Tampereen ammattikorkeakoulu

Mobile App Development 2

Study diary

Jingjing Yang

Contents

1 Week exercises

1.1 Written answers to questions

1.1.1 Cross-platform mobile development

Explain the term "Cross-platform mobile development".

Cross-platform mobile development refers to the creation of software applications that are compatible with multiple mobile operating systems like Android and iOS or even Windows and Blackberry, reducing the duplication of effort by developers.

Instead of creating separate versions of the same application for each individual operating system, cross-platform mobile development allows for the creation of a universal application. This not only streamlines the development process but also ensures a consistent user experience across all platforms.

1.1.2 Frameworks/options

What different frameworks/options you have when making a mobile application for Android and iOS?

Android frameworks include not only React Native, Flutter, and Xamarin, but also Kotlin, Java, and Ionic.

For iOS development, in addition to React Native and Flutter, Swift UI is also a popular choice. Other options include Objective-C and Cordova, which also provide reliable solutions for iOS app development.

Top 10: Flutter, Ionic, React Native, Xamarin, NativeScript, Node.Js, PhoneGap, Appcelerator, Corona SDK, Sencha Touch

1.1.3 React Native

Give a brief overview of React Native.

React Native is a JavaScript-based framework used to develop real, natively rendering mobile applications for iOS and Android. React Native uses the same basic UI building blocks as regular iOS and Android apps, making it a versatile tool for cross-platform app development.
Developed by Facebook, it allows developers to write code in JavaScript and render it with native code.
React Native is based on React, Facebook's JavaScript library for building user interfaces, but targets mobile platforms instead of the browser.
React Native development tools include text editors and IDEs like VSCode, Atom, Sublime, Vim, and Emacs. For debugging, Chrome Dev Tools or React Native Debugger can be used. Node.js and npm are employed to load packages into the project. Android Studio and Xcode are utilized for Android

and iOS build and development, respectively. Jest, a JavaScript testing framework, is used to write unit tests for React Native applications. Additionally, Expo, a free and open-source toolchain built around React Native, is available.
Basic tutorials for React Native can be found on various online platforms such as Codecademy, Udemy, and YouTube. Websites like freeCodeCamp or W3Schools also provide comprehensive guides.
The official documentation for React Native can be found on the React Native website (https://reactnative.dev/docs/getting-started).
Pros of React Native include code reusability across different platforms and a large supportive community. It also offers a faster and cost-effective development process as compared to native Java in Android. Cons inclued slower performance compared to native apps (con), and difficulty in accessing some native APIs.
Personally, I believe React Native is a efficient framework that significantly accelerates the development process of high-quality mobile applications. However, despite its various advantages, it does have its own set of limitations. For instance, it may not offer the same level of performance as native apps and accessing some native APIs could pose challenges. Therefore, before adopting React Native for a particular project, it's crucial to consider whether the framework aligns with the specific requirements and objectives of the project.

1.2 Code Lab - a simple React Native UI for currency converter

1.2.1 React Native code

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

const Converter = () => {
  const [inputValue, setInputValue] = useState('');
  const [resultValue, setResultValue] = useState('');

  const convertCurrency = () => {
    const result = inputValue * 7.81;
    setResultValue(result.toString());
  };

  return (
    <View style={styles.container}>
      <TextInput
        style={styles.input}
        onChangeText={text => setInputValue(text)}
        value={inputValue}
        keyboardType='numeric'
        placeholder='Enter amount in EUR'
      />
      <Button onPress={convertCurrency} title='Convert EUR to CNY' />
      <Text style={styles.resultText}>{"Converted amount in CNY:\n" + resultValue}</Text>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    padding: 16,
  },
  input: {
    height: 40,
    borderColor: 'gray',
    borderWidth: 1,
    marginBottom: 20,
    padding: 10,
  },
  resultText: {
    marginTop: 20,
    fontSize: 18,
    textAlign: 'center',
  },
});

export default Converter;
```

1.2.2 Test on Expo Snack

1.2.3 Thoughts and findings

The use of JavaScript and similar syntax (such as components, state, and props) in both React Native and React simplified my learning process for React Native, thanks to my previous experience with React.
React builds on the Virtual DOM and is used for web development, while React Native uses native components as its building blocks, which allows it to render mobile applications that are indistinguishable from an app built using Objective-C or Java. Therefore, React Native also introduced new concepts to learn.

1) Element vs. component: Instead of using HTML elements like div, h1, or button as in React, it uses native components provided by React Native, such as View, Text, TextInput, and Button.
2) Handling text input: In React, we use an <input type="text" /> element with an onChange event handler. In React Native, we use a <TextInput /> component with an onChangeText prop.
3) Setting Button Titles: In React, the title for a button is set using the children prop -<Button> Click me </Button>. However, in React Native, the title prop is used - <Button title="Click me" />.
4) Interaction Naming: In React, we use the onClick event handler to manage button clicks. On the other hand, React Native utilizes the onPress event handler for the same purpose.
5) StyleSheet: web development with React employs CSS classes or inline styles directly on HTML elements, typically using CSS syntax in separate files or styled-components. Conversely, In React Native, we use the StyleSheet.create() method to define styles as a JavaScript object, which can be applied to components via the style prop. This simplifies CSS and ensures style validity.
6) Styling property names: React Native uses JavaScript to style components with a syntax similar to CSS but with different property names as they follow a camelCase convention. For instance, 'background-color' in CSS is translated to 'backgroundColor' in React Native.

The code lab is generally straightforward, especially with the counter app example provided in the lesson. I also tested the app in my Expo Go. However, I still found it helpful to refer to the official React Native documentation. The Chat GPT tool was particularly useful for guiding me through setting up the development environment and understanding React Native's core concepts. Its ability to provide immediate, relevant responses significantly improved my learning experience and efficiency.

2 Week exercises

2.1 Written answers to questions

What alternatives are there for fetching?

Apart from using the Fetch API, other alternatives for fetching data in JavaScript include:

- **Axios:** A very popular JavaScript library for making HTTP requests. It supports both browser and server environment. It provides a more powerful and flexible feature set and it can handle requests and responses in JSON format automatically.
- **jQuery AJAX:** jQuery provides a set of AJAX functions that can be used for sending asynchronous requests in browser environment.
- **XMLHttpRequest:** This is an older way of making HTTP requests and has been largely replaced by the Fetch API and other libraries. It's still supported in all browsers, but it's more complex and verbose.
- **SuperAgent:** It's a lightweight, flexible and readable AJAX library. It handles data types automatically and can also be extended with plugins.
- **GraphQL Client Libraries (like Apollo or Relay):** If you're using GraphQL, you'll likely use a dedicated library to make requests, like Apollo Client or Relay.

2.2 Code Lab - Weather App for Android and iOS

2.2.1 Description of Weather App

- The application's user interface consists of three main components: the Header, the WeatherScreen, and the InputButton.
    - Components are arranged in separate JS files under the “Components” directory.
    - The Header component displays the current weather location which is passed as a prop.
    - The WeatherScreen component encompasses the display of the weather conditions, including an icon representing the weather, the temperature, and the wind speed.
    - The InputButton component includes a field for the user to input a location and a button to refresh the weather information based on the entered location.
- Styles are used throughout the application.
- The application's user interface is responsive, designed to scale to different portrait screen sizes by using percentage and flex-based layout techniques.
- The application fetches data from the OpenWeatherMap API using async/await and Fetch API.
- A feature where the user can select the city/location of the weather to be fetched is implemented.

2.2.2 React Native code

App.js

```
import { useState, useEffect } from 'react';
import { StyleSheet, View } from 'react-native';
import Header from './components/Header';
import WeatherScreen from './components/WeatherScreen';
import InputButton from './components/InputButton';

const App = () => {
  // State variables declaration
  const [location, setLocation] = useState('Beijing');
  const [weather, setWeather] = useState('');
  const [icon, setIcon] = useState('');
  const [temperature, setTemperature] = useState('');
  const [windSpeed, setWindSpeed] = useState('');
  const [inputValue, setInputValue] = useState('');

  // API key for OpenWeatherMap
  const API_KEY = '395dc02446c77c0ac922cb465d9a395b';

  // Function to fetch weather data from OpenWeatherMap API
  const getWeatherData = async (location) => {
    try {
      const response = await fetch(
        `https://api.openweathermap.org/data/2.5/weather?q=${location}&units=metric&appid=${API_KEY}`
      );

      if (!response.ok) {
        throw new Error(`HTTP error! Status: ${response.status}`);
      }

      const data = await response.json();
      const weather = data.weather[0].description;
      const icon = data.weather[0].icon;
      const temperature = data.main.temp;
      const windSpeed = data.wind.speed;

      // Return fetched data
      return { icon, weather, temperature, windSpeed };
    } catch (error) {
      console.error('Error fetching weather data:', error);
      throw error;
    }
  };

  // Fetching data whenever 'location' changes
  useEffect(() => {
    const fetchWeatherData = async () => {
      try {
        const weatherData = await getWeatherData(location);
        const { weather, icon, temperature, windSpeed } = weatherData;

        // Set state variables with fetched data
        setWeather(weather);
        setIcon(icon);
        setTemperature(temperature);
        setWindSpeed(windSpeed);
      } catch (error) {
        console.error('Error fetching weather data:', error);
      }
    };
    fetchWeatherData();
  }, [location]);

  // Function to update weather data based on user's input
  const refreshWeather = () => {
    if (inputValue) {
      setLocation(inputValue);
    }
  };
```

```
69
70    // Render components
71    return (
72      <View style={styles.container}>
73        <Header location={location} />
74        <WeatherScreen
75          icon={icon}
76          weather={weather}
77          temperature={temperature}
78          windSpeed={windSpeed}
79        />
80        <InputButton
81          inputValue={inputValue}
82          setInputValue={setInputValue}
83          refreshWeather={refreshWeather}
84        />
85      </View>
86    );
87  };
88
89  // Styles for the app
90  const styles = StyleSheet.create({
91    container: {
92      flex: 1,
93      paddingTop: '15%',
94      alignItems: 'center',
95      backgroundColor: '#F5F7F8'
96    }
97  });
98
99
100 // Exporting App component
101 export default App;
```

components/Header.js

```
1   import { View, Text, StyleSheet } from 'react-native';
2
3   const Header = ({ location }) => {
4     return (
5       <View style={styles.header}>
6         <Text style={styles.text}>{location}</Text>
7       </View>
8     );
9   };
10
11  const styles = StyleSheet.create({
12    header: {
13      height: '11%',
14      padding: '5%',
15      width: '100%',
16      backgroundColor: '#186F65',
17    },
18    text: {
19      color: '#fff',
20      fontSize: '36dp',
21      textAlign: 'center',
22    },
23  });
24
25
26  export default Header;
```

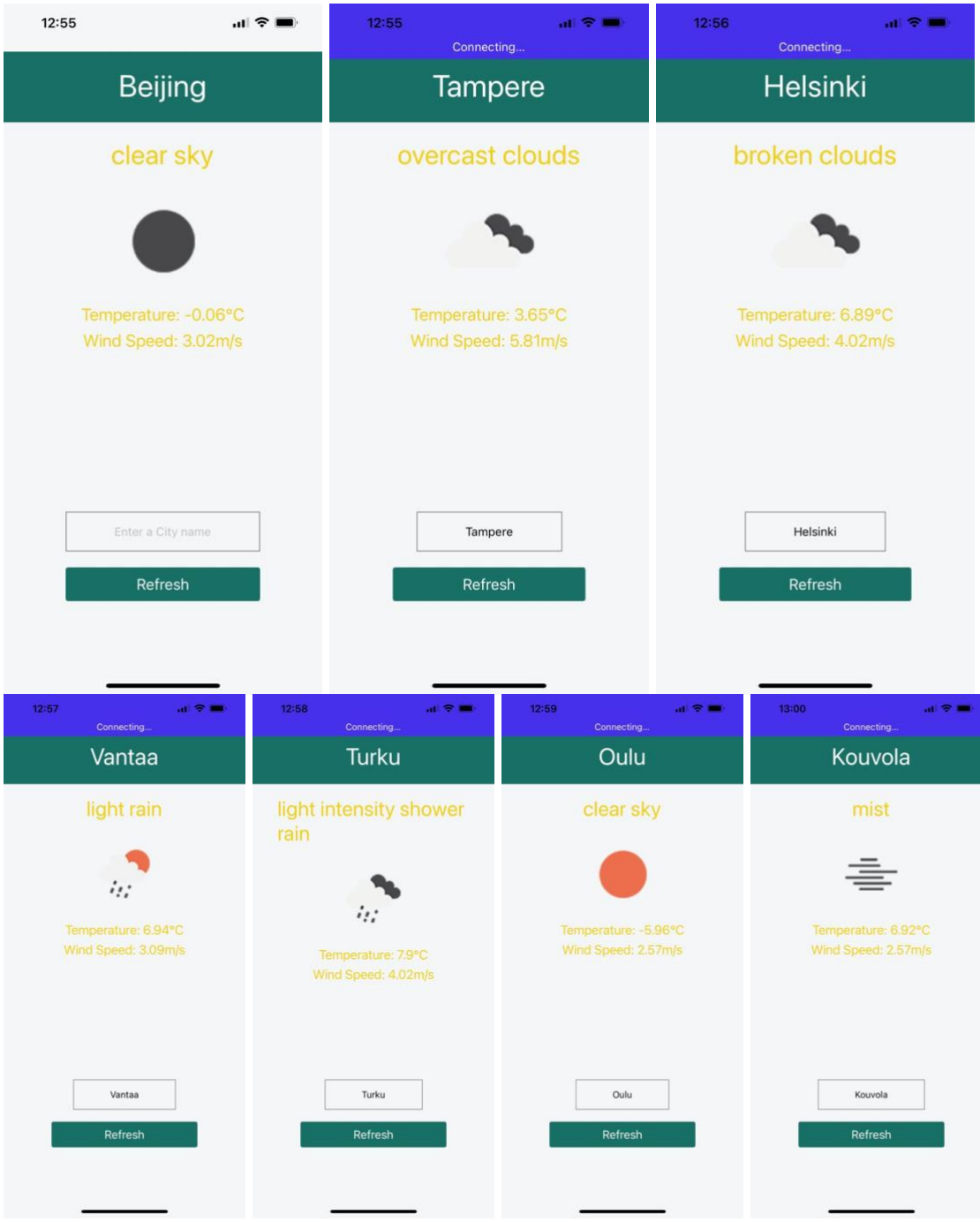
components/WeatherScreen.js

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

const WeatherScreen = ({ weather, icon, temperature, windSpeed }) => {

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

      <Text style={styles.textBig}>{weather}</Text>
    <Image source={{ uri: `https://openweathermap.org/img/wn/${icon}@2x.png` }}
style={styles.weatherIcon} />
      <Text style={styles.textSmall}>Temperature: {temperature}°C</Text>
      <Text style={styles.textSmall}>Wind Speed: {windSpeed}m/s</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'flex-start',
    alignItems: 'center',
    padding: '5%',
  },
  weatherIcon: {
    width: 150,
    height: 150,
  },
  textBig: {
    marginBottom: '2%',
    fontSize: '32dp',
    color: '#F4CE14',
  },
  textSmall: {
    marginBottom: '2%',
    fontSize: '20dp',
    color: '#F4CE14',
  },
});


export default WeatherScreen;
```

components/InputButton.js

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

const InputButton = ({ inputValue, setInputValue, refreshWeather }) => {
  return (
    <View style={styles.inputButtonContainer}>
      <TextInput
        style={styles.input}
        value={inputValue}
        onChangeText={(text) => setInputValue(text)}
        placeholder="Enter a City name"
      />
      <Button
        title="Refresh"
        onPress={refreshWeather}
        buttonStyle={styles.button}
        titleStyle={styles.buttonText}
      />
    </View>
  );
};

const styles = StyleSheet.create({
  inputButtonContainer: {
    flex: 1,
    justifyContent: 'center',
    paddingVertical: '2%',
  },
  input: {
    height: '15%',
    borderColor: 'gray',
    borderWidth: 1,
    marginBottom: '5%',
    paddingHorizontal: '15%',
    width: '100%',
    alignSelf: 'center', // to center the input
  },
  button: {
    backgroundColor: '#186F65',
    borderRadius: '2%',
    width: '100%',
    alignSelf: 'center', // to center the button
  },
  buttonText: {
    color: 'white',
  },
});

