-native-safe-area-context:** Modulo que permite detectar el viewport disponible en cada pantalla, respetando los márgenes “safe”. Se ha incluido por ser requerida en la instalación de “*react-navigation”.*
- **react-native-screens:** Modulo que permite gestionar las pantallas de la aplicación de react native. Se ha incluido por ser requerida en la instalación de “*react-navigation”.*
- **react-native-sqlite-2:** Librería que permite gestionar la persistencia de datos de la aplicación a nivel de base de datos. Se trata de una base de datos, con el motor sqlite, que almacena de forma segura la información de los usuarios de la aplicación y todos sus contenidos relacionados. Por tanto, nos permite crear tablas y trabajar con ellas.
- **react-native-vector-icons:** Librería que permite incluir iconos vectoriales en la aplicación, mejorando la identificación de ciertos elementos con los que interactuar, así como la estética.
- **uuid:** Librería que permite generar identificadores únicos de tipo alfanumérico. Se ha utilizado para crear tokens de sesión aleatorios, cada vez que el usuario inicia sesión. Si se modificara a mano el token generado en el fichero de *“async-storage”* podrían producirse errores de identificación.

3. Descripción funcional

A nivel funcional, la aplicación se desarrolla en un total de 5 flujos principales de navegación, cada uno de ellos consta de varias pantallas que analizaremos según esta clasificación.

3.1. Flujo de navegación de bienvenida

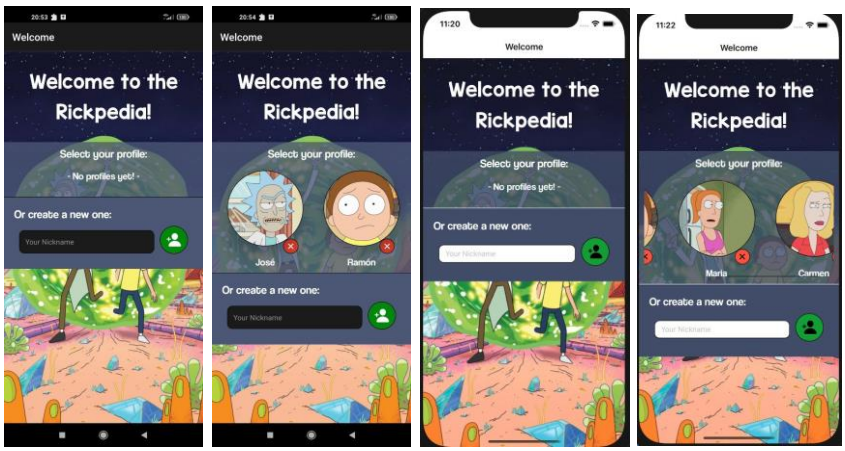
- **Bienvenida/Acceso**: Se trata del primer flujo que encontramos al acceder a la aplicación. Es el que nos permite acceder a nuestro perfil o crear uno nuevo si no lo hemos creado previamente. Superado este proceso se avanza hacia la siguiente pantalla y navegación, de Home/Inicio.

  Este flujo puede verse modificado la segunda vez que iniciamos la aplicación, pasando directamente de la bienvenida al home gracias a la sesión guardada en fichero de configuración (*AsyncStorage*), que se establece por defecto, si un usuario se ha identificado previamente y no ha hecho logout.

  Este primer flujo está compuesto por una sola vista:

  - Welcome: vista que nos permite realizar el comportamiento descrito con anterioridad. Para ello, está compuesta por un listado de elementos con nombre y avatar (asignado automáticamente), que, al pulsar sobre alguno de ellos, nos identifica y nos da acceso a la Home. El listado se crea con un FlatList Horizontal. También existe un botón de borrado, asociado a cada perfil. Si pulsamos en alguno, se nos mostrará una ventana para confirmar la acción.

    Si no hemos creado ningún perfil, este listado estará vacío y no habrá elementos en la lista con los que interactuar. Por otro lado, tenemos un campo de texto donde introducir el "*nickname"* del perfil a crear y un botón para confirmar. Al pulsar, se añadirá un elemento al listado, con el nombre introducido y un avatar asociado.



3.2. Flujos de navegación principal

Una vez analizada la pantalla de bienvenida e identificación, pasamos a analizar los principales flujos de navegación, disponibles para los usuarios identificados.

