



Mobile App Development 2

Study diary

Chou-Ping Ding

Contents

1	Week exercises	3
1.1	Written questions / answers React Native.....	3
1.2	Code lab.....	4
2	Week excercises.....	11
2.1	Weather App for Android and iOS.....	11
2.2	Implement functionality for your Weather Application.	16
2.3	Implement a feature where the user can select the city/location of the weather to be fetched.....	18
3	Week exercises	20
3.1	React Page Navigation	20
3.2	List items and FlatList component.....	27
4	Week exercises	31
4.1	FlexBox Layout	31
4.2	Platform Specific Components in React Native.....	32
4.3	Utilizing Device APIs – Part I	34
5	Week exercises	36
5.1	Utilizing Device APIs – Part II	36
5.2	Using the Linking Module.....	37
5.3	Yet some other Device APIs	39
5.4	Publishing APPs.....	40
	Final project	42
	Sources used with exercises.....	46

1 Week exercises

1.1 Written questions / answers React Native

1. *Explain the term “Cross-platform mobile development”*

Cross-platform mobile development basically meant developing a mobile application which can run on multiple mobile platforms, such as iOS, Android, with single code base. With this practice, there's no need to create different native based applications, but it allows developers to just write the code once and have it deployed to various platforms.

2. *What different frameworks/options you have when making a mobile application for*

- **Android**

- i. *Native Android with Java or Kotlin*
- ii. *Flutter*
- iii. *Xamarin*
- iv. *React Native*

- **iOS**

- i. *SwiftUI*
- ii. *Flutter*
- iii. *Firebase*
- iv. *React Native*

3. *Give a brief overview of React Native. Consider at least the following*

- *What is the programming language used?*
- *What are the development tools options?*
- *Where can you find basic tutorials (Hello World etc)?*
- *Where is the official documentation for React Native?*
- *What are the pros and cons of React Native (against pure native Java in Android)?*
- *Your own thoughts on the framework?*

The programming language used in React Native is based on JavaScript, there are different kinds of development tools that can be used for app development with React Native, common ones include VSCode, React Native CLI and Expo (what we are using now). The environment can be set, but at

the moment we use Expo online tool. The basic tutorial can be found on React Native official site (<https://reactnative.dev/docs/getting-started>), where it provides basic introduction of the framework and documentation needed within the development process can also be found on this website, including community information for further support.

There are some pros and cons about using React Native for Android app development, as on the good side, one of the big advantages is the cross-platform nature, which saves development time, it also have a larger developer community, providing support and open-source libraries and components. It also provides better code reusability as a large portion of the code can be shared between platforms and lowers the learning curves for web developers with it's use of JavaScript based language.

But there are still downsides about this framework, the first thing is learning curve can be longer while writing and integrating native modules, and although it can deliver near-native performances, there might still be the need to write platform specific code for performance-critical tasks. And some of the native features and APIs might not be available too, so it might require custom native module development.

I personally agree that with this framework, a lot of development time is saved, it also makes things easier for development for different platforms. But the downside is the language it used, which is not my favorite, and although the styling method might be easier for some, for me it's more troublesome compared to Android or iOS native development.

4. *How does React Native differ from React?*

Key differences: target platform (web vs mobile app), components (html vs native mobile elements), styling (CSS vs StyleSheet).

1.2 **Code lab**

Implement a simple React Native UI for converter. The conversion can be e.g. currency, temperature or some other of your choice. The UI can consist of simple input field for amount to convert (e.g current), then a button to do the conversion (from e.g. EUR to USD) or vice versa. The user inputs the currency amount and clicks a button and there should be a styled text field to show the conversion.

Implement the functionality by using functional components and hooks. Style your app so that it works responsively well in most of the (portrait) smartphones. Use styles for fonts, sizes, colours and layouts on the screen.

Add a screenshot and code in your study paper. Write your own thoughts and findings in the study paper. Document also, which sites/tutorials/AI tools (Chat GPT etc) you have used to find your solution.

Note! *You can use web sites / tutorials and Chat tools for help as it is part of modern development. However, you're required to return own solution for the most efficient learning results.*

```

1  import { useState } from 'react';
2  import { Text, View, Button, TextInput, StyleSheet } from 'react-native';
3
4  const MyConverterApp = () => {
5    const [inputC, setInputC] = useState('0');
6    const [inputF, setInputF] = useState('32');
7    const [conversionMode, setConversionMode] = useState('CtoF');
8
9    const converter = () => {
10     if (conversionMode === 'CtoF') {
11       const degreeC = parseFloat(inputC);
12       const degreeF = (degreeC * 9) / 5 + 32;
13       const roundF = degreeF.toFixed(2);
14       setInputF(roundF.toString());
15       setConversionMode('cal');
16     } else if (conversionMode === 'FtoC') {
17       const degreeF = parseFloat(inputF);
18       const degreeC = ((degreeF - 32) * 5) / 9;
19       const roundC = degreeC.toFixed(2);
20       setInputC(roundC.toString());
21       setConversionMode('cal');
22     }
23   };

```

```

24
25  return (
26    <View style={styles.container}>
27      <Text style={styles.paragraph}>Temperature Converter</Text>
28      <Text style={styles.paragraph}>Celsius: </Text>
29
30      <TextInput style={styles.paragraph}
31        onChangeText={(newText) => {
32          setInputC(newText);
33          setConversionMode('CtoF');
34        }}
35        value={inputC} ></TextInput>
36      <Text style={styles.paragraph}>Fahrenheit: </Text>
37
38      <TextInput style={styles.paragraph}
39        onChangeText={(newText) => {
40          setInputF(newText);
41          setConversionMode('FtoC');
42        }}
43        value={inputF} ></TextInput>
44      <Button title="Calculate" onPress={converter}></Button>
45    </View>
46  );
47  };

```

```

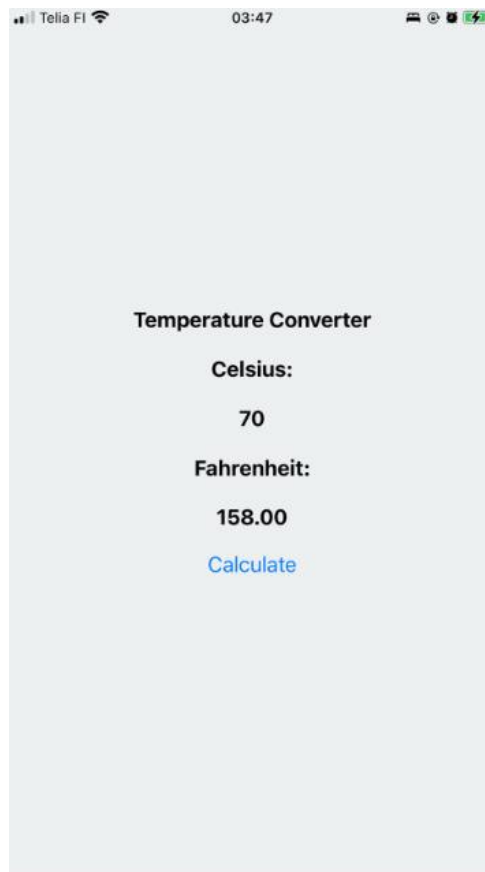
48
49  const styles = StyleSheet.create({
50    container: {
51      flex: 1,
52      justifyContent: 'center',
53      backgroundColor: '#ecf0f1',
54      padding: 4,
55    },
56    paragraph: {
57      margin: 10,
58      fontSize: 18,
59      fontWeight: 'bold',
60      textAlign: 'center',
61    },
62  });
63
64  export default MyConverterApp;
65

```

Android interface (provided by Expo):



iOS (own device screenshot):



In class reference:

RN App Building Blocks

- Components
 - Components are built with React Native components containing
 - View, Text, TextInput, Image, Button, ScrollView, FlatList...
- Styles
 - Components are styled with StyleSheets

```

1  import {useState} from 'react';
2  import { View, Text, Button, StyleSheet } from 'react-native'
3  // My click counter app
4
5  const MyCounterApp = () => {
6    const [counter, setCounter] = useState(0);
7
8    const sayHello = () => {
9      | setCounter( counter + 1);
10   }
11   // we return a component
12   return(
13     <View style={styles.viewStyle }>
14       <Text>Counter: {counter}</Text>
15       <Button title={"Increase counter"} onPress={sayHello}></Button>
16     </View>
17   );
18 }
19
20 const styles = StyleSheet.create({
21   viewStyle: {
22     flex: 1, justifyContent: "center", alignItems: "center"
23   },
24 });
25
26 export default MyCounterApp;

```

Basic Components

View The most fundamental component for building a UI.	Text A component for displaying text.	Image A component for displaying images.
TextInput A component for inputting text into the app via a keyboard.	ScrollView Provides a scrolling container that can host multiple components and views.	StyleSheet Provides an abstraction layer similar to CSS stylesheets.

Common UI Controls

Button A basic button component for handling touches that should render nicely on any platform.	Picker Renders the native picker component on iOS and Android.	Slider A component used to select a single value from a range of values.
Switch Renders a boolean input.	FlatList A component for rendering performant scrollable lists.	SectionList Like FlatList, but for sectioned lists.

Android specific components and APIs

BackHandler Detect hardware button presses for back navigation.	DatePickerAndroid Opens the standard Android date picker dialog.	DrawerLayoutAndroid Renders a <code>DrawerLayout</code> on Android.
PermissionsAndroid Provides access to the permissions model introduced in Android M.	ProgressBarAndroid Renders a <code>ProgressBar</code> on Android.	TimePickerAndroid Opens the standard Android time picker dialog.
ToastAndroid Create an Android Toast alert.	ToolbarAndroid Renders a <code>Toolbar</code> on Android.	ViewPagerAndroid Container that allows to flip left and right between child views.

iOS specific components and APIs

ActionSheetIOS API to display an iOS action sheet or share sheet.	AlertIOS Create an iOS alert dialog with a message or create a prompt for user input.	DatePickerIOS Renders a date/time picker (selector) on iOS.
ImagePickerIOS Renders a image picker on iOS.	NavigatorIOS A wrapper around <code>UINavigationController</code> , enabling you to implement a navigation stack.	ProgressViewIOS Renders a <code>UIProgressView</code> on iOS.
PushNotificationIOS Handle push notifications for iOS, including permission handling and icon badge number.	SegmentedControlIOS Renders a <code>UISegmentedControl</code> on iOS.	TabBarIOS Renders a <code>UITabBarController</code> on iOS. Use with <code>TabBarIOS.Item</code> .

2 Week excercises

2.1 Weather App for Android and iOS

Implement a weather app in React Native. Use various Components from React Native and divide your implementation also into separate components so that your weather screen is in its own JS file e.g. WeatherScreen.js. under “Components” directory.

