WELCOME TO scope

Welcome to the first Scope meeting for the Spring 2021 semester! Today we'll be going over some essential Scope information, and then beginning the first lesson of our React Native curriculum!

QR Code for Attendance

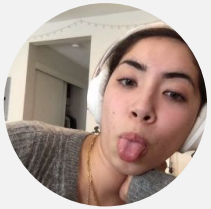
</>

Please fill out this attendance form! At the end of the semester, prizes for participation may or may not be awarded based upon attendance and other secret to-be-determined factors :^)

MEET THE BOARD

President

Claire



Curriculum

Julia

Curriculum

Ryan

People

Jeff

Events

Josh

Media

Jady

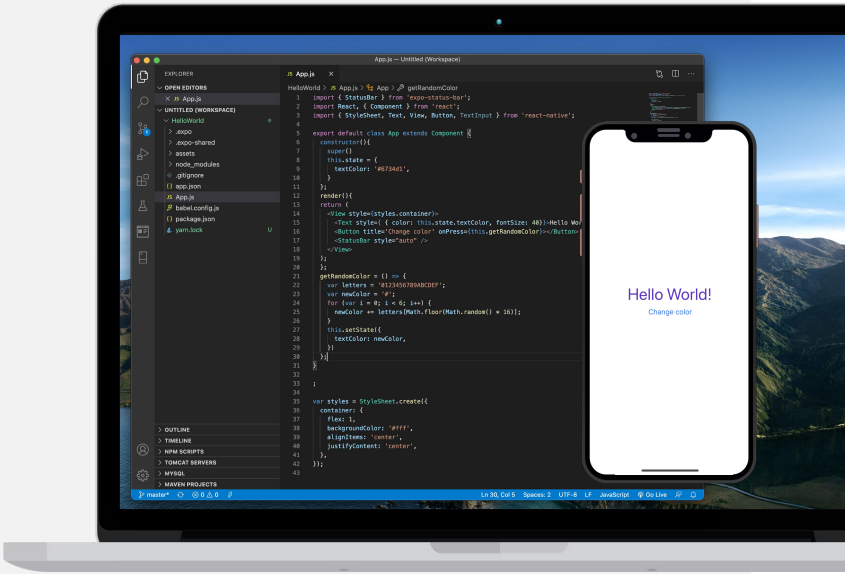
TODAY'S LESSON :^)

What you'll build:

A Hello World app using React Native UI components, with a colorful flair!

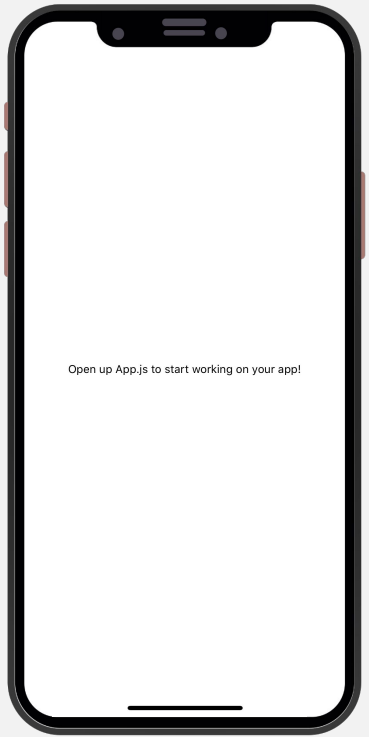
What you'll learn:

The basics: essential parts of React Native, and how to start creating apps



CREATE A PROJECT

- Run `expo init HelloWorld` in a folder where you want your Scope projects to live
- Select “blank”
- Run `cd HelloWorld`
- Run `npm start`



LOAD PREVIEW

On your phone:

1. Scan QR code generated by Expo with your phone
2. Launch preview in Expo Go
3. Exit preview by three-finger long-press

Or, in your browser:

1. Select the “Run on iOS Simulator” or “Run on Android device/emulator” options

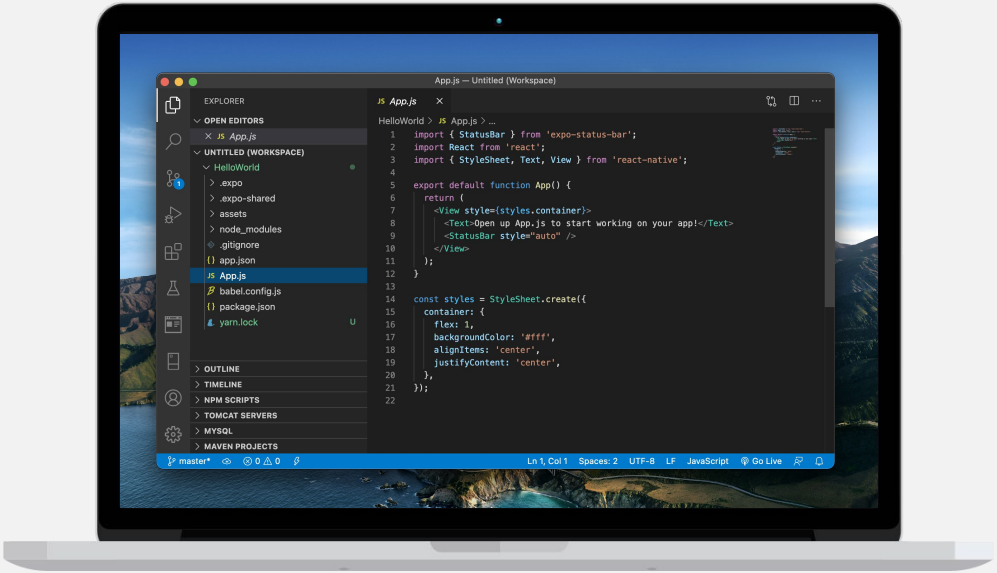
APP.JS

On computer:

1. Navigate to HelloWorld folder
2. Open App.js in text editor

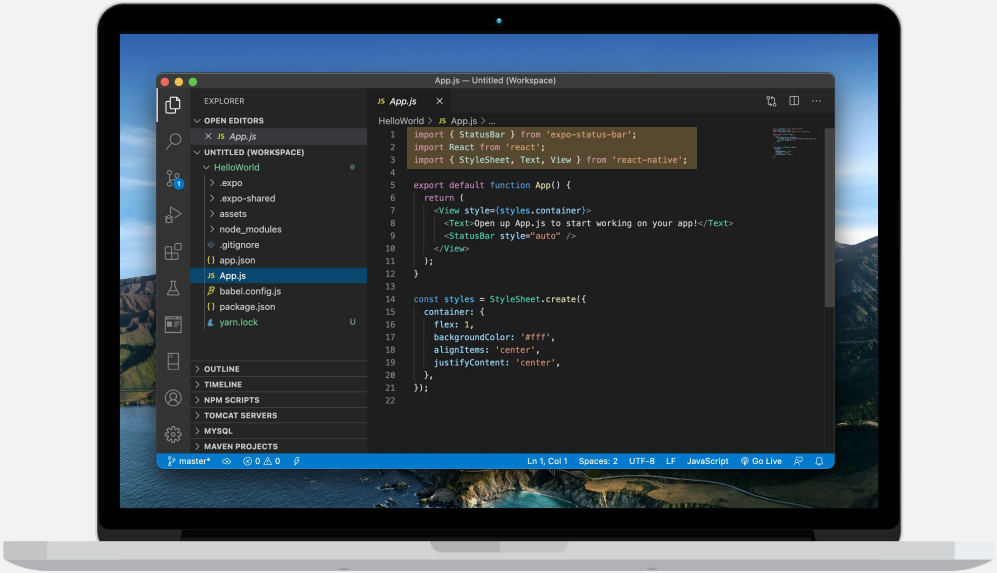
App.js is the entry point to our application. It contains:

1. Imports
2. Export App
3. Globals



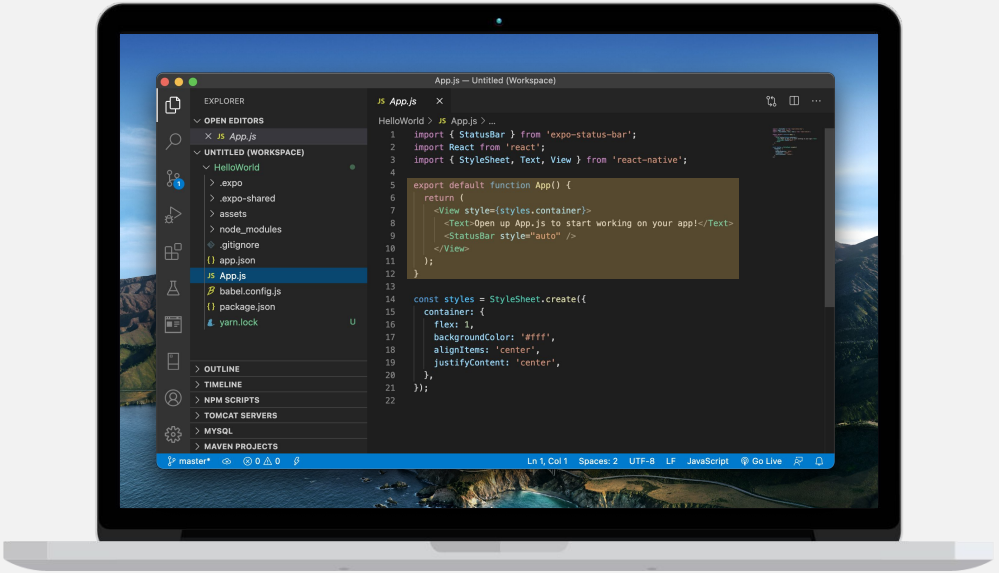
IMPORTS

- Allow us to use modules outside our current JS file
- External views, objects, UI components, etc.



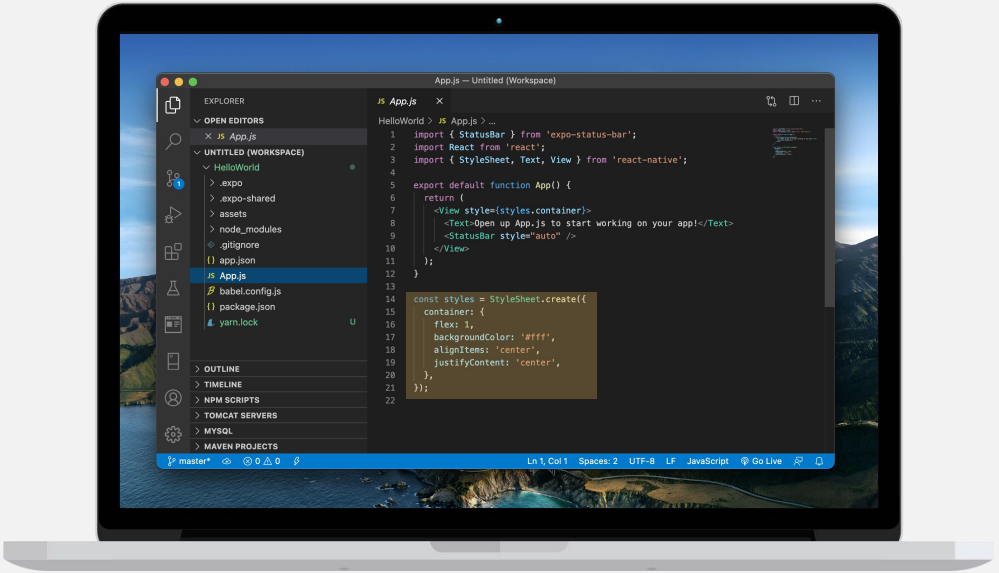
APP

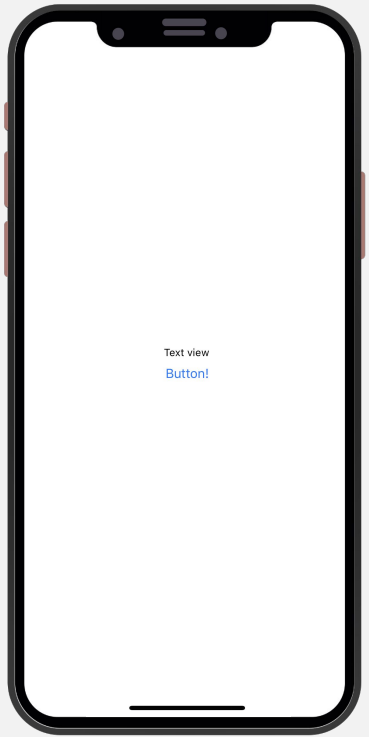
- Entry point of our application
- Returns a singular View
    - This can be a navigation controller!
- View contains all of the UI components of the main window
- Functions can be contained within the export