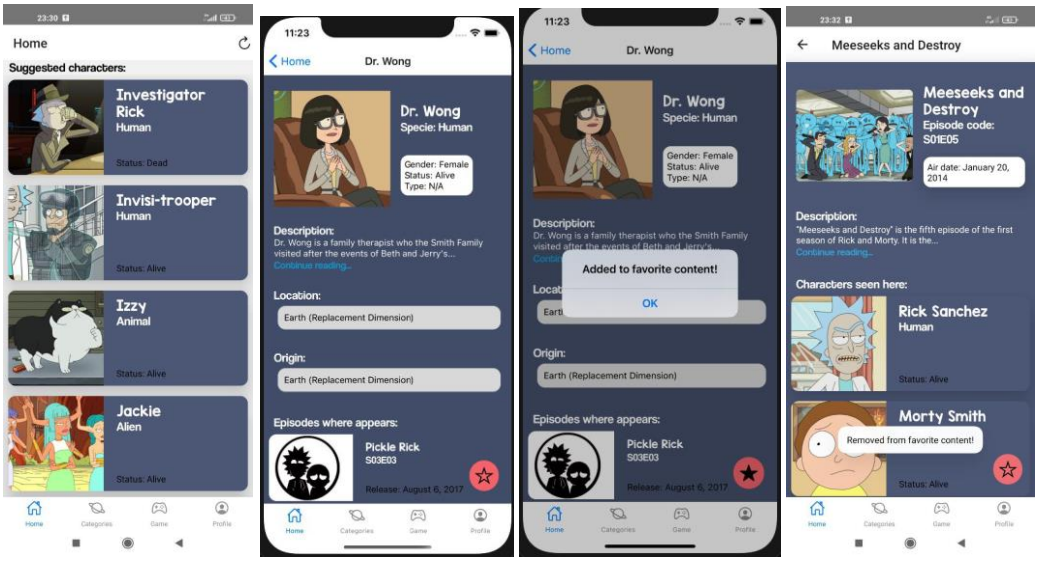
Son aquellos flujos correspondientes con las opciones que nos ofrece el BottomTabNavigator, es decir: Home, Categorías, Juego y Perfil. Cada uno de ellos se encuentra encapsulado dentro de un Stack o pila de navegación, que veremos a continuación:

3.2.1. Flujo de Inicio

- **Home/Inicio**: Se trata de uno de los flujos principales de la aplicación, ya que es el primero al que se accede una vez el usuario se ha identificado. Al cargar el componente podemos distinguir varios elementos:
    - En la barra superior, tenemos el título de sección y un botón para refrescar el contenido sugerido.
    - Título indicando el tipo de contenido sugerido.
    - Listado de elementos sobre el tipo de contenido sugerido. Siempre se muestran un total de 10 elementos independientemente del tipo. Cada uno de ellos se corresponde con el mismo componente, formado por una imagen y 3 textos. Este listado, se obtiene haciendo uso de FlatList con disposición vertical.
    - En la parte inferior se muestra el BottomNavigationTab, con las 4 opciones comentadas anteriormente.

  Si pulsamos en alguno de los ítems del listado, accedemos al detalle correspondiente. Se trata de una pantalla cuyo componente es reutilizado en muchos otros flujos de navegación principal, como veremos más adelante.
    - **Detalle:** Pantalla ampliando la información que se ve a simple vista del elemento pulsado. En ella tenemos:
        - Barra superior, con el título del contenido seleccionado.
        - Imagen del contenido.
        - Título del contenido.
        - Subtítulo del contenido.
        - Información adicional del contenido.
        - Descripción y enlace externo al artículo con más información.

- Listado de elementos relacionados con el contenido: En caso de ser un contenido de tipo personaje, el listado será de episodios y en otro caso, de personajes. Este listado está hecho con FlatList vertical y al pulsar en alguno de los elementos nos llevará al detalle de cada uno.
- Solo en caso de ser contenido de tipo personaje, se muestra también la ubicación de origen y la actual correspondientes. Al pulsar sobre ellas se redirige al detalle asociado de la ubicación.

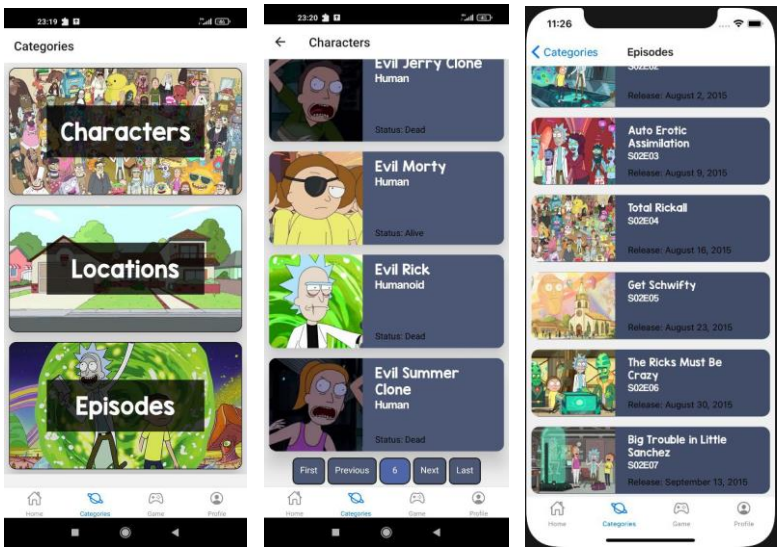


3.2.2. Flujo de Categorías

- **Categorías**: Flujo de navegación que permite visualizar las categorías en las que se encuentran clasificados los distintos contenidos. Al cargar el componente asociado, tenemos:
    - Barra superior con el título de sección.
    - Listado de categorías. En este caso tenemos siempre los 3 tipos de categorías comentadas: Personajes, Ubicaciones y episodios. Cada elemento de la lista tiene un ImageBackground y un texto asociado, con el nombre de la categoría. Nuevamente, el listado se crea con una FlatList en disposición vertical.
    - En la parte inferior se muestra el BottomNavigationTab.

