ACTIVIDADES PRÁCTICAS:

Título Primeros pasos en React Native

Objetivos

Familiarizarse con el código JSX y el uso de componentes en React Native.

Temporalización

Dedicación estimada: 1,5 horas.

Actividades:

En este primer bloque de actividades prácticas nos familiarizaremos con React Native. Para ello, no serviremos de los componentes 'core' importados del módulo 'react-native'.

Trabajaremos a partir del código que se genera automáticamente en el archivo App.js cuando generamos el proyecto desde la terminal trabajando en local. Si prefieres trabajar con Expo Snack, solo tienes que copiar el siguiente código y pegarlo en el archivo App.js de modo que substituya al código existente.

```
import { StatusBar } from 'expo-status-bar';
import { StyleSheet, Text, View } from 'react-native';

export default function App() {
  return (
    <View style={styles.container}>
      <Text>Open up App.js to start working on your app!</Text>
      <StatusBar style="auto" />
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
```

Puedes hacer cada ejercicio en un archivo App.js diferente e ir progresivamente guardándolos conforme los tengas terminados -en una carpeta aparte si trabajas en local o descargando el proyecto si lo haces en Expo Snack-.

Ten en cuenta que el componente App.js es el punto de entrada de la aplicación, desde donde va a renderizarse todo lo que desarrolles y por tanto no debes borrarlo.

Ejercicio 1.

En el archivo App.js, dentro de la función App(){…}, las líneas de código que están incluidas dentro de return(…) son de código JSX. Aunque guarda muchas similitudes, no es código HTML. Es código JavaScript transpilado. Se ha programado para que sintácticamente sea similar a las etiquetas HTML con el objetivo de que su aprendizaje resulte más sencillo. Las particularidades del código JSX están explicadas con mayor detenimiento en las transparencias, no obstante, vamos a ver los aspectos que difieren respecto a HTML y que, por ello, es necesario asimilar pronto para su correcta utilización.

Los componentes 'core' son aquellos que ya vienen programados en react native para que podamos utilizarlos en nuestros desarrollos. Si trabajas en local, están guardados dentro de la carpeta react-native que se encuentra a su vez en la carpeta node_modules. Para poder utilizarlos, están importados en la parte superior del archivo App.js, en la instrucción siguiente:

```
import { StyleSheet, Text, View } from 'react-native';
```

Como ves, en una misma línea se están importando tres funcionalidades –los componentes 'core' Text y View y la funcionalidad StyleSheet que nos permitirá definir estilos para los componentes-. Se indican entre llaves porque están contenidos en un archivo con varias funcionalidades.

A continuación vamos a trabajar con el componente 'core' Text. Del mismo modo que en HTML tenemos la etiqueta <h1> para introducir un titulo de primer nivel, con <Text> introducimos un bloque de texto.

En el archivo App.js, en el interior de la función App, hay implementado una instrucción return que devuelve código JSX. Borra todos los componentes existentes y dejan sólo un componente Text –asegúrate de que existe etiqueta de cierre, es decir <Text>Open up App.js to start working on you app!</Text>-. Modifica el contenido del componente Text para que muestre por pantalla una frase distinta –es decir, cambia el texto escrito en el interior de las etiquetas de apertura y cierre de Text-.

```
export default function App() {
  return (
      <Text>Ejercicio 1</Text>
  );
}
```

Ejercicio 2.

Ahora vamos a ver cómo aplicar estilos a este componente Text, ubicándolos "en línea" –es decir, en el mismo componente-. Para ello, copia la siguiente línea e introdúcela en tu componente Text:

style={{flex:1, backgroundColor: 'red', alignItems: 'center', justifyContent: 'center'}}

Como puedes ver son similares a los estilos CSS con la excepción de que se escriben siguiendo "camelCase" –es decir, en lugar de background-color, se utiliza backgroundColor-. Fíjate que se indican entre dos llaves, luego veremos porqué.