export default InputButton;
```

2.2.3 Test on my device through Expo app

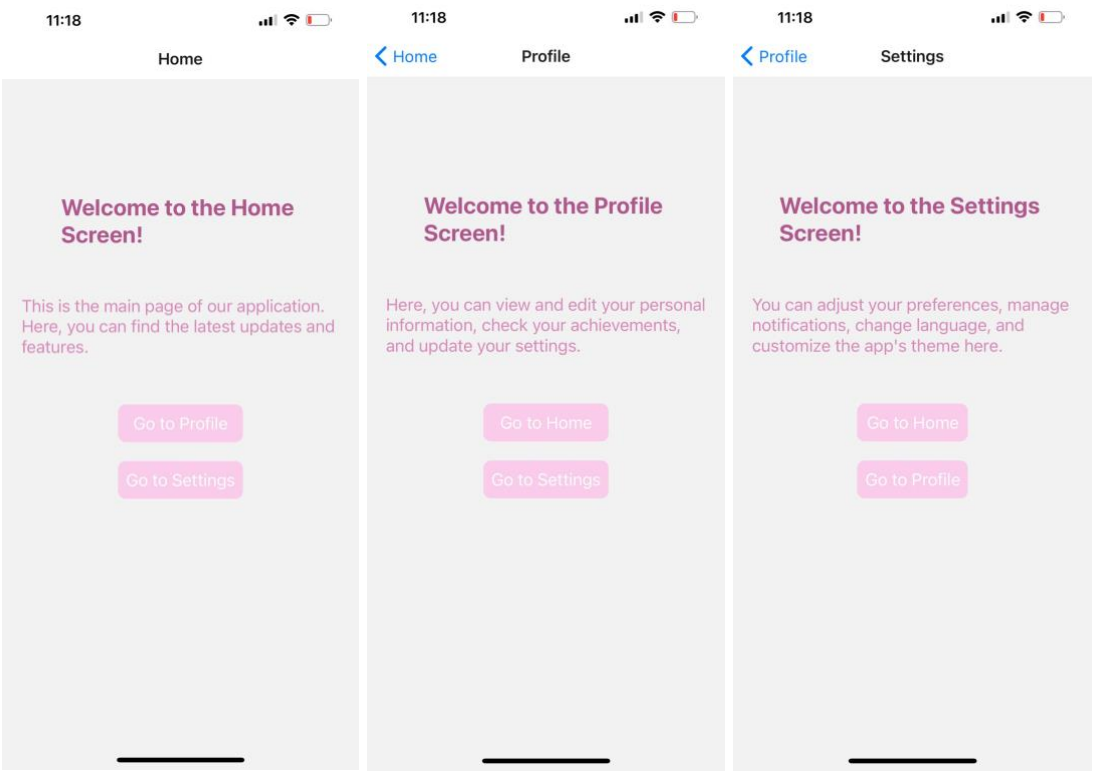


3 Week exercises

3.1 React Page Navigation

3.1.1 Stack Navigation

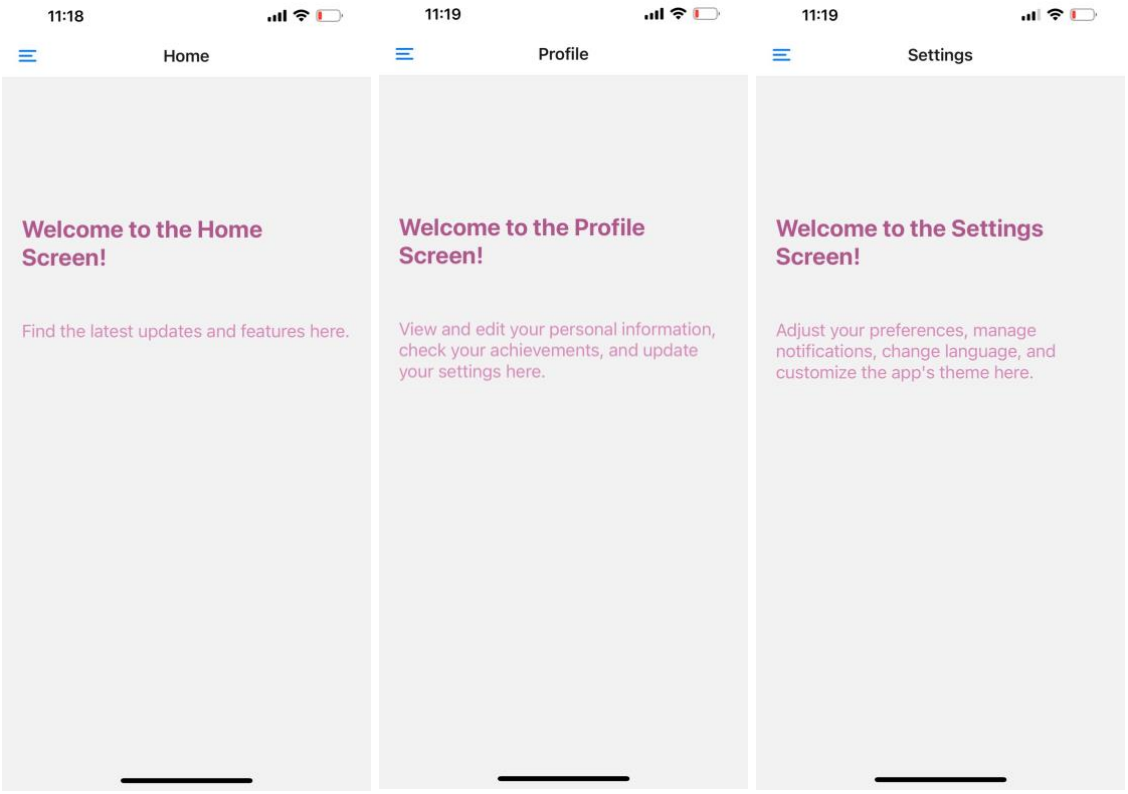
This code creates a stack navigator with three screens: Home, Profile, and Settings. Each screen is a function component that displays some text and two buttons to navigate to the other two screens. The stack navigator is used in the main App compone.

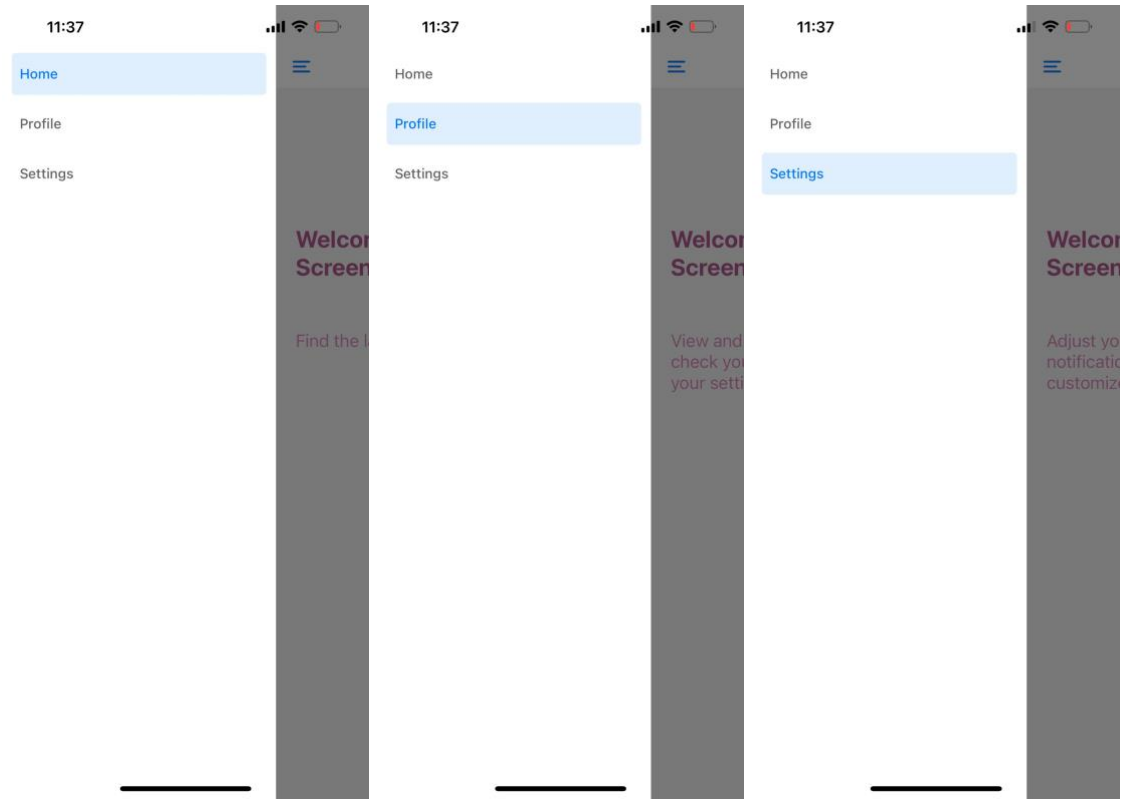


```jsx
import { View, Text, StyleSheet } from 'react-native';
import { Button } from 'react-native-elements';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';

function HomeScreen({ navigation }) {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Home Screen!</Text>
      <Text style={styles.description}>This is the main page of our application. Here, you can find the latest updates and features.</Text>
      <View>
        <Button buttonStyle={styles.button} title="Go to Profile" onPress={() => navigation.navigate('Profile')} />
        <Button buttonStyle={styles.button} title="Go to Settings" onPress={() => navigation.navigate('Settings')} />
      </View>
    </View>
  );
}

function ProfileScreen({ navigation }) {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Profile Screen!</Text>
      <Text style={styles.description}>Here, you can view and edit your personal information, check your achievements, and update your settings.</Text>
      <View>
        <Button buttonStyle={styles.button} title="Go to Home" onPress={() => navigation.navigate('Home')} />
        <Button buttonStyle={styles.button} title="Go to Settings" onPress={() => navigation.navigate('Settings')} />
      </View>
    </View>
  );
}

function SettingsScreen({ navigation }) {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Settings Screen!</Text>
      <Text style={styles.description}>You can adjust your preferences, manage notifications, change language, and customize the app's theme here.</Text>
      <View>
        <Button buttonStyle={styles.button} title="Go to Home" onPress={() => navigation.navigate('Home')} />
        <Button buttonStyle={styles.button} title="Go to Profile" onPress={() => navigation.navigate('Profile')} />
      </View>
    </View>
  );
}

const Stack = createStackNavigator();

function MyStack() {
  return (
    <Stack.Navigator>
      <Stack.Screen name="Home" component={HomeScreen} />
      <Stack.Screen name="Profile" component={ProfileScreen} />
      <Stack.Screen name="Settings" component={SettingsScreen} />
    </Stack.Navigator>
  );
}

export default function App() {
  return (
    <NavigationContainer>
      <MyStack />
    </NavigationContainer>
  );
}

const styles = StyleSheet.create({
  container: {
    padding: 20,
    marginTop:100,
    flex: 1,
    alignItems: 'center',
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 50,
    color:'#B0578D',
  },
  description: {
    fontSize: 18,
    marginBottom: 50,
    color:'#D988B9',
  },
  button: {
    backgroundColor: '#FACBEA',
    borderRadius: 8,
    marginBottom: 20,
  }
});
```

3.1.2 Drawer Navigation

This code creates a drawer navigator with three screens: Home, Profile, and Settings. Each screen is a function component that displays some text. The drawer navigator is used in the main App component.

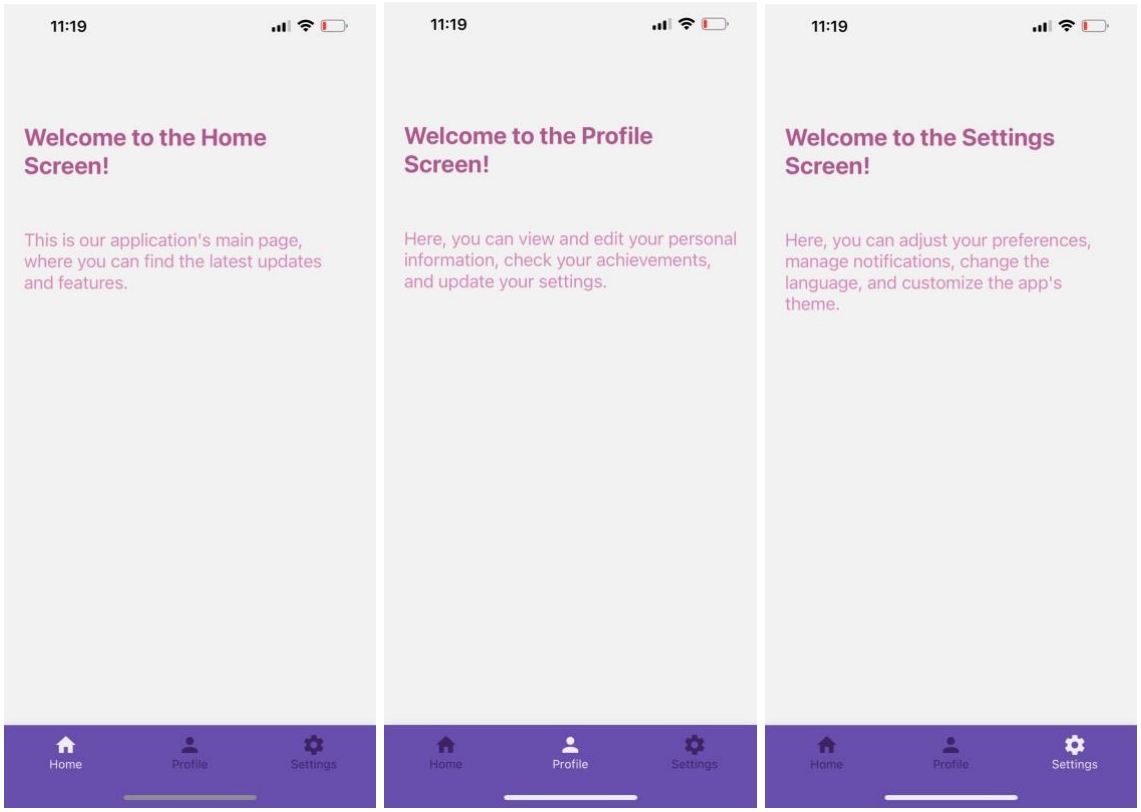




```
import { View, Text, StyleSheet } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createDrawerNavigator } from '@react-navigation/drawer';

function HomeScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Home Screen!</Text>
      <Text style={styles.description}>Find the latest updates and features here.</Text>
    </View>
  );
}

function ProfileScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Profile Screen!</Text>
      <Text style={styles.description}>View and edit your personal information, check your achievements, and update your settings here.</Text>
    </View>
  );
}

function SettingsScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Settings Screen!</Text>
      <Text style={styles.description}>Adjust your preferences, manage notifications, change language, and customize the app's theme here.</Text>
    </View>
  );
}

const Drawer = createDrawerNavigator();

function MyDrawer() {
  return (
    <Drawer.Navigator
      useLegacyImplementation
    >
      <Drawer.Screen name="Home" component={HomeScreen} />
      <Drawer.Screen name="Profile" component={ProfileScreen} />
      <Drawer.Screen name="Settings" component={SettingsScreen} />
    </Drawer.Navigator>
  );
}

export default function App() {
  return (
    <NavigationContainer>
      <MyDrawer />
    </NavigationContainer>
  );
}

const styles = StyleSheet.create({
  container: {
    padding: 20,
    marginTop: 100,
  },
  title: {
    marginTop: 20,
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 50,
    color: '#B0578D',
  },
  description: {
    fontSize: 18,
    marginBottom: 50,
    color: '#D988B9',
  },
});
```

3.1.3 Bottom Tabs Navigation

This code creates a material bottom tab navigator with three screens: Home, Profile, and Settings. Each screen is a function component that displays some text. The tab navigator is used in the main App component, and each tab is labeled with its name.



```
import { Text, View, StyleSheet } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createMaterialBottomTabNavigator } from '@react-navigation/material-bottom-tabs';

function HomeScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Home Screen!</Text>
      <Text style={styles.description}>This is our application's main page, where you can find the latest updates and features.</Text>
    </View>
  );
}

function ProfileScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Profile Screen!</Text>
      <Text style={styles.description}>Here, you can view and edit your personal information, check your achievements, and update your settings.</Text>
    </View>
  );
}

function SettingsScreen() {
  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Settings Screen!</Text>
      <Text style={styles.description}>Here, you can adjust your preferences, manage notifications, change the language, and customize the app's theme.</Text>
    </View>
  );
}

const Tab = createMaterialBottomTabNavigator();

export default function App() {
  return (
    <NavigationContainer>
      <Tab.Navigator
        initialRouteName="Home"
        activeColor="#f0edf6"
        inactiveColor="#3e2465"
        barStyle={styles.bar}
      >
        <Tab.Screen name="Home" component={HomeScreen} options={{ tabBarLabel: 'Home', tabBarIcon: 'home'}}
         />
        <Tab.Screen name="Profile" component={ProfileScreen}
          options={{ tabBarLabel: 'Profile', tabBarIcon: 'account'}}
         />
        <Tab.Screen name="Settings" component={SettingsScreen}
          options={{ tabBarLabel: 'Settings', tabBarIcon: 'cog' }}
        />
      </Tab.Navigator>
    </NavigationContainer>
  );
}

const styles=StyleSheet.create({
  container: {
    padding: 20,
    marginTop:100,
  },
  title: {
    fontSize: 24,
    fontWeight: 'bold',
    marginBottom: 50,
    color:'#B0578D',
  },
  description: {
    fontSize: 18,
    marginBottom: 50,
    color:'#D98889',
  },
  bar: {
    backgroundColor: '#694fad',
  }
});
```

3.2 Written answers to questions

Share your perspective on each of the navigation patterns from a user's point of view.

If your Weather Application had multiple screens, which navigation pattern would you choose?

From a usability viewpoint, the Stack navigation is simple and intuitive. It's ideal for apps with a linear flow, where users move sequentially from one screen to another.

The Drawer navigation was interesting as it provides an organized way to access different parts of the app from a side panel. I believe it's useful for apps with numerous sections or categories.

The Bottom Tabs navigation was user-friendly and provided quick access to different screens. It's great for apps where users frequently switch between different tabs.

For a Weather Application with multiple screens, I would choose the Bottom Tabs navigation. It allows users to easily switch between screens like current weather, forecast, and settings without needing to return to a home screen.

3.3 List Items and FlatList Component

The code below fetches weather forecast data for a specific location from the OpenWeatherMap API. After fetching the data, it processes and formats it for display. The forecast data includes the date and the day of the week, a weather icon, temperature, and wind speed.

The data is organized into an array of objects, each representing a day's forecast. The array is then passed to a FlatList component for rendering on the screen. Each item in the FlatList is rendered using a custom Item component, which displays the day, temperature, wind speed, and an icon representing the weather condition.

The FlatList component is part of the main App component, which fetches the data when it mounts using the useEffect hook and stores it in the local state using the useState hook.

