Curso de React y React Native

Clase 10

Agenda de la clase

Agenda

- React Native
  - Conceptos básicos.
  - Styling
  - Elementos
- Ejercicios.

React Native

React Native

React Native es un framework que permite construir aplicaciones *cross-platform* utilizando JavaScript. Comparte el mismo diseño que React, permitiendo componer la UI mobile partiendo de componentes declarativos.

```
import React, { Component } from 'react';
import { Text, View } from 'react-native';
class WhyReactNativeIsSoGreat extends Component {
 render() {
   return (
     <View>
       <Text>  If you like React on the web, you'll like React Native. </Text>
       <Text>
         Usamos componentes nativos como 'View' and 'Text', en vez de componentes web como 'div' and 'span'.
       </Text>
     </View>
   );
 }
}
```

React Native

No construiremos "Mobile web apps", ni "HTML5 apps", ni "Hybrid apps".

- En iOS, el componente de React Native ScrollView utiliza el UIScrollView nativo.
- En Android utiliza el ScrollView nativo.
- En iOS, el componente de React Native Image utiliza el UIImageView nativo.
- En Android utiliza el ImageView nativo.

React Native envuelve los componentes nativos fundamentales, permitiéndonos la performance de una app nativa, pero con el diseño de React y escribiéndola en JavaScript.

La documentación oficial se encuentra aquí.

React Native - Expo

La forma más sencilla de comenzar a trabajar con React Native es utilizando Expo, ya que no requiere instalación previa de Android Studio ni de Xcode. Con Expo solo tendremos un proyecto Javascript, al igual que en React web.

Advertencias: Debido a que Expo no genera ningún código nativo, no es posible incluir módulos nativos personalizados más allá de las API y componentes nativos de React que están disponibles en la aplicación cliente de Expo.

De todas formas, se puede utilizar el comando eject en un proyecto creado con Expo, y eso nos generará el proyecto nativo de Android e iOS como si hubiéramos creado el proyecto con react-native-cli

React Native - Expo

La documentación de Expo la podemos encontrar en este link.
Lo primero que necesitamos, el instalar el cliente de expo:

- npm install -g expo-cli

Luego, para crear un proyecto, ejecutamos el comando (en la carpeta deseada para el proyecto):

- expo init clase9
  - Cuando nos ofrece opciones de templates, por ahora crearemos el primero ‘blank’

Por último, entramos en la carpeta que creó y corremos el proyecto:

- cd clase9
- npm start

En MacOS es recomendable primero instalar Watchman

React Native - Expo

Podemos correr nuestras aplicaciones desde el Simulador de Android o de iOS, pero mejor aún, desde nuestro celular. Para eso necesitamos instalar la aplicación cliente de Expo.

También pueden registrarse/logearse en Expo por la terminal, y luego iniciar sesion en la app para acceder más rápido a sus proyectos.

