BUILDING REACT NAVIGATOR FROM SCRATCH

GOAL

- Build a React Native Navigator from scratch.
- Have animations between screens
- Have multiple screens to navigate between

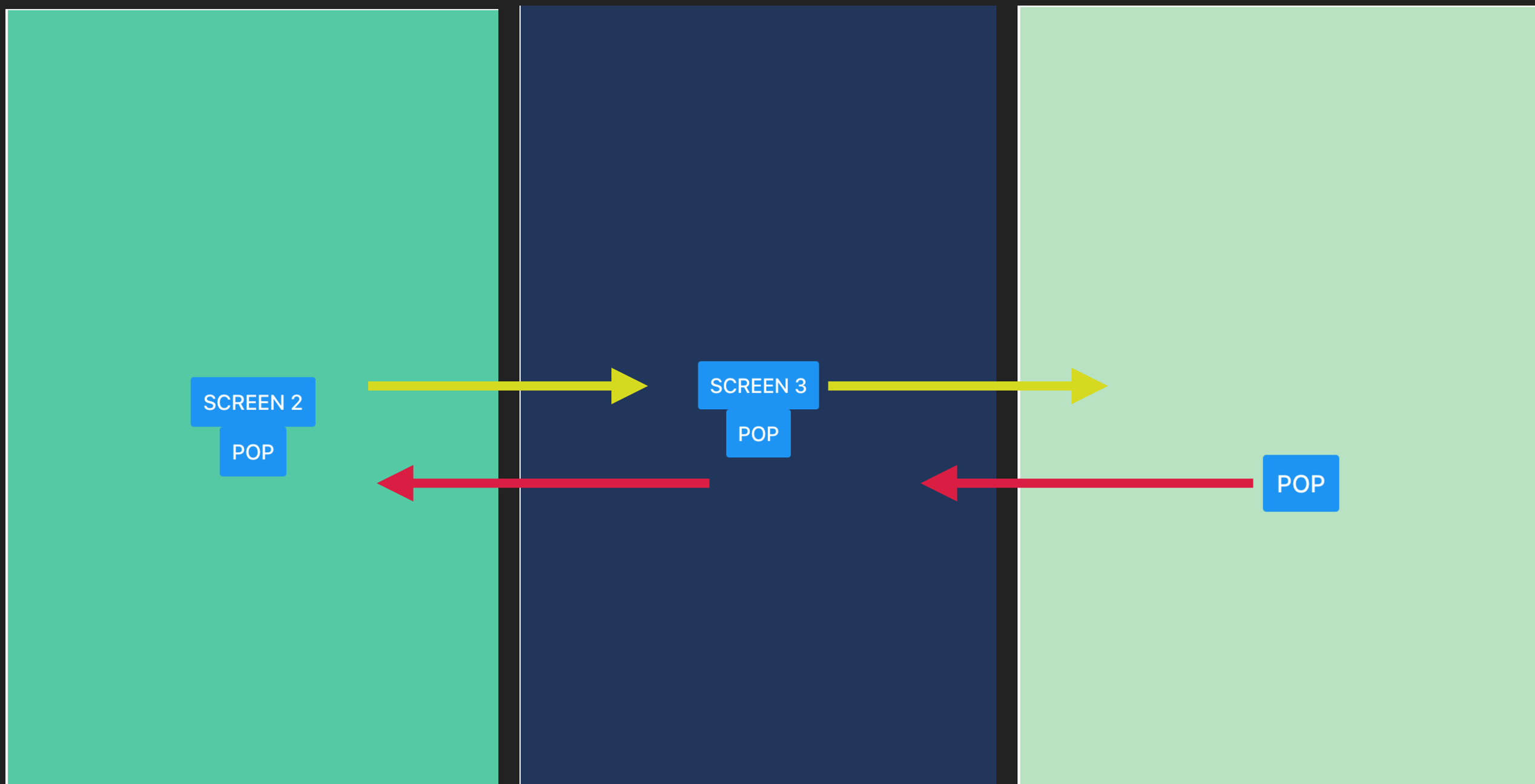
WHERE TO START.

- If you have your own environment set up (personal Mac or Windows laptop) that's great. For everyone using the student computers. Lets 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 lets look at the editor.