```
import { useState, useEffect } from 'react';
import { SafeAreaView, FlatList, View, Image, Text, StyleSheet } from 'react-native';

// Location and API key for the OpenWeatherMap API
const location = 'Tampere';
const API_KEY = '395dc02446c77c0ac922cb465d9a395b';

// Function to fetch weather data from the OpenWeatherMap API
async function fetchWeatherData() {
  try {
    // Fetch the weather data
    const response = await fetch(`https://api.openweathermap.org/data/2.5/forecast?q=${location}&units=metric&appid=${API_KEY}`);
    const data = await response.json();

    // Map the response data to a more usable format
    let forecastData = data.list.map((item) => ({
      id: String(item.dt),
      day: new Date(item.dt * 1000).toLocaleDateString('en-GB', { weekday: 'short' }),
      temperature: Math.round(item.main.temp),
      windSpeed: item.wind.speed,
      icon: item.weather[0].icon,
    }));

    // Group data by day
    const groupedData = forecastData.reduce((acc, curr) => {
      (acc[curr["day"]] = acc[curr["day"]] || []).push(curr);
      return acc;
    }, {});

    // Calculate average temperature, maximum wind speed and most common icon for each day
    forecastData = Object.keys(groupedData).map(day => {
      const dayData = groupedData[day];
      const avgTemp = Math.round(dayData.reduce((a, b) => a + b.temperature, 0) / dayData.length);
      const maxWindSpeed = Math.max(...dayData.map(item => item.windSpeed)).toFixed(0);
      const icon = dayData.reduce((a, b) => dayData.filter(item => item.icon === a.icon).length >= dayData.filter(item => item.icon === b.icon).length ? a : b).icon;
      return {
        id: dayData[0].id,
        day: day,
        temperature: avgTemp,
        windSpeed: maxWindSpeed,
        icon: icon
      }
    });

    return forecastData;
  } catch (error) {
    console.error(error);
  }
}

// Component to render each weather item
function Item({ day, temperature, windSpeed, icon }) {
  return (
    <View style={styles.item}>
      <Text style={styles.day}>{day}</Text>
      <Image source={{ uri: `https://openweathermap.org/img/wn/${icon}@2x.png` }} style={styles.weatherIcon} />
      <Text style={styles.data}>{temperature}°C</Text>
      <Text style={styles.data}>{windSpeed} m/s</Text>
    </View>
  );
}

// Main App component
export default function App() {
  const [data, setData] = useState([]);

  // Use an effect to fetch the data once on component mount
  useEffect(() => {
    fetchWeatherData().then(setData);
  }, []);

  return (
    <SafeAreaView style={styles.container}>
      <View style={styles.header}>
        <Text style={styles.headerText}>{location}</Text>
      </View>
      <FlatList
        data={data}
        renderItem={({ item }) => (
          <Item day={item.day} temperature={item.temperature} windSpeed={item.windSpeed} icon={item.icon} />
        )}
        keyExtractor={item => item.id}
      />
    </SafeAreaView>
  );
}
```

```
87
88  // Styles for the components
89  const styles = StyleSheet.create({
90    container: {
91      flex: 1,
92      backgroundColor: '#F0F0F0',
93    },
94    header: {
95      height: 60,
96      backgroundColor: '#81A1C1',
97      justifyContent: 'center',
98      alignItems: 'center',
99      marginBottom:20
100   },
101   headerText: {
102     fontSize: 32,
103     color: '#2E3440'
104   },
105   item: {
106     padding: 5,
107     marginHorizontal:10,
108     marginBottom:5,
109     backgroundColor: '#D8DEE9',
110     borderRadius: 5,
111     flexDirection: 'row',
112     justifyContent: 'space-between',
113     alignItems: 'center'
114   },
115   weatherIcon: {
116     width: 50,
117     height: 50
118   },
119   day: {
120     fontSize: 20,
121     fontWeight: 'bold',
122     marginLeft: 10,
123     color: '#2E3440'
124   },
125   data: {
126     fontSize: 16,
127     marginLeft: 10,
128     color: '#2E3440'
129   },
130 });
```

4 Week exercises

4.1 FlexBox Layout

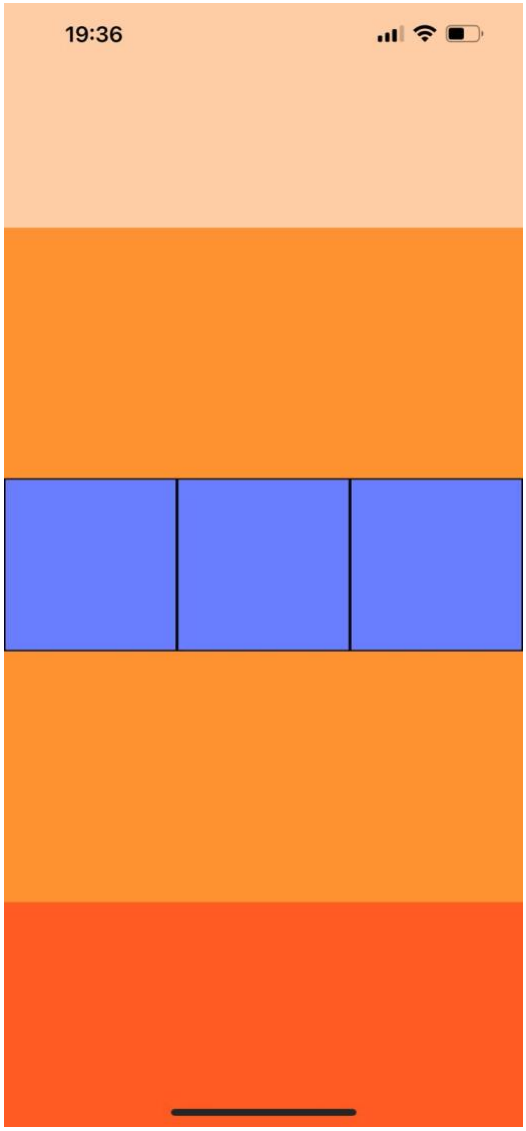
Create a React Native layout using Flexbox to achieve the following design:

Header Section: Should occupy 20% of the screen height at the top.

Content Section: Should take up 60% of the screen height in the middle, containing three equally spaced boxes horizontally.

Footer Section: Should occupy the remaining 20% of the screen height at the bottom.

Use View components to create the layout and apply Flexbox properties for alignment and spacing. Each box in the Content Section should be square (width equals height). Ensure the layout is responsive across different device sizes and customize the background color for clear visibility.


19:36

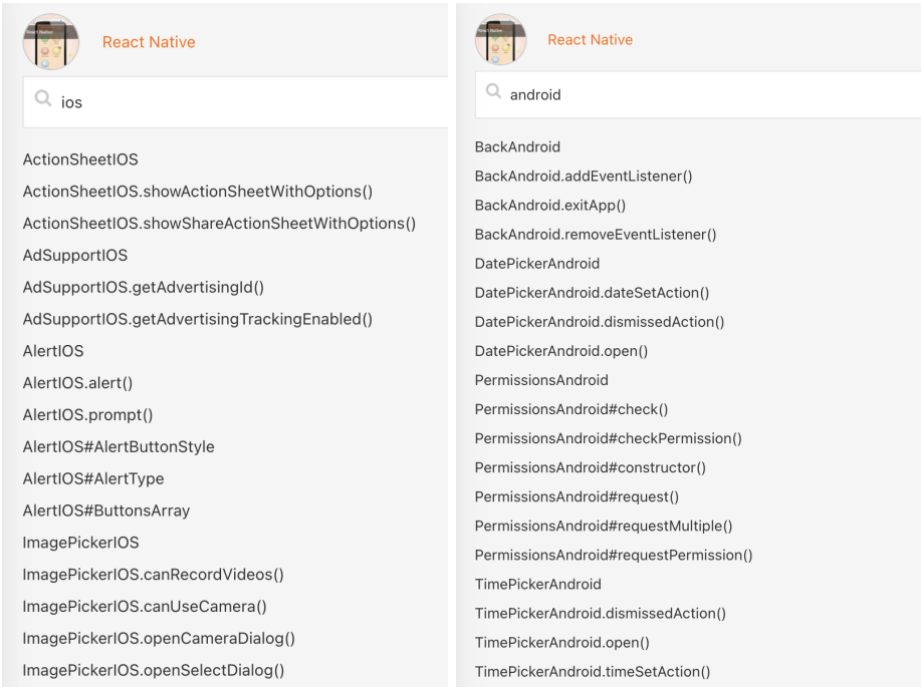
```jsx
import { View, StyleSheet } from 'react-native';

// Define the main App component
export default function App() {
    return (
      <View style={styles.container}>
        {/* Header section */}
        <View style={styles.header}></View>
        {/* Content section */}
        <View style={styles.content}>
          {/* Box 1 */}
          <View style={styles.box}></View>
          {/* Box 2 */}
          <View style={styles.box}></View>
          {/* Box 3 */}
          <View style={styles.box}></View>
        </View>
        {/* Footer section */}
        <View style={styles.footer}></View>
      </View>
    );
}

const styles = StyleSheet.create({
  container: {
    flex: 1, // Take all available space
  },
  header: {
    flex: 2, // Take 20% of the space
    backgroundColor: '#FECDA6',
  },
  content: {
    flex: 6, // Take 60% of the space
    flexDirection: 'row', // Arrange boxes horizontally
    justifyContent: 'center', // Center boxes horizontally
    alignItems: 'center', // Center boxes vertically
    backgroundColor: '#FF9130',
 },
  box: {
    flex: 1, // Each box takes one third of the content section
    aspectRatio: 1, // Make the box square (width equals height)
    margin: 'auto', // Center the box
    borderWidth: 1,
    backgroundColor: '#687EFF',
  },
  footer: {
    flex: 2, // Take 20% of the space
    backgroundColor: '#FF5B22',
  },
});
```

4.2 Platform Specific Components in React Native

Name some components that are available only in:

- ***Android***
- ***iOS***

For **Android**, exclusive components include ToastAndroid, ViewPagerAndroid, PermissionsAndroid, ProgressBarAndroid and TimePickerAndroid.
For **iOS**, exclusive components include ActionSheetIOS, DatePickerIOS, ImagePickerIOS, SegmentedControlIOS and StatusBarIOS.
We can find more on https://www.w3cschool.cn/doc_react_native/react_native-datepickerios.html



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

How can you determine in your code, on which platform your application is currently running? Provide a code example.

```
import { Platform, ToastAndroid, ActionSheetIOS, View, StyleSheet } from 'react-native';
import { Button } from 'react-native-elements';
import { useState } from 'react';

// Define the main App component
export default function App() {
    const [color, setColor] = useState('#687EFF'); // Initialize box colors

    const handlePress = () => {
        if (Platform.OS === 'android') {
            ToastAndroid.show('Button pressed!', ToastAndroid.SHORT);
        } else if (Platform.OS === 'ios') {
            ActionSheetIOS.showActionSheetWithOptions(
                {
                    options: ['Change Box Color to Red', 'Change Box Color to Green', 'Cancel'],
                    cancelButtonIndex: 2,
                },
                (buttonIndex) => {
                    if (buttonIndex === 0) {
                        // Change box colors
                        setColor('red');
                    }
                    else if (buttonIndex === 1) {
                        // Change box colors
                        setColor('green');
                    }
                }
            );
        }
    };

    return (
        <View style={styles.container}>
            {/* Header section */}
            <View style={styles.header}></View>
            {/* Content section */}
            <View style={styles.content}>
                {/* Box 1 */}
                <View style={[styles.box, {backgroundColor: color}]}></View>
                {/* Box 2 */}
                <View style={[styles.box, {backgroundColor: color}]}></View>
                {/* Box 3 */}
                <View style={[styles.box, {backgroundColor: color}]}></View>
            </View>
            {/* Footer section */}
            <View style={styles.footer}>
                {/* Button */}
                <Button
                    buttonStyle={styles.button}
                    title="Press me"
                    onPress={handlePress}
                />
            </View>
        </View>
    );
}
```

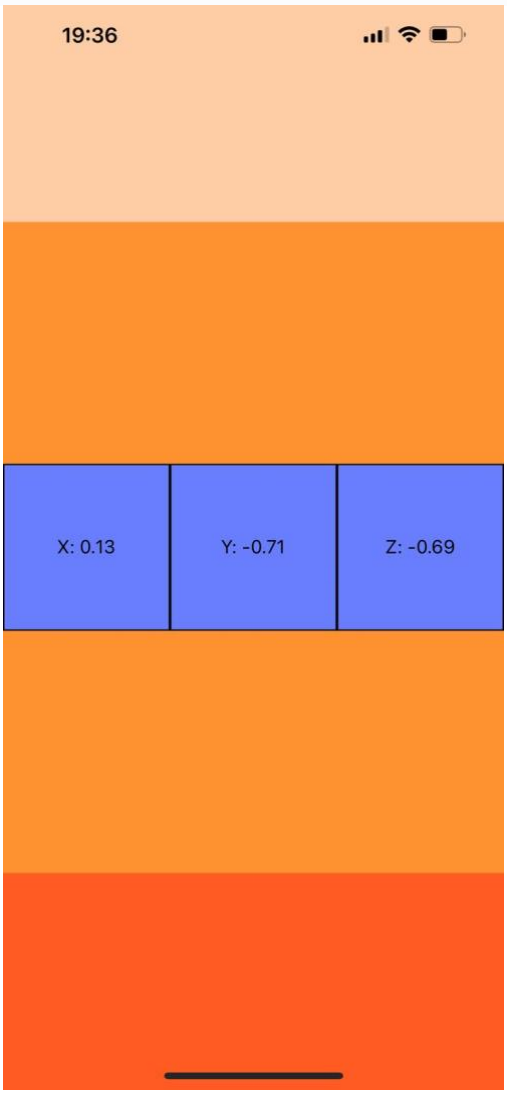
```
57
58  const styles = StyleSheet.create({
59      container: {
60          flex: 1,
61      },
62      header: {
63          flex: 2,
64          backgroundColor: '#FECDA6',
65      },
66      content: {
67          flex: 6,
68          flexDirection: 'row',
69          justifyContent: 'center',
70          alignItems: 'center',
71          backgroundColor: '#FF9130',
72      },
73      box: {
74          flex: 1,
75          aspectRatio: 1,
76          margin: 'auto',
77          borderWidth: 1,
78      },
79      footer: {
80          flex: 2,
81          backgroundColor: '#FF5B22',
82          justifyContent: 'center',
83          alignItems: 'center',
84      },
85      button: {
86      backgroundColor: '#687EFF',
87      borderRadius: 8,
88      marginBottom: 20,
89    }
90  });
```

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, for example, in the horizontal boxes of Exercise 1 (add a Text element inside each of the boxes for x, y, and z).

```
import { useState, useEffect } from 'react';
import { Text, View, StyleSheet } from 'react-native';
import { Accelerometer } from 'expo-sensors'; // Import Accelerometer from expo-sensors

export default function App() {
  const [data, setData] = useState({}); // Initialize state to hold accelerometer data

  useEffect(() => {
    let subscription = Accelerometer.addListener(accelerometerData => {
      setData(accelerometerData); // Set accelerometer data to state
    });

    return () => {
      subscription && subscription.remove(); // Cleanup function to remove the listener
    };
  }, []);

  let { x, y, z } = data; // Destructure x, y, and z values from state
  return (
    <View style={styles.container}>
      {/* Header section */}
      <View style={styles.header}></View>
      {/* Content section */}
      <View style={styles.content}>
        {/* Box 1 */}
        <View style={styles.box}>
          <Text>X: {round(x)}</Text>
        </View>
        {/* Box 2 */}
        <View style={styles.box}>
          <Text>Y: {round(y)}</Text>
        </View>
        {/* Box 3 */}
        <View style={styles.box}>
          <Text>Z: {round(z)}</Text>
        </View>
      </View>
      {/* Footer section */}
      <View style={styles.footer}></View>
    </View>
  );
}

function round(n) {
  if (!n) {
    return 0;
  }

  return Math.floor(n * 100) / 100; // Function to round the values to two decimal places
}

```

```
const styles = StyleSheet.create({
  container: {
    flex: 1,
  },
  header: {
    flex: 2,
    backgroundColor: '#FECDA6',
  },
  content: {
    flex: 6,
    flexDirection: 'row',
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#FF9130',
  },
  box: {
    flex: 1,
    aspectRatio: 1,
    margin: 'auto',
    borderWidth: 1,
    backgroundColor: '#687EFF',
    justifyContent: 'center', // center the text vertically
    alignItems: 'center', // center the text horizontally
  },
  footer: {
    flex: 2,
    backgroundColor: '#FF5B22',
  },
});
```



5 Week exercises

5.1 Utilizing Device APIs – Part II

5.1.1 react-native-permissions

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?

The react-native-permissions library is a third-party package that simplifies the process of requesting permissions on both iOS and Android. It provides a consistent API for handling permissions, and it supports various permission types.

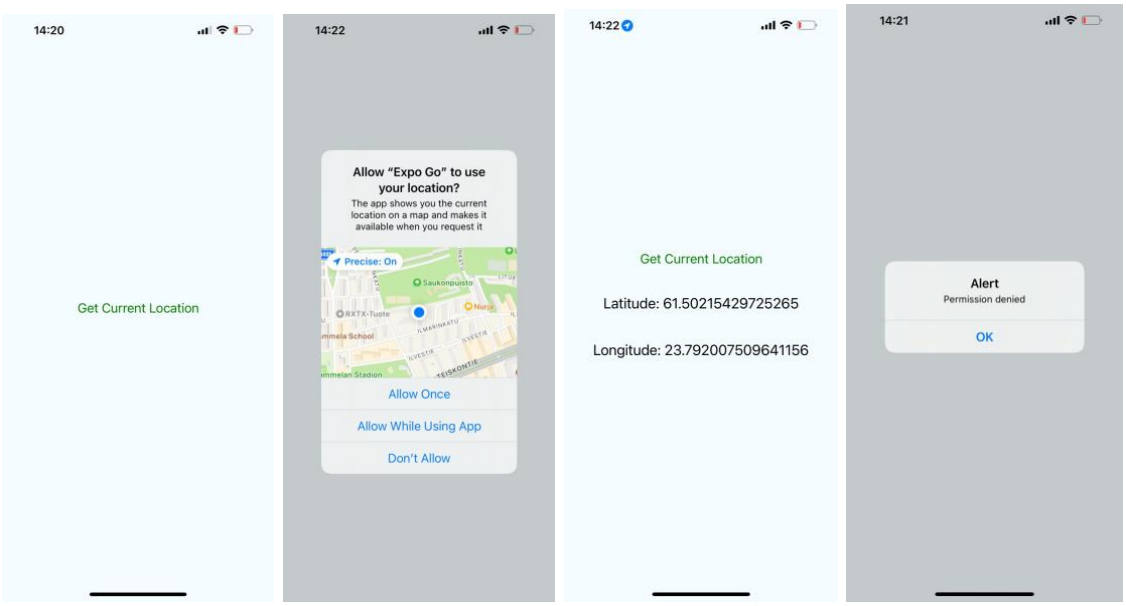
In Expo, handling permissions is simplified through the use of the expo-permissions module. Expo provides a set of APIs that abstract the underlying native implementations for requesting various permissions on both iOS and Android.

5.1.2 display current location

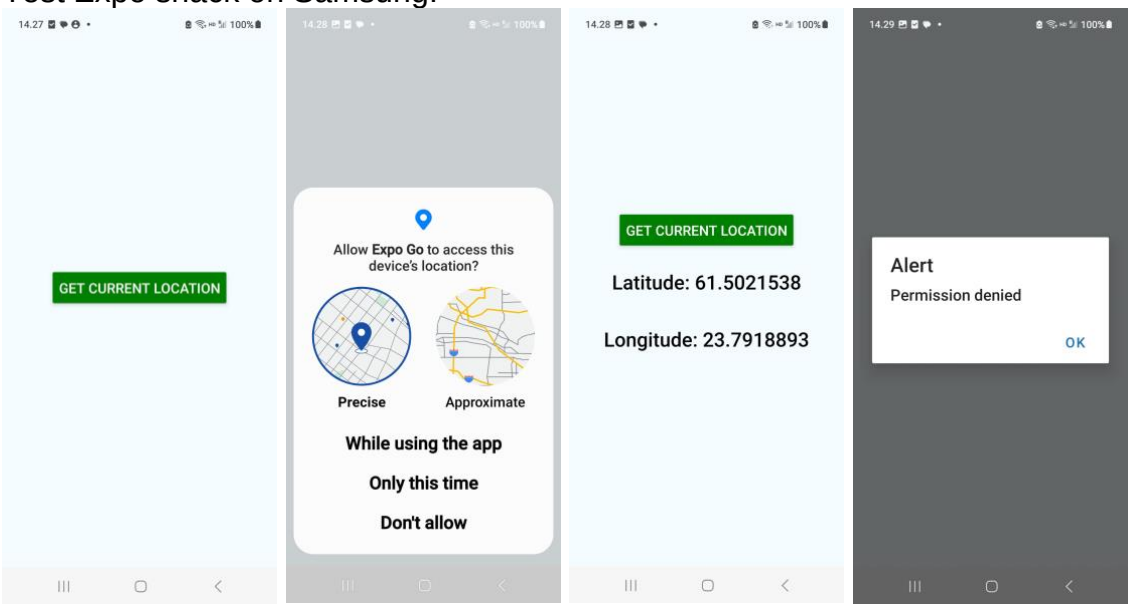
To your application, add a feature, which shows the current latitude and longitude on the display. The application should ask proper permissions from the user before accessing the Location API.