Requirements:

- The user interface should contain at least
 - Header (showing the weather location (city, other place)) in its own sub-component
 - Header.js for which you pass the location name as a props (this is basic React.js)
 - Weather icon (sunny, cloudy, rainy, fogg)
 - Temperature
 - Wind Speed
 - Location input where the user gets to select the city/place for asking the weather.
 - Button or other UI component to refresh the weather info
- Use styles throughout your application
- The user interface should be responsive (i.e. scale to different portrait screen sizes).
- Update a random weather data on the screen when the user clicks the “refresh” button

User Interface with fixed value first:

Main (App.js)

..... ✓
 ous saves. ✓

```

1  import { useState } from 'react';
2  import { Text, View, Button, TextInput, StyleSheet } from 'react-native';
3  import Header from './components/Header';
4  import WeatherInfo from './components/WeatherInfo';
5  import LocationInput from './components/LocationInput';
6
7  const MyWeatherApp = () => {
8    const [locationName, setLocationName] = useState('Tampere');
9    const handleCityChange = (newCity) => {
10      setLocationName(newCity); // Update the city name
11    };
12
13    return (
14      <View style={styles.container}>
15        <Header cityName={locationName}></Header>
16        <WeatherInfo></WeatherInfo>
17        <LocationInput onCityChange={handleCityChange} />
18      </View>
19    );
20  };
21
22  const styles = StyleSheet.create({
23    container: {
24      marginTop: 30,
25
26      flex: 1,
27      justifyContent: 'center',
28      backgroundColor: '#ecf0f1',
29      padding: 4,
30    },
31    paragraph: {
32      margin: 10,
33      fontSize: 18,
34      fontWeight: 'bold',
35      textAlign: 'center',
36    },
37  });
38
39  export default MyWeatherApp;
40

```

Header.js

[Go to previous slide](#)

```
1  import { Text, View, Button, TextInput, StyleSheet } from 'react-native';
2
3  const Header = ({ cityName }) => {
4    return (
5      <View style={styles.headerBackground}>
6        <Text style={styles.headerStyle}>{cityName}</Text>
7      </View>
8    );
9  };
10
11  const styles = StyleSheet.create({
12    headerBackground: {
13      flex: 1,
14      backgroundColor: '#ecf0f1',
15      padding: 5,
16    },
17    headerStyle: {
18      margin: 10,
19      fontSize: 40,
20      fontWeight: 'bold',
21      textAlign: 'center',
22    },
23  });
24
25  export default Header;
26
```

WeatherInfo.js

```

1  import { Image, Text, View, Button, TextInput, StyleSheet } from 'react-native';
2
3  const WeatherInfo = () => {
4    return (
5      <View style={styles.headerBackground}>
6        <Text style={styles.headerStyle}>Weather Info</Text>
7        <Image style={styles.logo} source={require('../assets/rain.jpg')} />
8        <Text style={styles.paragraph}>Temperature: -3</Text>
9        <Text style={styles.paragraph}>Wind: 2 m/s</Text>
10     </View>
11   );
12 };
13
14 const styles = StyleSheet.create({
15   headerBackground: {
16     flex: 2,
17     justifyContent: 'center',
18     alignItems: 'center',
19     backgroundColor: '#ecf0f1',
20     padding: 4,
21   },
22   headerStyle: {
23     margin: 5,
24     fontSize: 20,
25     fontWeight: 'bold',
26     textAlign: 'center',
27   },
28   logo: {
29     width: 150,
30     height: 150,
31   },
32   paragraph: {
33     margin: 5,
34     fontSize: 15,
35     textAlign: 'center',
36     padding: 3,
37   },
38 });
39
40 export default WeatherInfo;

```

LocationInput.js

```

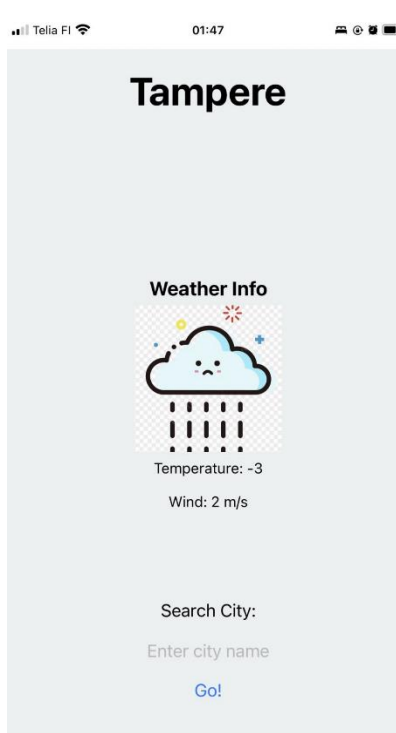
1  import { Text, View, Button, TextInput, StyleSheet } from 'react-native';
2  import { useState } from 'react';
3
4  const LocationInput = ({ onCityChange }) => {
5
6      const [input, setInput] = useState('');
7
8      // Pass the updated city back to the parent component
9      const handleLocation = () => {
10         onCityChange(input);
11         setInput(''); // Clear the input field after submitting
12     };
13
14     return (
15         <View style={styles.container}>
16             <Text style={styles.paragraph}>Search City:</Text>
17             <TextInput style={styles.paragraph} value={input} onChangeText={(text) => setInput(text)}
18                 placeholder="Enter city name"></TextInput>
19             <Button title="Go!" onPress={handleLocation}></Button>
20         </View>
21     );
22 };
23
24 const styles = StyleSheet.create({
25     container: {
26         flex: 1,
27         justifyContent: 'center',
28         backgroundColor: '#ecf0f1',
29         padding: 4,
30     },
31     paragraph: {
32         margin: 10,
33         fontSize: 18,
34         textAlign: 'center',
35     },
36 });
37
38 export default LocationInput;
39

```

Android (Expo):



iOS (own device):



2.2 Implement functionality for your Weather Application.

The application can fetch the data from e.g. OpenWeatherMap API. You can create your free account to the server with limited amount of fetches. Use Async/Await functionality and Fetch API here. What alternatives are there for fetching?

According to the official documentation of React Native, due to the fact that XMLHttpRequest API is built into Reactive Native, external libraries such as frisbee or axios which depend on it can be used, also the XMLHttpRequest can be directly.

Fetching data from OpenWeatherMap:

```

1  import { Image, Text, View, Button, TextInput, StyleSheet } from 'react-native';
2  import { useState } from 'react';
3
4  const WeatherInfo = ({ cityName }) => {
5    const [weatherData, setWeatherData] = useState(null);
6    const API_KEY = 'd213cb6747b9f0f6b320c3d78cfca46a';
7    const API_ENDPOINT = 'https://api.openweathermap.org/data/2.5/weather';
8
9    const fetchWeatherData = async (city) => {
10     try {
11       const response = await fetch(
12         `${API_ENDPOINT}?q=${city}&units=metric&appid=${API_KEY}`
13       );
14       const data = await response.json();
15       setWeatherData(data);
16     } catch (error) {
17       console.error('Error fetching weather data:', error);
18     }
19   };
20   const handleCityChange = (newCity) => {
21     fetchWeatherData(newCity);
22   };
23
24   // Fetch data when the cityName changes
25   if (cityName) {
26     handleCityChange(cityName);
27   }
28
29   //using the provided icon from the openweather api
30   const getWeatherIcon = () => {
31     if (weatherData) {
32       const weatherIcon = weatherData.weather[0].icon;
33       return `http://openweathermap.org/img/w/${weatherIcon}.png`;
34     } else {
35       return require('../assets/snack-icon.png');
36     }
37   };

```

Use static location first in the main view:

```

<View style={styles.container}>
  <Header cityName={locationName}></Header>
  <WeatherInfo cityName={"Tampere"}></WeatherInfo>
  <LocationInput onCityChange={handleCityChange} />
</View>

```

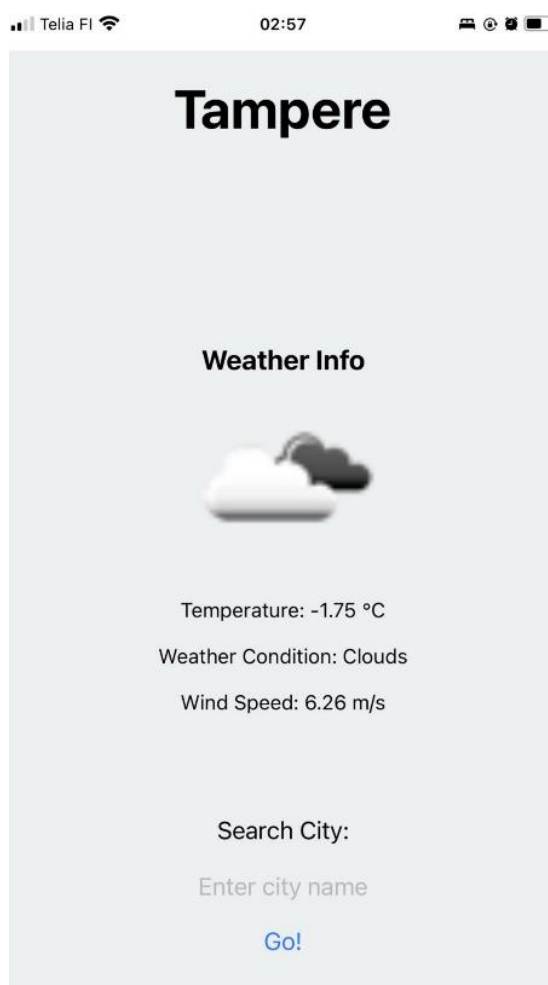

Display in view:

```

38
39   return (
40     <View style={styles.headerBackground}>
41       <Text style={styles.headerStyle}>Weather Info</Text>
42       <Image style={styles.logo} source={{ uri: getWeatherIcon() }} />
43       {weatherData && (
44         <View>
45           <Text style={styles.paragraph}>
46             Temperature: {weatherData.main.temp} °C
47           </Text>
48           <Text style={styles.paragraph}>
49             Weather Condition: {weatherData.weather[0].main}
50           </Text>
51           <Text style={styles.paragraph}>
52             Wind Speed: {weatherData.wind.speed} m/s
53           </Text>
54         </View>
55       )}
56     </View>
57   );
58 };
59
60 const styles = StyleSheet.create({

```

The fetching works for real device (iOS), not in Expo device.



2.3 Implement a feature where the user can select the city/location of the weather to be fetched.

Use “lifting state up” functionality for fetching part of the UI so that the city name and update button are together in their own component in the UI.