Puedes probar a cambiar las propiedades partiendo de los conocimientos

que ya tienes de CSS y ver qué efecto tienen. Acuérdate de respetar la nomenclatura camelCase.

```
export default function App() {
  return (
      <Text style={{flex:1, backgroundColor: 'red', alignItems:
'center', justifyContent: 'center'}}>Ejercicio 2</Text>
  );
}
```

Ejercicio 3.

Los estilos en línea resultan prácticos, pero tal y como sucede con CSS, la convención es tenerlos aparte para poder reutilizarlos para todos los elementos que tengan los mismos estilos. Para ello hacemos uso de la funcionalidad StyleSheet importada de la librería react-native, que guarda los estilos en un objeto que por defecto nombraremos styles y donde definiremos las distintas propiedades, trabajando de forma muy similar a como lo hacemos en los archivos *.CSS. Para este ejercicio vamos a cambiar los estilos que hemos puesto en el ejercicio anterior y los volveremos a dejar en styles.container, tal y cómo estaba originalmente en el componente View que se crea por defecto al crear nuestro proyecto.

Puedes copiarlos de aquí:

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Observa que tenemos una constante styles donde guardamos un objeto que a su vez puede contener varios objetos. En este caso tenemos sólo un objeto –container, que guarda las propiedades ‘flex’, ‘backgroundColor’, ‘alignItems’ y ‘justifyContent’-. Para utilizarlas recurrimos a style={styles.container} de modo que estamos accediendo al objeto container incluido dentro de la constante styles. Podemos definir más objetos de estilos, tantos como necesitemos. Por ejemplo, podríamos definir otro objeto llamado footer donde indicaríamos los estilos del footer. Accederíamos a este objeto con style={styles.footer}.

```
export default function App() {
  return (
      <Text style={styles.container}>Ejercicio 3</Text>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Ejercicio 4.

A continuación, crea otro componente Text y ubícalo debajo del anterior. Aparecerá un error indicando:

Adjacent JSX elements must be wrapped in an enclosing tag. Did you want a JSX fragment <>...</>?

Es importante aprender a interpretar los errores. Este error surge porque todo lo que vaya a renderizarse dentro de return debe estar contenido dentro de un componente, de este modo, siempre deberá haber un componente que contenga el resto de componentes que vayamos definiendo. La forma de solucionar este error es ubicando un componente exterior dentro del cual queden ubicados los dos componentes. En este caso vamos a poner <></> (vacío).

```
export default function App() {
  return (
      <Text style={styles.container}>Ejercicio 4</Text>
      <Text style={styles.container}>Ejercicio 4</Text>
  );
}
```

Error:

```
export default function App() {
  return (
      <>
      <Text style={styles.container}>Ejercicio 4</Text>
      <Text style={styles.container}>Ejercicio 4</Text>
      </>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Ejercicio 5.

Otra opción que debes conocer es la de ubicar los dos componentes Text dentro de un View. Este componente es el equivalente a un *div* en HTML. Impleméntalo y cambia los estilos existentes de Text para que sólo apliquen a View, de este modo que los estilos se aplican a todos los componentesubicados dentro de View.

```
export default function App() {
  return (
      <View style={styles.container}>
      <Text>Ejercicio 5</Text>
      <Text>Ejercicio 5</Text>
      </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Ejercicio 6.

Anteriormente hemos visto que introducíamos dos llaves para poder definir los estilos en línea. Las llaves las utilizamos cuando queremos introducir código JavaScript en el interior de return(); A su vez, entre return y el inicio de la función App(), podemos incluir código. En este ejercicio vamos a ver cómo hacerlo. Introduce entre function App() y return(...) una variable -las variables pueden declararse con let y const, siendo let aquellas que se les van a asignar distintos valores durante el desarrollo y const las que permanecerán constantes-.

Para este ejercicio declárala con let y asígnale una cadena de texto (string) de tu elección. Posteriormente, en el interior de return ubica un componente Text contenido dentro de otro componente View. En el componente <Text></Text> indica entre llaves el nombre de la variable –es decir, <Text>{nombreDeLaVariable}</Text>-. Observarás que se renderiza por pantalla el contenido de la cadena de texto guardada en la variable.