```
// The application below initializes a state variable to hold the location,
// then defines a function that requests location permissions.
// If granted, it retrieves the current location and sets it in the state.
// The latitude and longitude are then displayed in a Text component.
// A Button component triggers the getLocation function.

import { useState } from 'react';
import { Button, Text, View, StyleSheet } from 'react-native';
import * as Permissions from 'expo-permissions';
import * as Location from 'expo-location';

export default function App() {
  const [location, setLocation] = useState(null);
  const [loading, setLoading] = useState(false);

  const getLocation = async () => {
    setLoading(true);
    const { status } = await Permissions.askAsync(Permissions.LOCATION_FOREGROUND);
    if (status === 'granted') {
      const location = await Location.getCurrentPositionAsync({});
      setLocation(location);
    } else {
      // Permission denied
      alert("Permission denied");
    }
    setLoading(false);
  };

  return (
    <View style={styles.container}>
      <Button title="Get Current Location" onPress={getLocation} color="green"/>
      {loading ? (
        <Text style={styles.text}>Loading...</Text>
      ) : (
        location && (
          <>
            <Text style={styles.text}>Latitude: {location.coords.latitude}</Text>
            <Text style={styles.text}>Longitude: {location.coords.longitude}</Text>
          </>
        )
      )}
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    backgroundColor: '#F5FCFF',
  },
  text: {
    fontSize: 20,
    textAlign: 'center',
    margin: 10,
    marginTop:30,
  },
});
```

Test Expo snack on iPhone:



Test Expo snack on Samsung:



5.1.3 permissions for Android or iOS

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)?

For Android, React Native offers the PermissionsAndroid API to request permissions. It provides a set of methods to request various Android permissions.

On iOS, permissions are usually specified in the app's Info.plist file. React Native doesn't provide a specific API for handling permissions on iOS, as it relies on the native iOS permissions system.

5.2 Using the Linking Module

5.2.1 written answers to questions

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.

Adding a feature in React Native to open Maps at the current location requires the use of the Linking module. First, import the Linking module from React Native. Then, a function should be created that uses Linking.openURL with a URL scheme corresponding to the maps app.

To implement routing to a specific destination, append the destination to the geo URL scheme.

To add a button that dials a phone number specified in a TextInput element, a function should be created that reads the input value (the phone number), and then calls Linking.openURL with a tel URL scheme.

Which permissions would your app need in order to perform a phone call?

In a React Native Expo project, when the Linking.openURL method is used to initiate a phone call, the CALL_PHONE permission is not explicitly requested as would be done in native Android development. The Linking.openURL method essentially opens the default phone dialer app on the device, and the call can be initiated from there.

In iOS, making a phone call using the Linking.openURL method generally doesn't require specific permission requests. The tel: scheme used in the URL is a standard way to open the Phone app, and iOS allows this without requiring special permission. However, it does provide a user confirmation step to ensure that users are aware and in control when the app tries to initiate a phone call.

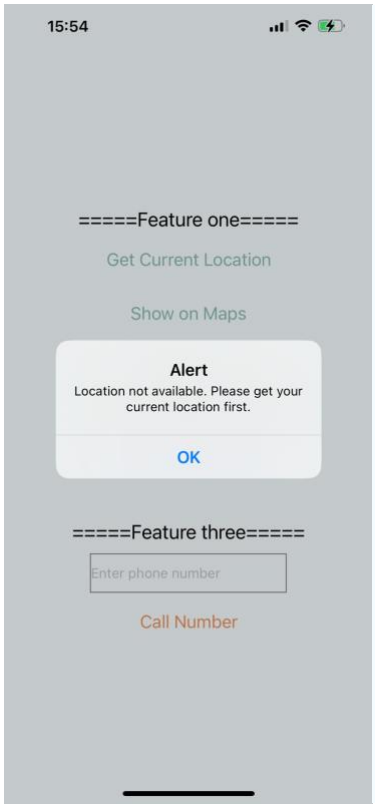
5.2.2 Expo snack code for Linking module usage

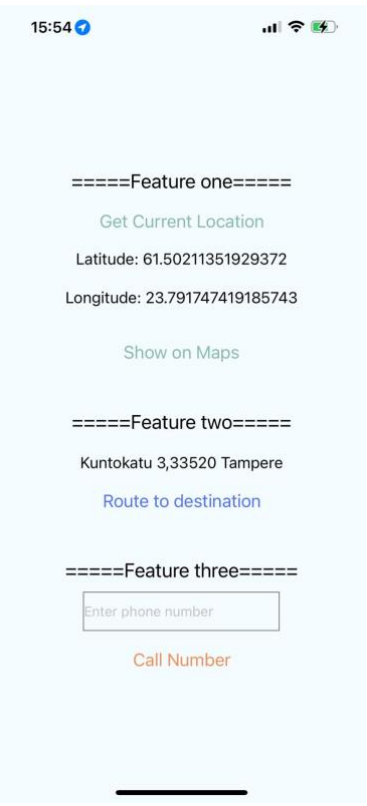
```
// The code below includes functions to open maps at current location, route to a specific destination,
// and dial a phone number provided in a TextInput element.
// Additionally, a regular expression validates if the input phone number is in a valid international format,
// with or without the '+' sign.
// If validated, the app dials the number; otherwise, an alert about an invalid phone number is displayed.

import { useState } from 'react';
import { Button, Text, TextInput, View, StyleSheet, Linking, Platform } from 'react-native';
import * as Permissions from 'expo-permissions';
import * as Location from 'expo-location';

const App = () => {
  const [location, setLocation] = useState(null);
  const [loading, setLoading] = useState(false);
  const [phoneNumber, setPhoneNumber] = useState('');

  const getLocation = async () => {
    setLoading(true);
    const { status } = await Permissions.askAsync(Permissions.LOCATION_FOREGROUND);
    if (status === 'granted') {
      const location = await Location.getCurrentPositionAsync({});
      setLocation(location);
    } else {
      alert("Permission denied");
    }
    setLoading(false);
  };

  const openMaps = () => {
    if(!location) {
      alert('Location not available. Please get your current location first.');
      return;
    }
    const url = Platform.select({
      android: `geo:${location.coords.latitude}, ${location.coords.longitude}`,
      ios: `maps://?q=${location.coords.latitude}, ${location.coords.longitude}`,
    });
    Linking.openURL(url);
  };

```

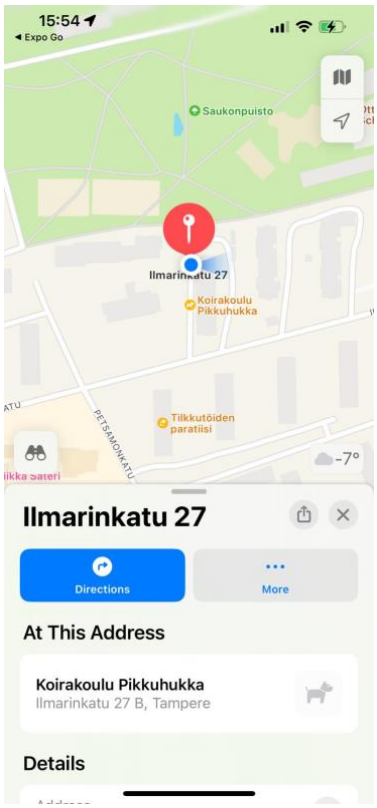
```
  const routeToDestination = () => {
    const url = Platform.select({
      android: "geo:0,0?q=Kuntokatu+3,+33520+Tampere",
      ios: "maps://0,0?q=Kuntokatu+3,+33520+Tampere",
    });
    Linking.openURL(url);
  };

  const callNumber = () => {
    const phoneNumberRegex = /^[\+]?[(]?[0-9]{3}[)]?[-\s\.]?[0-9]{3}[-\s\.]?[0-9]{4,6}$/;
    if (phoneNumberRegex.test(phoneNumber)) {
      const url = `tel:${phoneNumber}`;
      Linking.openURL(url);
    } else {
      alert('Please enter a valid phone number');
    }
  };