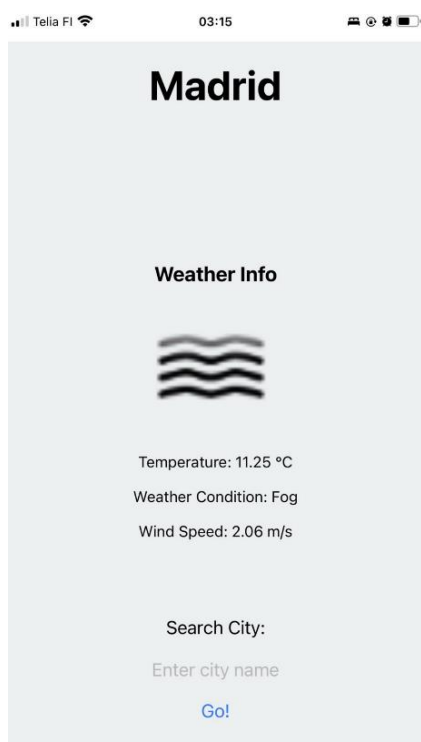
Change the value so that it will update with user input.

```
import { useState } from 'react';
import { Text, View, Button, TextInput, StyleSheet } from 'react-native';
import Header from './components/Header';
import WeatherInfo from './components/WeatherInfo';
import LocationInput from './components/LocationInput';

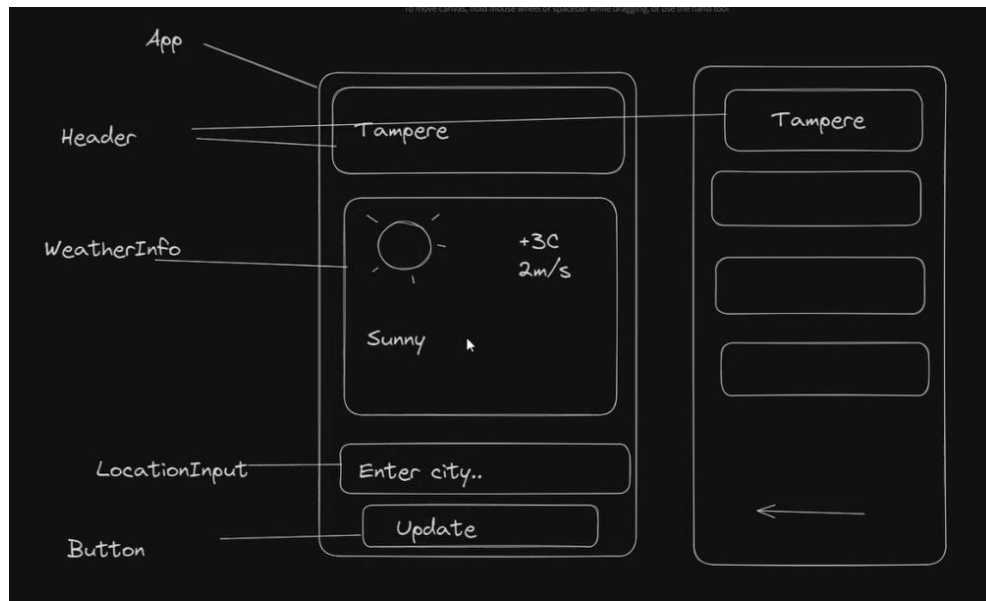
const MyWeatherApp = () => {
  const [locationName, setLocationName] = useState('Tampere');
  const handleCityChange = (newCity) => {
    setLocationName(newCity); // Update the city name
  };

  return (
    <View style={styles.container}>
      <Header cityName={locationName}></Header>
      <WeatherInfo cityName={locationName}></WeatherInfo>
      <LocationInput onCityChange={handleCityChange} />
    </View>
  );
};
```

Again works for real device (iOS), not in Expo device.



In class reference:



3 Week exercises

3.1 React Page Navigation

Get familiar with React Native Page Navigation components. Try out three different navigation patterns by implementing a simple navigation (of 2 to 3 pages, which can contain some random components) in an application.

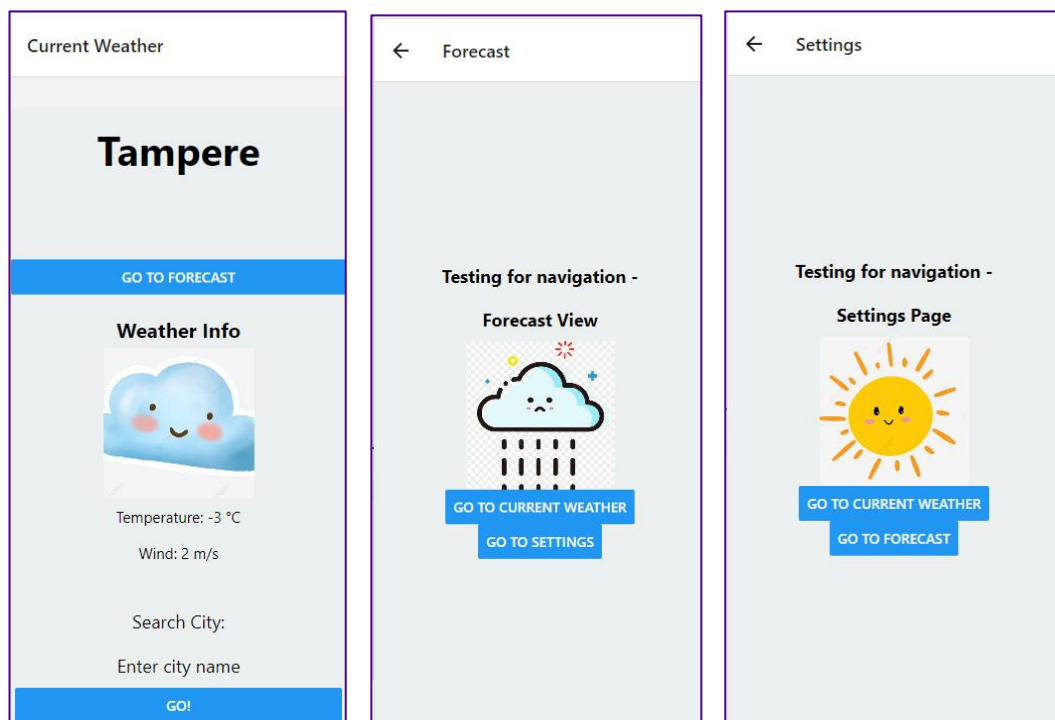
- *Stack*
- *Drawer*
- *Some other of your choice (e.g. Bottom Tabs, Material Top Tabs)*

Add code and screenshots to the study diary. Give your own opinion of each of the navigation patterns from user's point of view. Which navigation pattern would you use in your Weather Application if there would be several screens?

****This exercise is carried out with static data since when I use the free weather API, the updating will exceed the connection limits and effect the layout of the view. ****

For testing methods, I added two simple views with just text and image. Buttons were added to assist navigation for stack method also, but dropped as they were not needed in other methods.

1. Stack:



App.js:

```

1  import { useState } from 'react';
2  import { View, StyleSheet } from 'react-native';
3  import { createStackNavigator } from '@react-navigation/stack';
4  import { NavigationContainer } from '@react-navigation/native';
5
6
7  import CurrentWeatherView from '../components/CurrentWeatherView';
8  import SettingView from '../components/SettingView';
9  import ForecastView from '../components/ForecastView';
10
11  const Stack = createStackNavigator();
12
13  function MyStack() {
14    return (
15      <Stack.Navigator>
16        <Stack.Screen name="Current Weather" component={CurrentWeatherView} />
17        <Stack.Screen name="Forecast" component={ForecastView} />
18        <Stack.Screen name="Settings" component={SettingView} />
19      </Stack.Navigator>
20    );
21  }
22  const MyWeatherApp = () => {
23    const [locationName, setLocationName] = useState('Tampere');
24    const handleCityChange = (newCity) => {
25      setLocationName(newCity); // Update the city name
26    };
27
28    return (
29      <NavigationContainer>
30        <MyStack />
31      </NavigationContainer>
32      // <View style={styles.container}>
33      // <ForecastView/>
34      // </View>
35    );
36  };
37

```

CurrentWeatherView.js:

```

1  import { useState } from 'react';
2  import { Text, View, Button, TextInput, StyleSheet, SafeAreaView } from 'react-native';
3
4  import Header from '../components/Header';
5  import WeatherInfo from '../WeatherInfo';
6  import LocationInput from '../LocationInput';
7
8  const CurrentWeatherView = ({navigation}) => {
9    const [locationName, setLocationName] = useState('Tampere');
10    const handleCityChange = (newCity) => {
11      setLocationName(newCity); // Update the city name
12    };
13
14    return (
15      <SafeAreaView style={styles.container}>
16        <Header cityName={locationName}></Header>
17        <Button title="Go to Forecast" onPress={() => navigation.navigate('Forecast')} />
18        <WeatherInfo cityName={locationName}></WeatherInfo>
19        <LocationInput onCityChange={handleCityChange} />
20      </SafeAreaView>
21    );
22  };
23

```

ForecastView.js:

```

1  import { Text, Button, StyleSheet, SafeAreaView, Image } from 'react-native';
2
3  const ForecastView = ({navigation}) => {
4
5    return (
6      <SafeAreaView style={styles.headerBackground}>
7        <Text style={styles.paragraph}>Testing for navigation -</Text>
8        <Text style={styles.paragraph}>Forecast View</Text>
9        <Image style={styles.logo} source={require('../assets/rain.jpg')} />
10       <Button style={styles.button} title="Go to Current Weather" onPress={() => navigation.navigate('Current
Weather')} />
11       <Button style={styles.button} title="Go to Settings" onPress={() => navigation.navigate('Settings')} />
12     </SafeAreaView>
13   );
14 };
15

```

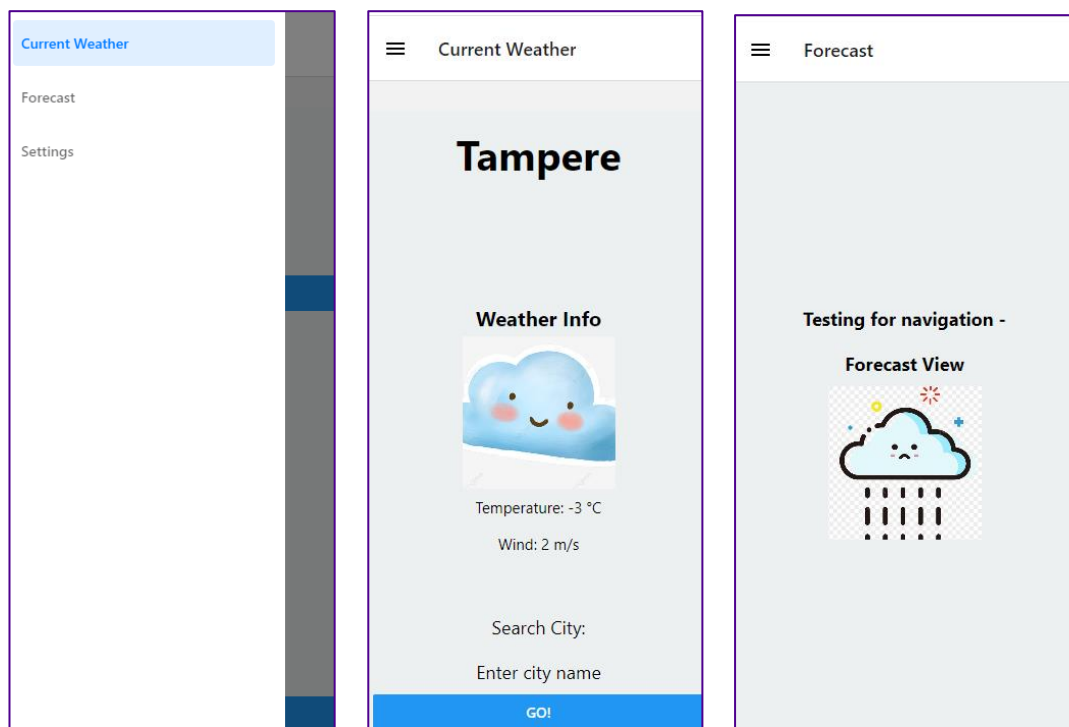
SettingView.js:

```

1  import { Text, Button, StyleSheet, SafeAreaView, Image } from 'react-native';
2
3  const SettingView = ({navigation}) => {
4
5    return (
6      <SafeAreaView style={styles.headerBackground}>
7        <Text style={styles.paragraph}>Testing for navigation -</Text>
8        <Text style={styles.paragraph}>Settings Page</Text>
9        <Image style={styles.logo} source={require('../assets/sun.jpg')} />
10       <Button style={styles.button} title="Go to Current Weather" onPress={() => navigation.navigate('Current
Weather')} />
11       <Button style={styles.button} title="Go to Forecast" onPress={() => navigation.navigate('Forecast')} />
12     </SafeAreaView>
13   );
14 };
15