```
export default function App() {
  let myString = "Esto es una cadena de texto";
  return (
      <View style={styles.container}>
      <Text>{myString}</Text>
      <Text>Ejercicio 6</Text>
      </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Ejercicio 7.

En este ejercicio vamos a repasar objetos en JavaScript para ver cómo podemos utilizarlos para trabajar con React Native. Si lo has olvidado, los objetos en JavaScript se declaran de la siguiente manera:

```
let person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};
```

Donde ‘person’ es el nombre del objeto y ‘firstName’, ‘secondName’, ‘age’ y ‘eyeColor’ son las propiedades del objeto. Accedemos a cada propiedad como se ha comentado en el ejercicio 3, por ejemplo: person.firstName nos devolverá el valor de la propiedad ‘firstName’ del objeto ‘person’.

A continuación define un objeto 'content' -con let- en el interior de la función App() y declara las propiedades 'paragraphOne' y 'paragraphTwo'. A cada una de ellas asígnale como valor una cadena de texto de tu elección.

Por lo general las variables las puedes declarar en el idioma que prefieras, no obstante es recomendable que te acostumbres a declararlas en inglés. Desde return, crea dos componentes Text dentro de un componente View y renderiza en uno de ellos el contenido de paragraphOne y en el otro el de paragraphTwo.

```
export default function App() {
  let content = {
    paragraphOne: "parrafo uno",
    paragraphTwo: "parrafo dos"
  };

  return (
      <View style={styles.container}>
      <Text>{content.paragraphOne}</Text>
      <Text>{content.paragraphTwo}</Text>
      </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
});
```

Ejercicio 8.

Añade un componente Text al contenido realizado en el anterior ejercicio y al objeto ‘content’ añádele la propiedad 'title' y asígnale una cadena de texto. En la constante styles fuera de la función donde se han definido los estilos, añade un objeto 'title' y asígnale un tamaño de letra de 12 puntos, negrita cursiva y subrayado. Recuerda que al ser código JavaScript debe ir en camelCase. Renderiza en cada componente Text una propiedad del objeto, de modo que en el primero se renderice el contenido de la propiedad ‘title’ del objeto ‘content’, en el segundo el contenido de la propiedad ‘paragraphOne’ y en el último el de ‘paragraphTwo’.

```jsx
export default function App() {
  let content = {
    title: "Este es mi titulo",
    paragraphOne: "parrafo uno",
    paragraphTwo: "parrafo dos"
  };

  return (
      <View style={styles.container}>
        <Text style={styles.title}>{content.title}</Text>
        <Text>{content.paragraphOne}</Text>
        <Text>{content.paragraphTwo}</Text>
      </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: '#fff',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 12,
    fontWeight: 'bold',
    fontStyle: 'italic',
    textDecorationLine: 'underline'
  }
});
```

Ejercicio 9.

Vamos ahora a ver cómo podemos realizar nuestro primer componente. Para ello vamos a partir del contenido realizado en el ejercicio anterior. Modifica el color de fondo de la propiedad 'container' de los estilos y asígnale un valor distinto a blanco para que se destaque del color de fondo. A continuación crea una función llamada Article -la primera letra con mayúsculas, al ser un componente se define con la primera letra en mayúscula- en el espacio entre function App() y return(). De este modo, la función Article –function Article()- estará contenida dentro de la función App(). Copia el contenido que en el ejercicio anterior tenías dentro de return (return incluido) y pégalo dentro de la función Article(). Nota: en Article tendrás un return con el siguiente código JSX: un componente View y tres componentes Text.

```
export default function App() {
  let content = {
    title: "Este es mi titulo",
    paragraphOne: "parrafo uno",
    paragraphTwo: "parrafo dos"
  };

  function Article(){
    return (
      <View style={styles.container}>
      <Text style={styles.title}>{content.title}</Text>
      <Text>{content.paragraphOne}</Text>
      <Text>{content.paragraphTwo}</Text>
    </View>
    );
  }
  return (
    <Article></Article>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'green',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 12,
    fontWeight: 'bold',
    fontStyle: 'italic',
    textDecorationLine: 'underline'
  }
});
```

Ejercicio 10.

