Práctica 3 - React Native

Introducción y Objetivos

Requisitos:

- Nociones básicas de las tecnologías explicadas en clase (HTML, CSS, JS, React, **React Native**)
- Editor de código Visual Studio Code (o similar)
- Tener instalado node.js versión 16 (LTS), yarn, git y Expo
- Tener disponible un dispositivo físico con Expo Go o un simulador iOS o Android.

Motivación:

El cliente necesita que la aplicación que hemos desarrollado anteriormente (en la Práctica 2) esté disponible en las tiendas de aplicaciones App Store y Google Play. En el equipo habéis decidido utilizar React Native para hacer el desarrollo ya que os permitirá aprovechar el código desarrollado anteriormente y tener la aplicación disponible con muy poco esfuerzo. De nuevo para que el cliente esté satisfecho (y por lo tanto obtener el 10/10) debemos cumplir con las especificaciones de este enunciado y para asegurarlo nos proporcionan una batería de tests a pasar.

Objetivo:

El objetivo de esta práctica es realizar una aplicación nativa para Android e iOS de un catálogo de productos. Las funcionalidades son las mismas que en la aplicación React desarrollada en la Práctica 2 con el requisito adicional de que debe poder funcionar como aplicación nativa en dispositivos móviles. Para su desarrollo usaremos Expo y partiremos del código de la práctica anterior.

Preparación del entorno de desarrollo

Para poder hacer esta aplicación se proporciona un esqueleto o scaffold en github con el resultado de ejecutar el comando “expo init P3_RN_productos” y añadir la herramienta de autocorrección y los tests a pasar para superar esta práctica.

1. Clona el esqueleto de la aplicación a tu ordenador: `git clone https://github.com/REACT-UPM/P3_RN_productos_MOOC.git`
2. Entra en el directorio creado: `cd P3_RN_productos_MOOC`
3. Instala los paquetes necesarios que indica el package.json: `yarn install`
4. Inicia el servidor de desarrollo: `npm start`

IMPORTANTE instalar dependencias con `yarn` y no con `npm`. Para instalar yarn hay que ejecutar `npm install -g yarn`

A partir de ahora, podemos usar un simulador iOS/Android o un dispositivo físico con Expo Go para probar la práctica.

Ahora hay que abrir la carpeta P3_RN_productos_MOOC en el editor de textos y empezar a desarrollar nuestra aplicación.

Desarrollo

Recuerda los pasos que hemos seguido en clase para transformar una aplicación React en una de React Native. No es recomendable copiar y modificar todos los componentes de golpe sino ir progresivamente asegurándonos de que cada pieza va funcionando.

El estado, llamadas a APIs, fichero de configuración y la lógica de la aplicación será la misma que en React. Sin embargo, se han simplificado algunas de las funcionalidades para que sea más sencillo realizar la transformación, por ejemplo el desarrollo del selector de categorías es opcional. En caso de que desee implementarlo puede usar alguno de los paquetes proporcionados por la comunidad. Por ejemplo el componente "react-native-picker" mantenido por la comunidad y cuya documentación puede encontrarse en https://github.com/ouroboroscoding/react-native-picker.

Todos los componentes de la aplicación tendrán los mismos nombres que en React y deben estar en una carpeta llamada "components" excepto el componente principal App.js de React Native. El componente principal de la aplicación React se renombrará a HomeScreen.js y estará también en la carpeta "components".

La navegación se implementará usando Stack Navigator y debe inicializarse y configurarse en el componente App.js de la aplicación React Native. Utilice **Yarn** para instalar los paquetes necesarios para la navegación.

La renderización de la lista de productos se hará mediante un componente FlatList.

IMPORTANTE: Para la implementación de tests en aplicaciones React Native, los componentes se identifican mediante un atributo llamado `testID`. Por ejemplo:

```
<Text testID="mainView" >Hola!</Text>
```

Requisitos que debe cumplir nuestra aplicación

La práctica se valorará sobre un máximo de 10 puntos. La mayoría de los requisitos son de funcionalidad, la interfaz de usuario puede maquetarse como se desee aunque se valorará que los estilos y usabilidad sean adecuados.

