

3SA04 – React Native

1. เครื่องมือที่จำเป็น

- Chocolatey (for Windows), Brew (for OSX)
- Node.js
- Yarn
- Git
- create-react-native-app CLI
- Visual Studio Code
- Android Studio

หลังจากที่ได้ติดตั้ง Chocolatey ในเครื่องแล้ว สามารถติดตั้ง Node.js, Yarn และ Git ได้ผ่าน Chocolatey ผ่าน Command Prompt (ที่รันด้วยสิทธิ์ Administrator)

```
>> choco install nodejs
>> choco install yarn
>> choco install git
```

ในการติดตั้ง create-react-app CLI สามารถทำได้ผ่านการใช้คำสั่ง yarn ผ่าน Command Prompt (ที่รันด้วยสิทธิ์ Administrator)

```
>> yarn global add create-react-native-app
```

** สำหรับคอมพิวเตอร์ให้ห้องปฏิบัติการ เครื่องมือข้างต้นได้ติดตั้งไว้ล่วงหน้าเรียบร้อยแล้ว

ในการทดลองนี้ นศ. จะต้องติดตั้งโปรแกรม Expo (มีทั้งบน Android และ iOS) ลงบนสมาร์ตโฟนที่ใช้ในการรันโปรแกรม

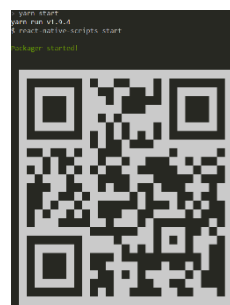
2. Hello world

สร้างโครงร่างโปรเจกต์สำหรับการพัฒนา React Native ด้วย create-react-native-app

```
>> create-react-native-app wt-app
>> cd wt-app
>> yarn upgrade react-native@0.55.4
```

ทำการรันโปรแกรมโปรแกรมขึ้นมา แล้วใช้สมาร์ตโฟนรันโปรแกรม expo แล้วสแกน QR Code ที่ได้จากคำสั่ง yarn start

```
>> yarn start
```



Open up App.js to start working on your app!
Changes you make will automatically reload.
Shake your phone to open the developer menu.



*คำสั่ง `yarn upgrade` ก่อนหน้าใช้เพื่ออัปเดต React Native ให้เป็นเวอร์ชัน 0.55.4 อย่างไรก็ตาม ในการทดลองนี้ ตั้งสมมติฐานว่า `create-react-native-app` ทำการติดตั้ง React Native เวอร์ชัน 0.55.2 เป็น default หากมีการเปลี่ยนแปลง ให้ค้น ทำการศึกษา Document เพิ่มเติมจาก <https://github.com/react-community/create-react-native-app/blob/master/VERSIONS.md>

Source Code

เปิด Source Code ของโปรเจกต์ wt-app ด้วย Visual Studio Code หรือ Text Editor ที่ต้องการ แก้ไข App.js เหมือนโค้ดข้างล่าง

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

export default class App extends React.Component {
  doIt = () => {
    console.log("Hello from console")
  }
  render() {
    return (
      <View style={styles.container}>
        <Text onPress={this.doIt}>Hello World</Text>
      </View>
    );
  }
}

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

สังเกตผลลัพธ์ที่ได้บนโปรแกรม Expo ในสมาร์ตโฟน

Import & Export Components

สร้างไฟล์เตอร์ components ในโปรเจ็ค แล้วสร้างไฟล์ Weather.js

ให้ copy code ทั้งหมดจาก App.js แล้วเปลี่ยนชื่อคลาส App เป็นคลาส Weather

เปลี่ยนโค้ดใน App.js ให้เป็นไปตามโค้ดข้างล่าง

```
import React from 'react';
import Weather from './components/Weather'

export default class App extends React.Component {
  render() {
    return (
      <Weather/>
    );
  }
}
```

3. Components

Passing Props

คอมโพเนนต์ Weather รับ Props ชื่อ zipCode โดยให้กำหนด zipCode เป็น 90110

ไฟล์ App.js

```
<Weather zipCode="90110"/>
```

ไฟล์ Weather.js

```
export default class Weather extends React.Component {
  render() {
    return (
      <View style={styles.container}>
        <Text>{this.props.zipCode}</Text>
      </View>
    );
  }
}
```