Colocamos dos componentes Article y hacemos los ajustes necesarios para que se renderice dos veces -la ubicación de los componentes en la pantalla no es importante por el momento, lo importante es entender la mecánica de los componentes-. Recuerda que los componentes se enuncian como las etiquetas HTML, es decir: <Article></Article> o bien <Article/>

```
export default function App() {
  let content = {
    title: "Este es mi titulo",
    paragraphOne: "parrafo uno",
    paragraphTwo: "parrafo dos"
  };

  function Article(){
    return (
      <View style={styles.container}>
      <Text style={styles.title}>{content.title}</Text>
      <Text>{content.paragraphOne}</Text>
      <Text>{content.paragraphTwo}</Text>
    </View>
    );
  }
  return (
    <View>
      <Article></Article>
      <Article></Article>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 0,
    backgroundColor: 'green',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 12,
    fontWeight: 'bold',
    fontStyle: 'italic',
    textDecorationLine: 'underline'
  }
});
```

Ejercicio 11.

Creamos otro componente Articles y mediante un 'for' hacemos los ajustes necesarios para renderizar cuatro componentes Article.

Para ello deberás crear un array vacío donde ir añadiendo componentes Article en cada iteración –mediante push()-. En React Native los componentes que ubicamos en un array deben identificarse con key -es decir, debemos añadir en su interior un campo key como se indica a continuación <Article key={...}> y el valor que asignemos para key debe ser único. Una buena práctica es utilizar el valor de la posición del array convertida a string -usando toString()-. Más adelante veremos formas más óptimas de realizar este mismo ejercicio.

```
export default function App() {
  let content = {
    title: "Este es mi titulo",
    paragraphOne: "parrafo uno",
    paragraphTwo: "parrafo dos"
  };

  function Article(){
    return (
      <View style={styles.container}>
      <Text style={styles.title}>{content.title}</Text>
      <Text>{content.paragraphOne}</Text>
      <Text>{content.paragraphTwo}</Text>
    </View>
    );
  }
function Articles(){
  const articulos = [];
  for (let i = 0; i < 4; i++) {
    articulos.push(
      <Article key={i.toString()}/>
    );
}
  return articulos;
}

  return (
    <Articles/>
  );
}
```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'green',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 12,
    fontWeight: 'bold',
    fontStyle: 'italic',
    textDecorationLine: 'underline'
  }
});
```

Ejercicio 12.

Ahora vamos a crear una carpeta llamada Components dentro del directorio de nuestro proyecto –si trabajas en Expo Snack, la tendrás creada por defecto-. En esta carpeta creamos un archivo llamado Article.js

Esta carpeta la utilizaremos para ubicar aquellos componentes que vayamos a utilizar varias veces durante nuestros desarrollos –es decir, reutilizar-.

En el archivo Article.js copia el contenido de la función Article realizada en el ejercicio anterior y haz los ajustes pertinentes con el objeto donde habíamos guardado la información del componente. En la parte superior del archivo copiamos también las importaciones que tenemos en el archivo App.js. También los estilos del componente.

Para que el componente pueda utilizarse en otros archivos de nuestra aplicación, debemos exportarlo. Para ello ubicamos al final del archivo como última línea de código la frase export default Arcticle; o bien escribimos export default antes de la declaración de la función Article tal y cómo vemos en la función App contenida en el archivo App.js. En el archivo App.js, importamos Article siguiendo la nomenclatura import Article from y especificamos la ruta donde hemos ubicado el componente.

Hemos realizado una exportación por defecto –export default- porque únicamente

exportamos un componente o funcionalidad del archivo. Si quisiéramos exportar varios componentes ya no sería una exportación por defecto y utilizaríamos por ejemplo para exportar dos componentes Article y Title la instrucción export {Article, Title};

En este caso, además, al tener exportados varios elementos de un mismo archivo, para importarlos utilizaríamos las llaves. Es decir:

import { Article, Title } from ‘./Components/Article.js’;

Si quisiéramos importar únicamente un componente, también utilizaríamos las llaves:

import { Article } from ‘./Components/Article.js’;

Es importante entender cómo funcionan las importaciones y exportaciones de componentes porque suelen ser fuente de errores –diferenciar cuando se indican las llaves (exportación de varios componentes) y cuando no (exportación por defecto, con un único componente)-.

Finalmente desde el return de App renderizamos el componente Article importado.

