COMP7506 Individual Assignment

Leung Ga Wai – 3035097428

Topic: Smartphone Apps Development Using React Native and Expo

What is React Native?

React Native is a JavaScript framework for building cross platform native mobile apps. It allows developers to develop with React framework, together with native platform functionalities. As React framework is popular in website development, web developers may find it easy to build an mobile app with React native.

In this tutorial, we are also using Expo. It is a framework and platform that provides sets of tools and services for React Native projects. According to official React Native website, Expo CLI (Command-line interface) is recommended to beginners of mobile development.

There is no specific IDE for React Native project. Any text editors would work. I am using Visual Studio Code as my personal preference.

About this Tutorial

We will be building a simple app to illustrate simple concepts about React Native. A user can enter 3 numbers and a button on the screen. When the button is pressed, the sum of the 3 numbers will be displayed. This should be a good starting point for learning the framework.

Set up the Environment

Before any coding, we have to set up the development environment first.

Install Necessary Tools

Since React Native is a JavaScript framework, please make sure Node.js (14 LTS or greater) and npm are installed. You may follow the instruction in the website.

We are using Expo CLI. Please have this installed.

shell
```
npm install -g expo-cli
```

Initializing a Project

Run the following in terminal to create a new React Native Project named “ReactNativeDemo”. Choose the option “blank” (default) when asked to choose a template.

shell
```
expo init ReactNativeDemo
cd ReactNativeDemo
```

A blank app is created! However, in order to see the app, you will need to view it through a physical device or simulator / emulator. This is described below.

View the App

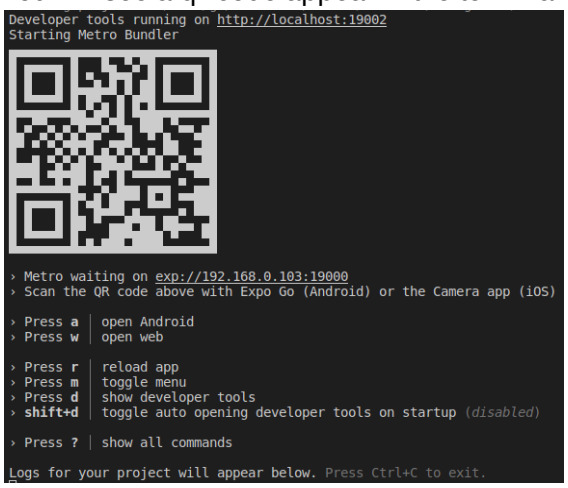
View the app with physical device

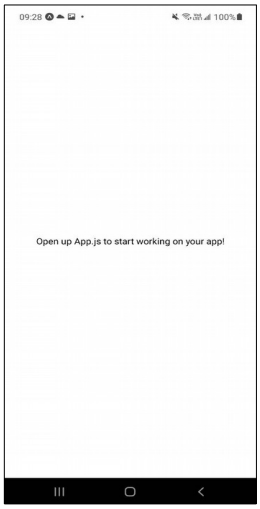
First, please have Expo Go App installed on your smartphone (Android / iOS).

Then, run the following in the terminal to start the development server:

shell
```
expo start
```

You will see a qr-code appear in the terminal.





Scan the qr-code the Expo Go App (Android) or default Apple Camera App (iOS) to view the app in the Expo Go. Please ensure both the smartphone and computer are in the same wifi network.

View in Android Studio Emulator

Please follow the steps provided by Expo. Basically you need to set up Android studio's tools, add and environment variable pointing to the Android SDK location, and set up a virtual device.

Run `expo start` in terminal. Press `a` to view the app in emulator.

View in iOS Simulator

Please follow the steps provided by Expo. Basically you need to install Xcode and Xcode command line tools.

Run `expo start` in terminal. Press `i` to view the app in emulator.