- Android Playstore
- App Store

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

export default class App extends React.Component {
 render() {
   return (
     <View style={styles.container}>
       <Text>Open up App.js to start working on your app! </Text>
     </View>
   );
 }
}
const styles = StyleSheet.create({
 container: {
   flex: 1,
   backgroundColor: '#fff',
   alignItems: 'center',
   justifyContent: 'center',
 },
});
```

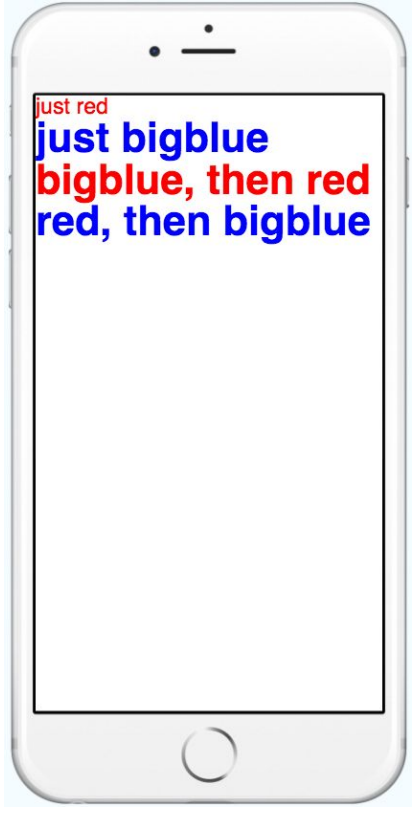
React Native - Styling

Todos los componentes que nos brinda React Native aceptan una *prop* llamada style, que recibe un objeto que permite describir los estilos que queramos aplicarle al componente.

Las claves y valores del objeto de Style suelen corresponderse con las propiedades de css de web, con la diferencia de que se escriben utilizando camelcase.

Por ejemplo, utilizamos `backgroundColor` en vez de `background-color`.

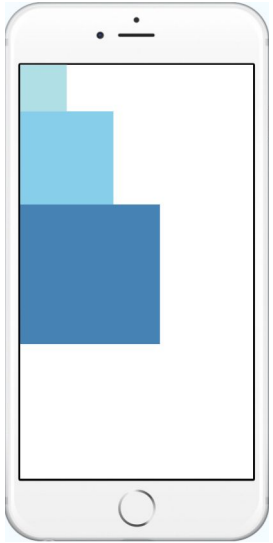
```
export default class LotsOfStyles extends Component {
   render() { return (
       <View>
           <Text style={styles.red}>just red</Text>
           <Text style={styles.bigblue}>just bigblue</Text>
           <Text style={[styles.bigblue, styles.red]}>bigblue, then red</Text>
           <Text style={[styles.red, styles.bigblue]}>red, then bigblue</Text>
       </View> );
   }
}
const styles = StyleSheet.create({
   bigblue: {
       color: 'blue',
       fontWeight: 'bold',
       fontSize: 30,
   },
   red: {
       color: 'red',
   }
});
```



React Native – Height y Width

El `height` y `width` de un componente en React Native determinan su tamaño en pantalla. Todas las dimensiones en React Native son sin unidad, representan píxeles independientes a la densidad del celular.

```
export default class FixedDimensionsBasics extends Component {
 render() {
   return (
     <View>
       <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />
       <View style={{width: 100, height: 100, backgroundColor: 'skyblue'}} />
       <View style={{width: 150, height: 150, backgroundColor: 'steelblue'}} />
     </View>
   );
 }
}
```



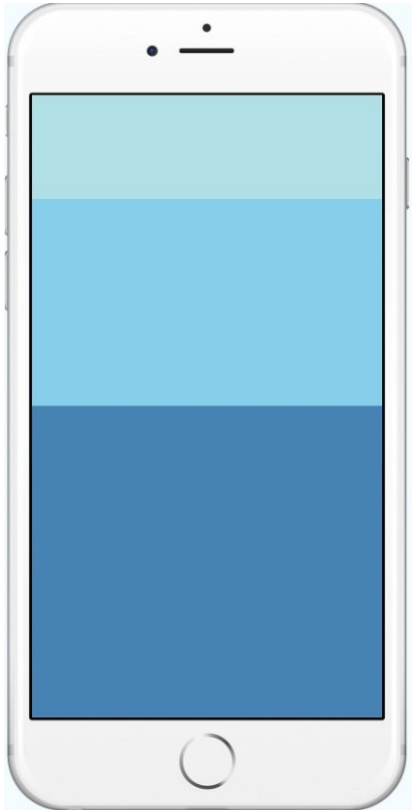
React Native – Flex

Si queremos que un componente se expanda y se contraiga según el tamaño de pantalla y el espacio libre, utilizaremos la propiedad `flex`. Si se le da un valor de `flex: 1`, el componente se expande y ocupará todo el espacio disponible, compartido uniformemente entre los demás hermanos.

Un detalle importante es que un componente tomará el espacio disponible basándose en el tamaño de su padre, por lo que si éste no tiene `height` y `width`, o `flex`, y ocupa 0x0, los hijos no se dibujarán.

React Native – Flex

```
export default class FlexDimensionsBasics extends Component {
 render() {
   return (
     <View style={{flex: 1}}>
       <View style={{flex: 1, backgroundColor: 'powderblue'}} />
       <View style={{flex: 2, backgroundColor: 'skyblue'}} />
       <View style={{flex: 3, backgroundColor: 'steelblue'}} />
     </View>
   );
 }
}
```



Si el View padre no tiene `flex: 1`, no se dibuja nada, y si tuviera un `height` fijo, se distribuyen los hijos en ese `height`.