```

2. Drawer:



App.js

```

1  import { useState } from 'react';
2  import { View, StyleSheet } from 'react-native';
3  import { createDrawerNavigator } from '@react-navigation/drawer';
4  import { NavigationContainer } from '@react-navigation/native';
5
6  import CurrentWeatherView from '../components/CurrentWeatherView';
7  import SettingView from '../components/SettingView';
8  import ForecastView from '../components/ForecastView';
9
10 const Drawer = createDrawerNavigator();
11
12 function MyDrawer() {
13   return (
14     <Drawer.Navigator useLegacyImplementation>
15       <Drawer.Screen name="Current Weather" component={CurrentWeatherView} />
16       <Drawer.Screen name="Forecast" component={ForecastView} />
17       <Drawer.Screen name="Settings" component={SettingView} />
18     </Drawer.Navigator>
19   );
20 }
21
22 const MyWeatherApp = () => {
23   const [locationName, setLocationName] = useState('Tampere');
24   const handleCityChange = (newCity) => {
25     setLocationName(newCity); // Update the city name
26   };
27
28   return (
29     <NavigationContainer>
30       <MyDrawer />
31     </NavigationContainer>
32   );
33 };
34

```

CurrentWeatherView.js

```

1  import { useState } from 'react';
2  import { Text, View, Button, TextInput, StyleSheet, SafeAreaView } from 'react-native';
3
4  import Header from '../components/Header';
5  import WeatherInfo from '../WeatherInfo';
6  import LocationInput from '../LocationInput';
7
8  const CurrentWeatherView = () => {
9    const [locationName, setLocationName] = useState('Tampere');
10    const handleCityChange = (newCity) => {
11      setLocationName(newCity); // Update the city name
12    };
13
14    return (
15      <SafeAreaView style={styles.container}>
16        <Header cityName={locationName}></Header>
17        <WeatherInfo cityName={locationName}></WeatherInfo>
18        <LocationInput onCityChange={handleCityChange} />
19      </SafeAreaView>
20    );
21  };

```

ForecastView.js

```

1  import { Text, Button, StyleSheet, SafeAreaView, Image } from 'react-native';
2
3  const ForecastView = () => {
4
5    return (
6      <SafeAreaView style={styles.headerBackground}>
7        <Text style={styles.paragraph}>Testing for navigation -</Text>
8        <Text style={styles.paragraph}>Forecast View</Text>
9        <Image style={styles.logo} source={require('../assets/rain.jpg')} />
10      </SafeAreaView>
11    );
12  };

```

SettingView.js

```

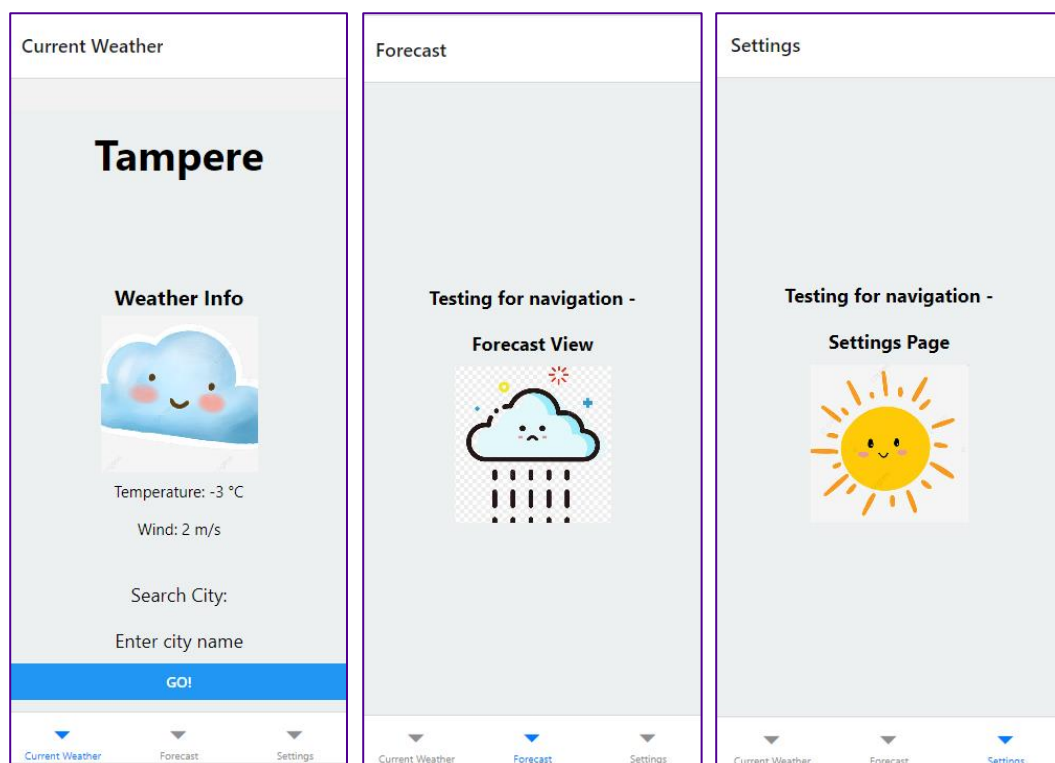
import { Text, Button, StyleSheet, SafeAreaView, Image } from 'react-native';

const SettingView = () => {

  return (
    <SafeAreaView style={styles.headerBackground}>
      <Text style={styles.paragraph}>Testing for navigation -</Text>
      <Text style={styles.paragraph}>Settings Page</Text>
      <Image style={styles.logo} source={require('../assets/sun.jpg')} />
    </SafeAreaView>
  );
};

```

3. Bottom Tabs:



App.js:

```

1  import { useState } from 'react';
2  import { View, StyleSheet } from 'react-native';
3  import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
4  import { NavigationContainer } from '@react-navigation/native';
5
6  import CurrentWeatherView from './components/CurrentWeatherView';
7  import SettingView from './components/SettingView';
8  import ForecastView from './components/ForecastView';
9
10 const Tab = createBottomTabNavigator();
11
12 function MyTabs() {
13   return (
14     <Tab.Navigator useLegacyImplementation>
15       <Tab.Screen name="Current Weather" component={CurrentWeatherView} />
16       <Tab.Screen name="Forecast" component={ForecastView} />
17       <Tab.Screen name="Settings" component={SettingView} />
18     </Tab.Navigator>
19   );
20 }
21
22 const MyWeatherApp = () => {
23   const [locationName, setLocationName] = useState('Tampere');
24   const handleCityChange = (newCity) => {
25     setLocationName(newCity); // Update the city name
26   };
27
28   return (
29     <NavigationContainer>
30       <MyTabs />
31     </NavigationContainer>
32   );
33 };

```

CurrentWeatherView.js

```

1  import { useState } from 'react';
2  import { Text, View, Button, TextInput, StyleSheet, SafeAreaView } from 'react-native';
3
4  import Header from './components/Header';
5  import WeatherInfo from './WeatherInfo';
6  import LocationInput from './LocationInput';
7
8  const CurrentWeatherView = () => {
9    const [locationName, setLocationName] = useState('Tampere');
10    const handleCityChange = (newCity) => {
11      setLocationName(newCity); // Update the city name
12    };
13
14    return (
15      <SafeAreaView style={styles.container}>
16        <Header cityName={locationName}></Header>
17        <WeatherInfo cityName={locationName}></WeatherInfo>
18        <LocationInput onCityChange={handleCityChange} />
19      </SafeAreaView>
20    );
21  };

```

ForecastView.js

```

1  import { Text, Button, StyleSheet, SafeAreaView, Image } from 'react-native';
2
3  const ForecastView = () => {
4
5    return (
6      <SafeAreaView style={styles.headerBackground}>
7        <Text style={styles.paragraph}>Testing for navigation -</Text>
8        <Text style={styles.paragraph}>Forecast View</Text>
9        <Image style={styles.logo} source={require('../assets/rain.jpg')} />
10      </SafeAreaView>
11    );
12  };
13

```

SettingView.js

```

import { Text, Button, StyleSheet, SafeAreaView, Image } from 'react-native';

const SettingView = () => {

  return (
    <SafeAreaView style={styles.headerBackground}>
      <Text style={styles.paragraph}>Testing for navigation -</Text>
      <Text style={styles.paragraph}>Settings Page</Text>
      <Image style={styles.logo} source={require('../assets/sun.jpg')} />
    </SafeAreaView>
  );
};

```

Conclusion:

By trying out all three of navigation methods, I would say stack navigation took more time and effort to use. This is due to its need of implementing navigational buttons to each of the pages. This method also requires a strong hierarchy structure which user cannot navigate with clear vision of how many pages there are and where they are at. This method I felt might be more suitable for maybe questionnaires or some sort of forms for user to fill in which requires some sort of order.

As for the other two types of methods (drawer and bottom tab), they are quite simple to implement, it only needs to be implemented at the root level without much setting to each sub-page. They both supply a very clear view of all pages and user can easily jump from page to page as they like. One of the main differences between the two methods is that the space available for the list/tabs. Since bottom tab only has the bottom space of the view, it can work

quite well with smaller amount of sub pages to navigate, but in the case larger amount of pages, it can be quite crowded and user will not be able to navigate through. The drawer method on the other hand can conquer this problem since the list can be much longer, especially if the scroll function is implemented. It provides more room for larger list of pages to navigate. The down side of this method is that extra step needs to be taken to view the list and it also obstructs the view while showing the list.

Out of all three methods and considering the content might be in the weather app we are aiming to make, I'd first consider bottom tab method for navigation, since at the moment there isn't any plan for a large amount of pages, and the pages doesn't need (or have to) be in certain navigational order, the bottom tab will provide the most clarity and freedom to users.