GLOBALS

- Any global constants or variables you need for your program
- Stylesheets are usually included here, can be constants or variables
    - Similar to HTML

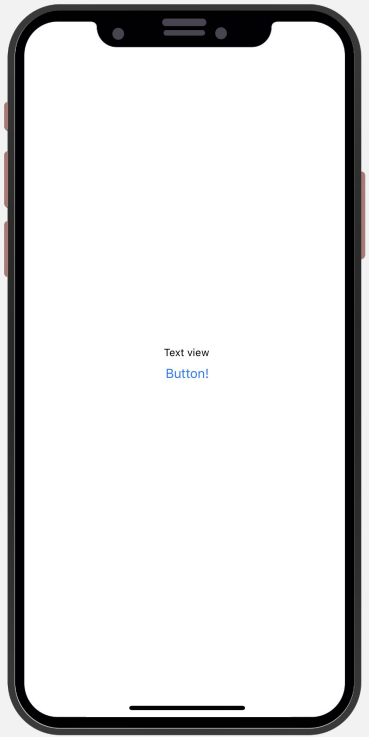




REACT NATIVE UI

Basic UI components:

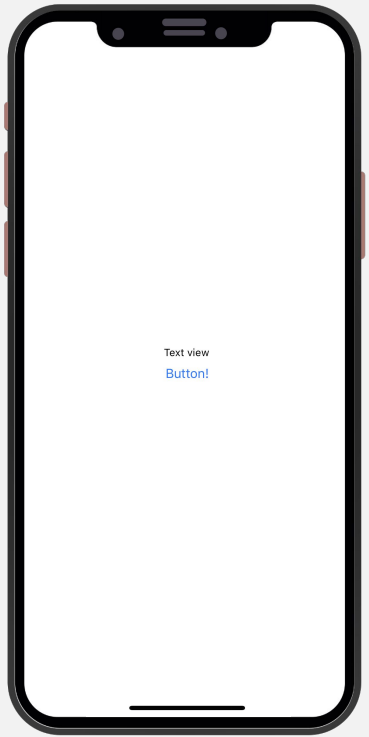
1. View
2. Text
3. Button
4. TextInput – Entry
5. Image
6. Switch
7. Many more!



VIEW

A view is essentially a list of all the UI components on a screen. Views can be nested, and can have unlimited children.

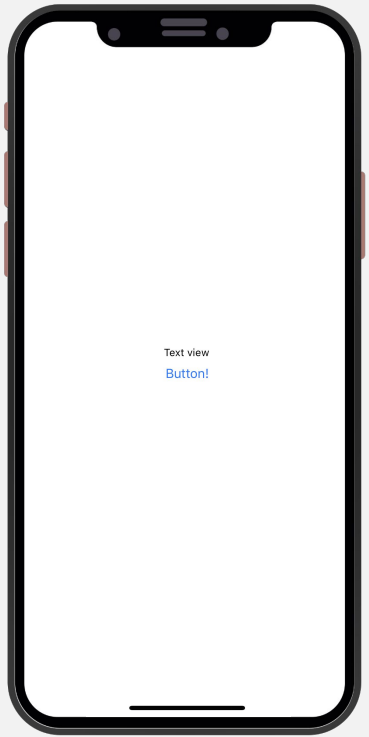
```
<View style={styles.container}>
    {/* Objects inside your view go here */}
</View>
```



TEXT

It's just text. Here's how you do it. Styling tips in a bit.

```
<Text>Text view</Text>
```



BUTTON

It's just a button. You can put text in it and you can make it execute some function. Riveting.

```
<Button title='Button!'></Button>
```

PROGRAMMING BASICS

React Native uses JavaScript. While JavaScript can be similar to other native compiled languages we learn in class, both JS and React Native have their own programming conventions.