```
import { AppRegistry, View, StyleSheet, Dimensions, Platform } from "react-native";
const deviceHeight = Dimensions.get("window").height;
const deviceWidth = Dimensions.get("window").width;
export default class FlexDimensionsBasics extends Component {
 render() {
   return <View style={styles.container}>
       <View style={{ flex: 1, backgroundColor: "powderblue" }} />
       <View style={{ flex: 2, backgroundColor: "skyblue" }} />
       <View style={{ flex: 3, backgroundColor: "steelblue" }} />
     </View>
 }
}
const styles = StyleSheet.create({
 container: {
   width: deviceWidth - 100,
   height: deviceHeight - 30,
   left: Platform.OS === "android" ? 5 : 30
 }
});
```

También podemos acceder a las dimensiones de la pantalla, e incluso aplicar distintos estilos según la plataforma.

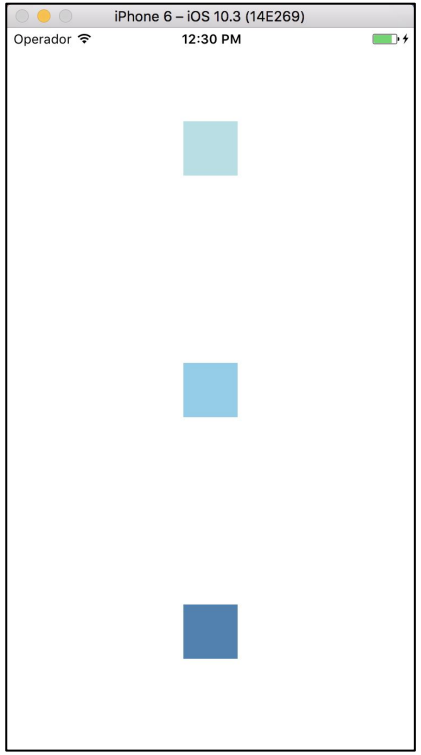
React Native – Flexbox

En cada componente podemos utilizar Flexbox (Guide to Flexbox css)para determinar el layout de los hijos, de forma de garantizar un layout que se mantenga consistente entre distintas pantallas.

Las propiedades que más comúnmente se utilizan son:

- `flexDirection`: Determina el eje principal, si es horizontal (`row`) o vertical (`column`). Por defecto es **vertical**.
- `justifyContent`: Determina la distribución de los hijos a lo largo del eje principal, ya sea al comienzo, al final, en el centro, separados equitativamente, etc.
- `alignItems`: Determina la distribución de los hijos a lo largo del eje secundario.

```
class AlignItemsBasics extends Component {
 render() {
   return (
     <View
       style={{
         flex: 1,
         flexDirection: "column",
         justifyContent: "space-around",
         alignItems: "center"
       }}
     >
       <View style={{ width: 50, height: 50, backgroundColor: "powderblue" }} />
       <View style={{ width: 50, height: 50, backgroundColor: "skyblue" }} />
       <View style={{ width: 50, height: 50, backgroundColor: "steelblue" }} />
     </View>
   );
 }
}
```



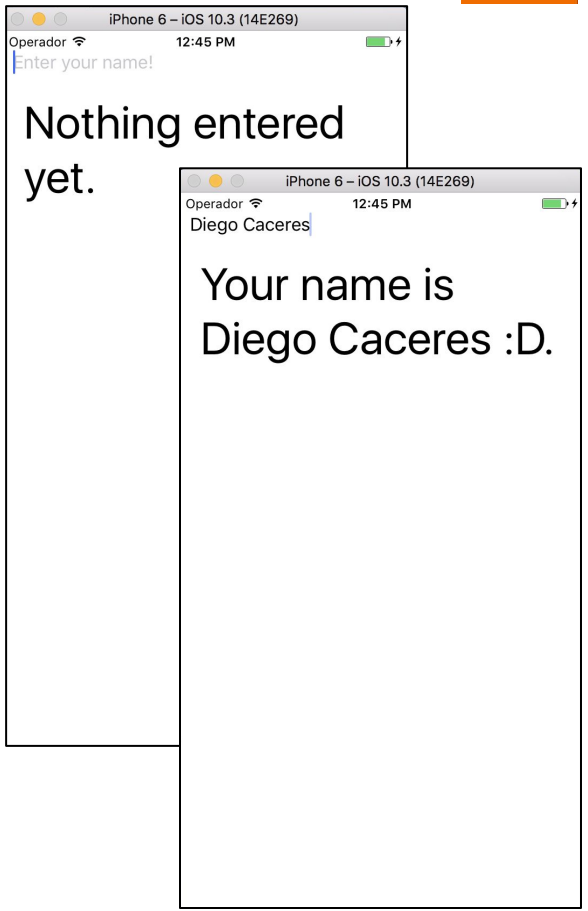
React Native – TextInput

Para crear inputs en React Native se utiliza el componente TextInput.

A diferencia de la web, donde se tiene el evento `onChange` para los `<input>`, al utilizar `TextInput` se cuenta con el evento `onChangeText`.

También existen otras propiedades comunes como `placeholder`, `multiline`, `numberOfLines`, `secureTextEntry`, `keyboardType`, etc.