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



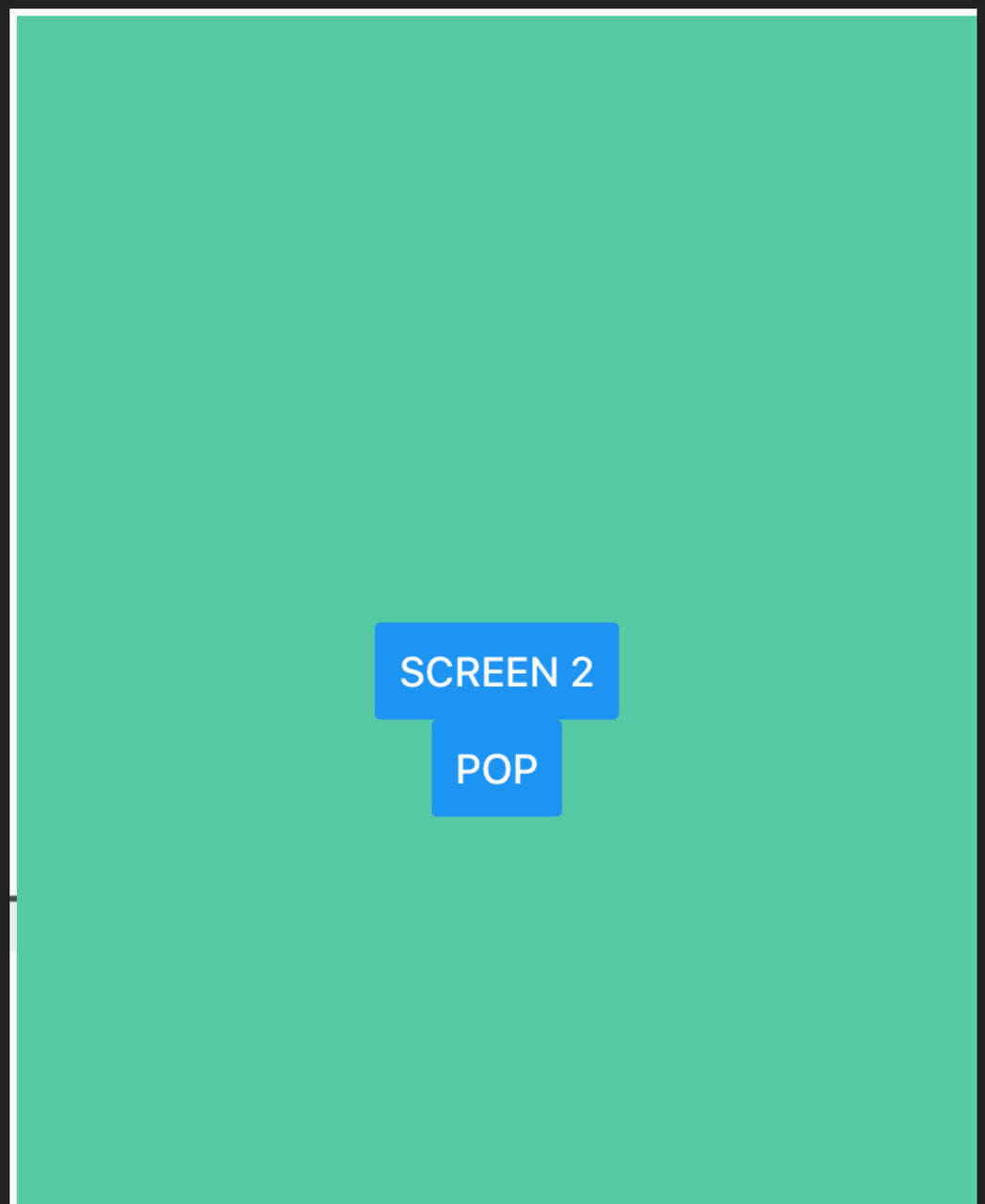
- First things first… lets delete everything in the editor and start from scratch. Then lets build out the base of our component.

```
1    import React, { Component } from 'react';
2    import { StyleSheet, View, Button } from 'react-native';
```

- Now lets build out our component... Right now we only want a functional component that will render the first navigation screen.

```
4   const Screen1 = (props) => (
5     <View style={[styles.screen, { backgroundColor: '#59C9A5' }]}>
6       <Button title="Screen 2" />
7       <Button title="Pop" />
8     </View>
9   );
10
11  export default Screen1;
12
13  const styles = StyleSheet.create({
14    screen: {
15      flex: 1,
16      alignItems: 'center',
17      justifyContent: 'center',
18    },
19  });
```

- This is what our app should look like if all is typed in correctly… Two buttons with names, but when click the buttons nothing happens.



