SPACE APP IN REACT NATIVE

BY QCC - TECH WORKS

GOAL

- Build a React Native application that will display pictures from space.
- Have the ability to go back and forth through the pictures one at a time.
- Apply styling using the StyleSheet component.
- Use methods for handling click events.

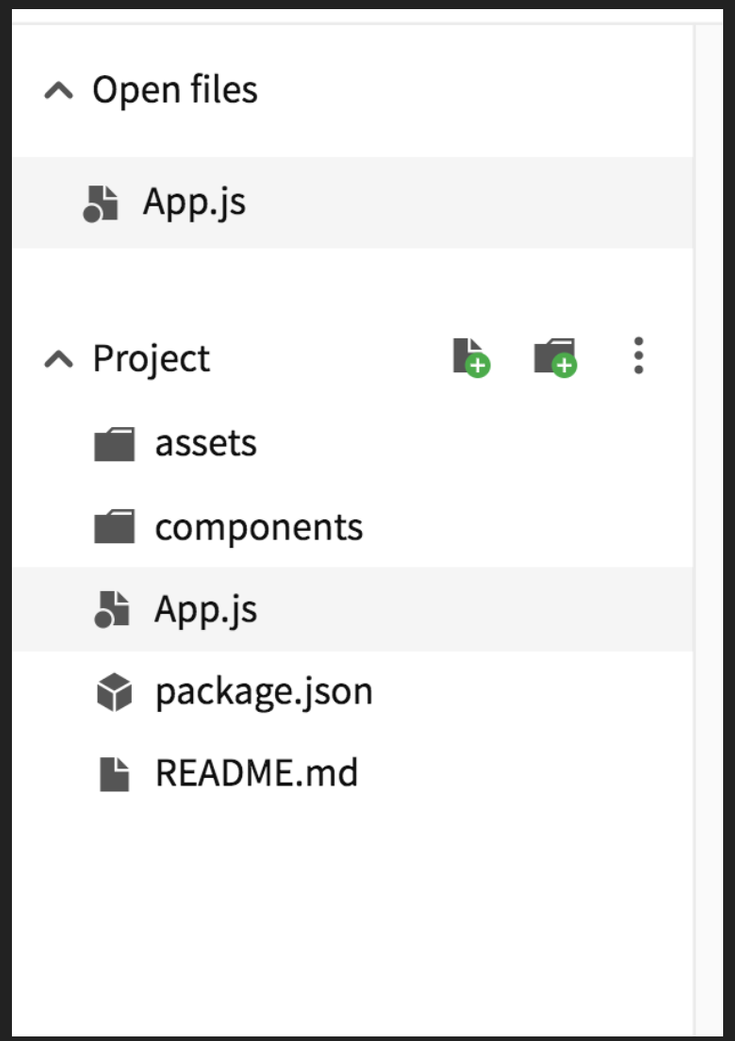
WHERE TO START.

- If you have your own environment set up (personal Mac or Windows laptop) that’s great. For everyone using the student computers. Let's use an online editor and emulator.

- https://snack.expo.io

- Sign up with expo - to make sure you can save your projects.
- We can also through the QR codes run this app from your phone if you install the Expo app on your Android phone (Sorry iOS it doesn’t work anymore)
- Now let's look at the editor.

- First, we have to the far left our files listed below our project.
- This should look pretty familiar. Looks like a simple React application directory structure.