VARIABLES & STATE

Local class variables in React are usually stored in the “state”, which is data tied to our component that will change throughout its life cycle. State is normally put in a constructor.

FUNCTIONS

Functions are written normally using JavaScript conventions. They can be local methods, belonging to objects, or global functions.

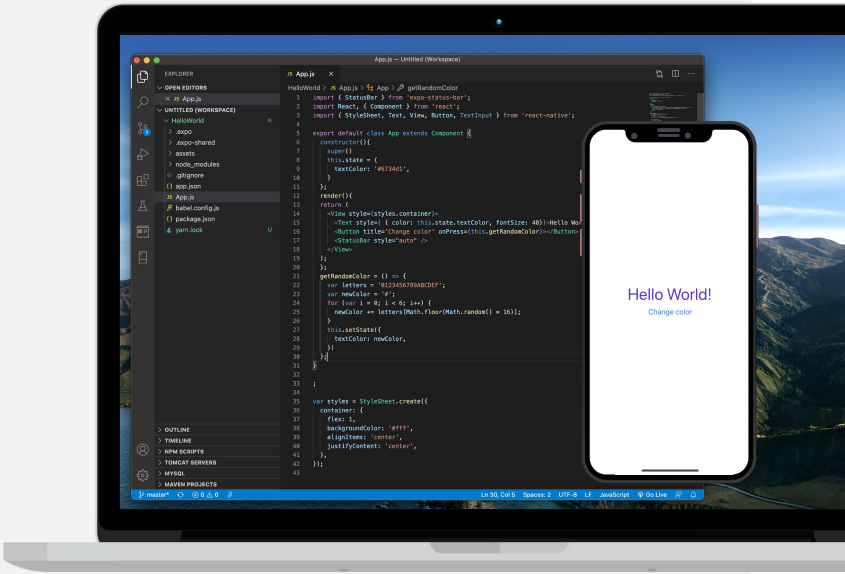
HELLO WORLD APP

Skills we are using:

- How to create a React Native app using Expo CLI
- Basic implementations of View, Button and Text UI components
- State to store variables

Final product:

Basic Hello World app with a button that changes the ‘Hello World’ text color



SETTING UP

First, we need to take several steps to set up our App.js before we start adding UI to the view.

1

Export a class, not a function

Replace the line

```
export default function App() {
```

With:

```
export default class App extends Component {
```

This allows us to place functions within the App class, rather than App being a singular function.

2

Import Component

In the

```
import React from 'react';
```

line, add `React, { Component } from 'react';` so that we are importing Component from React. A component is a type of class that provides additional features.

3

Create the render() method

Surround the `return ( );` method with the render method. Add `render(){` on line 6 before the 'return', and add a closing curly brace after return's closing parenthesis on line 11 (before the semicolon). The return method is required to implement the Component.

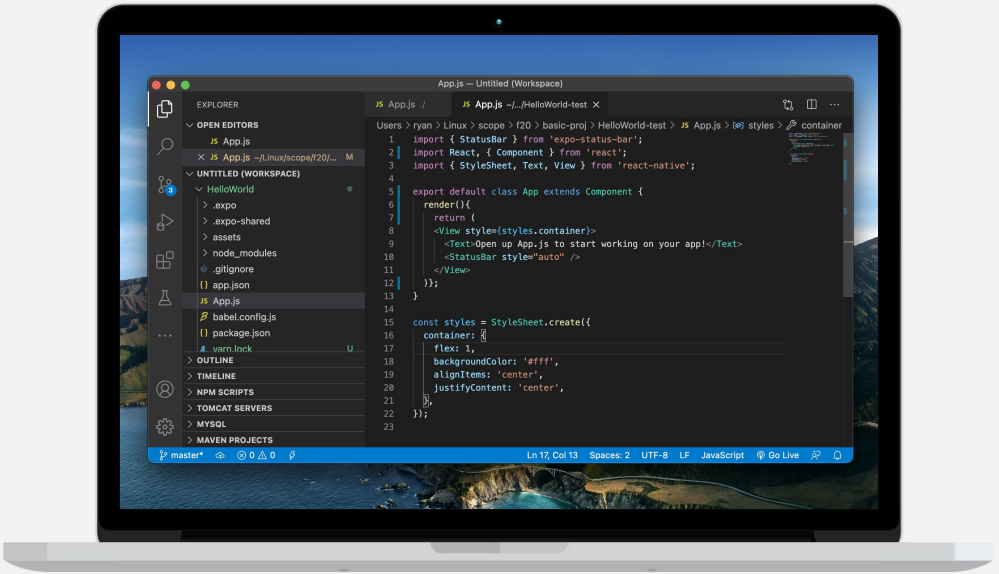
CODE

Your current code should read as shown:

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

export default class App extends Component {
 render(){
   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',
 },
});
```



ADD ELEMENTS

Now we can add a button, and implement some color-changing functionality.

1

Import the button

Add

`Button`

To the list of React element imports at the top of App.js, so the line reads::

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

2

Add the button

Below the `</Text>` element, add a title so the button appears onscreen with its own text:

```
<Button title='Change color'></Button>
```

3

Add a State for the text color

We need a state for the App class so we can track the color of the text, since it's a variable. Add the following to the App class:

```
constructor(){
   super()
   this.state = {
     textColor:
'#6734d1',
   }
 };
```

4

Add style to the text

Change the text inside to read whatever you want, and add the following to the text element:

```
style={{ color: this.state.textColor, fontSize: 40}}
```

FUNCTIONALITY

Finally, we can add the method to change the color of the text, and link it to the button we just created

1

Color-changing function

Finally, we need a function to generate a random hex color code for our text. This function procedurally chooses a base-16 digit for each position of the color. Add it to our

```
getRandomColor = () => {
  var letters = '0123456789ABCDEF';
  var newColor = '#';
  for (var i = 0; i < 6; i++) {
    newColor += letters[Math.floor(Math.random() * 16)];
  }
  this.setState({
    textColor: newColor,
  })
};
```

2

Link function to button

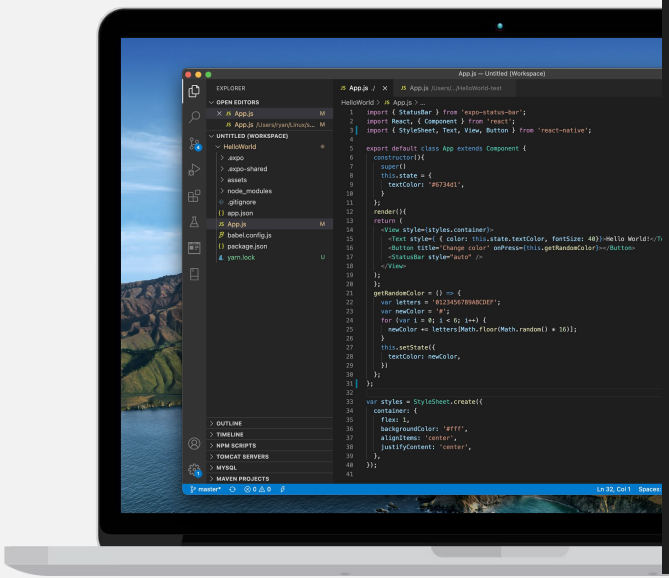
In order to tell the button what to do when pressed, we need to add the onPress attribute, and tell the button which function to call.

Add the following as an attribute to our Change Color button:

```
onPress={this.getRandomColor}
```

FINAL CODE

Your current code should read as shown:



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

export default class App extends Component {
 constructor(){
   super()
   this.state = {
     textColor: '#6734d1',
   }
 };
 render(){
 return (
   <View style={styles.container}>
     <Text style={ { color: this.state.textColor, fontSize: 40}}>Hello World!</Text>
     <Button title='Change color' onPress={this.getRandomColor}></Button>
     <StatusBar style="auto" />
   </View>
 );
 };
 getRandomColor = () => {
   var letters = '0123456789ABCDEF';
   var newColor = '#';
   for (var i = 0; i < 6; i++) {
     newColor += letters[Math.floor(Math.random() * 16)];
   }
   this.setState({
     textColor: newColor,
   })
 };
};

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

WRAP UP

Hopefully today taught you some basics of React Native, and how to start prototyping apps! Next week we'll go a little more in depth