- Now we need to create our file that we are going to use to accomplish the navigation. Make a new file and call this Navigator.js.

```
1   import React from 'react';
2
3   export const Route = () => null;
4
5   export class Navigator extends React.Component {
6     render() {
7       return null;
8     }
9   }
```

- Now that we have our navigator component we need to make sure to import this into our App.js file. Open the App.js file and add:

```
3    import { Navigator, Route } from './Navigator';
```

- Now what we're going to do is build out our Routes. We're going to do this by making a class component with-in our app.js file. Lets start off by replacing

```
12    export default Screen1;
```

With

```
12  export default class App extends React.Component {
13    render() {
14      return (
15        <Navigator>
16          <Route name="Screen1" component={Screen1} />
17          <Route name="Screen2" component={Screen2} />
18          <Route name="Screen3" component={Screen3} />
19        </Navigator>
20      );
21    }
22  }
```

- Dont worry if you're getting an error that's expected. What we need to do is create screen two and three.

```
12  const Screen2 = props => (
13    <View style={[styles.screen, { backgroundColor: '#23395B' }]}>
14      <Button title="Screen 3" />
15      <Button title="Pop" />
16    </View>
17  );
18
19  const Screen3 = props => (
20    <View style={[styles.screen, { backgroundColor: '#B9E3C6' }]}>
21      <Button title="Pop" />
22    </View>
23  );
```

CREATING A NAVIGATOR IN REACT NATIVE

- Now we want to go back into our Navigator.js file and we want to start building out the logic that is going to handle all the routing for us. Since we're using Navigator as a parent component and Route as child components - all of our Screens are available as props.

```
12  export default class App extends React.Component {
13    render() {
14      return (
15        <Navigator>
16          <Route name="Screen1" component={Screen1} />
17          <Route name="Screen2" component={Screen2} />
18          <Route name="Screen3" component={Screen3} />
19        </Navigator>
20      );
21    }
22  }
```

See how navigator is the parent of the 3 routes?

- What we can do now is set a current Scene in our Navigator by accessing the props passed into the child component and then access the props.component to then render the whole component.

```
5   export class Navigator extends React.Component {
6     render() {
7     console.log(this.props.children[0].props.component);
8       const CurrentScene = this.props.children[0].props.component;
9       return <CurrentScene />;
10    }
11  }
```

- We should now see our first Screen now that we access the child component and are able to render that component. This works, but accessing everything via this.props.children won't work very well going forward. We're going to need some internal state.

- First, lets create a stack array that will track the current stack of rendered screens. It should default to the first child of Navigator.

```
5  const buildSceneConfig = (children = []) => {
6    const config = {};
7
8    children.forEach(child => {
9      config[child.props.name] = { key: child.props.name, component: child.props.component };
10   });
11
12   return config;
13 };
```

- Lets check out whats going on in repl.it
- We're building out an object based on the child components coming in to help set up a stack. To where our result is:

```
{
  Scene1: {
    key: 'Scene1',
    component: Scene1,
  },
  Scene2: {
    key: 'Scene2',
    component: Scene2,
  },
  Scene3: {
    key: 'Scene3',
    component: Scene3,
  },
}
```

- Now we can use this function in the constructor of the Navigator component, and store the result in state. We can also populate our stack with the first screen.

```
constructor(props) {
  super(props);

  const sceneConfig = buildSceneConfig(props.children);
  const initialSceneName = props.children[0].props.name;

  this.state = {
    sceneConfig,
    stack: [sceneConfig[initialSceneName]],
  };
}
```

- Now lets use this.state.stack to render out our screen

```
31    render() {
32    console.log(this.state.stack[0].component);
33      const CurrentScene = this.state.stack[0].component;
34      return <CurrentScene />;
35    }
```

- Time to handle our Push and Pop functionality. Right now If you press the “Screen 2 “ button you’ll get an error “Cannot read property ‘push’ of undefined”. We aren’t yet passing a navigator prop down to the scene. I n this navigator prop we’ll pass the push action. Let’s write that push handler now.

```
30    handlePush = (sceneName) => {
31      this.setState(prevState => ({
32        ...prevState,
33        stack: [...prevState.stack, prevState.sceneConfig[sceneName]],
34      }));
35    }
```