3.2 List items and FlatList component

Implement a simple screen for showing weather forecast as a list like (mon, tue, wed). Implement a list item component (e.g. WeatherForecastListItem.js) which shows the weather forecast list item with at least day, temperature, wind speed and a weather icon. Place the items in FlatList component of React Native. Fetch a weather forecast from e.g. OpenWeatherMap Api and populate the forecast with your FlatList and ListItem. You can also make some other example app (not necessarily a Weather app) if you prefer using some other REST API.

After some research about the scrolling function, it appears that the FlatList itself comes with scrolling function, and this is confirmed with testing the app in the on my own device. And with official documentation, the ScrollView and FlatList in a way are similar methods that can be used under different circumstances. I first implemented the flat list with static data to ensure the structure works, then I used the OpenWeatherMap API, which only provide the 5 day 3 hr forecast for the free subscription. I haven't implemented the update on city selection yet, but can be implemented later on.

First try with static data:

```

1  import { Text, SafeAreaView, StyleSheet, FlatList } from 'react-native';
2  import WeatherListItem from './WeatherListItem';
3
4  const WeatherDataView = () => {
5    return (
6      <SafeAreaView>
7        <Text style={styles.headerStyle}><Flat List with Static Data</Text>
8        <FlatList
9          data={WeatherData}
10         renderItem={({ item }) => (
11           <WeatherListItem
12             time={item.time}
13             description={item.description}
14             temperature={item.temperature}
15             icon={item.icon}
16           />
17         )}></FlatList>
18      </SafeAreaView>
19    );
20  };
21
22  const styles = StyleSheet.create({
23
24    headerStyle: {
25      margin: 5,
26      fontSize: 20,
27      fontWeight: 'bold',
28      textAlign: 'center',
29    },
30  });
31
32  > const WeatherData = [ ...
83  ];
84
85  export default WeatherDataView;
86

```

Weather Data

Flat List with Static Data

monday

sunny

-3

01d



tuesday

snow

-5

05d

wednesday

sunny

-8

02d



thursday

Implementing OpenWeatherMap API:

```

1  import { Text, SafeAreaView, StyleSheet, FlatList, Button } from 'react-native';
2  import WeatherListItem from './WeatherListItem';
3  import { useState } from 'react';
4
5  const WeatherDataView = () => {
6    const [weatherData, setWeatherData] = useState([]);
7    const city = 'Tampere'; // Set the city to the desired Location
8
9    const API_ENDPOINT = 'https://api.openweathermap.org/data/2.5/forecast';
10   const API_KEY = 'd213cb6747b9f0f6b320c3d78cfca46a';
11
12   const fetchWeatherData = async (city) => {
13     try {
14       const response = await fetch(
15         `${API_ENDPOINT}?q=${city}&units=metric&appid=${API_KEY}`
16       );
17       const data = await response.json();
18       setWeatherData(data.list);
19     } catch (error) {
20       console.error('Error fetching weather data:', error);
21     }
22   };
23
24   // Fetch weather data when the component renders
25   fetchWeatherData(city);
26

```

```

26
27   return (
28     <SafeAreaView>
29       <Text style={styles.headerStyle}>Flat List Forecast Data </Text>
30       <Text style={styles.headerStyle}>Tampere</Text>
31       <Button
32         title="Fetch Weather Forecast"
33         onPress={() => fetchWeatherData(city)}
34       />
35       <FlatList
36         data={weatherData}
37         keyExtractor={(item) => item.dt.toString()}
38         renderItem={({ item }) => (
39           <WeatherListItem
40             time={item.dt_txt}
41             description={item.weather[0].description}
42             temperature={item.main.temp}
43             icon={item.weather[0].icon}
44           />
45         )}></FlatList>
46     </SafeAreaView>
47   );
48 };
49
50 const styles = StyleSheet.create({
51   headerStyle: {
52     margin: 5,
53     fontSize: 20,
54     fontWeight: 'bold',
55     textAlign: 'center',
56   },
57 });
58
59 > const WeatherData = [ ...
109 ];
110
111 export default WeatherDataView;

```

Weather Data

Flat List Forecast Data

Tampere


FETCH WEATHER FORECAST

2023-11-23 03:00:00

overcast clouds

0.79

04n




2023-11-23 06:00:00

overcast clouds

0.31

04n




2023-11-23 09:00:00

broken clouds

-0.44

04d



▼

▼

▼

▼

Weather Data

Current Weather

Forecast

Flat List

4 Week exercises

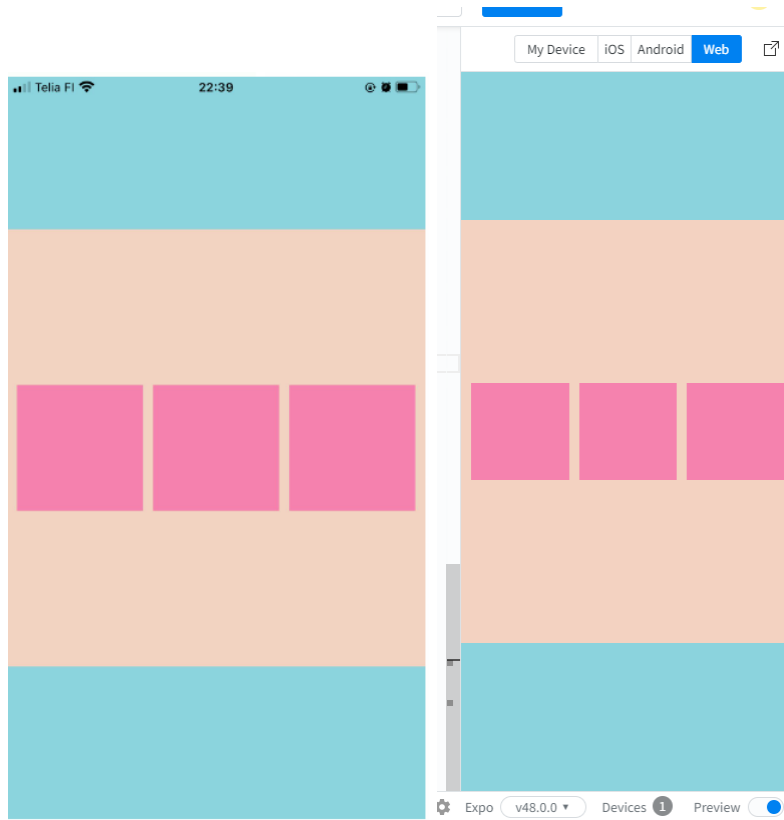
4.1 FlexBox Layout

Create a React Native layout using Flexbox that achieves the following design:

- **Header Section:** Occupies 20% of the screen height at the top.
- **Content Section:** Takes up 60% of the screen height in the middle. It should contain three equally spaced boxes horizontally.
- **Footer Section:** Occupies the remaining 20% of the screen height at the bottom.

Use **View** components to create the layout. Apply Flexbox properties to align and space the elements appropriately. Each box in the Content Section should be square (width equals height).

Ensure the layout is responsive across different device sizes. Customize the background color for each section for clear visibility.



```

1  import { Text, View, StyleSheet } from 'react-native';
2
3  const App = () => {
4    return (
5      <View style={styles.backGround}>
6        <View style={styles.headerZone}></View>
7        <View style={styles.contentZone}>
8          <View style={styles.contentBox}></View>
9          <View style={styles.contentBox}></View>
10         <View style={styles.contentBox}></View>
11        </View>
12        <View style={styles.footerZone}></View>
13      </View>
14    );
15  };
16
17  };
18
19  export default App;
20

```

```

21  const styles = StyleSheet.create({
22    backGround: {
23      flex: 1,
24    },
25
26    headerZone: {
27      flex: 1,
28      padding: 5,
29      backgroundColor: '#8bd3dd',
30    },
31
32    contentZone: {
33      flex: 3,
34      flexDirection: 'row',
35      justifyContent: 'space-around',
36      alignItems: 'center',
37      padding: 5,
38      backgroundColor: '#f3d2c1',
39    },
40
41    contentBox: {
42      flex: 1,
43      margin: 5,
44      aspectRatio: 1, // Keeps the width and height equal
45      backgroundColor: '#f582ae',
46    },
47
48    footerZone: {
49      flex: 1,
50      padding: 5,
51      backgroundColor: '#8bd3dd',
52    },
53  });
54

```

4.2 Platform Specific Components in React Native

Name some components that are available only in:

- Android
- iOS

In your weather (or other) application, implement something that uses Android specific components on Android platform and iOS specific components on iOS.

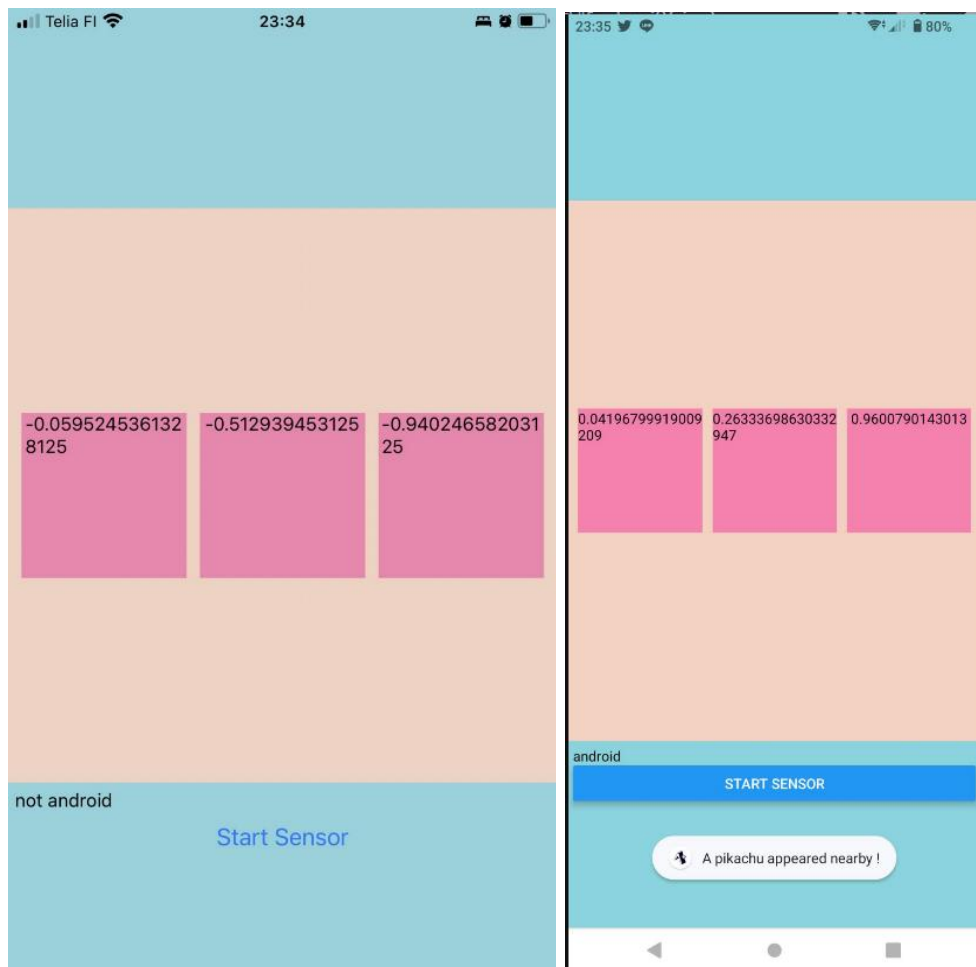
How can you know inside your code on which platform is your application currently running? Give a code example.