Si pulsamos en alguno de los ítems de categorías, accedemos al listado de contenido de la categoría seleccionada. Se trata de una pantalla cuya vista es muy similar a la de Home, por mostrar un listado de contenidos, en este caso filtrado según la selección:

- **Listado de contenido por categoría:**
  - Barra superior, con el título de la categoría seleccionada.
  - Listado de elementos de la categoría. Se muestran siempre un máximo de 20 elementos por página. Se utiliza para ello un FlatList vertical.
  - Footer con la paginación disponible de elementos: Primera, anterior, actual, siguiente y última página. Permite organizar el contenido e mostrando poco a poco, de 20 en 20, actualizándolo cada vez que se pulsa en algún ítem de la paginación. En algunos casos, estos botones se deshabilitan, cuando no tienen más contenido que mostrar.
- **Detalle de contenido:** Como ya ocurría en la vista de home, si pulsamos sobre algún elemento de la lista, accedemos al detalle de contenido de este. La vista es la misma, al reutilizar el componente ya visto.



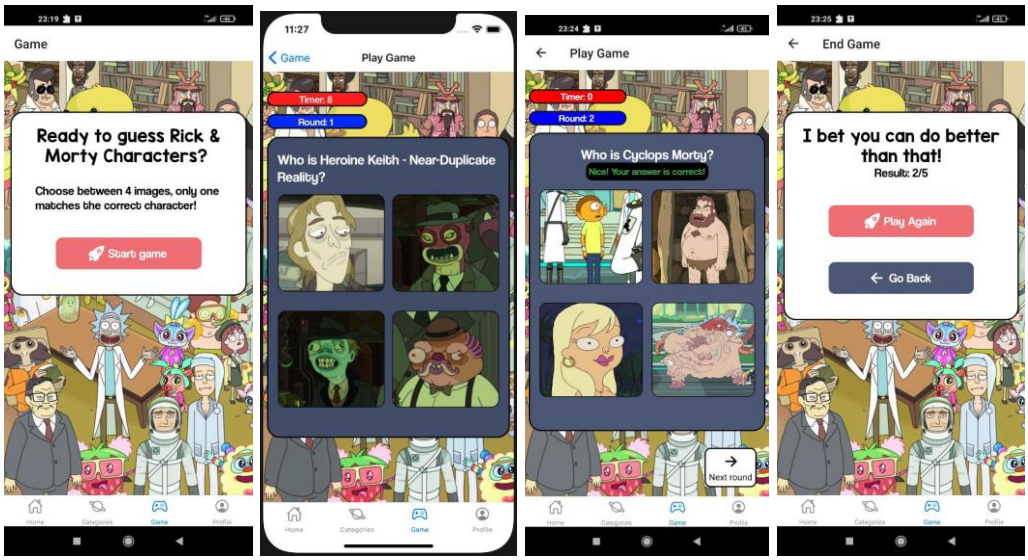
3.2.3. Flujo de Juego

- **Juego:** Flujo de navegación que permite jugar a adivinar la imagen del personaje de Rick y Morty, conociendo su nombre. El primer componente principal consta de:
  - Barra superior con el título de sección.
  - Componente contenedor principal: Consta de un título de presentación del juego, un texto descriptivo y un botón para jugar.
  - En la parte inferior se muestra el BottomNavigationTab.

  Si pulsamos sobre el botón de jugar, seremos redirigidos a la siguiente pantalla en la que comienza el juego.

- **Partida:** Pantalla en la que jugamos a adivinar el personaje. En primer lugar, se mostrará un contenedor con un texto, indicando una cuenta atrás de 3 segundos. Transcurrido ese tiempo, pasamos a analizar los elementos visibles:
  - Barra superior con el título de sección.
  - Indicadores de cuenta atrás y ronda de la partida.
  - Componente contenedor principal: Consta de un texto con la pregunta y de 4 imágenes seleccionables. Estas se crean usando un FlatList.

    Las imágenes se corresponden con el personaje correcto y 3 personajes aleatorios. Si pulsamos en alguna de las imágenes antes de que el tiempo se acabe, se nos muestra debajo de la pregunta si la respuesta indicada es correcta o no. En caso de que el tiempo se acabe también se nos indica en esta sección.
  - Botón flotante: Al finalizar una ronda, aparece en la parte inferior derecha de la pantalla un botón para pasar a la siguiente ronda.
- **Fin de partida:** Pantalla en la que se nos indica el final de partida con el resultado obtenido. Los elementos visibles son:
  - Barra superior con el título de sección.
  - Componente contenedor principal: Consta de un título descriptivo sobre el resultado obtenido, un texto indicando el número de respuestas correctas entre el número de rondas.
  - Botón para volver a jugar: Se reinician todos los parámetros y empieza la cuenta atrás.
  - Botón para volver atrás: Volver a la pantalla principal de juego.