- So what we're doing where is accepting a sceneName, which should correspond to a name prop given to one of our Route components and then finding the corresponding scene config for that route and adding it to the stack.

```
30    handlePush = (sceneName) => {
31      this.setState(prevState => ({
32        ...prevState,
33        stack: [...prevState.stack, prevState.sceneConfig[sceneName]],
34      }));
35    }
```

- We now make the push function available to the current scene.

```
37    render() {
38      const CurrentScene = this.state.stack[0].component;
39      return <CurrentScene navigator={{ push: this.handlePush }} />
40    }
```

- Now when we push the screen 2 … no errors occur. But we still don't navigate to our next screen. What we need to do is loop over our this.state.stack and render the screens. Lets first make sure we have the correct imports in our Navigator.js file

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

- Now we'll loop over this.state.stack and render each scene.

```
38  render() {
39    return (
40      <View >
41        {this.state.stack.map((scene, index) => {
42          const CurrentScene = scene.component;
43          return (
44            <CurrentScene
45              key={scene.key}
46              navigator={{ push: this.handlePush }}
47            />
48          );
49        })}
50      </View>
51    )
52  }
```

- Let also add some styling to the bottom of our module.

```
56    const styles = StyleSheet.create({
57      container: {
58        flex: 1,
59        flexDirection: 'row',
60      },
61    });
```

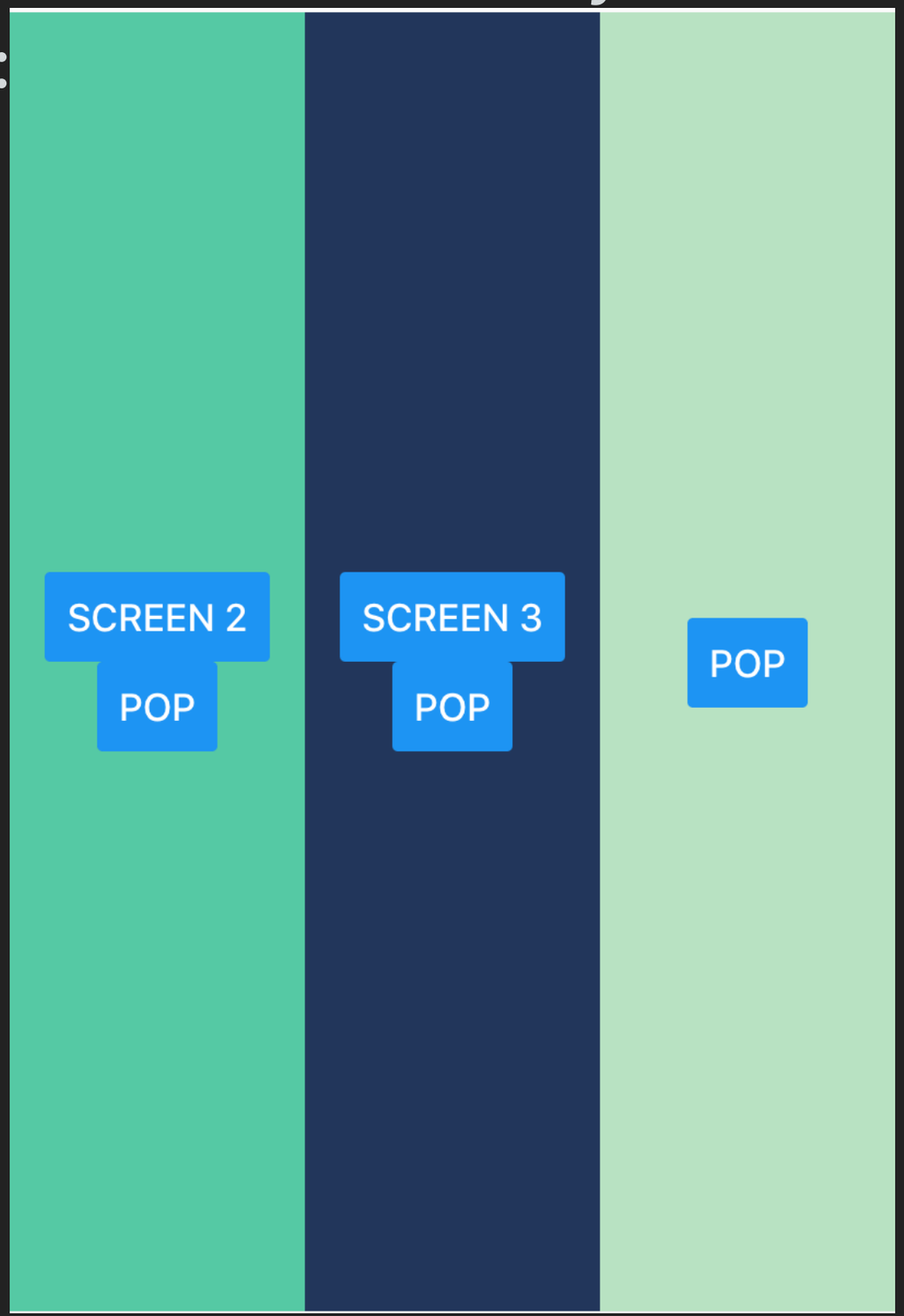
- And then add it to our View component

```
41          <View style={styles.container}>
```