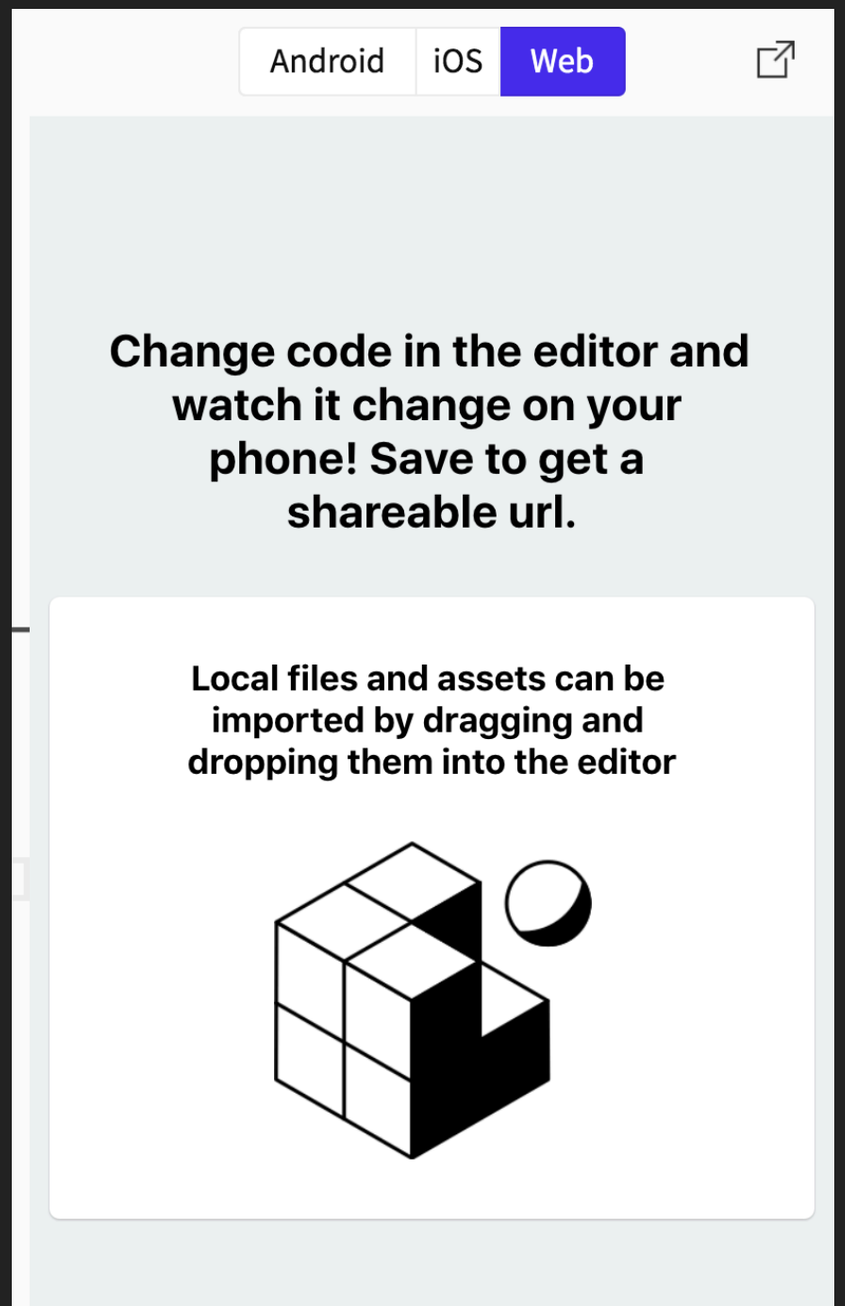


SPACE APP IN REACT NATIVE

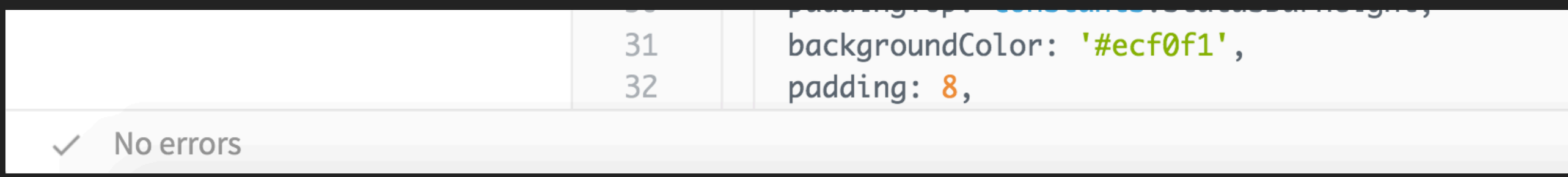
- Our main window is obviously our main development environment. This is where we'll be adding in our logic for our RN applications.

```
1   import * as React from 'react';
2   import { Text, View, StyleSheet } from 'react-native';
3   import Constants from 'expo-constants';
4
5   // You can import from local files
6   import AssetExample from './components/AssetExample';
7
8   // or any pure javascript modules available in npm
9   import { Card } from 'react-native-paper';
10
11  export default class App extends React.Component {
12    render() {
13      return (
14        <View style={styles.container}>
15          <Text style={styles.paragraph}>
16            Change code in the editor and watch it change on your phone! Save to get a shareable
    url.
17          </Text>
18          <Card>
19            <AssetExample />
20          </Card>
21        </View>
22      );
23    }
24  }
25
26  const styles = StyleSheet.create({
27    container: {
28      flex: 1,
29      justifyContent: 'center',
30      paddingTop: Constants.statusBarHeight,
```