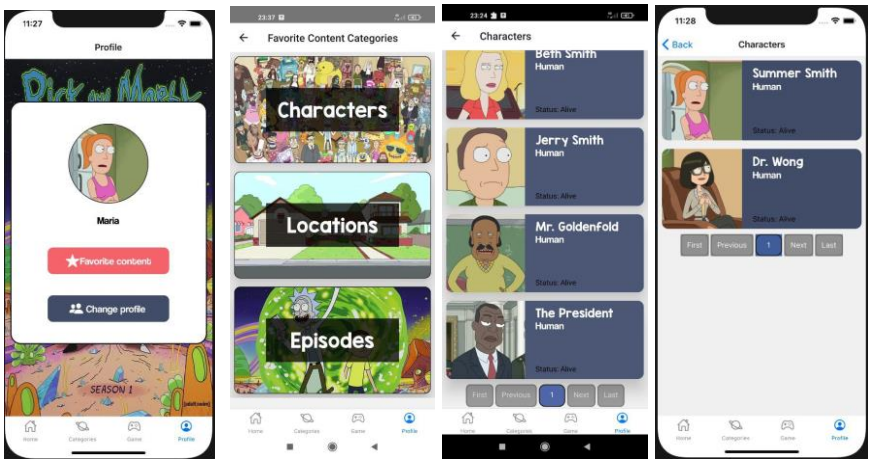


3.2.4. Flujo de Perfil

- **Perfil:** Flujo de navegación que permite acceder a las opciones del perfil de usuario identificado. El primer componente principal consta de:
    - Barra superior con el título de sección.
    - Contenedor principal: Consta del avatar de perfil y el texto con el *nickname* de usuario. Además, hay dos botones: uno para acceder al contenido favorito y otro para cambiar de perfil.
    - En la parte inferior se muestra el BottomNavigationTab.

  Si pulsamos en el botón de cambiar de perfil, se cerrará la sesión actual y volveremos a la pantalla de bienvenida.

  Si, por el contrario, pulsamos el botón de contenido favorito, pasaremos a la pantalla de selección de categoría del contenido.
    - **Categorías de contenido favorito:** Se trata de una pantalla que reutiliza el componente principal de la lista de categorías. Por ello, los elementos y las funcionalidades son las mismas, a diferencia que, al pulsar sobre alguna de las categorías, seremos redirigidos a una pantalla diferente, que nos mostrará otro listado de contenido.
    - **Lista de contenido favorito por categoría:** Pantalla a la que accedemos tras pulsar en alguna de las categorías anteriores. En ella, se muestra un listado de contenido, como ya ocurría en el Listado de contenido por categoría, haciendo uso de una FlatList vertical. En este caso, se nos muestra el contenido que hayamos marcado previamente como favorito, desde la pantalla de detalle. Si pulsamos en algún elemento de la lista accederemos a su detalle. Si no tenemos contenido de ese tipo marcado como favorito, la lista se mostrará vacía.
    - **Detalle de contenido.**



4. Descripción técnica

En este punto se va a describir de forma técnica las operaciones realizadas en código para los procesos y las transiciones descritas anteriormente.

4.1. Análisis Modelo de datos

Antes de realizar análisis a nivel de lógica de negocio, vamos a comentar el modelo de datos definido en la aplicación. El modelo de datos a nivel de base de datos, está formado por las siguientes entidades:

- **User:** Almacena información relativa a los perfiles de usuario creados en el dispositivo. Los campos de la tabla son:
  - Id: Identificador de la entidad de Usuario. Se trata de un campo numérico incremental.
  - nickname: Nombre introducido por el usuario en el momento de la creación del perfil. Se utiliza principalmente a la hora de identificar al usuario para restaurar su sesión junto al userToken.
  - userToken: Identificador único de sesión obtenido a partir del inicio de sesión del usuario en la aplicación. Los valores introducidos se generan a partir de la librería uuid.
  - Avatar: Campo encargado de guardar la imagen de usuario, que corresponde con su avatar. Este avatar, se asigna automáticamente al usuario entre los 5 disponibles.
  - createDate: Fecha y hora de creación del usuario en base de datos.

- **Content:** Almacena información relativa a los contenidos que los usuarios han añadido a sus listas de contenidos favoritos. Los campos de la tabla son:
  - Id: Identificador de la entidad de Contenido. Se trata de un campo numérico incremental.
  - apiId: Identificador único del contenido en la API de Rick y Morty utilizada. Permite asociar el registro en base de datos con la información alojada en la API, para ser consumida en cualquier momento.