  return (
    <View style={styles.container}>
      <Text style={styles.text}>=====Feature one=====</Text>
      <Button title="Get Current Location" onPress={getLocation} color="#89B9AD"/>
      {loading ? (
        <Text style={styles.text}>Loading...</Text>
      ) : (
        location && (
          <>
            <Text style={styles.textsmall}>Latitude: {location.coords.latitude}</Text>
            <Text style={styles.textsmall}>Longitude: {location.coords.longitude}</Text>
          </>
        )
      )}
      <Text>      </Text>
      <Button title="Show on Maps" onPress={openMaps} color="#89B9AD" />
      <Text style={styles.text}>=====Feature two=====</Text>
      <Text style={styles.textsmall}>Kuntokatu 3,33520 Tampere</Text>
      <Button title="Route to destination" onPress={routeToDestination} color="#5272F2" />
      <Text style={styles.text}>=====Feature three=====</Text>
      <TextInput
        style={styles.input}
        value={phoneNumber}
        onChangeText={setPhoneNumber}
        placeholder="Enter phone number"
      />
      <Button title="Call Number" onPress={callNumber} color="#EC8F5E" />
    </View>
  );
};

```

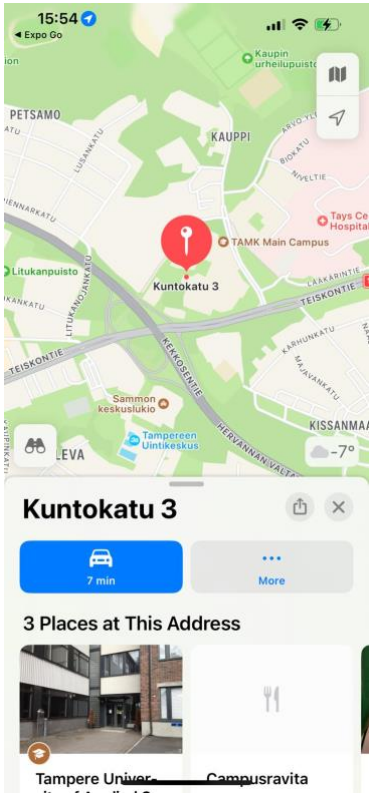
```
90   const styles = StyleSheet.create({
91     container: {
92       flex: 1,
93       justifyContent: 'center',
94       alignItems: 'center',
95       backgroundColor: '#F5FCFF',
96     },
97     text: {
98       fontSize: 20,
99       textAlign: 'center',
100      margin: 10,
101      marginTop:40,
102    },
103    textsmall: {
104      fontSize: 16,
105      textAlign: 'center',
106      margin: 10,
107    },
108    input: {
109      height: 40,
110      width:200,
111      borderColor: 'gray',
112      borderWidth: 1,
113      marginBottom: 10,
114    },
115  });
116
117  export default App;
```

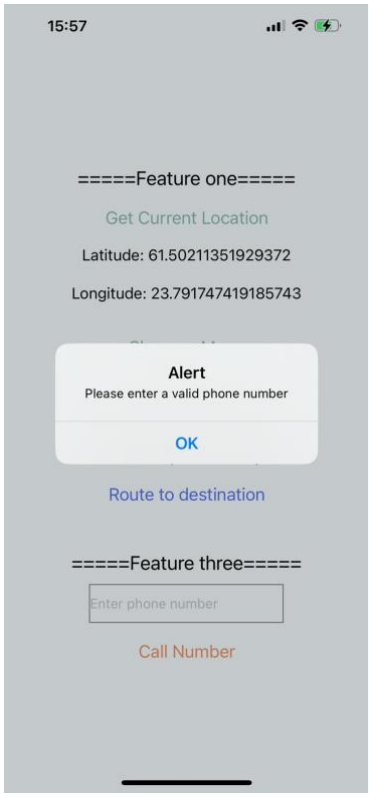
5.2.3 Test Expo snack on iPhone

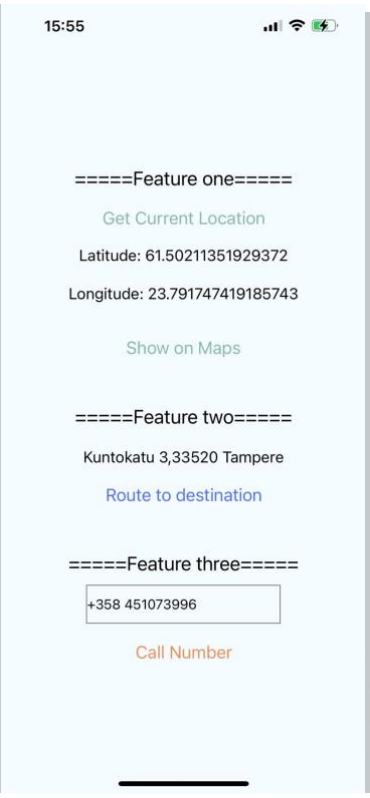


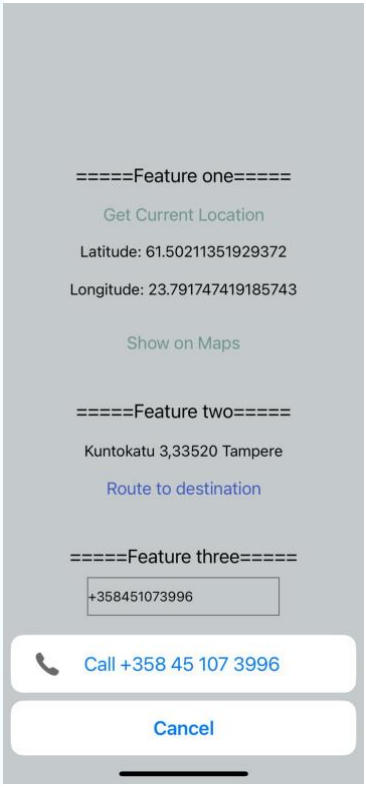




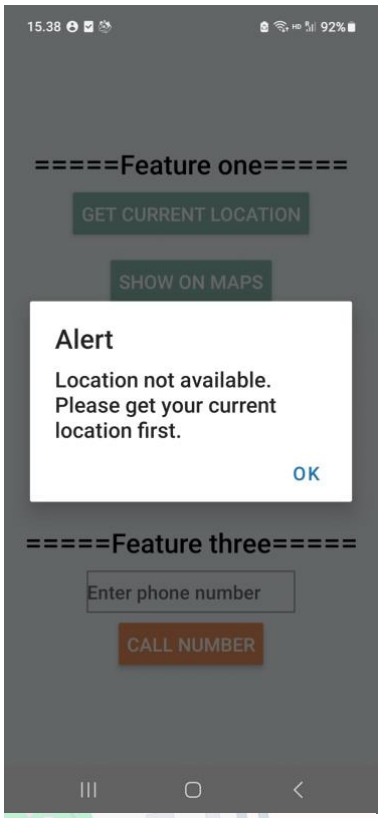

15:54
Expo Go
PETSAMO
KAUPPI
TAMK Main Campus
Kuntokatu 3
Kuntokatu 3
7 min
More
3 Places at This Address
Campusravita

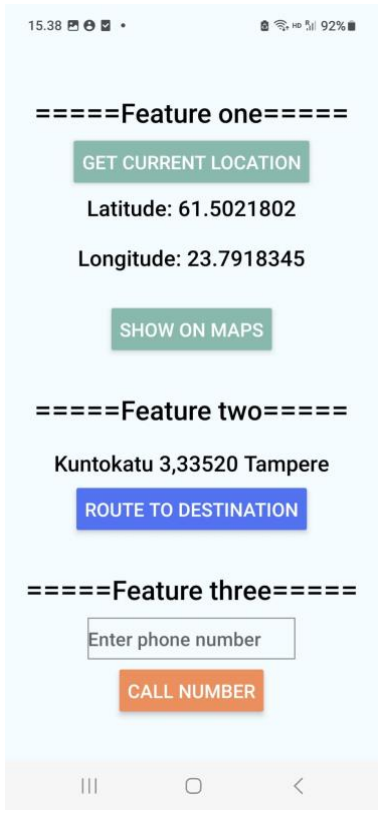

15:57
=====Feature one=====
Get Current Location
Latitude: 61.50211351929372
Longitude: 23.791747419185743
Alert
Please enter a valid phone number
OK
Route to destination
=====Feature three=====
Enter phone number
Call Number

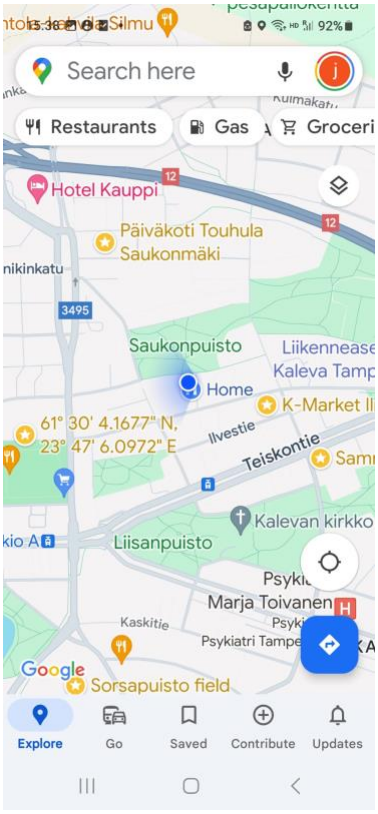

15:55
=====Feature one=====
Get Current Location
Latitude: 61.50211351929372
Longitude: 23.791747419185743
Show on Maps
=====Feature two=====
Kuntokatu 3,33520 Tampere
Route to destination
=====Feature three=====
+358 451073996
Call Number

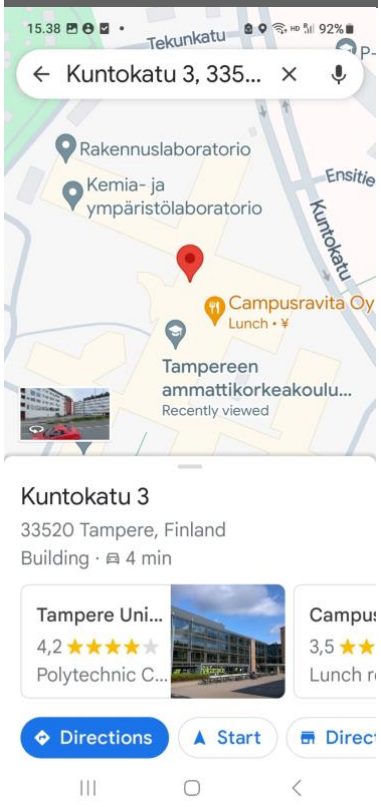

=====Feature one=====
Get Current Location
Latitude: 61.50211351929372
Longitude: 23.791747419185743
Show on Maps
=====Feature two=====
Kuntokatu 3,33520 Tampere
Route to destination
=====Feature three=====
+358451073996
Call +358 45 107 3996
Cancel

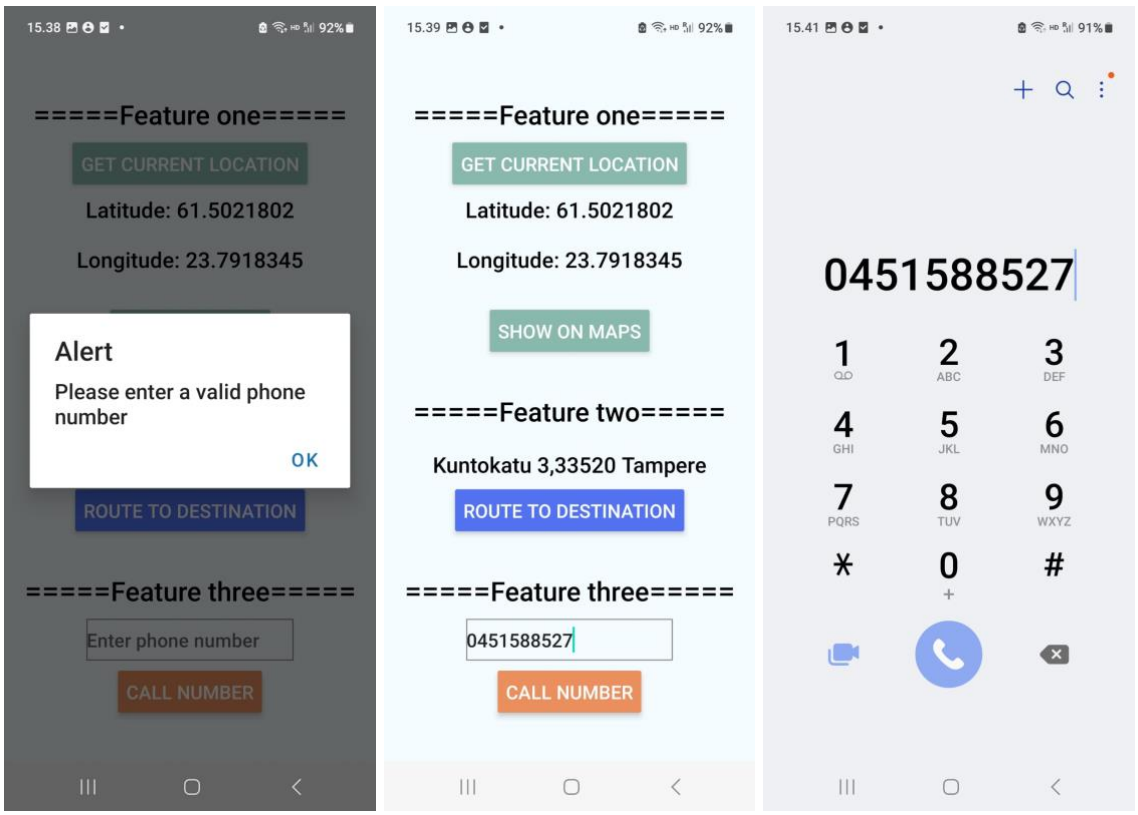
5.2.4 Test Expo snack on Samsung











5.3 Exploring Other Device APIs

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

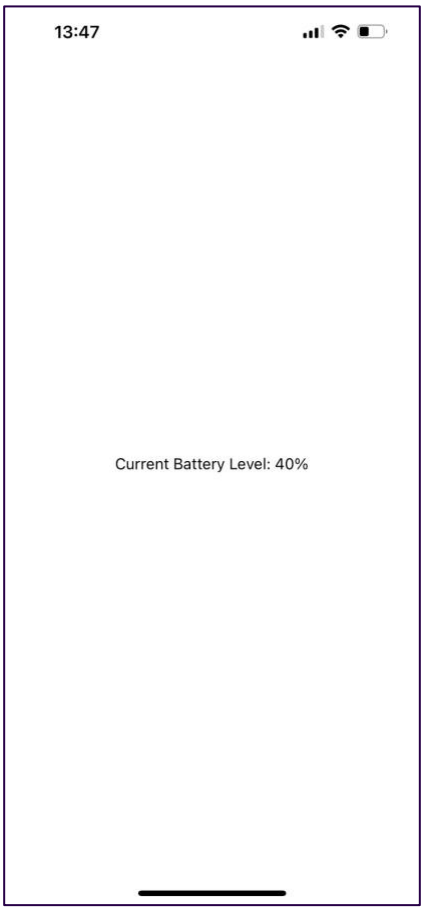
- ***Discover and list the Bluetooth devices in range***
- ***Take a picture with Camera and display it on your app***
- ***Read and monitor the current battery status and display the current battery percentage on your app display***

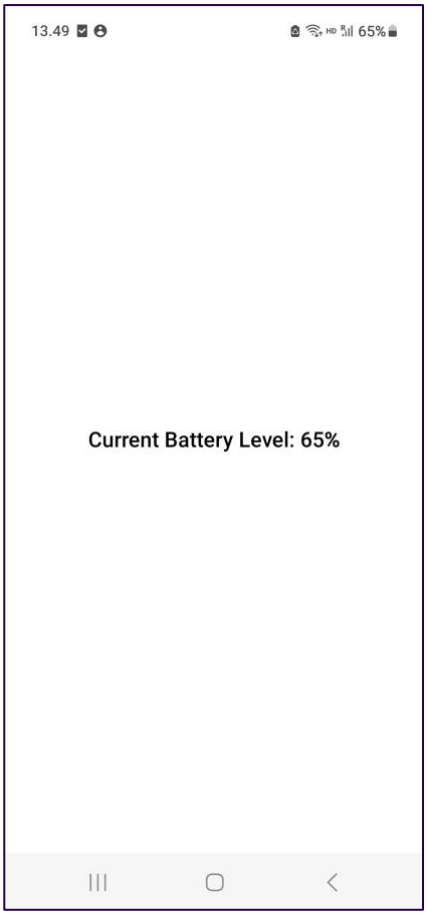
5.3.1 Read and monitor the current battery status

The 'expo-battery' library can be used. The battery level can be obtained by calling the 'getBatteryLevelAsync()' function. To monitor the battery status, an event listener can be added using the 'addBatteryLevelListener()' function which triggers every time the battery level changes. The current battery level can be displayed using a 'Text' component.

```
import { useEffect, useState, useCallback } from 'react';
import * as Battery from 'expo-battery';
import { StyleSheet, Text, View } from 'react-native';

const App = () => {
  const [batteryLevel, setBatteryLevel] = useState(null); // Initialize batteryLevel state
  const [subscription, setSubscription] = useState(null); // Initialize subscription state

  const _subscribe = async () => { // Define a subscribe function
    const batteryLevel = await Battery.getBatteryLevelAsync(); // Get the battery level
    setBatteryLevel(batteryLevel); // Set the battery level to state

    // Subscribe to battery level changes
    setSubscription(
      Battery.addBatteryLevelListener(({ batteryLevel }) => {
        setBatteryLevel(batteryLevel); // Update the state every time the battery level changes
        console.log('batteryLevel changed!', batteryLevel); // Log battery level changes
      })
    );
  };

  const _unsubscribe = useCallback(() => { // Define an unsubscribe function
    subscription && subscription.remove(); // Remove the subscription if it exists
    setSubscription(null); // Reset the subscription state
  }, [subscription]);

  useEffect(() => { // React useEffect hook
    _subscribe(); // Subscribe when the component mounts
    return () => _unsubscribe(); // Unsubscribe when the component unmounts
  }, [_unsubscribe]); // Run useEffect whenever _unsubscribe updates

  // Render the battery level or 'Loading...' if it's not yet loaded
  return (
    <View style={styles.container}>
      <Text>Current Battery Level: {batteryLevel !== null ?
        Math.round(batteryLevel * 100) + '%' : 'Loading...'}</Text>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    marginTop: 15,
    alignItems: 'center',
    justifyContent: 'center',
  },
});

export default App; // Export the BatteryScreen component
```

Test on Expo Go iPhone and Samsung





5.3.2 Take a photo with the camera and display it in the app

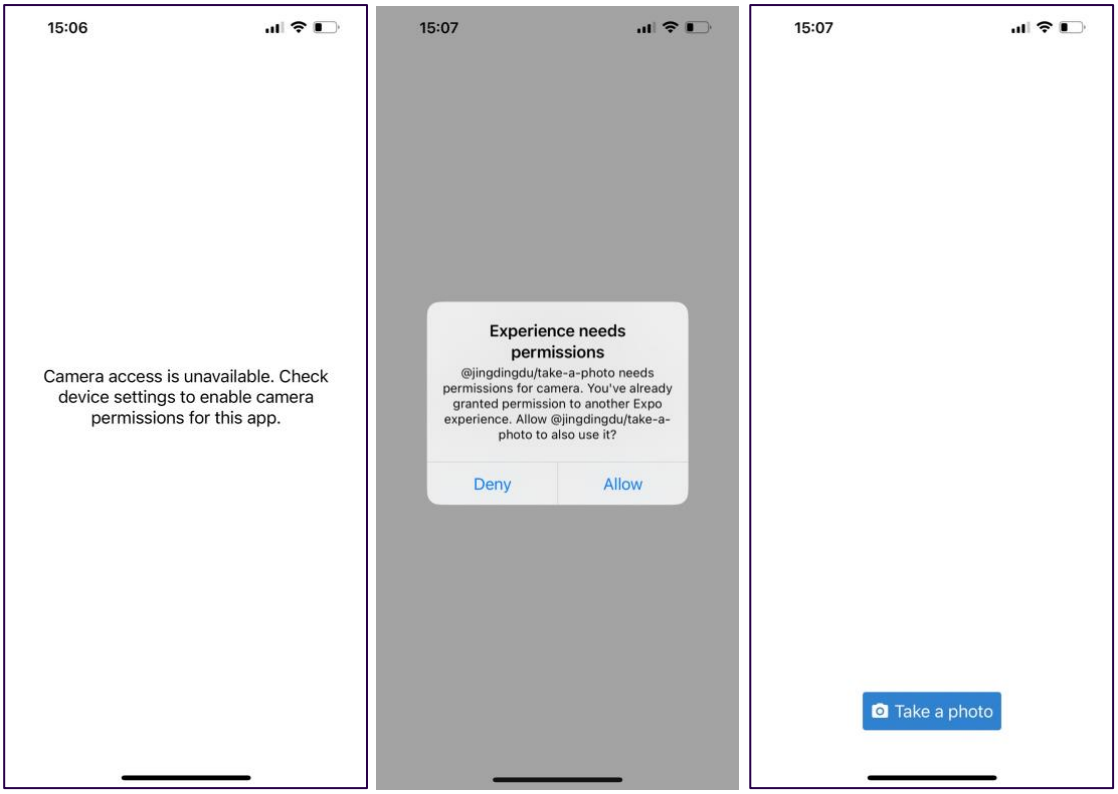
The 'expo-camera' library can be used. First, camera permissions need to be requested. If granted, the 'takePictureAsync()' function can capture a photo. The photo can be displayed using an 'Image' component and passing the URI of the photo to the 'source' prop.

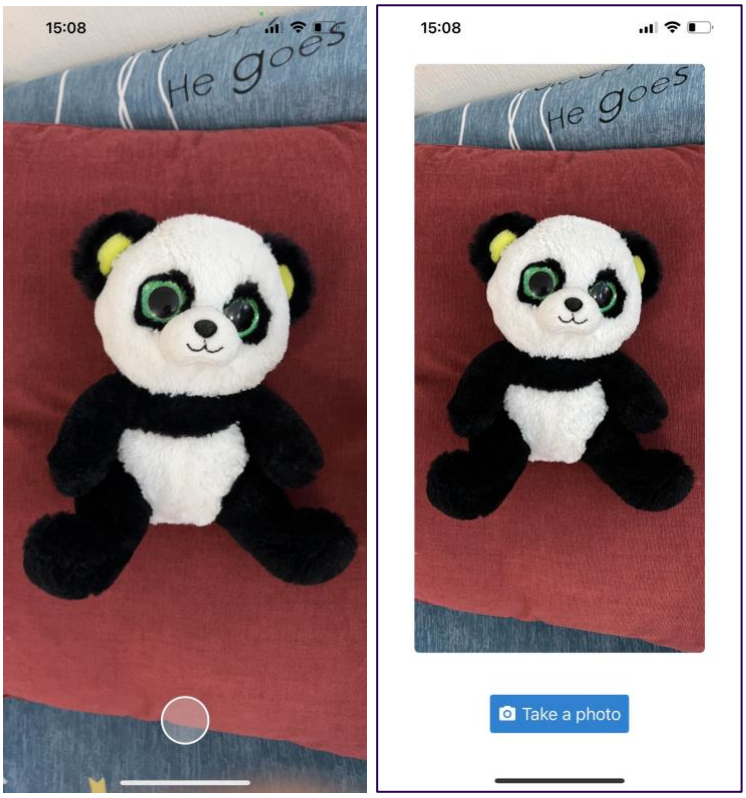
```
import { useState, useEffect } from 'react';
import { View, Text, TouchableOpacity, Image, StyleSheet } from 'react-native';
import { Button, Icon } from 'react-native-elements';
import { Camera } from 'expo-camera';

export default App = () => {
  const [hasPermission, setHasPermission] = useState(null);
  const [cameraRef, setCameraRef] = useState(null);
  const [capturedPhoto, setCapturedPhoto] = useState(null);
  const [startCamera, setStartCamera] = useState(false);

  useEffect(() => {
    const requestCameraPermissions = async () => {
      const { status } = await Camera.requestCameraPermissionsAsync();
      setHasPermission(status === 'granted');
    };

    requestCameraPermissions();
  }, []);

```

```
  const handleStartCamera = () => {
    setStartCamera(true);
  };

  const takePicture = async () => {
    if (cameraRef) {
      const photo = await cameraRef.takePictureAsync();
      setCapturedPhoto(photo.uri);
      setStartCamera(false);
    }
  };

  if (hasPermission === null) {
    return <View />;
  }
  if (hasPermission === false) {
    return (
      <View style={styles.container}>
        <Text style={styles.noAccessText}>Camera access is unavailable.
          Check device settings to enable camera permissions for this app.</Text>
      </View>
    );
  }