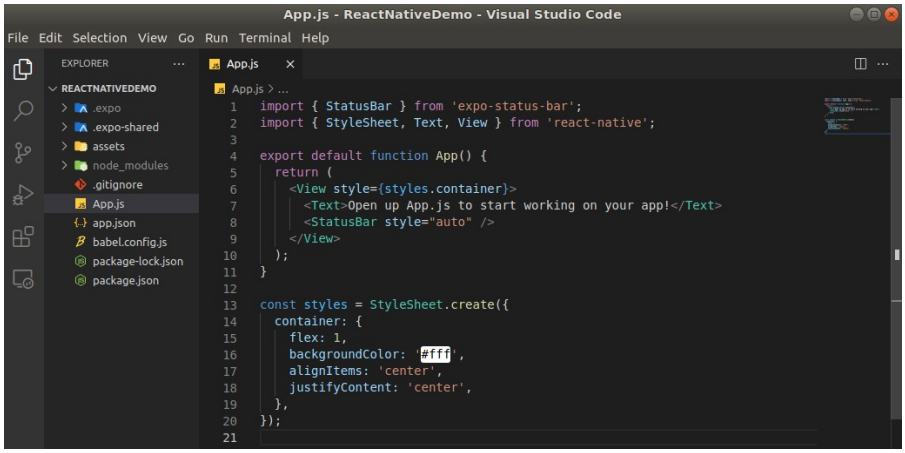
Fast refresh

To allow the app to refresh once the code is changed, please keep the terminal on after running `expo start`. The terminal will also display errors if there is any. In this case, you can fix the code and save the file. The app will be refreshed if the errors are fixed. In case the app fails to refresh automatically, you can press `r` in the terminal to force refresh.

Basic React Native App

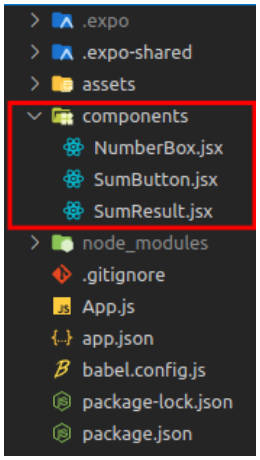
Components

Below shows the current folder structure and content of App.js.



`App.js` is the entry point of the app. In the function `App()` , there are core components like `View` and `Text`. They are basic building blocks for the app, from controls to activity indicators. They can be found in the documented API section.

Users can create custom components too. Create a folder `components/` in the project folder and create 3 files named `NumberBox.jsx` , `SumResult.jsx` , and `SumButton.jsx` . These will be the custom components for the input boxes, result box, and the button to press for summing the numbers.



The codes for the 3 custom components are as below.

NumberBox.jsx

```
import React from 'react'
import { TextInput } from 'react-native'

export default NumberBox = ({}) => {
  return (
    <TextInput
      style={{backgroundColor: 'lightgrey', margin: 2}}
      keyboardType="numeric"
    />
  )
}
```

SumResult.jsx

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

```
import { Text } from 'react-native'

export default NumberBox = ({result}) => {
  return (
    <Text
      style={{backgroundColor: 'lightblue', margin: 2}}
    > {result} </Text>
  )
}
```

SumButton.jsx

```
import React from 'react'
import { Text, TouchableOpacity } from 'react-native'

export default SumButton = ({onPressBtn}) => {
  return (
    <TouchableOpacity
      style={{backgroundColor: 'coral'}}
      onPress={onPressBtn}
    >
      <Text> + </Text>
    </TouchableOpacity>
  )
}
```

In `App.js`, we can import the custom components and add the components in the `App()` function:

App.js

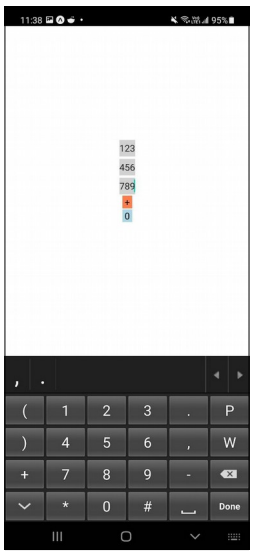
```
import React from 'react'
import { StatusBar } from 'expo-status-bar'
import { StyleSheet, Text, View } from 'react-native'
import NumberBox from './components/NumberBox'
import SumResult from './components/SumResult'
import SumButton from './components/SumButton'

export default function App() {
  return (
    <View style={styles.container}>
      <StatusBar style="auto" />
      <NumberBox />
      <NumberBox />
      <NumberBox />
      <SumButton />
      <SumResult result={0} />
    </View>
  );
}
//...
```

Here is a simple explanation on the custom components. For example, to define NumberBox component, first we import React and Reac Native's TextInput Core Component. Then, the component starts with a function, for which it returns is rendered as a React element. Hence, the NumberBox component will render a TextInput element.

Props is defined in SumResult component. "result" is a prop, that is passed from the parent (App) to the child (SumResult). Currently it is a placeholder for future use. Props can be virtually anything, including values, functions, or objects.

Below will be the current app layout. We have just defined the components and it is not working yet. And the appearance is pretty bad.



Styles and Flexbox

With React Native, we can style the app with JavaScript. We can pass a prop named `style` to define the style of core components. Web developers may find this familiar, as it is analogous to CSS on the web.

An example for styling can be found in file `App.js`. It uses `StyleSheet.create()` to create a stylesheet for components to refer to. We can also specify styles with plain JavaScript object or arrays, just like in the current implementation of the 3 custom components. Notice that styles can be passed as props. They are highlighted in **blue** in the codes below for reference.