- type: Campo numérico que permite identificar el tipo de contenido almacenado en base de datos.
- userId: Campo identificador del usuario al que pertenece el contenido. Se utiliza como clave ajena, a la tabla de User, y permite relacionar el contenido de ambas tablas, siendo una relación N-N. En caso de borrar el usuario, se borraría también su información almacenada en la tabla de contenido.
- createDate: Fecha y hora de creación del contenido en base de datos.

En base a estas entidades, se va almacenando la información durante el flujo de la aplicación.

Para cada uno de ellos se ha creado un "*manager"*, es decir un gestor donde centralizar las operaciones que se pueden realizar en base de datos, para cada tabla en cuestión.

Por lo que respecta al modelo de datos de la API, tenemos 4 manejadores encargados de interactuar con las diferentes entidades existentes en las 2 APIs utilizadas. Según la API utilizada, tenemos:

- Rick and Morty API: tiene toda la información estructurada y relacionada sobre los contenidos utilizados en aplicación. La url principal con toda la documentación es: https://rickandmortyapi.com/. Dispone de los 3 tipos/categorías de contenido comentadas anteriormente:
  - Personajes: https://rickandmortyapi.com/api/character
  - Ubicaciones: https://rickandmortyapi.com/api/location
  - Episodios: https://rickandmortyapi.com/api/episode

  Se ha creado un manejador para cada uno de ellos con las diferentes peticiones a realizar (todas de tipo GET).

- Fandom API: se trata de la API de una de las páginas con más información relativa a la serie de animación. Aunque no tiene la información tan estructurada como la anterior, permite ampliar los datos y descripciones sobre el contenido obtenido en la otra API.

  En este caso, se utiliza la siguiente url https://rickandmorty.fandom.com/api/v1/Articles/Details, filtrando por el título del contenido, para obtener 3 datos (en caso de estar disponible): Imagen,

descripción y enlace al artículo en Fandom. Estos dos últimos se utilizan en el detalle de todo tipo de contenidos. La imagen se recupera siempre en contenidos distintos a los personajes, ya que la Api de Rick y Morty, no tiene imágenes para esos dos tipos de contenido.

4.2. Análisis del código

El código de la aplicación se ha tratado de dividir en diferentes bloques, por organización y derivación de responsabilidades. Así pues, analizando las carpetas del proyecto contenidas en *"src"* podemos distinguir:

- ***Application***: Contiene todo el código relacionado con la lógica y modelo de la aplicación, apartado de la interfaz gráfica de componentes.
  - Modelo: Diferenciado en dos carpetas:
    - Base de datos: Con acceso a las tablas de las entidades vistas anteriormente.
    - Api: Donde se gestionan las peticiones a las APIs vistas anteriormente.
  - Managers: Podemos distinguir dos tipos de manejadores:
    - Los encargados de gestionar la lógica interna de cada componente
    - Aquellos encargados de comunicarse con las entidades del modelo.
  - Context: Scripts encargados de gestionar el contexto de la aplicación y controlar si el usuario actual está identificado y tiene autorización para acceder o no.
  - Container: Tiene un script para invertir y exportar las dependencias de los managers principales de la aplicación, es decir, aquellos que hacen referencia a las entidades de la API o de base de datos. Estos se "inyectan" en aquellos puntos del código que son necesarios, principalmente en los managers asociados a los componentes.
  - Data: Estructuras y tipos de datos que pertenecen al dominio de la aplicación.

- ***Screens***: Se trata de aquellos componentes principales que no son reutilizables y que constituyen las principales pantallas de la aplicación

- ***Components***: Se trata de aquellos componentes cuya vista puede ser reutilizada por otros componentes en diferentes pantallas.
- ***Navigation***: Scripts encargados de definir el orden y estructura de las pantallas de navegación asociadas a los distintos componentes.
- ***Utils***: Mezcla todo aquello que, sin tener lógica específica, puede ser reutilizado en distintos puntos de la aplicación. Entre ellos: estilos globales, constantes y comportamientos genéricos reutilizables por todos los componentes.

En la carpeta raíz, fuera del resto, encontramos el componente principal ***App.js***, que se encarga de obtener el contexto de la aplicación y en base a él renderizar la pila de navegación de bienvenida o la de usuario identificado.

Por lo que respecta a la parte visual, se ha tratado, en la medida de lo posible, de reutilizar estilos e interfaces haciendo uso de componentes genéricos, layouts y estilos css globales, aprovechando las opciones de reutilización que nos ofrece React.
Además de esto, se han aplicado ciertas animaciones a componentes con *Views*, que modifican la opacidad, escala y posición de los elementos en pantalla. Estas animaciones están asociadas a los estilos CSS y pueden verse la primera vez que se monta un componente cuando abrimos la App.