  return (
    <View style={{ flex: 1 }}>
      {capturedPhoto && !startCamera &&
        <Image source={{ uri: capturedPhoto }} style={styles.image} resizeMode="cover" />}
      {startCamera ? (
        <Camera style={styles.camera} type={Camera.Constants.Type.back}
          ref={(ref) => setCameraRef(ref)}  ratio={"16:9"}>
          <View style={styles.view}>
            <TouchableOpacity
              style={styles.touchableOpacity}
              onPress={takePicture}
            >
              <View style={styles.circle} />
            </TouchableOpacity>
          </View>
        </Camera>
      ) : (
        <View style={styles.buttonContainer}>
          <Button
            title=" Take a photo"
            icon={
              <Icon
                name="photo-camera"
                size={20}
                color="white"
              />
            }
            buttonStyle={{backgroundColor: "#3081D0"}}
            onPress={handleStartCamera}
          />
        </View>
      )}
    </View>
  );
};

```

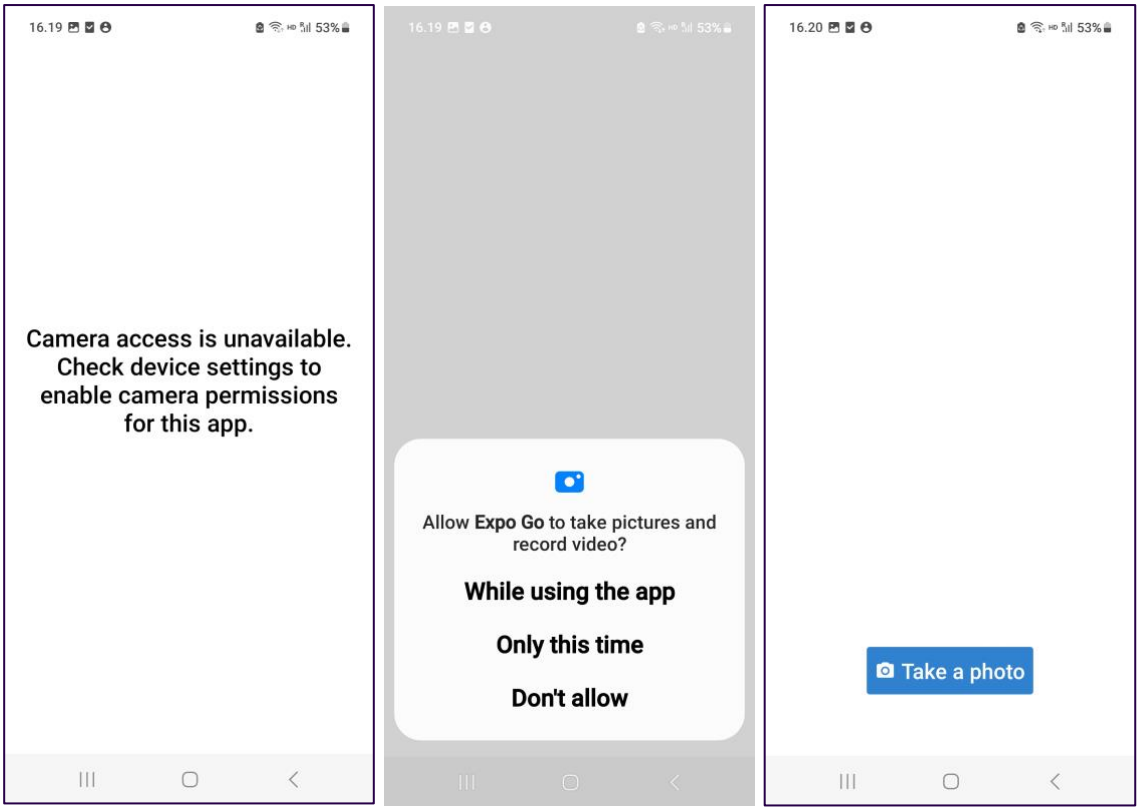
```
// Define styles
const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: "center",
    alignItems: "center",
  },
  image: {
    width: '80%',
    height: '75%',
    borderRadius: 5,
    alignSelf: 'center',
    marginTop: 60,
  },
  camera: {
    flex: 1
  },
  view: {
    flex: 1,
    backgroundColor: 'transparent',
    justifyContent: 'flex-end'
  },
  touchableOpacity: {
    alignSelf: 'center',
    marginBottom: 20
  },
  circle: {
    borderWidth: 2,
    borderRadius: 50,
    borderColor: 'white',
    height: 50,
    width:50,
    backgroundColor: 'rgba(255, 255, 255, 0.3)', // semi-transparent white
    display: 'flex',
    marginBottom: 30,
  },
  buttonContainer: {
    justifyContent: 'flex-end',
    alignItems: 'center',
    alignSelf: 'center',
    flex: 1,
    marginBottom: 60,
    width: "60%",
  },
  noAccessText: {
    fontSize: 18,
    textAlign:'center',
    margin:20,
  }
});
```

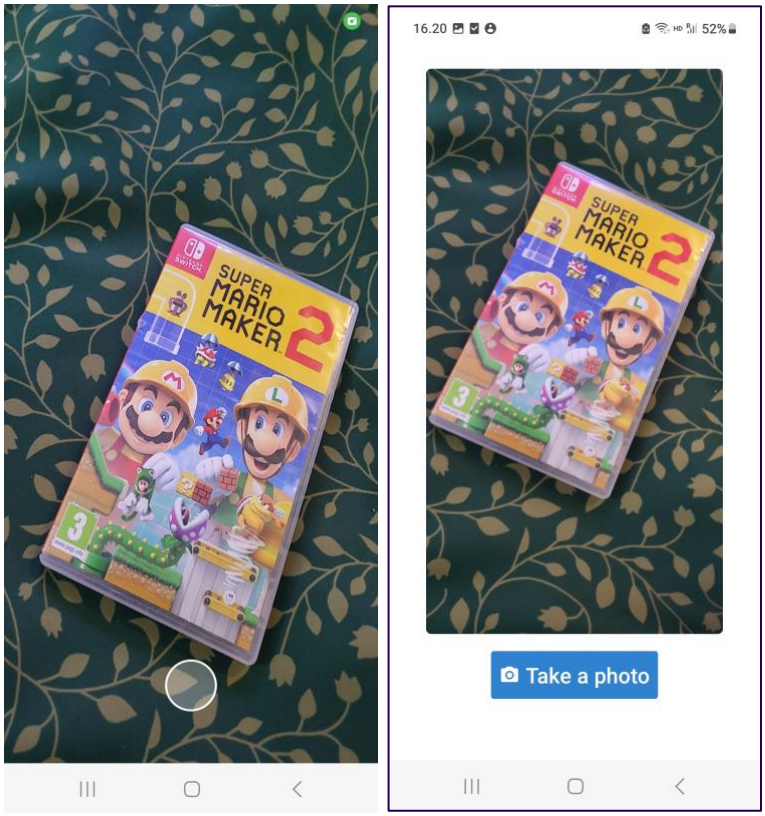
Test on Expo Go iPhone





Test on Expo Go Samsung





5.3.3 Discover and list the Bluetooth devices within range

Currently, Expo Go does not support Bluetooth functionality directly. To use Bluetooth features, the app needs to be ejected from Expo or a standalone app

needs to be built with a library that supports Bluetooth, such as 'react-native-ble-plx' or 'react-native-ble-manager'. These libraries provide methods to start scanning for devices, connect to a device, and read/write characteristics.

5.4 Publishing Apps

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

5.4.1 To publish on the Apple App Store

1. Build a standalone IPA for the project by running expo build:ios in the project directory.
2. Choose whether to have Expo handle the process or to upload own credentials.
3. After running this command, Expo provides a URL for downloading the IPA once it's ready.
4. Visit the Apple Developer website and access the App Store Connect section.
5. Create a new iOS app by filling out the required details.
6. Navigate to the "Build" section in the App Store Connect record of the app and select the "Select a build before submitting the app" option.
7. Upload the IPA file using Xcode or Application Loader.
8. Complete the remaining steps and submit the app for review. Note that Apple's app review process can take a few days.

5.4.2 To publish on the Google Play Store

1. Build an APK or AAB for the project using the expo build: android command in the project directory.
2. Choose whether to let Expo handle the process or upload a personal keystore.
3. After running this command, a URL is provided where the signed APK or AAB can be downloaded.
4. Visit Google Play Console and create a new application.
5. Complete the required details about the application and upload the APK or AAB file that was downloaded earlier.
6. After filling out all necessary fields and agreeing to the terms of service, the app can be published. Note that it might take a few hours to a couple of days before the app is live on the Play Store.

6 Final project

6.1 Project description

This project is a user-friendly React Native weather app using the OpenWeatherMap API. It consists of three screens: Home, Forecast, and Settings. Navigation between these screens is managed by the BottomTabNavigator.

- The Home screen fetches and displays current weather data like temperature, wind speed, and overall weather conditions for a selected location. It also allows users to manually input a location to get weather data or auto-detect their current location using device location API.
- The Forecast screen offers an hourly forecast for the location selected on the Home screen. It fetches data from the API and displays the temperature, wind speed, and weather icon for each hour of the day in a neat format.
- The Settings screen provides a customization feature that allows users to select their preferred theme color. This chosen theme will then be applied throughout the app, altering the color scheme and overall appearance according to the user's preference.

The app is designed for both Android and iOS.

For handling invalid user input or when the app doesn't have location permissions, the application uses platform-specific components. For iOS, it uses the Alert component from 'react-native' to display an alert dialog with a message, and for Android, it uses ToastAndroid from 'react-native' to display a short notification message.

On the Settings screen, the app employs different components based on the operating system. For Android, it uses the Picker component for selections, whereas for iOS, it utilizes the ActionSheetIOS component. This approach caters to the unique interfaces and user experiences of the two platforms.

The application could benefit from implementing a responsive design. This ensures optimal usability and aesthetics across different device sizes and orientations. Additionally, providing multi-language support would make the app more accessible to a global audience.

6.2 Project structure

```
.
├── App.js
├── README.md
├── app.json
├── assets
│   ├── adaptive-icon.png
│   ├── favicon.png
│   ├── icon.png
│   ├── snack-icon.png
│   └── splash.png
├── babel.config.js
├── components
│   ├── Context
│   │   ├── LocationContext.js
│   │   └── ThemeContext.js
│   ├── Current
│   │   ├── CurrentLocation.js
│   │   ├── CurrentWeatherHeader.js
│   │   ├── CurrentWeatherInfo.js
│   │   └── CurrentWeatherLocationUpdate.js
│   ├── Forecast.js
│   ├── Home.js
│   └── Settings.js
└── package.json
```

6.3 Package.json

```
{
  "dependencies": {
    "expo-location": "~15.1.1",
    "expo-permissions": "~14.1.1",
    "@expo/vector-icons": "^13.0.0",
    "react-native-elements": "*",
    "@react-navigation/material-bottom-tabs": "6.2.1",
    "@react-native-picker/picker": "2.4.8",
    "@react-navigation/native": "6.0.10",
    "react-native-safe-area-context": "4.5.0",
    "react-native-paper": "^4.7.2"
  }
}
```

6.4 Expo Snack code

6.4.1 App.js

```
import React, { useState } from "react";
import { NavigationContainer } from "@react-navigation/native";
import { createMaterialBottomTabNavigator } from "@react-navigation/material-bottom-tabs";

import HomeScreen from "./components/Home";
import ForecastScreen from "./components/Forecast";
import SettingsScreen from "./components/Settings";

import { LocationContext } from "./components/Context/LocationContext";
import { ThemeContext } from "./components/Context/ThemeContext";

const Tab = createMaterialBottomTabNavigator();

export default function App() {
  const [theme, setTheme] = useState("#5B9A8B");
  const [location, setLocation] = React.useState("Helsinki");

  return (
    <ThemeContext.Provider value={{ theme, setTheme }}>
      <LocationContext.Provider value={{ location, setLocation }}>
        <NavigationContainer>
          <Tab.Navigator
            initialRouteName="Home"
            activeColor="white"
            inactiveColor="#445069"
            barStyle={{ backgroundColor: theme }} // Apply theme styles to the bar
            shifting={true} // Enable shifting behavior for icons on Android
            labeled={true} // Show labels below icons
          >
            <Tab.Screen
              name="Home"
              component={HomeScreen}
              options={{ tabBarLabel: "Home", tabBarIcon: "home" }}
            />
            <Tab.Screen
              name="Forecast"
              component={ForecastScreen}
              options={{ tabBarLabel: "Forecast", tabBarIcon: "calendar" }}
            />
            <Tab.Screen
              name="Settings"
              component={SettingsScreen}
              options={{ tabBarLabel: "Settings", tabBarIcon: "cog" }}
            />
          </Tab.Navigator>
        </NavigationContainer>
      </LocationContext.Provider>
    </ThemeContext.Provider>
  );
}
```

6.4.2 components/Home.js

```
import React, { useState, useEffect } from "react";
import {
  StyleSheet,
  SafeAreaView,
  Platform,
  Alert,
  ToastAndroid,
} from "react-native";

import * as Permissions from "expo-permissions";
import * as Location from "expo-location";

import Header from "./Current/CurrentWeatherHeader";
import WeatherInfo from "./Current/CurrentWeatherInfo";
import LocationUpdate from "./Current/CurrentWeatherLocationUpdate";
import CurrentLocation from "./Current/CurrentLocation";
import { LocationContext } from "./Context/LocationContext";

const Home = () => {
  // State variables declaration
  const { location, setLocation } = React.useContext(LocationContext);
  const [weather, setWeather] = useState("");
  const [icon, setIcon] = useState("");
  const [temperature, setTemperature] = useState("");
  const [windSpeed, setWindSpeed] = useState("");
  const [inputValue, setInputValue] = useState("");
  const [isInputFocused, setIsInputFocused] = useState(false);

  // API key for OpenWeatherMap
  const API_KEY = "395dc02446c77c0ac922cb465d9a395b";

  // Function to fetch weather data from OpenWeatherMap API
  const getWeatherData = async (location) => {
    try {
      const response = await fetch(
        `https://api.openweathermap.org/data/2.5/weather?q=${location}&units=metric&appid=${API_KEY}`
      );

      if (!response.ok) {
        throw new Error(`HTTP error! Status: ${response.status}`);
      }

      const data = await response.json();
      const weather = data.weather[0].description;
      const icon = data.weather[0].icon;
      const temperature = Math.round(data.main.temp);
      const windSpeed = Number(data.wind.speed.toFixed(1));

      // Return fetched data
      return { icon, weather, temperature, windSpeed };
    } catch (error) {
      console.error("Error fetching weather data:", error);
      throw error;
    }
  };

```

```
  // Fetching data whenever 'location' changes
  useEffect(() => {
    const fetchWeatherData = async () => {
      try {
        const weatherData = await getWeatherData(location);
        const { weather, icon, temperature, windSpeed } = weatherData;

        // Set state variables with fetched data
        setWeather(weather);
        setIcon(icon);
        setTemperature(temperature);
        setWindSpeed(windSpeed);
      } catch (error) {
        console.error("Error fetching weather data:", error);
      }
    };
    fetchWeatherData();
  }, [location]);

  // Function to update weather data based on user's input
  const refreshWeather = async () => {
    if (inputValue.trim()) {
      try {
        // Check if the entered location is valid by attempting to fetch weather data
        const weatherData = await getWeatherData(inputValue.trim());

        // If weather data is successfully fetched, update the location
        setLocation(inputValue.trim());
      } catch (error) {
        // If there's an error fetching weather data, show an alert or toast based on platform
        const errorMessage = "Please enter a valid location.";

        if (Platform.OS === "ios") {
          Alert.alert("Invalid Input", errorMessage);
        } else if (Platform.OS === "android") {
          ToastAndroid.show(errorMessage, ToastAndroid.SHORT);
        }
      }
    } else {
      const errorMessage = "Please enter a valid location.";

      if (Platform.OS === "ios") {
        Alert.alert("Invalid Input", errorMessage);
      } else if (Platform.OS === "android") {
        ToastAndroid.show(errorMessage, ToastAndroid.SHORT);
      }
    }
  };

```

```
  const getCurrentLocation = async () => {
    const { status } = await Permissions.askAsync(
      Permissions.LOCATION_FOREGROUND
    );
    if (status === "granted") {
      const geo_location = await Location.getCurrentPositionAsync({});
      const geocode = await Location.reverseGeocodeAsync(geo_location.coords);
      setLocation(geocode[0].city);
      setInputValue(""); // clear the input value
    } else {
      // Permission denied
      if (Platform.OS === "ios") {
        Alert.alert("Permission denied");
      } else if (Platform.OS === "android") {
        ToastAndroid.show("Permission denied", ToastAndroid.SHORT);
      }
    }
  };

  // Render components
  return (
    <SafeAreaView style={styles.container}>
      <Header location={location} />
      {!isInputFocused && (
        <WeatherInfo
          icon={icon}
          weather={weather}
          temperature={temperature}
          windSpeed={windSpeed}
        />
      )}
      <LocationUpdate
        inputValue={inputValue}
        setInputValue={setInputValue}
        refreshWeather={refreshWeather}
        onFocus={() => setIsInputFocused(true)}
        onBlur={() => setIsInputFocused(false)}
      />
      <CurrentLocation getCurrentLocation={getCurrentLocation} />
    </SafeAreaView>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    paddingTop: 40,
    backgroundColor: "#F5F7F8",
  },
});

// Exporting Home component
export default Home;
```

6.4.3 components/Forecast.js

```
import React, { useState, useEffect } from "react";
import {
  SafeAreaView,
  FlatList,
  View,
  Image,
  Text,
  StyleSheet,
} from "react-native";

import { LocationContext } from "./Context/LocationContext";
import { ThemeContext } from "./Context/ThemeContext";

// Location and API key for the OpenWeatherMap API
const API_KEY = "395dc02446c77c0ac922cb465d9a395b";

// Component to render each weather item
function Item({ day, time, temperature, windSpeed, icon }) {
  return (
    <View style={styles.item}>
      <Text style={styles.day}>{day}</Text>
      <Text style={styles.time}>{time}:00</Text>
      <Image
        source={{ uri: `https://openweathermap.org/img/wn/${icon}@4x.png` }}
        style={styles.weatherIcon}
      />
      <Text style={styles.data}>{temperature}°C</Text>
      <Text style={styles.data}>{windSpeed} m/s</Text>
    </View>
  );
}

const Forecast = () => {
  const { theme, setTheme } = React.useContext(ThemeContext);
  const { location } = React.useContext(LocationContext);
  const [data, setData] = useState([]);

  // Use an effect to fetch the data once on component mount
  useEffect(() => {
    // Function to fetch weather data from the OpenWeatherMap API
    async function fetchWeatherData() {
      try {
        // Fetch the weather data
        const response = await fetch(
          `https://api.openweathermap.org/data/2.5/forecast?q=${location}&units=metric&appid=${API_KEY}`
        );
        const data = await response.json();

        // Map the response data to a more usable format
        let forecastData = data.list.map((item) => ({
          id: String(item.dt),
          day: ["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"][
            new Date(item.dt * 1000).getDay()
          ],
          time: new Date(item.dt * 1000)
            .toLocaleTimeString([], {
              hour: "2-digit",
              minute: "2-digit",
              hour12: false,
            })
            .slice(0, -3),
          temperature: Math.round(item.main.temp),
          windSpeed: Number(item.wind.speed.toFixed(1)),
          icon: item.weather[0].icon,
        }));

        return forecastData;
      } catch (error) {
        console.error(error);
      }
    }
    fetchWeatherData().then(setData);
  }, [location]);

```

```
  return (
    <SafeAreaView style={styles.container}>
      <View style={[styles.header, { backgroundColor: theme }]}>
        <Text style={styles.title}>Hourly Forecast</Text>
        <Text style={styles.subtitle}>{location}</Text>
      </View>
      <FlatList
        data={data}
        renderItem={({ item }) => (
          <Item
            day={item.day}
            time={item.time}
            temperature={item.temperature}
            windSpeed={item.windSpeed}
            icon={item.icon}
          />
        )}
        keyExtractor={(item) => item.id}
      />
    </SafeAreaView>
  );
};