Android:

BackHandler, DrawerLayoutAndroid, PermissionsAndroid, ToastAndroid

iOS:

ActionSheetIOS



as ago. [See previous saves.](#) ✓

```

1  import { Text, View, StyleSheet, Button, ToastAndroid, Platform } from 'react-native';
2  import { useState } from 'react';
3  import { Accelerometer } from 'expo-sensors';
4
5  :
6  const App = () => {
7
8      const [sensorValues, setSensorValues] = useState({
9          x: 0,
10         y: 0,
11         z: 0
12     });
13     const [displayText, setDisplayText] = useState('what device are you using');
14     const [subscription, setSubscription] = useState(null);
15     const startSensors = () =>{
16         setSubscription(Accelerometer.addListener(setSensorValues));
17     }
18     if (Platform.OS === 'android'){
19         ToastAndroid.show('A pikachu appeared nearby !', ToastAndroid.SHORT);
20         setDisplayText('android');
21     }
22     else {
23         setDisplayText('not android');
24     }
25 }
26
27 return (
28     <View style={styles.backGround}>

```

In the code I used the Platform component in order to verify which device specific component (or text display) to use. I've tested it on both Android and iOS devices, the result are shown above.

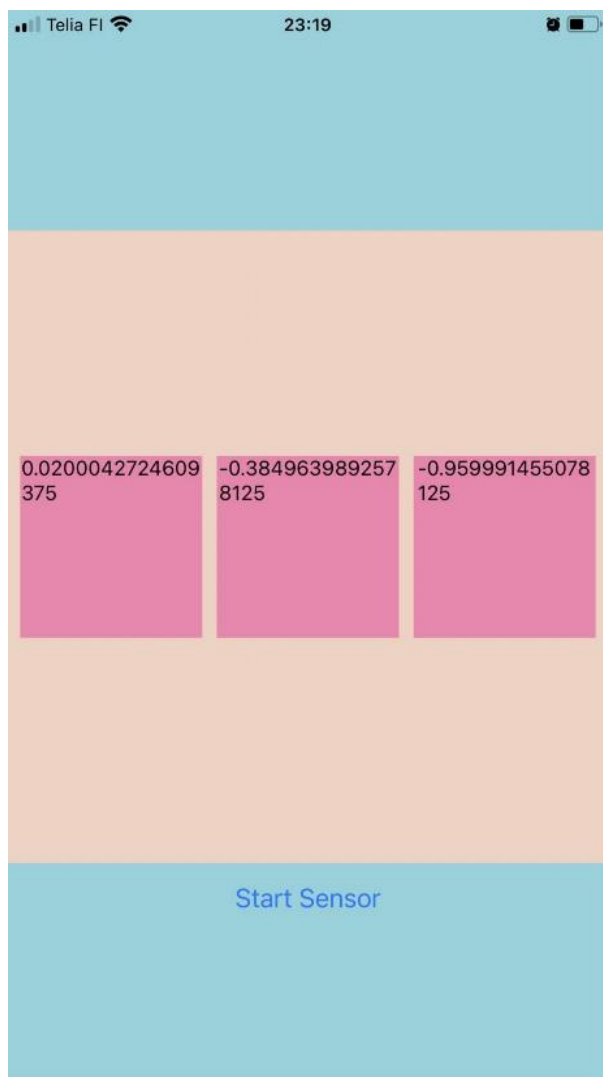
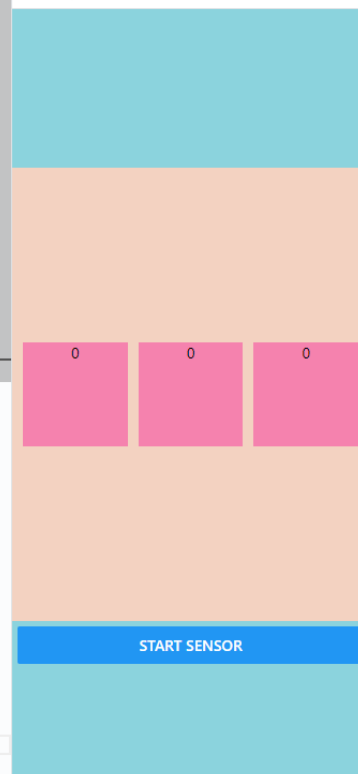
4.3 Utilizing Device APIs – Part I

Create a React Native app using Expo that displays real-time accelerometer data. You can display the data e.g., in the horizontal boxes of the exercise 1 (add a Text element inside each of the boxes for x, y and z).

```

1  import { Text, View, StyleSheet, Button } from 'react-native';
2  import { useState } from 'react';
3  import { Accelerometer } from 'expo-sensors';
4
5
6  const App = () => {
7
8    const [sensorValues, setSensorValues] = useState({
9      x: 0,
10     y: 0,
11     z: 0
12   })
13
14   const [subscription, setSubscription] = useState(null);
15   const startSensors = () => {
16     setSubscription(Accelerometer.addListener(setSensorValues));
17   }
18
19   return (
20     <View style={styles.background}>
21       <View style={styles.headerZone}></View>
22       <View style={styles.contentZone}>
23         <View style={styles.contentBox}>
24           <Text>{sensorValues.x}</Text>
25         </View>
26         <View style={styles.contentBox}>
27           <Text>{sensorValues.y}</Text>
28         </View>
29         <View style={styles.contentBox}>
30           <Text>{sensorValues.z}</Text>
31         </View>
32       </View>
33       <View style={styles.footerZone}>
34         <Button title='Start Sensor' onPress={startSensors}/>
35       </View>
36     </View>
37   );
38
39
40 };
```

My Device iOS Android Web



5 Week exercises

5.1 Utilizing Device APIs – Part II

Explain how permissions work through React Native. Which components are available in React Native for handling run-time permissions for e.g. location, camera or phone calls?

To your application, add a feature, which shows the current latitude and longitude on the display. The application should ask proper permissions from the user before accessing the Location API.

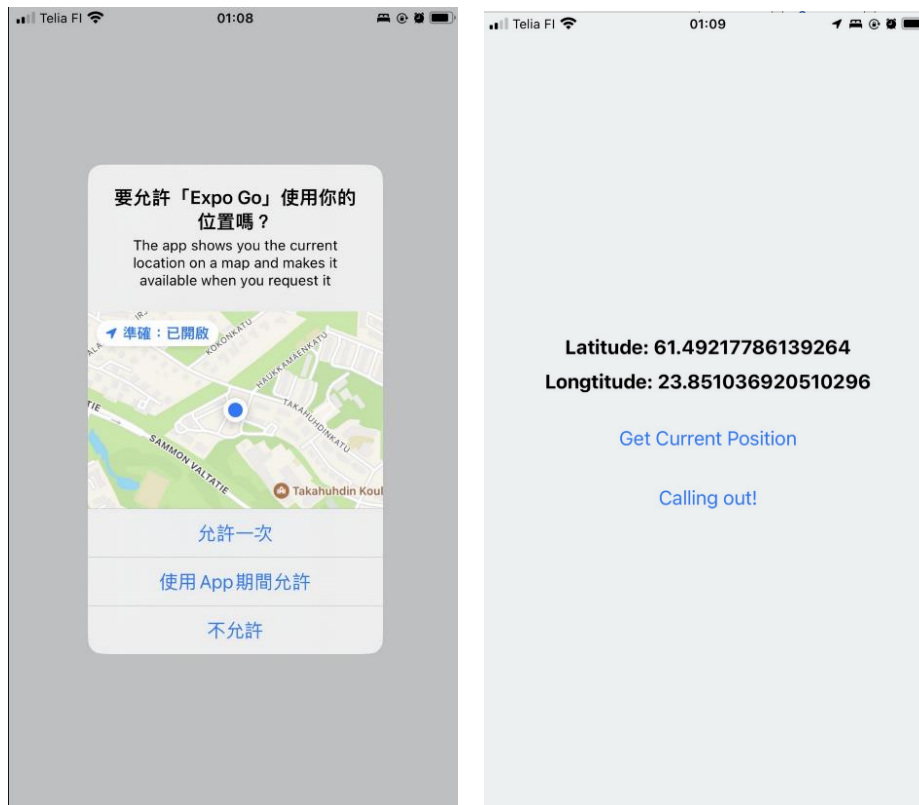
What do you need to do on Android or iOS platforms in order to get the permissions done (if you are not working through Expo project)?

With React Native and beside Expo libraries, there are functions that can obtain permission under Default Permissions section, there are also platform specific permissions like PermissionsAndroid

```

1  import { Text, SafeAreaView, Button, Linking, Platform, StyleSheet } from 'react-native';
2  import * as Location from 'expo-location';
3
4  import {useState} from 'react';
5
6  export default App = () => {
7
8    const [currentLocation, setCurrentLocation] = useState ({
9      currentLatitude: 'Unknown',
10     currentLongitude: 'Unknown',
11   });
12
13   const getCurrentPosition = async ()=>{
14     const result =await Location.requestForegroundPermissionsAsync();
15
16     console.log(result);
17
18     console.log("getting info");
19
20     const location =await Location.getCurrentPositionAsync({});
21     setCurrentLocation({
22       currentLatitude: location.coords.latitude,
23       currentLongitude: location.coords.longitude,
24     });
25     console.log(location);
26
27   }
28
29
30   return (
31     <SafeAreaView style={styles.container}>
32       <Text style = {styles.paragraph}>
33         {'Latitude: '+ currentLocation.currentLatitude}
34       </Text>
35       <Text style = {styles.paragraph}>
36         {'Longitude: '+ currentLocation.currentLongitude}
37       </Text>
38       <Text>{' '} </Text>
39       <Button
40         title='Get Current Position'
41         onPress={getCurrentPosition}
42         style = {styles.buttons}
43       />
44       <Text>{' '} </Text>
45       <Button
46         title='Calling out!'
47         onPress={getCurrentPosition}
48         style = {styles.buttons}
49       />
50     </SafeAreaView>
51   );
52 }