Para el análisis técnico de los flujos vistos en la descripción funcional, se va a tratar de seguir el mismo orden, es decir, en base a los 2 bloques principales:

- Flujo de navegación de bienvenida
- Flujos de navegación principal

4.2.1. Flujo de navegación de bienvenida

Al iniciar la aplicación, se carga el contexto y todas las acciones disponibles en él.
Durante esta carga, se intentan recuperar los datos de sesión del AsynStorage. En caso de existir, se recupera el *nickname* y el *userToken* y se llama al gestor de usuario, para consultar que esos valores coincidan con los almacenados en base de datos. De ser así, se ejecuta la acción de *signIn*, que redirige al usuario identificado a la Home. En caso contrario, no se hace signIn, por lo que se carga el flujo de navegación por defecto, es decir, el de bienvenida.

Al cargar la vista de este flujo, con una animación de opacidad, se consulta en base de datos todos los perfiles disponibles en la tabla de User, que serán enviados al

componente para ser pintados en la FlatList, reutilizando el componente AvatarProfileRow en el listado.
Si el usuario pulsa sobre alguno de estos componentes, se enviará un evento de click hasta el componente padre, que llamará a la acción *signIn* con la información del perfil seleccionado. Si todo va bien, se establecerá la sesión AsyncStorage y el contexto se encargará de redirigir al usuario a la Home. Además, se generará un nuevo userToken que será actualizado en base de datos, para el perfil en cuestión.
Por otro lado, si el usuario pulsa en el icono de borrado de algún perfil, se llamará al mostrado de una alerta con opción de confirmar o cancelar. Se cancela se cierra el modal y no ocurre nada. Pero si confirma, se llama al manejador de usuarios para hacer un delete en base de datos con el id del usuario en cuestión.
Por lo que respecta al formulario de creación de perfil, si el usuario introduce un *nickname* válido y pulsa en confirmar, se llamará al manejador de usuarios, para hacer un insert en la tabla de usuarios con el valor introducido y el resto de los campos. Pueden darse algunos errores en el proceso de creación de perfil que serán notificados en una alerta, por ejemplo: *nickname* duplicado, se ha excedido en máximo de perfiles (5), etc.

4.2.2. Flujos de navegación principal

Los flujos de navegación principal comienzan al cargar el BottomTabNavigator con las distintas pestañas de navegación disponibles. Por defecto, se carga al principio la de Home, que es la que analizaremos en primer lugar, pero también tenemos Categorías, Juego y Perfil.

4.2.2.1. Flujo de Inicio

El flujo de inicio empieza cuando carga el componente de Home, con el listado de contenidos sugeridos. Para realizar esta acción, en cuanto el componente está montado, se llama al "*manager*" de home para realizar una serie de generaciones aleatorias de valores, que seleccionan el tipo de contenido y la página a partir de la que va a obtener registros. Para conocer el número de páginas disponibles, es posible que la primera vez se requiera lanzar dos peticiones a la API. Por tanto, una vez conocido el número de páginas en la API y generados los valores aleatorios, se llama al gestor de API del contenido correspondiente. Con el resultado, se pinta un listado de 10 elementos en el FlatList de la Home. El componente reutilizado para este listado puede ser:

- CharacterRow en caso de ser contenido de personajes

- O bien, ContentRow, en caso de ser contenido de ubicación o episodio. Esto es así, porque ambos comparten una estructura muy similar a la hora de pintar la información, por lo que se ha decidido reutilizarlo. En cambio, los personajes, tienen algunos estilos diferentes, por lo que se ha preferido separarlo.

Además, se realizarán unas peticiones adicionales a la API de fandom, en caso de tratarse de un contenido diferente a personaje, ya que de este modo podemos recuperar imágenes que no nos ofrece la primera API para estos tipos de contenido. La petición se realiza por cada título de contenido. En caso de existir en Fandom, se recupera la imagen y se actualiza en el listado de la Home.

Al pulsar sobre alguno de los elementos de la lista, se emite un evento hasta el componente padre, que gestiona la navegación hacia el detalle de contenido. Esto es posible gracias a la prop que reciben los componentes “screen”, generada por la librería de *react-navigation.* Simplemente indicamos el nombre de la screen incluido en el stack de navegación y la librería asocia la nueva pantalla con el componente a renderizar, según le hayamos indicado.

Pasamos a analizar el detalle de contenido:

- **Detalle de contenido:** Al cargar el componente se realizan distintas operaciones:
  - Se realiza una petición a la API de Fandom para obtener todos los metadatos del contenido: imagen, descripción y url del artículo, a partir del título. Se incluirá esta información en la pantalla del detalle, en caso de existir.
  - Se realiza una petición a la API de Rick y Morty para obtener toda la lista de contenidos relacionados con el detalle de contenido en cuestión. Se agrupan los ids de los contenidos relacionados para ser consultados en una única petición. Si los contenidos son de tipo personaje se pintarán con un CharacterRow y en caso contrario con un ContentRow.
  - Finalmente, se realiza una consulta a base de datos con el id del usuario identificado y el id del contenido al que se ha accedido. De esta forma, se trata de recuperar de la tabla de Content, si se incluye entre los contenidos favoritos del usuario. De ser así se actualiza el icono del botón flotante, rellenándose para indicar que sí está incluido. Si no, se queda como está (sin rellenar).