Flexbox is an algorithm for specifying the layout of child components. It is designed to provide consistent layouts of devices of different sizes. It works in a similar way in React Native as in CSS in web development. Flexbox settings are specified just like other styles does: in other stylesheet, or plain JavaScript objects. In the codes below, items related to flexbox algorithm are highlighted in **red**. Please see the documentation for detail.

NumberBox.jsx

```
import React from 'react'
import { TextInput, StyleSheet } from 'react-native'

export default NumberBox = ({textStyle}) => {
  return (
    <TextInput
      style={[styles.inputText, textStyle]}
      keyboardType="numeric"
    />
  )
}

const styles = StyleSheet.create({
  inputText: {
    backgroundColor: 'lightgrey',
    minWidth: 150
  }
})
```

SumResult.jsx

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

```
export default NumberBox = ({result, textStyle}) => {
  return (
    <Text
      style={[styles.resultText, textStyle]}
    > {result} </Text>
  )
}

const styles = StyleSheet.create({
  resultText: {
    backgroundColor: 'lightblue',
    minWidth: 150
  }
})
```

SumButton.jsx

```
import React from 'react'
import { Text, TouchableOpacity, StyleSheet } from 'react-native'

export default SumButton = ({onPressBtn, textStyle}) => {
  return (
    <TouchableOpacity
      style={styles.sumBtn}
      onPress={onPressBtn}
    >
      <Text style={[textStyle, styles.plusSign]}> + </Text>
    </TouchableOpacity>
  )
}

const styles = StyleSheet.create({
  sumBtn: {
    backgroundColor: 'coral',
    width: 50,
    height: 50,
    margin: 2
  },
  plusSign: {
    color: 'oldlace',
    height: 'auto'
  }
})
```

App.js

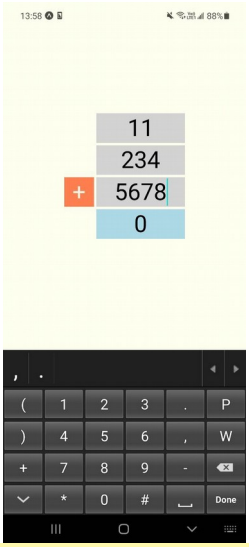
```
import React from 'react'
import { StatusBar } from 'expo-status-bar'
import { StyleSheet, Text, View } from 'react-native'
import NumberBox from './components/NumberBox'
import SumResult from './components/SumResult'
import SumButton from './components/SumButton'

export default function App() {
  return (
    <View style={styles.container}>
      <View style={{alignItems: 'flex-end'}}>
        <StatusBar style="auto" />
        <NumberBox textStyle={styles.texts}/>
        <NumberBox textStyle={styles.texts}/>
        <View style={{flexDirection: 'row'}}>
          <SumButton textStyle={styles.texts}/>
          <NumberBox textStyle={styles.texts}/>
```

```
        </View>
        <SumResult result={0} textStyle={styles.texts}/>
      </View>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'ivory',
    alignItems: 'center',
    justifyContent: 'center',
  },
  texts: {
    textAlign: 'center',
    textAlignVertical: 'center',
    includeFontPadding: false,
    fontSize: 36,
    margin: 2,
    height: 50,
    color: 'black'
  }
});
```

Below is the appearance of the app. It looks much nicer, but the function is yet to be implemented.



Functionality and State Hook

States

Both props and states are data that control a component. However, props are fixed throughout the lifetime of components, and states are data that is about to change, i.e. mutable. Components with data reference to those states will update whenever the data changes. In our app, we would like some data to be mutable, specifically the array of 3 numbers, and the sum of 3 numbers.

In the code, we can declare a state variable like below:

`const [sum, setSum] = React.useState(null)`

We have a variable named `sum`. The `React.useState()` function has created the variable, the function `setSum` for setting the variable, and defined an initial value (`null`) for the variable. Whenever we would like to change the value of the variable `sum`, we have to use the function `setSum`. Assume we would like the variable sum to change to 567, we can do this by `setSum(567)`.

Please update `NumberBox.jsx` and `App.js` as below.