```
function Article(){
    let content = {
        title: "Este es mi título",
        paragraphOne: "Párrafo uno",
        paragraphTwo: "Párrafo dos"
      };

    return (
      <View style={styles.container}>
      <Text style={styles.title}>{content.title}</Text>
      <Text>{content.paragraphOne}</Text>
      <Text>{content.paragraphTwo}</Text>
    </View>
    );
  }

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'green',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 12,
    fontWeight: 'bold',
    fontStyle: 'italic',
    textDecorationLine: 'underline'
  }
});

export default Article;
```

En App.js:

```
export default function App() {

  return (
    <Article/>
  );
}
```

Ejercicio 13.

Ahora que ya estás familiarizado con la creación de componentes básicos, es momento para que pongas en práctica los conceptos aprendidos y realices una modificación a tu componente Article. Para ello, añadirás una imagen, partiendo del componente 'core' de react-native Image, cuya documentación debes consultar para ir familiarizándote con la lectura de documentación técnica:

https://reactnative.dev/docs/image

La imagen que implementarás será de un link de internet. El componente, además, tendrá un título en la parte superior de la pantalla bajo el cual se ubicará la imagen y a continuación, un párrafo descriptivo de la imagen.

En Article.js:

```
function Article() {
  let content = {
    title: "Este es mi titulo",
    image: "https://media.istockphoto.com/photos/beautiful-sunset-
over-the-tropical-sea-picture-
id1172427455?k=20&m=1172427455&s=170667a&w=0&h=qU7HPik2cdJKmvN271uOe
0U9EcXe-59YN0yaMMkZ8wQ=",
    paragraph: "bla bla bla bla bla bla"
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>{content.title}</Text>
      <Image style={styles.image} source={{uri: content.image}} />
      <Text>{content.paragraph}</Text>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'green',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 12,
```

```
    fontWeight: 'bold',
    fontStyle: 'italic',
    textDecorationLine: 'underline'
  },
  image: {
    width: 200,
    height: 200,
  },
});

export default Article;
```

En App.js:

```
export default function App() {

  return (
    <Article/>
  );
}
```

Ejercicio 14.

Este último ejercicio deberás cambiar la imagen por alguna de las que están ubicadas en la carpeta 'assets'. Consulta la documentación oficial para ver cómo puedes acceder a las imágenes.

En Article.js:

```
function Article() {
  let content = {
    title: "Este es mi titulo",
    image: require('../assets/favicon.png'),
    paragraph: "bla bla bla bla bla bla"
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>{content.title}</Text>
      <Image style={styles.image} source={content.image} />
      <Text>{content.paragraph}</Text>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'green',
    alignItems: 'center',
    justifyContent: 'center',
  },
  title: {
    fontSize: 12,
    fontWeight: 'bold',
    fontStyle: 'italic',
    textDecorationLine: 'underline'
  },
  image: {
    width: 200,
    height: 200,
  },
});

export default Article;
```

En App.js:

```
export default function App() {

  return (
    <Article/>
  );
}
```