Se controla aquí que el usuario pulse sobre el botón de añadir a favoritos. Esto supondrá una llamada al manager de Contenido, para insertar en base de datos un registro asociado el contenido con el usuario en cuestión, representando así que está en su lista de contenido favorito. En caso de ya estar en la lista, se realiza un borrado en la tabla de este registro y se actualiza la información a nivel de componente.

Si por el contrario el usuario pulsa sobre otro contenido de la lista relacionada, se hará un *push* en la prop de navegación para acceder al detalle de contenido con la información recibida en el evento del click.

4.2.2.2. Flujo de Categorías

Este flujo se inicia en cuanto el usuario pulsa sobre la pestaña de categorías, situada en la BottomNavigationTabs. Al cargar el componente principal del flujo, se pintan con una animación el listado de categorías de contenido disponibles. Esto se realiza simplemente al array contenido en el objeto ContentCategoryTypes, que se encuentra en la carpeta "*data"* de nuestra aplicación. Contiene los identificadores, los títulos y las imágenes de cada una de las 3 categorías. En la FlatList se pinta el componente reutilizable CategoryRow, y al hacer click en alguno de ellos, se nos redirigirá al listado de contenidos de la categoría seleccionada.

Pasamos a analizar ese listado de contenidos:

- **Listado de contenido por categoría:** El funcionamiento aquí es muy similar al ya visto en el listado de contenidos de la Home, a diferencia que el contenido renderizado no es aleatorio sino en base a la categoría seleccionada previamente. Se pinta por defecto la página 1 y se controla la posibilidad de desplazarse entre páginas, modificando el contenido de la FlatList. El componente que hace posible la paginación se llama Pagination y permite a partir de conocer el total de páginas y la página actual, desplazarse entre todas las opciones disponibles. Cada vez que pulsamos en un botón de paginación habilitado, se lanza una petición a la API de Rick y Morty con la página actualizada. En este caso se muestran 20 registros por página. Al igual que ocurría en la vista de Home, si pulsamos sobre algún contenido accedemos a su detalle, gracias a la prop de navegación.
- **Detalle de contenido:** Ya descrito en Home.

4.2.2.3. Flujo de Juego

Flujo que se inicia cuando el usuario pulsa sobre la pestaña de juego. En cuanto el componente principal de juego se monta, se muestra una animación con el contenido de la vista, pero no se realiza ninguna operación adicional.
En cuanto el jugador pulsa en el botón de jugar, se emite un evento para navegar a la siguiente pantalla de juego, la partida.

- **Partida:** Al cargar este componente se inicia un Interval de 3 segundos. Al terminar, se realiza una petición a la API de Rick y Morty, con el tipo de contenido de personaje y con una página aleatoria. Del contenido recibido se seleccionan 4 personajes y solo uno como opción correcta. Se presenta al jugador como imágenes en una lista, además del nombre del personaje de la opción correcta y se inicia otro Interval de unos 12 segundos. Si se termina el tiempo o el jugador responde, se finaliza la ronda y se muestra si es correcta la respuesta o no. En este momento, se limpian los intervals y solo manteniendo el contador de la ronda y las respuestas correctas del usuario. Una vez se completen las 5 rondas de preguntas, se redirige al componente de fin de partida.
- **Fin de partida:** Al cargar, se reciben como prop el número de respuestas correctas y el número de rondas totales. Esta información será mostrada al usuario además de un feedback. Aquí el usuario tiene la opción de volver a la pantalla principal de juego, o bien volver a jugar otra partida, proceso que limpiará todos los datos de la partida anterior para empezar de 0 una nueva.

4.2.2.4. Flujo de Perfil

Flujo que se inicia cuando el usuario pulsa sobre la pestaña de perfil. En cuanto el componente principal de juego se crea, se hace una consulta al *AsyncStorage* para recuperar de base de datos el avatar y el *nickname* de usuario identificado. Se muestra esta información en la vista del componente.
Aquí el usuario puede, o bien pulsar el botón de cambiar de perfil, opción que ejecuta la acción *signOut* del contexto y limpia el token del A*syncStorage*. Esto supone una redirección al Stack de bienvenida.
Por otro lado, si el usuario decide pulsar sobre el botón de contenido favorito, se emite un evento para que el prop de navegación redirija a la pantalla de Categorías de contenido favorito.

- **Categorías de contenido favorito:** Ocurre lo mismo que ya se ha descrito en la pantalla de categorías, pero al pulsar sobre un elemento, el evento emitido supone navegar al Listado de contenido favorito por categoría.