- Okay so we did all this work and our app still isn't navigating. What we need to do now is add our event handlers into our buttons so they push the navigation correctly. First we ned to navigate to our App.js file and add these onPress event handlers.

```
5   const Screen1 = props => (
6     <View style={[styles.screen, { backgroundColor: '#59C9A5' }]}>
7       <Button title="Screen 2" onPress={() => props.navigator.push("Screen2")}/>
8       <Button title="Pop" />
9     </View>
10  );
11
12  const Screen2 = props => (
13    <View style={[styles.screen, { backgroundColor: '#23395B' }]}>
14      <Button title="Screen 3" onPress={() => props.navigator.push("Screen3")} />
15      <Button title="Pop" />
16    </View>
17  );
```

- Okay… if this is done successfully. We should see something like:



- We can push but we can pop back to our original or previous scene. Time to add in the functionality to pop our stack to the previous scene. This is a method that should go right under the handlePush method.

```
handlePop = () => {
  this.setState(prevState => {
    const { stack } = prevState;
    if (stack.length > 1) {
      return {
        stack: stack.slice(0, stack.length - 1),
      };
    }

    return prevState;
  });
}
```

- In the handlePop function we are checking if the stack has more than one screen and, if so, we remove the last screen in that stack.

```
handlePop = () => {
  this.setState(prevState => {
    const { stack } = prevState;
    if (stack.length > 1) {
      return {
        stack: stack.slice(0, stack.length - 1),
      };
    }

    return prevState;
  });
}
```

- Now we need to make sure to pass this handlePop method as a prop to our current scene in our Navigator.js file.

```
<CurrentScene
  key={scene.key}
  navigator={{ push: this.handlePush, pop: this.handlePop }}
/>
```

- Now we need to add that onPress event to our buttons that pop us back on the stack.

```
12  const Screen2 = props => (
13    <View style={[styles.screen, { backgroundColor: '#23395B' }]}>
14      <Button title="Screen 3" onPress={() => props.navigator.push('Screen3')} />
15      <Button title="Pop" onPress={() => props.navigator.pop()} />
16    </View>
17  );
18
19  const Screen3 = props => (
20    <View style={[styles.screen, { backgroundColor: '#B9E3C6' }]}>
21      <Button title="Pop" onPress={() => props.navigator.pop()} />
22    </View>
23  );
```

- Now our pop and push methods are working correctly and we can navigate through our components properly and we're happy right??? NO!!! Your Professor is never happy… Until we have proper styling.
- What we need to do is ensure that the screens are rendering correctly. We need to use absolute positioning for this. and we need to wrap our CurrentScene component in a View component with the added styling. We also need a flex: 1 so the view takes up the entire screen.

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
    flexDirection: 'row',
  },
  scene: {
    ...StyleSheet.absoluteFillObject,
    flex: 1,
  },
});
```

- Lets now wrap our CurrentScene component in a View and then add the key and style to the view component.

```
render() {
  return (
    <View style={styles.container}>
      {this.state.stack.map((scene, index) => {
        const CurrentScene = scene.component;
        return (
          <View key={scene.key} style={styles.scene}>
            <CurrentScene
              navigator={{ push: this.handlePush, pop: this.handlePop }}
            />
          </View>
        );
      })}
    </View>
  )
}
```

- Now we should be able to navigate between these Scenes properly and the whole component should populate the screen.