Requisitos:

- **1 punto**: La aplicación tiene un componente Header con el logo y el mensaje de bienvenida con tu nombre (el nombre debe coincidir con el que le hemos dado al autocorrector la primera vez que lo ejecutamos).
    - El componente Header debe renderizar un elemento View con testID "cabecera"
    - Dentro de dicho View debe haber una imagen con testID "logo" y un Text con testID "mensaje"

    - El Text debe contener el texto que desees pero al menos debe poner tu nombre. Por ejemplo “Bienvenido a la página de Enrique Barra”
- **1 punto**: La aplicación, mientras carga, muestra una imagen con testID “loading”
    - Esto se puede hacer con un componente Image que muestre un gif o similar.
- **1 punto**: La aplicación tiene un componente SearchPage con el cuadro buscador con al menos un Text, un TextInput y un Button.
    - el Text debe tener el testID “catalogo” y contenido al menos la palabra “Catálogo”, el TextInput debe tener el testID “filtro” y el Button debe tener el testID “buscador”
- **2 puntos**: La aplicación tiene un componente SearchPage que renderiza los productos que recibe
    - El componente SearchPage tiene que recibir una prop “theproducts” con un array con los productos.
    - Los productos deben renderizarse con un componente FlatList.
    - La FlatList renderizará un item por cada producto. Cada item será un componente padre View con los hijos correspondientes a la imagen del producto, su título, etc.
        - Dicho componente View debe tener testID “item_<id_producto>” (por ejemplo “item_7”).
        - Uno de los hijos debe ser un componente Text con el título del producto y testID “title_<id_producto>” (por ejemplo “title_7”).
        - Otro de los hijos debe ser un componente Button para ver el detalle del producto y debe tener testID “button_<id_producto>” (por ejemplo “button_7”)
- **2 puntos**: La aplicación maneja el valor del input y filtra los resultados por su título al pulsar el button
    - Llamar a una función que coge el array de productos y utiliza el método “filter” de array para dejar en el array solo los que contienen el texto que ha introducido el usuario (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter)
- **2 puntos**: La aplicación tiene un componente para la pantalla de un producto
    - Esta página debe tener al menos un Text con testID “detalle” que contenga el título del producto y un Button con testID “volver” para poder volver a la página principal una vez visualizado el producto
- **1 punto**: La aplicación hace fetch de datos del servidor remoto
    - cuando el parámetro use_server vale true en el fichero de config se hace la petición al servidor y se utilizan los datos que devuelve
    - la url del servidor se obtiene también del fichero de configuración del parámetro server_url

Pruebas manuales y capturas de pantalla

Como se ha comentado se provee una batería de tests que la aplicación debe pasar y que serán los que produzcan la nota final. Pero es importante desarrollar la aplicación viendo poco a poco el resultado y visualizando cada cambio introducido hasta que funcione. Considerando los test provistos por el autocorector como una serie de requisitos “estrictos” o “estáticos”

como por ejemplo que un componente tenga determinado testID o que haya determinados textos en la aplicación.

No se recomienda por lo tanto ir desarrollando sin visualizar lo que hacemos y pasar el autocorector a cada paso “a ver si se consiguen los puntos”, porque los errores que dan las baterías de tests son más crípticos que lo que veremos en el dispositivo o en la consola por nosotros mismos al ejecutar la aplicación.

Adicionalmente a pasar la batería de tests y obtener un 10/10 hay que hacer dos capturas de pantalla con la aplicación completa, es decir en la versión final antes de entregar. Dichas capturas se tienen que ubicar en formato png, jpg o pdf en el directorio “miscapturas”. Y el autocorector las subirá junto con el código de la práctica y el resto de evidencias a Moodle. Estas capturas son obligatorias y deben ser personales, en ellas se debe ver que la cabecera de la práctica pone el nombre del alumno y que el estilo es el entregado en el código. Estas capturas serán revisadas por el profesor para comprobar que se ha realizado la funcionalidad correctamente.

En esta práctica hay que subir dos capturas equivalentes a las de la Práctica 2, es decir una de la página principal y otra de la vista de un producto.

Pruebas con el autocorector

El autocorector es la herramienta que permite pasar la batería de tests a la práctica y producir una nota. También subirla a Moodle junto con el código desarrollado, las capturas y otras evidencias de evaluación.

Ejecute el autocorector tantas veces como desee en la práctica y suba la nota a Moodle también tantas veces como desee hasta que se cierre la entrega. La nota válida será la última subida.

Dudas y tutorías

Para dudas sobre el enunciado y sobre la práctica en general vamos a utilizar el foro de la asignatura. Lo único que está prohibido es subir vuestra solución completa al foro en un zip o enlace o similar. Si se pueden compartir trozos de código para pedir ayuda o para ilustrar lo que os está ocurriendo.