- **Lista de contenido favorito por categoría:** Al igual que ocurría en el Listado de contenido por categoría, se cargan los contenidos según la categoría seleccionada. Sin embargo, el listado en este caso dependerá de la información recuperada de base de datos. En el gestor de la entidad Content, se consulta por tipo y usuario y se obtiene un array con el contenido añadido. A continuación, se realiza una petición a la API de Rick y Morty para obtener la información a partir de los ids recuperados. Finalmente se obtienen las imágenes de los contenidos, en caso de no ser de tipo personaje.
  El usuario tiene la opción de pulsar sobre algun elemento de la lista, que le redirigirá al detalle.
- **Detalle de contenido.**

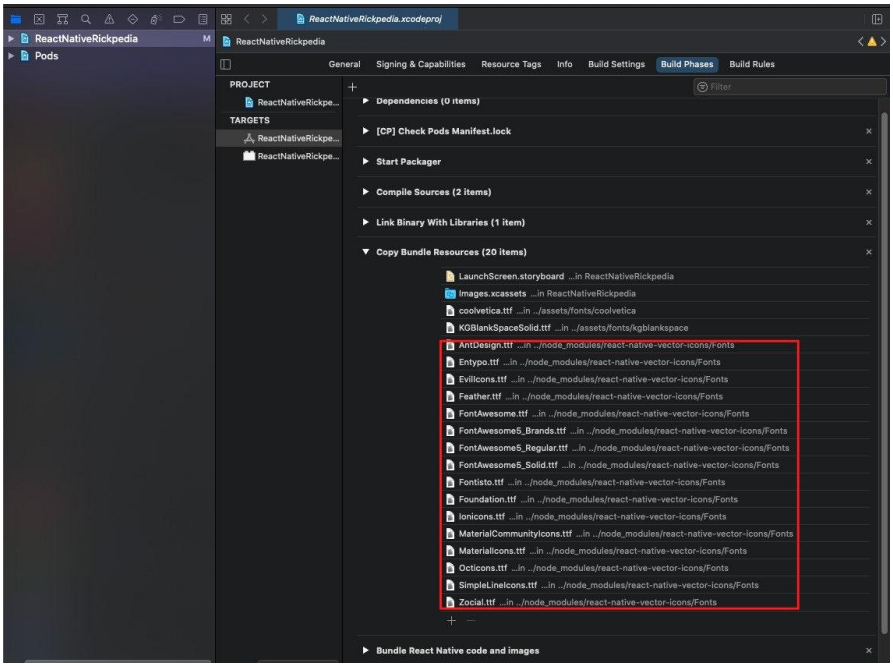
4.3. Inconvenientes técnicos del despliegue en ambas plataformas

Cabe decir que se han encontrado algunos problemas a la hora de hacer funcionar la aplicación en ambas plataformas. El desarrollo se ha hecho durante la mayor parte del tiempo trabajando en un Android. Pese a las primeras configuraciones del entorno, con Java, JDK y otras dependencias propias de Android Studio, no ha habido grandes problemas aquí.

Sin embargo, a la hora de probar la app en IOS, si se han encontrado algunas trabas. Las principales se han derivado de la librería *"react-native-vector-icons",* que generaba iconos desde varios puntos durante el proceso de build. Se ha conseguido solucionar desactivando el "linkado" automático de react-native, mediante el siguiente comando:

react-native link react-native-vector-icons

Hecho esto, se han eliminado los ficheros duplicados desde la siguiente sección de Xcode:



Con esto finalizado, el npm install completado y realizado el pod install en la carpeta de IOS, se ha conseguido hacer la build y arrancar correctamente el proyecto.

El siguiente "problema" encontrado ha sido con sqllite, que por algún motivo daba error al lanzar la sentencia de base de datos CREATE TABLE IF NOT EXISTS, en IOS si y en Android no. Se ha añadido una query previa para validar que existan las tablas antes de lanzar esta sentencia. De esta forma todo funciona correctamente, en ambos entornos.

5. Conclusiones

Este proyecto ha sido de gran utilidad para asimilar los conocimientos vistos en clase y ponerlos en práctica, aprendiendo muchas cosas en el camino.

Por lo general, estoy bastante satisfecho con el resultado obtenido, ya que se ha conseguido desarrollar una app en React Native que funciona tanto en Android como en IOS, y cumple los requisitos obligatorios planteados por la entrega. Aunque ya tenia algunos conocimientos previos de React, ha sido de gran utilidad poder realizar una aplicación en esta tecnología y ver sus particularidades, aprovechando todas las opciones que nos ofrece y la gran reutilización de código que facilita.

Sin embargo, se podría haber mejorado un poco el resultado desarrollando alguna vista o algún modulo nativo. También me gustaría haber incluido la lista de partidas jugadas como entidad de Base de datos, con la que registrar y consultar las partidas realizadas por cada uno de los usuarios de la aplicación en el dispositivo. La falta de tiempo y organización en estos últimos meses no ha permitido finalizarlo con todo lo esperado. Aun así, el resultado obtenido cumple con los objetivos propuestos.

Intentaré investigar por mi cuenta los requisitos opcionales pendientes, para así terminar de asimilar las posibilidades que ofrece esta tecnología.