```



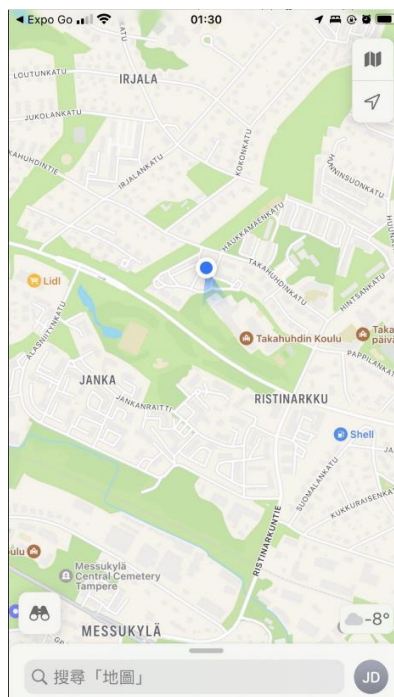
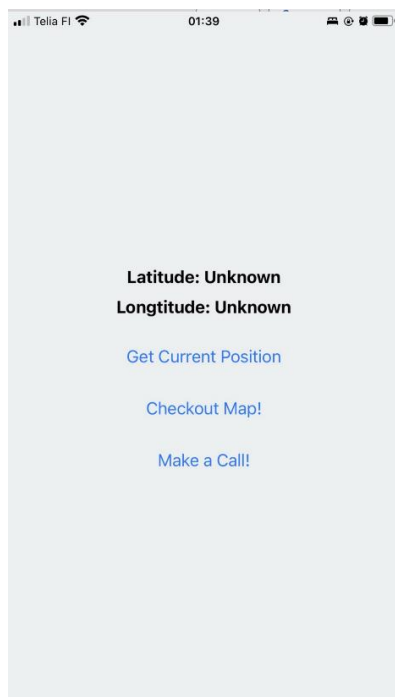
5.2 Using the Linking Module

Get familiar with Linking module of React Native. To your application, add a feature, where the user can click to open Maps on the current location. How would you implement routing to a specific destination in you React Native Expo app? Also add another button to your GUI to call to a phone number specified in a separate TextInput element. Which permissions would your app need in order to perform a phone call?

```

27   }
28
29   const showMapWithCurrentLocation = () => {
30     const url = Platform.select({
31       android: `geo:${currentLocation.currentLatitude},${currentLocation.currentLongitude}`,
32       ios: `maps:${currentLocation.currentLatitude},${currentLocation.currentLongitude}`,
33     });
34     Linking.openURL(url);
35   };
36
46   <Button
47     title='Get Current Position'
48     onPress={getCurrentPosition}
49     style = {styles.buttons}
50   />
51   <Text>{' '} </Text>
52   <Button
53     title='Checkout Map!'
54     onPress={showMapWithCurrentLocation}
55     style = {styles.buttons}
56   />
57   <Text>{' '} </Text>
58   <Button
59     title='Make a Call!'

```



```

    };

    const callSomeOne = () => {
      const url = `tel:+1234567`;
      Linking.openURL(url);
    };

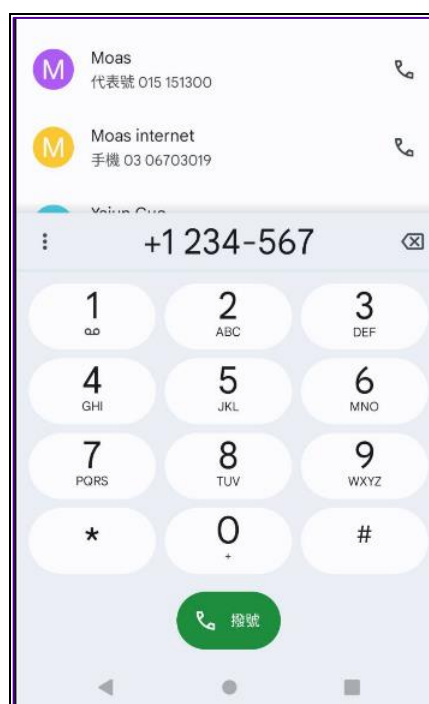
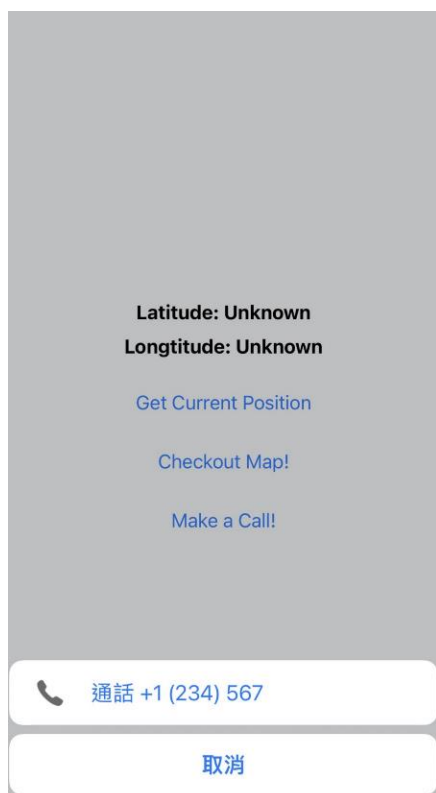
    return (
      <SafeAreaView style={styles.container}>

```

```

      style = {styles.buttons}
    />
    <Text>{' '} </Text>
    <Button
      title='Make a Call!'
      onPress={callSomeOne}
      style = {styles.buttons}
    />
  </SafeAreaView>
);

```



In the implementation of making a phone call, it's not necessary to ask user for phone call in both iOS and Android device if it's just directing to dial the number provided, but permission is needed if the app needs to access the user's contact list.

5.3 Yet some other Device APIs

Select to implement one of the following in your React Native Expo App:

- *Discover and list the Bluetooth devices in range*
- *Take a picture with Camera and display it on your app*
- *Read and monitor the current battery status and display the current battery percentage on your app display*

```

1  import {
2    Text,
3    SafeAreaView,
4    View,
5    Button,
6    Linking,
7    Platform,
8    Image,
9    TouchableOpacity,
10   StyleSheet,
11 } from 'react-native';
12
13 import * as Location from 'expo-location';
14 import { Camera, CameraType } from 'expo-camera';
15
16 import { useState, useRef } from 'react';

```



```

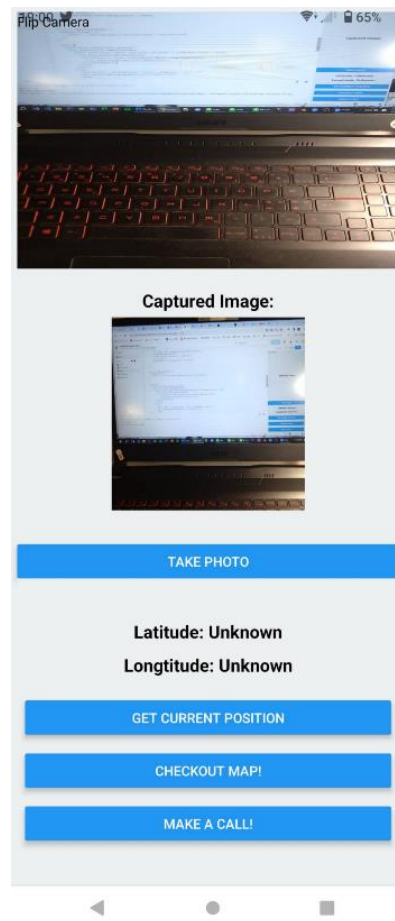
54
55 //activating camera area
56 const [type, setType] = useState(CameraType.back);
57 const [capturedImage, setCapturedImage] = useState(null);
58 const cameraRef = useRef(null);
59
60 const takePhoto = async () => {
61   // Request camera permissions when taking a photo
62   const cameraAllow = await Camera.requestCameraPermissionsAsync();
63   console.log(cameraAllow);
64   console.log('Camera is on!');
65
66   if (cameraAllow.status !== 'granted') {
67     console.log('Camera permission not granted');
68     return;
69   }
70
71   if (cameraRef.current) {
72     console.log('getting ready for photo!');
73     try {
74       const photo = await cameraRef.current.takePictureAsync();
75       setCapturedImage(photo.uri);
76       console.log('Capturing photo...', photo.uri);
77     } catch (error) {
78       console.error('Error taking picture:', error);
79     }
80   }
81 };
82
83 function toggleCameraType() {
84   setType((current) =>
85     current === CameraType.back ? CameraType.front : CameraType.back
86   );
87 }
88

```

```

<View style={styles.buttonContainer}>
  <TouchableOpacity style={styles.button} onPress={toggleCameraType}>
    <Text style={styles.text}>Flip Camera</Text>
  </TouchableOpacity>
</View>
</Camera>
<View
  style={{ flex: 1, justifyContent: 'center', alignItems: 'center' }}>
  <Text style={styles.paragraph}>Captured Image:</Text>
  <Image
    source={{ uri: capturedImage }}
    style={{ width: 200, height: 200 }}
  />
</View>
<Text> </Text>
<Button title="Take Photo" onPress={takePhoto} style={styles.buttons} />
</View>

```



5.4 Publishing APPs

Explain the steps on how you could publish your React Native Expo app in the Play store?

With React Native, Steps are different for Google Play and Apple App store, detailed steps with all then commends can be found on the official React Native documentation site:

<https://reactnative.dev/docs/signed-apk-android>

<https://reactnative.dev/docs/publishing-to-app-store>

But there's another path via using Expo.

<https://docs.expo.dev/deploy/build-project/>

it starts with creating a production build, it is submitted to app stores for release to the general public with the intension of "Test-flight". Then developer account will be needed, and this part have to follow the platform's instructions. After obtaining the account, developers need to generate or provide app signing credentials, this can also be done following the instructions (the links are in the expo documentation related pages). After these steps, the app can be submitted to the stores via commend line.

Final project

Implement a React Native application, which utilizes some data from selected web service. The remote service can be for example weather, traffic, University live info or anything that is openly downloadable (HTTP/JSON etc.). The app should use more than one screen and should use some navigation between the screens. Use also some app wide settings in the app as well and some selected device APIs like location or sensors (as an example). The app should use some Android specific components on Android and iOS specific components on iOS.