```
export default class TextInputExample extends Component {
 constructor(props) {
   super(props);
   this.state = { text: "" };
 }
 render() {
   return (
     <View style={{ padding: 10 }}>
       <TextInput style={{ height: 40 }} placeholder="Enter your name!"
         onChangeText={text => this.setState({ text })}
       />
       <Text style={{ padding: 10, fontSize: 42 }}>
         {this.state.text ? `Your name is ${this.state.text} :D.`
           : "Nothing entered yet."}
       </Text>
     </View>
   );
 }
}
```



React Native – Button

En React Native tenemos un componente `Button` que se dibujara de forma distinta para cada plataforma, siguiendo los estándares de la plataforma.

```
import { AppRegistry, View, Button, Alert } from "react-native";
export default class ButtonExample extends Component {
 render() {
   return (
     <View style={{ padding: 10 }}>
       <Button onPress={() => { Alert.alert("You tapped the button!"); }} title="Press Me" />
     </View>
   );
 }
}
```

React Native – Button

En el caso de que se quiera personalizar la visualización el botón o se quiera que algún otro componente sea "clickeable", se puede utilizar alguno de estos otros componentes que nos brinda React Native:

- TouchableHighlight
- TouchableOpacity
- TouchableWithoutFeedback

React Native – Extras

- En el caso de React Native, para poder inspeccionar y debugguear nuestras aplicaciones, podemos hacer uso de las React Dev Tools separadas de Chrome.

- Otra buena herramienta es React Native Debugger que incluye también las Redux Dev Tools.

- Comandos:
  - Command + R (iOS) / R+R (Android) => Recarga la aplicación.
  - Command + D (iOS) / Command + M (Android) => Abre el menú de desarrollo con varias opciones.
  - Shake (dispositivos físicos) => Abre el menú de desarrollo con varias opciones.

React Native – Extras

- A veces cuando tengamos un error que no podamos identificar, cambiando de un proyecto a otro, conviene realizar una ‘limpieza’ de watchman, del cache de npm y de react native, por lo que el siguiente comando puede ser útil:

```
watchman watch-del-all && rm -rf $TMPDIR/react-* && rm -rf node_modules/ && npm cache clean --force
```

Ejercicios

Ejercicio

1. Crear un proyecto de react-native con expo. `expo-cli` es particularmente pesado así que lo descargamos por única vez globalmente: `npm install -g expo-cli` y luego podrán corremos `expo init` cuando quieran para crear un proyecto.
2. En el proyecto recién creado, poner un `<TextInput>` que reciba un valor numérico como **prop** llamada `defaultCount`. Dicha prop será el **valor por defecto** del `<input>`.
3. Luego, agregar un `<Button>` que diga START.
4. El usuario podrá ingresar en dicho `<TextInput>` un valor alternativo, que reemplazará el valor que el componente tenía por defecto al ser invocado.
5. Cuando el usuario clickea en START aparecerá en pantalla un nuevo componente: el cual irá mostrando una cuenta regresiva de segundos. Dicha cuenta regresiva comenzará, sí, lo adivinaron; con el valor del `<TextInput>` anterior, hasta llegar a 0.

Ejercicio

Además:

- Cuando sólo queden 5 segundos, poner el contador de color ROJO.
- No repetir los bugs del ejercicio análogo de la clase 2.
- Probar las distintas alineaciones posibles con flexbox.