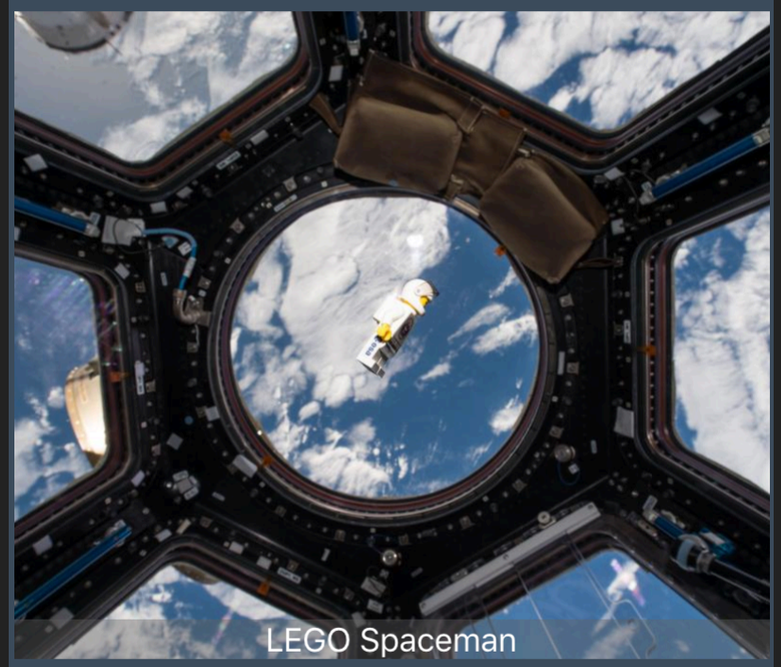
- To the far right is our preview pane. This is where we can see our application come to life. This is where everything is rendered... either in an Android simulator / iOS simulator or a web view.



- We also have access to a console. Which is at the bottom of the IDE… that should always say No Errors :). You can click on this window and it will open the logs and errors window that can help you troubleshoot your program.



- Okay - now that we are friends with this editor. We need to start thinking about what we're trying to create.



We want to make an application that will have a pictures that we can click through - of something in space, and has a caption of that picture.

- First things first... let's delete everything in the editor and start from scratch. Then, let's build out the base of our component.

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


export default class App extends Component {
 render() {
   return (
    <View style={styles.container}></View>
   );
 }
}

const styles = StyleSheet.create({
 container: {
   flex: 1,
   justifyContent: 'center',
   alignItems: 'center',
   backgroundColor: '#F5FCFF',
 },
```

- Okay now that we have a blank slate... lets think about what we want this app to do...
  - We want to pull in some images of space
  - We want to display one image at a time
  - We want to be able to cycle through the images in both directions based on where the image is clicked (left or right side of the image)
  - What we want for the caption of the image

- First - lets make an object that we can use to store the data for the images.
- Add this to your app.js file below the exports and above your class component. We want to use this array as a global variable in the module so we can reference it in our app.

- Let's add some styling to our app before we start adding logic into our project.
  - First let's change that background color - I know we really don’t have to but… its good practice.
  - Change the background color hex to ‘#3D4B5E We should see the background color reflected in our app preview.
  - Then let's add two new style objects to handle our image and our caption.

- React Native uses flex box to arrange the layout. The numbers determine which component to prioritize when rendering. We're giving the container priority over the image since we want the image inside the container when the app renders.