// Styles for the components
const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: "#F5F7F8",
    paddingTop: 40,
  },
  header: {
    width: "100%",
    padding: 15,
  },
  title: {
    textAlign: "center",
    fontSize: 22,
    color: "#fff",
    fontWeight: "bold",
  },
  subtitle: {
    textAlign: "center",
    fontSize: 16,
    color: "#fff",
    marginTop: 5,
  },
  item: {
    padding: 5,
    marginHorizontal: 10,
    marginTop: 5,
    backgroundColor: "#D8DEE9",
    borderRadius: 5,
    flexDirection: "row",
    justifyContent: "space-between",
    alignItems: "center",
  },
  day: {
    width: 50,
    fontSize: 16,
    fontWeight: "bold",
    color: "#445069",
    textAlign: "center",
  },
  time: {
    width: 50,
    fontSize: 14,
    color: "#445069",
  },
  weatherIcon: {
    width: 40,
    height: 50,
  },
  data: {
    width: 70,
    fontSize: 14,
    color: "#445069",
  },
});

// Exporting Forecast component
export default Forecast;
```

6.4.4 components/Settings.js

```
import React from "react";
import { Platform, View, Text, StyleSheet, ActionSheetIOS } from "react-native";
import { Button } from "react-native-elements";
import { Picker } from "@react-native-picker/picker";
import { ThemeContext } from "./Context/ThemeContext";

const Settings = () => {
  const { theme, setTheme } = React.useContext(ThemeContext);

  const themeOptions = {
    Green: "#5B9A8B",
    Orange: "#EA906C",
    Pink: "#DC8686",
    Purple: "#8E8FFA",
    Brown: "#BCA37F",
  };

  const handleThemeChangeIOS = () => {
    ActionSheetIOS.showActionSheetWithOptions(
      {
        options: Object.keys(themeOptions),
        title: "Choose Theme",
      },
      (buttonIndex) => {
        setTheme(Object.values(themeOptions)[buttonIndex]);
      }
    );
  };

  return (
    <View style={styles.container}>
      <Text style={styles.title}>Welcome to the Settings Screen!</Text>
      <Text style={[styles.description, { color: theme }]}>
        Here, you can customize the app's theme.
      </Text>
      {Platform.OS === "ios" ? (
        <Button
          title="Change Theme"
          onPress={handleThemeChangeIOS}
          buttonStyle={[styles.button, { backgroundColor: theme }]}
        />
      ) : (
        <>
          <Text style={styles.pickerTitle}>Select a Theme:</Text>
          <Picker
            selectedValue={theme}
            onValueChange={(itemValue) => setTheme(itemValue)}
            style={[styles.picker, { backgroundColor: theme }]}
          >
            {Object.entries(themeOptions).map(([colorName, colorValue]) => (
              <Picker.Item
                key={colorName}
                label={colorName}
                value={colorValue}
              />
            ))}
          </Picker>
        </>
      )}
    </View>
  );
};

```

```
const styles = StyleSheet.create({
  container: {
    padding: 20,
    marginTop: 100,
  },
  title: {
    fontSize: 24,
    fontWeight: "bold",
    marginBottom: 50,
    color: "#445069",
  },
  description: {
    fontSize: 18,
    marginBottom: 50,
  },
  pickerTitle: {
    fontSize: 18,
    fontWeight: "bold",
    marginBottom: 10,
  },
  picker: {
    height: 50,
    width: 200,
    borderColor: "#445069",
    borderWidth: 1,
    borderRadius: 4,
  },
  button: {
    borderRadius: 5,
    width: 200,
  },
});

// Exporting Settings component
export default Settings;
```

6.4.5 components/Context/ThemeContext.js

```
import React from 'react';

export const LocationContext = React.createContext();
```

6.4.6 components/Context/LocationContext.js

```
import React from 'react';

export const ThemeContext = React.createContext();
```

6.4.7 components/Current/CurrentWeatherHeader.js

```
import React from 'react';
import { View, Text, StyleSheet } from 'react-native';
import { ThemeContext } from '../Context/ThemeContext';

const CurrentWeatherHeader = ({ location }) => {
  const { theme } = React.useContext(ThemeContext);

  const currentDate = new Date().toLocaleDateString('fi-FI');

  return (
    <View style={[styles.header, {backgroundColor: theme}]}>
      <Text style={styles.title}>Current weather</Text>
      <Text style={styles.subtitle}>{location}   {currentDate}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  header: {
    width: '100%',
    padding: 15,
  },
  title: {
    textAlign: 'center',
    fontSize: 22,
    color: '#fff',
    fontWeight: 'bold',
  },
  subtitle: {
    textAlign: 'center',
    fontSize: 16,
    color: '#fff',
    marginTop: 5,
  },
});


export default CurrentWeatherHeader;
```

6.4.8 components/Current/CurrentWeatherInfo.js

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

const CurrentWeatherInfo = ({ weather, icon, temperature, windSpeed }) => {
  return (
    <View style={styles.container}>
      <View style={styles.upperContainer}>
        <Image source={{ uri: `https://openweathermap.org/img/wn/${icon}@4x.png` }}
          style={styles.weatherIcon} />
        <View style={styles.dataContainer}>
          <Text style={styles.textSmall}>{temperature}°C</Text>
          <Text style={styles.textSmall}>{windSpeed}m/s</Text>
        </View>
      </View>
      <Text style={styles.textBig}>{weather}</Text>
    </View>
  );
};

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
    alignItems: 'center',
    padding: 15,
  },
  upperContainer: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
  },
  dataContainer: {
    marginLeft: 30,
    marginTop: 24,
  },
  weatherIcon: {
    width: 150,
    height: 150,
  },
  textSmall: {
    marginBottom: 20,
    fontSize: 24,
    color:'#445069',
  },
  textBig: {
    fontSize: 32,
    textAlign: 'center',
    color:'#445069',
  },
});

export default CurrentWeatherInfo;
```

6.4.9 components/Current/CurrentWeatherLocationUpdate.js

```
import React from "react";
import { View, TextInput, StyleSheet } from "react-native";
import { Button } from "react-native-elements";
import { ThemeContext } from "../Context/ThemeContext";

const CurrentWeatherLocationUpdate = ({
  inputValue,
  setInputValue,
  refreshWeather,
  onFocus,
  onBlur,
}) => {
  const { theme } = React.useContext(ThemeContext);

  const handleFocus = () => {
    if (onFocus) {
      onFocus();
    }
  };

  const handleBlur = () => {
    if (onBlur) {
      onBlur();
    }
  };

  return (
    <View style={styles.inputButtonContainer}>
      <TextInput
        style={styles.input}
        value={inputValue}
        onChangeText={(text) => setInputValue(text)}
        placeholder="Enter a city"
        onFocus={handleFocus}
        onBlur={handleBlur}
      />
      <Button
        title="Refresh"
        onPress={refreshWeather}
        buttonStyle={[styles.button, { backgroundColor: theme }]}
      />
    </View>
  );
};

const styles = StyleSheet.create({
  inputButtonContainer: {
    flex: 1,
    justifyContent: "center",
    paddingVertical: 15,
    alignSelf: "center",
  },
  input: {
    height: 30,
    borderColor: "gray",
    borderWidth: 1,
    marginBottom: 20,
    paddingHorizontal: 20,
    width: 200,
    alignSelf: "center", // to center the input
  },
  button: {
    borderRadius: 5,
    width: 200,
  },
});

export default CurrentWeatherLocationUpdate;
```

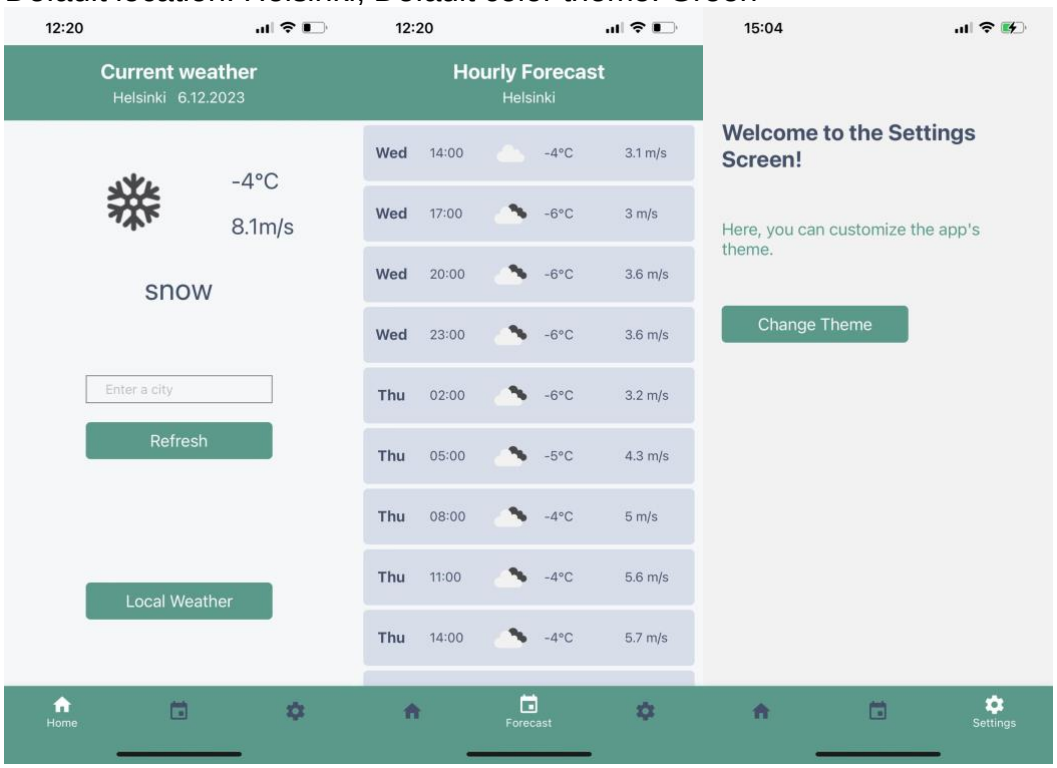
6.4.10 components/Current/CurrentLocation.js

```
import React from 'react';
import { View, StyleSheet } from 'react-native';
import { Button } from 'react-native-elements';
import { ThemeContext } from '../Context/ThemeContext';

const CurrentLocation = ({ getCurrentLocation }) => {
  const { theme } = React.useContext(ThemeContext);

  return (
    <View style={styles.container}>
      <Button title="Local Weather" onPress={getCurrentLocation}
        buttonStyle={[styles.button, {backgroundColor: theme}]}/>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    justifyContent: 'center',
  },
  button: {
    borderRadius: 5,
    width: 200,
    alignSelf: 'center',
  },
});

// Exporting CurrentLocation component
export default CurrentLocation;
```

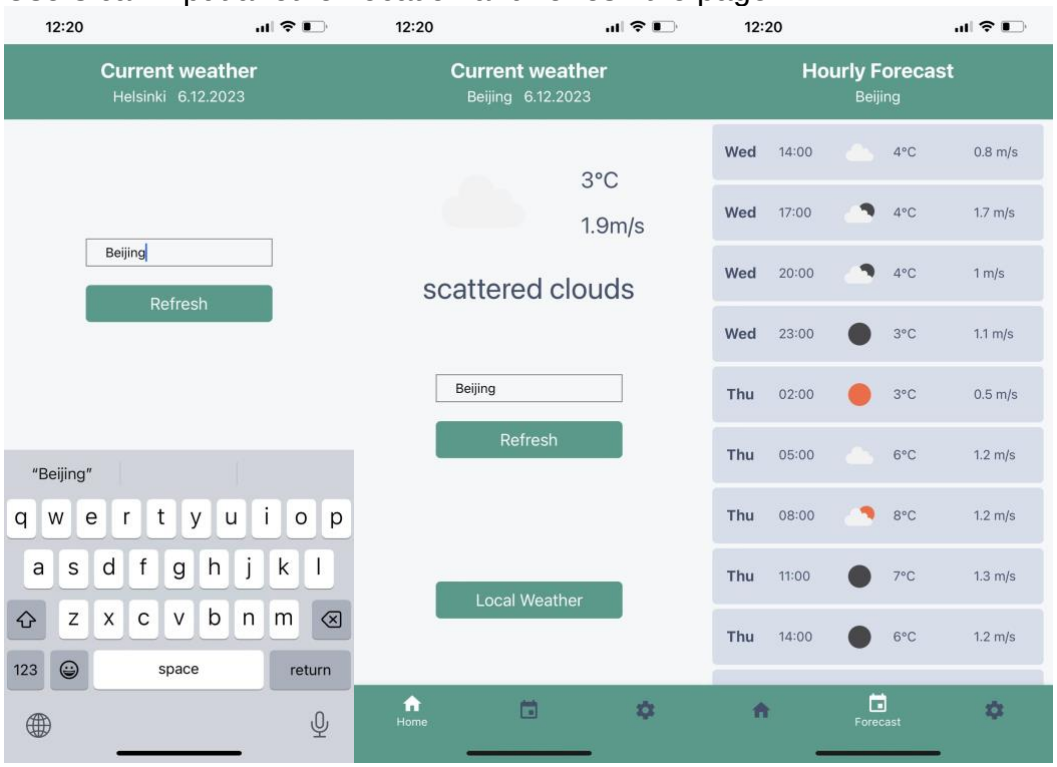
6.5 Test via Expo Go

6.5.1 On an ios device (iPhone 13 Pro Max, v17.1.2)

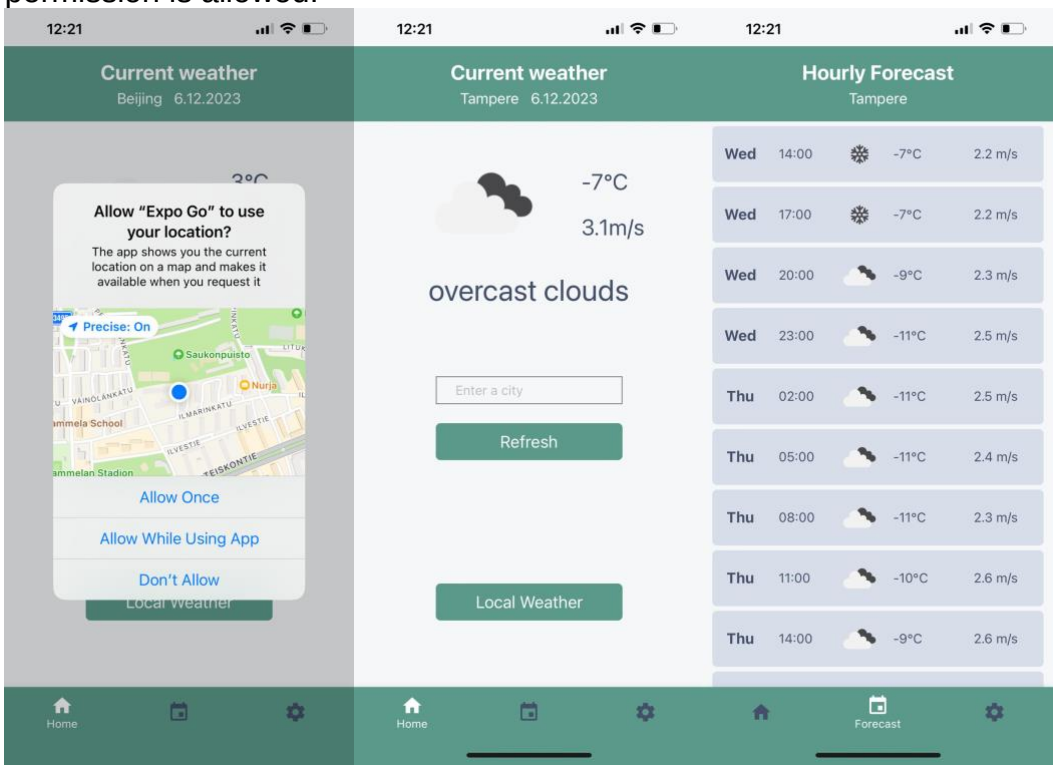
Default location: Helsinki, Default color theme: Green



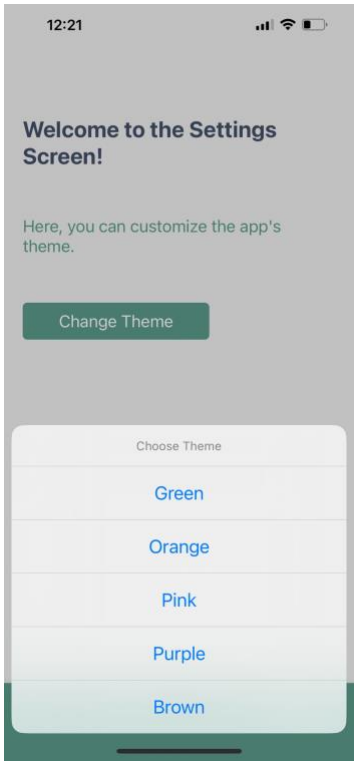
Users can input another location and refresh the page:



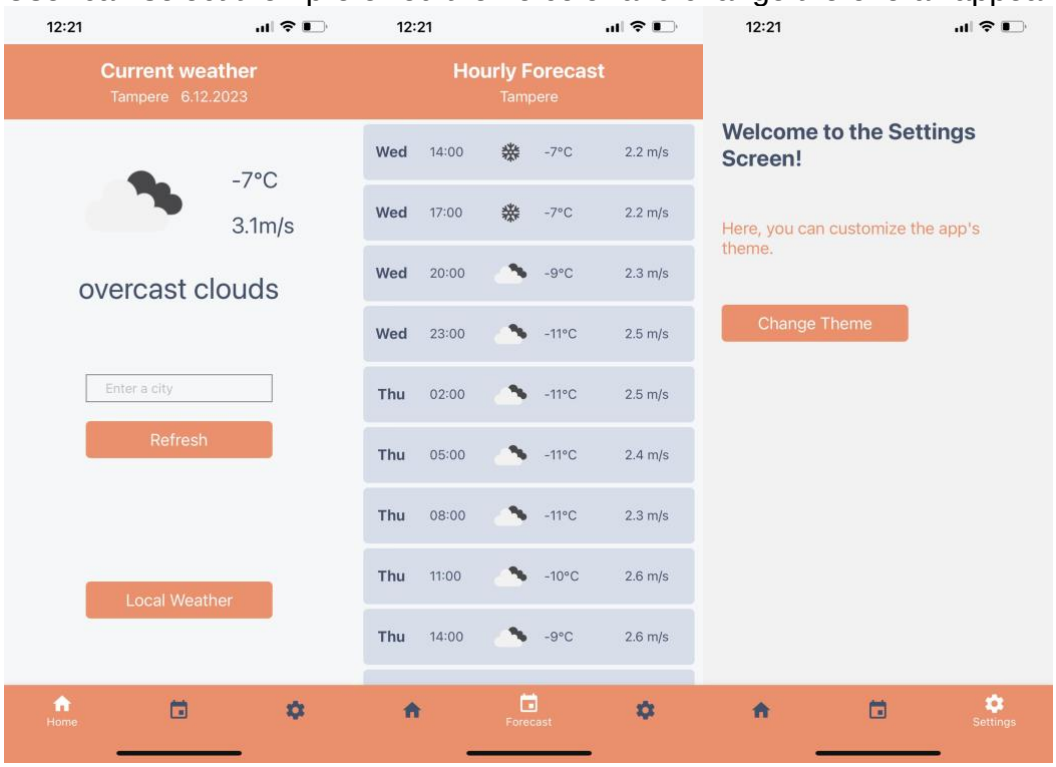
Users can click the “Local Weather” button to see local weather if LOCATION permission is allowed:



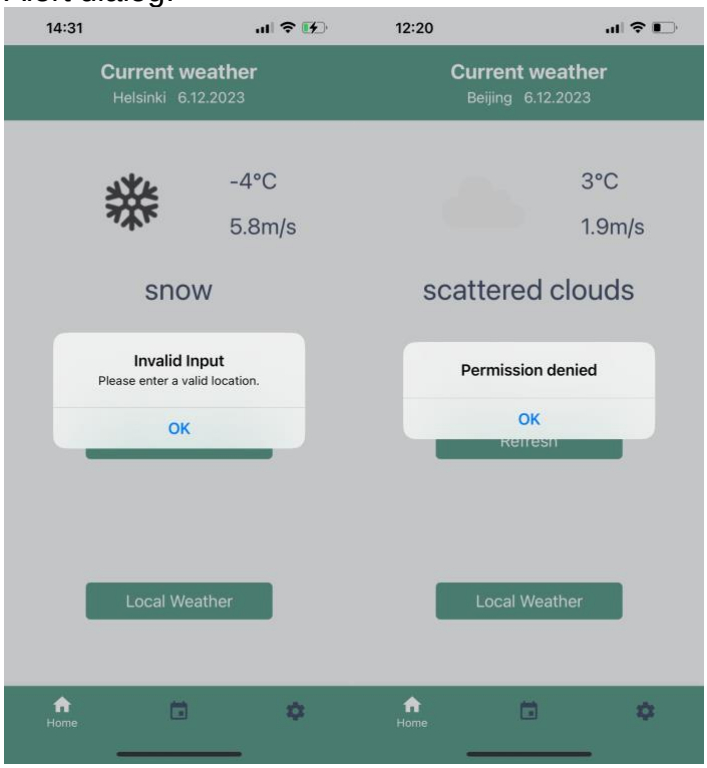
ActionSheetIOS for selections:



User can select their preferred theme color and change the overall appearance:

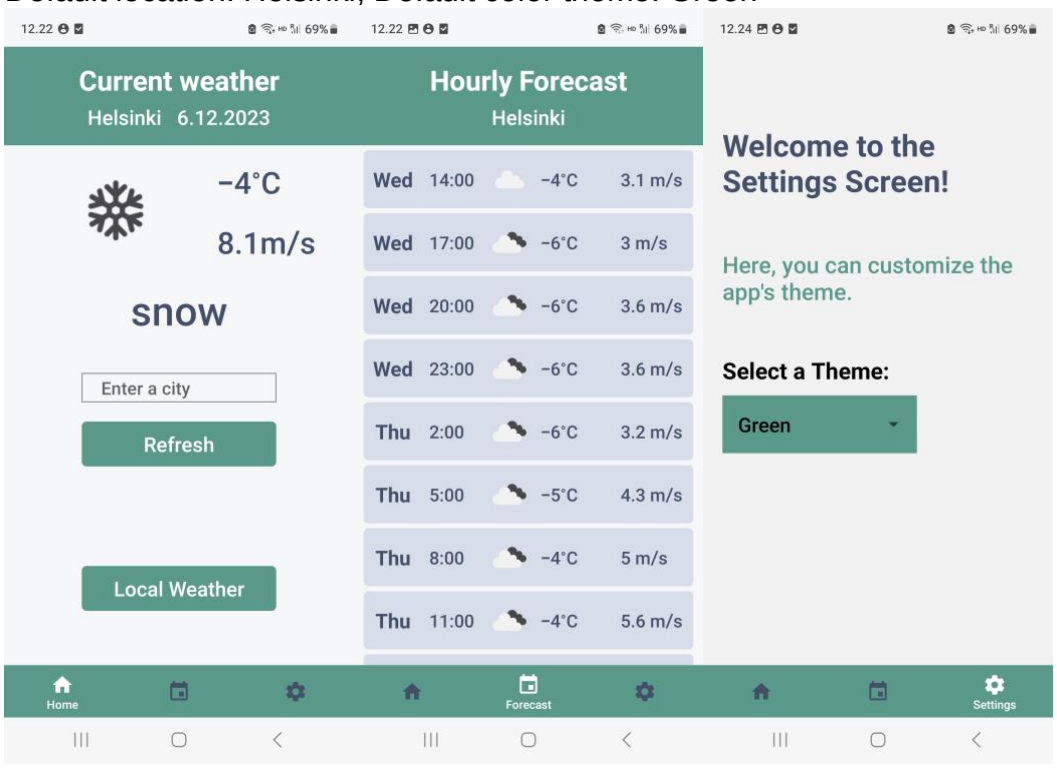


Alert dialog:

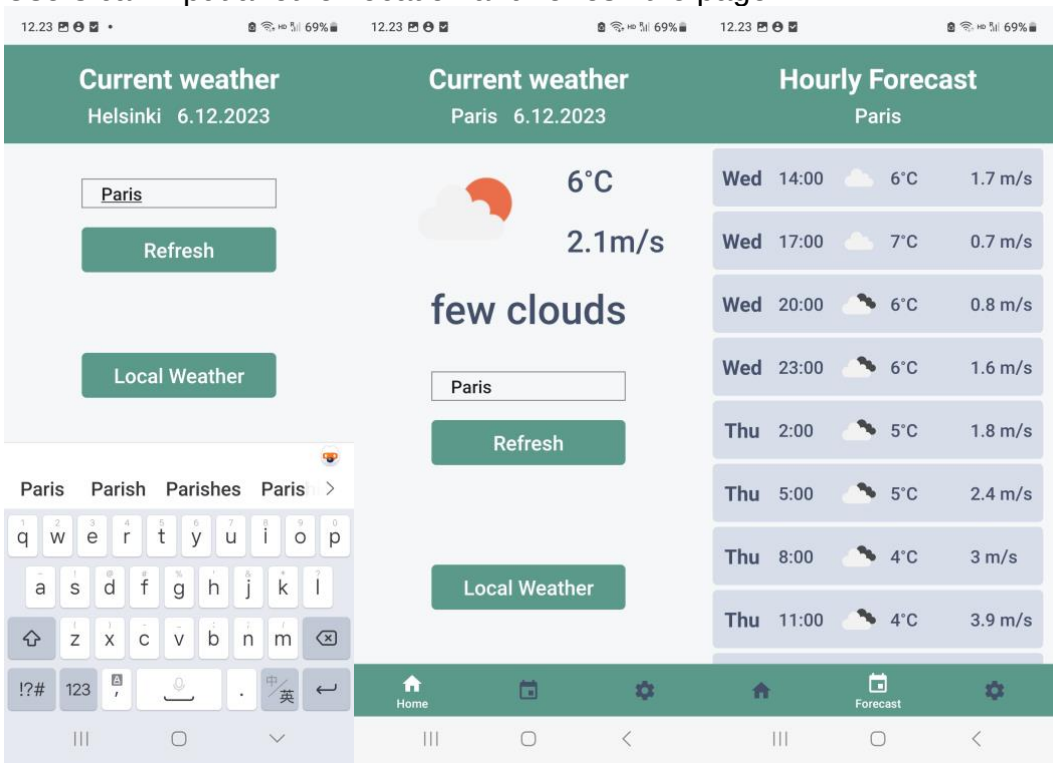


6.5.2 On Android device (Samsung S21 FE, v13)

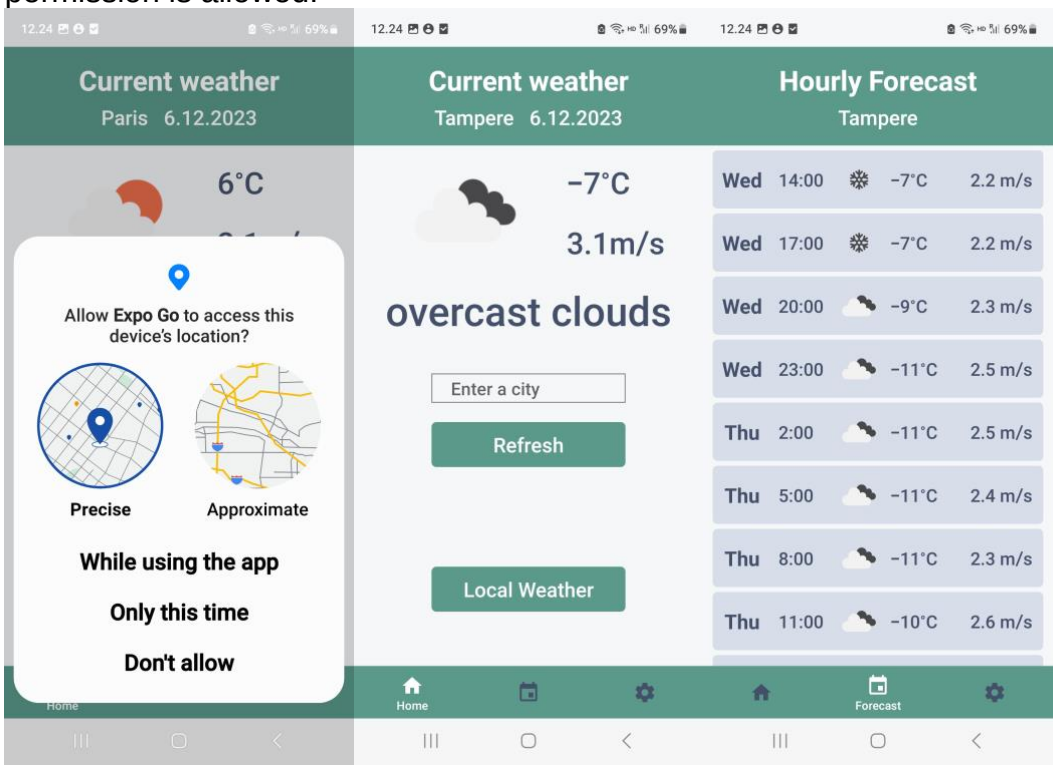
Default location: Helsinki, Default color theme: Green



Users can input another location and refresh the page:

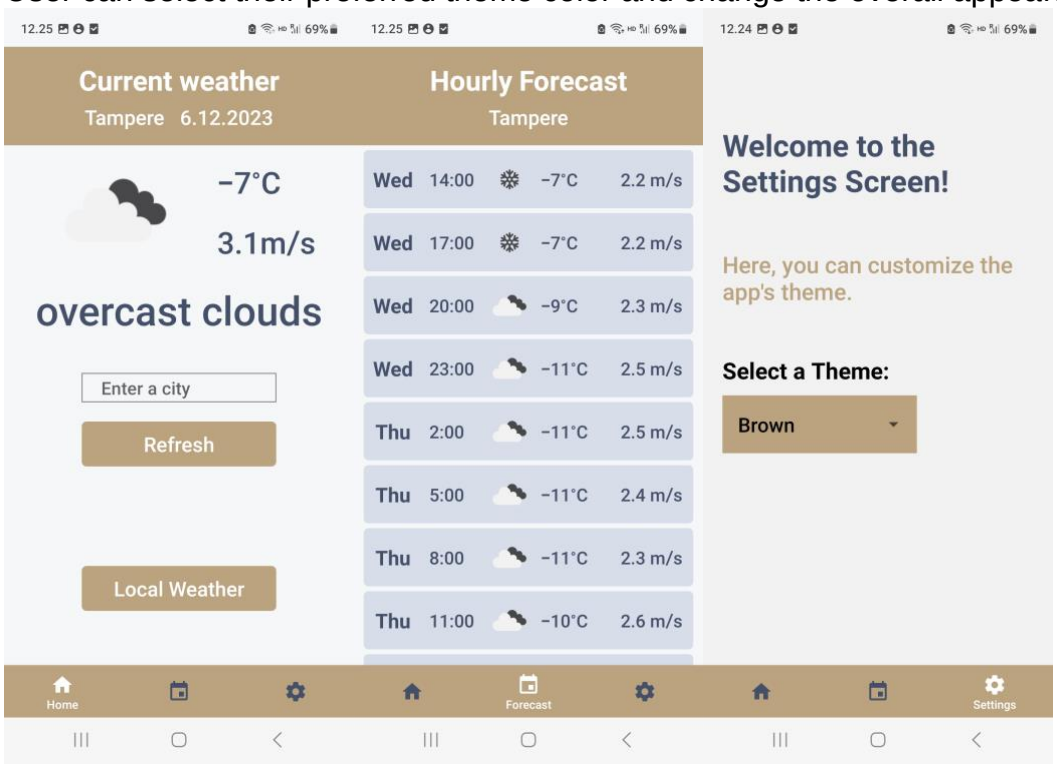


Users can click the “Local Weather” button to see local weather if LOCATION permission is allowed:

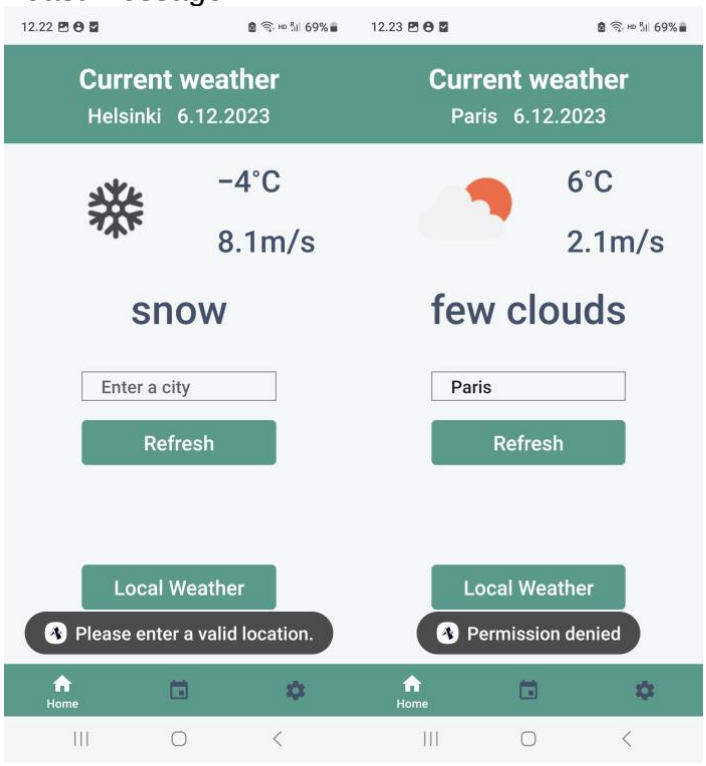


Picker for selections:

User can select their preferred theme color and change the overall appearance:



Toast message:



Sources used with exercises

https://kotlinlang.org/docs/cross-platform-mobile-development.html#different-approaches-to-mobile-app-development

https://appinventiv.com/blog/cross-platform-app-frameworks/

https://reactnative.dev/docs/getting-started?guide=web

https://www.webscrapingapi.com/fetch-alternative-js

https://codenameuriel28.medium.com/an-alternative-to-fetch-using-axios-b2387a5b721

https://reactnative.dev/docs/navigation

https://blog.logrocket.com/react-native-navigation-tutorial/

https://reactnavigation.org/docs/stack-navigator/

https://reactnavigation.org/docs/drawer-navigator

https://reactnavigation.org/docs/material-bottom-tab-navigator

OpenWeatherMap API Tutorial 2021
https://www.youtube.com/watch?v=nGVoHEZojiQ

Getting started with OpenWeatherMap API - python script
https://www.youtube.com/watch?v=PWZKTWJ9bJE

https://reactnative.dev/docs/flexbox

https://reactnative.dev/docs/platform-specific-code

https://www.w3cschool.cn/doc_react_native/react_native-datepickerios.html

https://react-native-sensors.github.io/docs/API.html

https://docs.expo.dev/guides/permissions/

https://reactnative.dev/docs/permissionsandroid

https://reactnative.dev/docs/linking?language=javascript

https://reactnative.dev/docs/linking#openurl

https://ihateregex.io/expr/phone/

https://docs.expo.dev/versions/latest/sdk/battery/

https://docs.expo.dev/versions/latest/sdk/camera/

https://www.youtube.com/watch?v=9EoKurp6V0I

https://blog.expo.dev/so-you-want-to-build-a-bluetooth-app-with-react-native-and-expo-6ea6a31a151d

https://github.com/expo/config-plugins/tree/main/packages/react-native-ble-plx

https://blog.logrocket.com/using-react-native-ble-manager-mobile-app/

https://pagepro.co/blog/publishing-expo-react-native-app-to-ios-and-android/

https://www.youtube.com/watch?v=oBWBDaqNuws

https://www.youtube.com/watch?v=LE4Mgkrf7Sk

https://legacy.reactjs.org/docs/context.html

https://www.w3cschool.cn/doc_react_native/react_native-navigatorios.html?lang=en

https://www.w3cschool.cn/doc_react_native/react_native-drawerlayoutandroid.html?lang=en