NumberBox.jsx

```
import React from 'react'
import { TextInput, StyleSheet } from 'react-native'

export default NumberBox = ({value, onChange, textStyle}) => {
  return (
    <TextInput
      style={[styles.inputText, textStyle]}
      keyboardType="numeric"
      onChangeText={onChange}
      value={value}
    />
  )
}

const styles = StyleSheet.create({
  inputText: {
    backgroundColor: 'lightgrey',
    minWidth: 150
  }
})
```

App.js

```
import React from 'react'
import { StatusBar } from 'expo-status-bar'
import { StyleSheet, View, Alert } from 'react-native'
import NumberBox from './components/NumberBox'
import SumResult from './components/SumResult'
import SumButton from './components/SumButton'

export default function App() {
  const [numArr, setNumArr] = React.useState([0, 0, 0])
  const [sum, setSum] = React.useState(null)

  const strToNum = (idx, v) =>{
    let num = Number(v)
    let tempArr = [...numArr]
    tempArr[idx] = num
    setNumArr(tempArr)
    setSum(null)  // reseting the sum when user inputs new numbers
  }

  const onPressBtn = () => {
    const arrSum = numArr[0] + numArr[1] + numArr[2]
    if (isNaN(arrSum)){
      Alert.alert('There is an error in your input.')
    } else {
      setSum(arrSum)
    }
  }

  return (
    <View style={styles.container}>
      <View style={{alignItems: 'flex-end'}}>
        <StatusBar style="auto" />
        <NumberBox num={numArr[0]} onChange={v => strToNum(0, v)}
          textStyle={styles.texts}/>
        <NumberBox num={numArr[1]} onChange={v => strToNum(1, v)}
          textStyle={styles.texts}/>
        <View style={{flexDirection: 'row'}}>
          <SumButton onPressBtn={onPressBtn} textStyle={styles.texts}/>
```

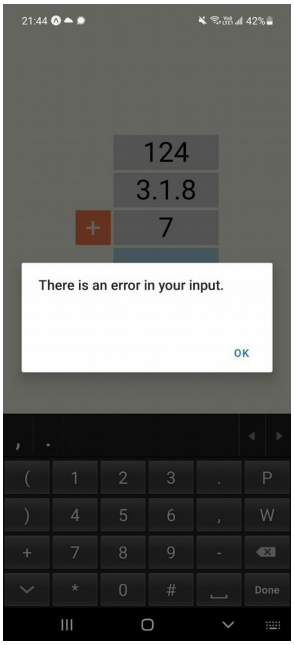
```
          <NumberBox num={numArr[2]} onChange={v => strToNum(2, v)}
            textStyle={styles.texts}/>
        </View>
        <SumResult result={sum} textStyle={styles.texts}/>
      </View>
    </View>
  );
}

const styles = StyleSheet.create({
  container: {
    flex: 1,
    backgroundColor: 'ivory',
    alignItems: 'center',
    justifyContent: 'center',
  },
  texts: {
    textAlign: 'center',
    textAlignVertical: 'center',
    includeFontPadding: false,
    fontSize: 36,
    margin: 2,
    height: 50,
    color: 'black'
  }
});
```

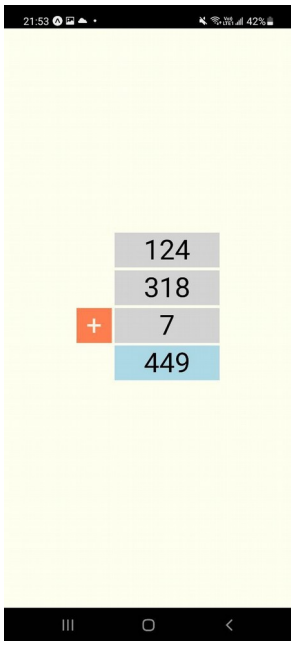
In the code above, the code related to states are highlighted in **purple**. These includes the initialization of variables and setting new values to the states.

The `TextInput` values are strings, which has to be converted to numbers. The strings are converted to numbers in the App component (`strToNum`). When the value of `TextInput` changes, it triggers **onChangeText** callback of the TextInput component. It is then passed back to the App component for convertion and store to the numArray state variable. The related codes are highlighted in **orange**.

We need to be cautious about what users enter. When users enters some values (like alphabets) into the `TextInput`, they may be unable to be converted to numbers. Hence, we have to catch the errors if there is improper inputs. The error-catching codes are highlighted in **red**. When user click the button and there is a invalid input, an alert box will be prompted. For example, when one of the box is entered "3.1.8", which is an invalid number, a prompt is displayed as below.



Congratulations, you have now created a simple React Native App with Expo!



What's Next?

This tutorial is just a very beginning of learning React Native and Expo. We cannot cover everything. Here are some topics you can learn next.

Publishing

When you have built an amazing app, you still have to publish it. You may refer to the documentation offered by Expo.

Using Libraries

We do not want to reinvent the wheel. It is good to use some libraries when building practical apps. There are different types of libraries for Expo projects, like React Native libraries, Expo SDK libraries, and third party libraries. Please refer to the documentation on using libraries offered by Expo.