```
17  const styles = StyleSheet.create({
18    container: {
19      flex: 1,
20      justifyContent: 'center',
21      alignItems: 'center',
22      backgroundColor: '#3D4B5E',
23    },
24    image: {
25      flex: 2,
26      width: 320,
27      justifyContent: 'flex-end',
28      alignItems: 'center'
29    },
30    imageCaption: {
31        textAlign: 'center',
32        backgroundColor: 'rgba(100, 100, 100, 0.5)',
33        color: 'white',
34        width: 320
35    },
36    empty: {
37        flex: 1
38    }
39  });
```

- The preview is going to give us errors... but that’s okay we’ll deal with them in a minute.
- Now let's... initialize our state. What we need in our state is the index that the image is currently at. This way we can change our index to render the proper image.

```
8   export default class App extends Component {
9     constructor(props) {
10      super(props);
11      this.state = {index: 0};
12    }
13    render() {
14      return (
15       <View style={styles.container}></View>
16      );
17    }
18  }
```

- Next, let's create an image variable in our render method to grab the image that is at the index of our state. And then we'll add some JSX to render the image on our screen.
- We're using ImageBackground in the JSX, which is a React Native component for displaying nested images inside the View component... be sure to add it to your React Native imports at the top.

```
47  render() {
48    const image = SpaceImages[this.state.index];
49
50    return (
51      <View style={styles.container}>
52        <View style={styles.empty} />
53        <ImageBackground source={{uri: image.url}} style={styles.image}>
54          <Text style={styles.imageCaption}>{image.caption}</Text>
55        </ImageBackground>
56        <View style={styles.empty} />
57      </View>
58    );
59  }
```

- Now let's think about the events we need and how we're going to change state.
- In order to get the functionality for our app that we want. We need to add another value to state, and add a couple methods.

- I'm going to create a newImage method and add imageWidth to state
- imageWidth will help the different image widths as we cycle through them on the screen

```
46    this.state = {
47      index: 0,
48      imageWidth: null
49    };
```

- Next, we create a newImage event, which takes the current index and imageWidth of the current image, calculates where the images is touched (on the left half or the right half of the image), and move forward or back one image in the array.

```
49    newImage = (event) => {
50        //deconstruct the state
51        const { index, imageWidth } = this.state
52        //grab the location of where someone touches the picutre
53        const X = event.nativeEvent.locationX
54        //caculate if its on the right or left half of the image to go back or forth
55        const touchCalc = (X < imageWidth/2) ? -1 : +1;
56        //caculate the newIndex based on the touchCalc
57        let newIndex = (index + touchCalc) % SpaceImages.length;
58        //handle the case if the touch was negative
59        if (newIndex < 0) {
60            newIndex = SpaceImages.length - Math.abs(newIndex);
61        }
62        //set the state with the new index to display the new image
63        this.setState({
64            index: newIndex
65        });
66    }
```

- Then we create a method onNewLayout that takes an event and is simply going to update the imageWidth the next image's width and keep track of it.

```
71    onNewLayout = (event) => {
72        this.setState({
73            imageWidth: event.nativeEvent.layout.width
74        });
75    }
```

SPACE APP IN REACT NATIVE

- Now - we're ready to make the changes to our UI so we can use our methods.
- Let's use TouchableHighlight, which is a React Native component used to wrap touchable areas. Make sure to add it to your React Native imports at the top.
- We're using the onPress handler in TouchableHighlight to execute the newImage method.
- Also, notice the onLayout event handler in ImageBackground, which we're using to execute our onNewLayout event. onLayout is a RN method that is called anytime the app layout is rendered.

```
81      return (
82          <View style={styles.container}>
83              <View style={styles.empty} />
84              <TouchableHighlight onPress={this.newImage} style={styles.image}>
85                <ImageBackground source={{uri: image.url}} style={styles.image}
86                  onLayout={this.onNewLayout}>
87                    <Text style={styles.imageCaption}>{image.caption}</Text>
88                </ImageBackground>
89              </TouchableHighlight>
90              <View style={styles.empty} />
91          </View>
92      );
```

- The end