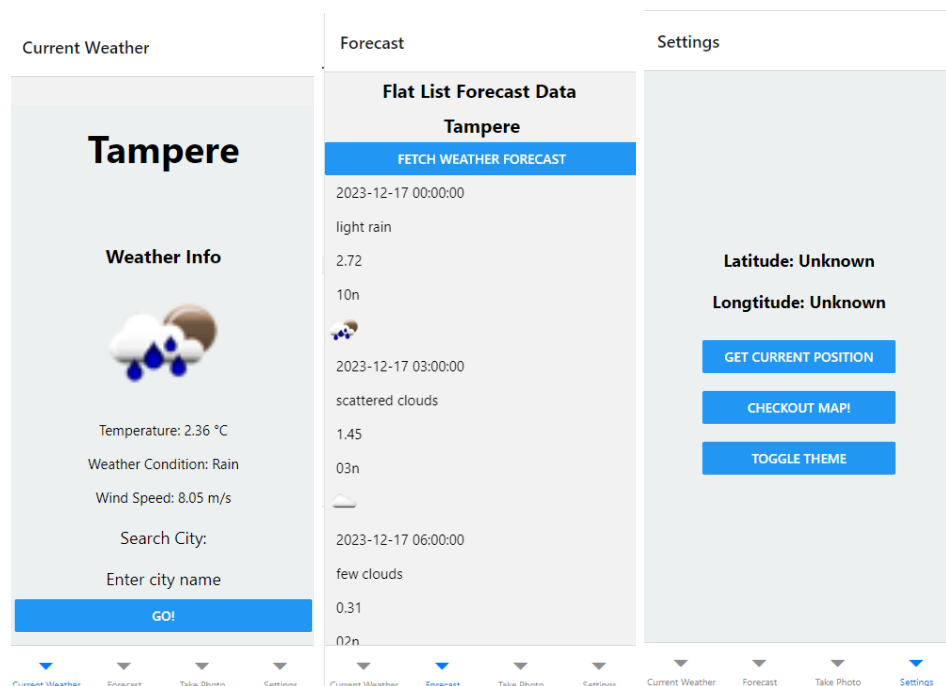
1. Data from selected web service: OpenWeatherMap API

```

3
4  const WeatherInfo = ({ cityName }) => {
5    const [weatherData, setWeatherData] = useState(null);
6    const API_KEY = 'd213cb6747b9f0f6b320c3d78cfa46a';
7    const API_ENDPOINT = 'https://api.openweathermap.org/data/2.5/weather';
8
9    const fetchWeatherData = async (city) => {
10     try {
11       const response = await fetch(
12         `${API_ENDPOINT}?q=${city}&units=metric&appid=${API_KEY}`
13       );
14       const data = await response.json();
15       setWeatherData(data);
16     } catch (error) {
17       console.error('Error fetching weather data:', error);
18     }
19   };
20   const handleCityChange = (newCity) => {
21     fetchWeatherData(newCity);
22   };
23
24   // Fetch data when the cityName changes
25   if (cityName) {
26     handleCityChange(cityName);
27   }
28
29   //using the provided icon from the openweather api
30   const getWeatherIcon = () => {

```

2. More than one screen and navigation: BottomTabNavigator / Current Weather view, Forecast View, Take Photo view, Settings view



```

import { createBottomTabNavigator } from '@react-navigation/bottom-tabs';
import { NavigationContainer } from '@react-navigation/native';

import CurrentWeatherView from './components/CurrentWeatherView';
import SettingView from './components/SettingView';
import ForecastView from './components/ForecastView';
import TakePhoto from './components/TakePhoto';

const Tab = createBottomTabNavigator();

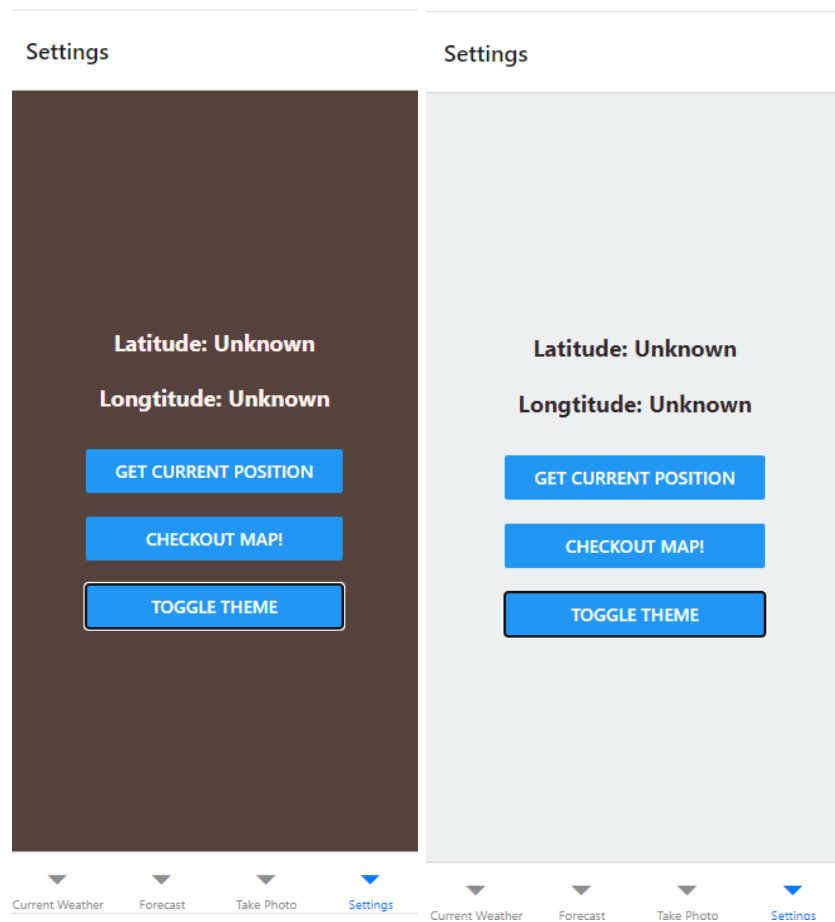
function MyTabs() {
  return (
    <Tab.Navigator useLegacyImplementation>
      <Tab.Screen name="Current Weather" component={CurrentWeatherView} />
      <Tab.Screen name="Forecast" component={ForecastView} />
      <Tab.Screen name="Take Photo" component={TakePhoto} />
      <Tab.Screen name="Settings" component={SettingView} />
    </Tab.Navigator>
  );
}

const MyWeatherApp = () => {
  return (
    <NavigationContainer>
      <MyTabs />
    </NavigationContainer>
  );
};

export default MyWeatherApp;

```

3. App wide settings: dark / light theme toggle



Set the toggle function and style variable for dark and light theme in ThemeConext.js:

```

1  import React, { createContext, useContext, useState } from 'react';
2  import { useColorScheme } from 'react-native';
3
4  const ThemeContext = createContext();
5
6  export const useTheme = () => {
7    return useContext(ThemeContext);
8  };
9
10 export const ThemeProvider = ({ children }) => {
11   const initialColorScheme = useColorScheme();
12   const [colorScheme, setColorScheme] = useState(initialColorScheme);
13
14   const toggleTheme = () => {
15     const newColorScheme = colorScheme === 'light' ? 'dark' : 'light';
16     setColorScheme(newColorScheme);
17   };
18
19   const theme = {
20     colorScheme,
21     container: {
22       flex: 1,
23       alignItems: 'center',
24       justifyContent: 'center',
25       backgroundColor: colorScheme === 'dark' ? '#55423d' : '#ecf0f1',
26     },
27     text: {
28       color: colorScheme === 'dark' ? '#fff3ec' : '#33272a',
29       textAlign: 'center',
30     },
31   },
32   toggleTheme,
33 };
34
35 return (
36   <ThemeContext.Provider value={theme}>
37     {children}
38   </ThemeContext.Provider>
39 );
40

```

Application of the method in ThemeView.js:

```

1  // ThemedView.js
2  import { View, StyleSheet, Text } from 'react-native';
3  import { useTheme } from '../ThemeContext';
4
5  const ThemedView = ({ children }) => {
6    const { container, text } = useTheme();
7    return <View style={container}><Text style={text}>{children}</Text>
8  </View>;
9  };
10
11 const styles = StyleSheet.create({
12   container: {
13     flex: 1,
14   },
15 });
16
17 export default ThemedView;
18

```

Application of the the theme mode in each tab:

```

1  import { useState } from 'react';
2  import { StyleSheet } from 'react-native';
3
4  import Header from '../components/Header';
5  import WeatherInfo from '../WeatherInfo';
6  import LocationInput from '../LocationInput';
7  import ThemedView from '../ThemedView';
8
9
10 const CurrentWeatherView = () => {
11   const [locationName, setLocationName] = useState('Tampere');
12   const handleCityChange = (newCity) => {
13     setLocationName(newCity); // Update the city name
14   };
15
16   return (
17     <ThemedView>
18       <Header cityName={locationName}></Header>
19       <WeatherInfo cityName={locationName}></WeatherInfo>
20       <LocationInput onCityChange={handleCityChange} />
21     </ThemedView>
22   );
23 };
24
25

```

Implementation of the ThemeProvider to enable app-wide manipulation of theme mode:

```

10
11 import { ThemeProvider } from './components/ThemeContext';
12
13 const Tab = createBottomTabNavigator();
14
15 function MyTabs() {
16
17   return (
18     <Tab.Navigator useLegacyImplementation>
19       <Tab.Screen name="Current Weather" component={CurrentWeatherView} />
20       <Tab.Screen name="Forecast" component={ForecastView} />
21       <Tab.Screen name="Take Photo" component={TakePhoto} />
22       <Tab.Screen name="Settings" component={SettingView} />
23     </Tab.Navigator>
24   );
25 }
26 const MyWeatherApp = () => {
27   // const [LocationName, setLocationName] = useState('Tampere');
28   // const handleCityChange = (newCity) => {
29   //   setLocationName(newCity); // Update the city name
30   // };
31
32   return (
33     <ThemeProvider>
34       <NavigationContainer>
35         <MyTabs />
36       </NavigationContainer>
37     </ThemeProvider>
38   );
39 }
40
41 };

```

4. Selected device API: get current position

```

7 //Location request
8 const [currentLocation, setCurrentLocation] = useState({
9   currentLatitude: 'Unknown',
10   currentLongitude: 'Unknown',
11 });
12
13 const getCurrentPosition = async () => {
14   const result = await Location.requestForegroundPermissionsAsync();
15
16   console.log(result);
17
18   console.log('getting info');
19
20   const location = await Location.getCurrentPositionAsync({});
21   setCurrentLocation({
22     currentLatitude: location.coords.latitude,
23     currentLongitude: location.coords.longitude,
24   });
25   console.log(location);
26 };

```

5. Platform specific components: opening map function

```

//open map request
const showMapWithCurrentLocation = () => {
  const url = Platform.select({
    android: `geo:${currentLocation.currentLatitude},${currentLocation.currentLongitude}`,
    ios: `maps:${currentLocation.currentLatitude},${currentLocation.currentLongitude}`,
  });
  Linking.openURL(url);
};

```

Sources used with exercises

1. React Native <https://reactnative.dev/>
2. 8 Best Android Frameworks for App Development in 2023 <https://www.intellectsoft.net/blog/best-android-frameworks/>
3. Platform architecture <https://developer.android.com/guide/platform>
4. Develop apps for iOS <https://developer.apple.com/tutorials/app-dev-training/>
5. Top 4 iOS App Development Frameworks In 2023 <https://www.linkedin.com/pulse/top-4-ios-app-development-frameworks-2023-jignesh-thanki/>
6. Chat GPT <https://chat.openai.com/>
7. Networking – React Native <https://reactnative.dev/docs/network#using-fetch>
8. Permissions <https://docs.expo.dev/guides/permissions/>
9. Publishing to Apple App Store <https://reactnative.dev/docs/publishing-to-app-store>
10. Publishing to Google Play Store <https://reactnative.dev/docs/signed-apk-android#macos>
11. Expo Camera <https://docs.expo.dev/versions/latest/sdk/camera/>
12. Build your project for app stores <https://docs.expo.dev/deploy/build-project/>
13. React Native - Appearance <https://reactnative.dev/docs/appearance>
14. Expo Documentation - Color themes <https://docs.expo.dev/develop/user-interface/color-themes/>