Components and Image Background

กำหนด State ให้กับคอมโพเนนต์ใน constructor แล้วใช้เป็น props ส่งผ่านไปยังคอมโพเนนต์ Forecast ที่สร้างขึ้นใหม่

ไฟล์ Forecast.js (สร้างขึ้นใหม่ – ให้นำค. Import ให้ถูกต้อง)

```
export default class Forecast extends React.Component {
  render() {
    return (
      <View>
        <Text>{this.props.main}</Text>
        <Text>{this.props.description}</Text>
        <Text>{this.props.temp}</Text>
        <Text>°C</Text>
      </View>
    );
  }
}
```

ไฟล์ Weather.js และรูป background ที่เหมาะสม

```
export default class Weather extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      forecast: {
        main: '-', description: '-', temp: 0
      }
    }
  }

  render() {
    return (
      <View style={styles.container}>
        <ImageBackground source={require('./bg.jpeg')} style={styles.backdrop}>
          <Text>Zip code is {this.props.zipCode}</Text>
          <Forecast {...this.state.forecast} />
        </ImageBackground>
      </View>
    );
  }
}
```

CHECK POINT #1

แสดงผลการทำงานของรันบนมือถือ

```
const styles = StyleSheet.create({
  container: { paddingTop: 25 },
  backdrop: { width: '100%', height: '100%' },
});
```

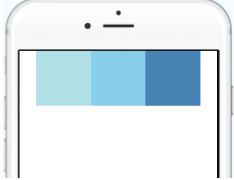
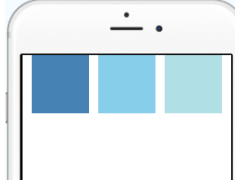
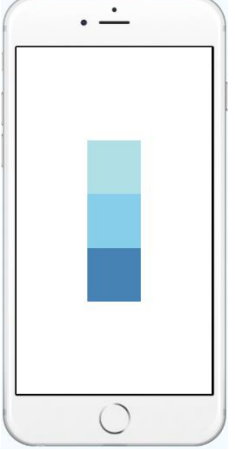
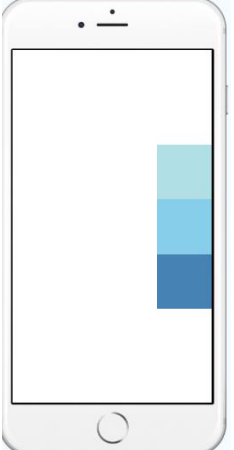
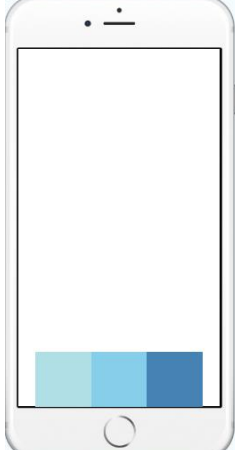
4. Flex Box

การจัด Layout บน React Native จะใช้ Flex Box ในการจัดการ แม้ว่า เทคนิคการจัด Flex Box จะรองรับความซับซ้อนสูง (React Native ไม่สนับสนุนทุก features ของ Flex Box) อย่างไรก็ตามคุณสมบัติที่ถูกใช้บ่อยในการจัด Layout มี 3

คุณสมบัติ คือ

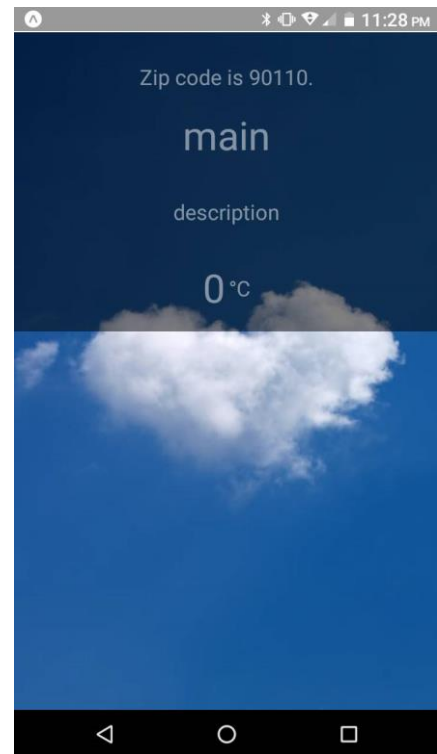
- flex – เป็นตัวเลข น้ำหนักในการแบ่งพื้นที่ เช่นถ้าคอมโพเนนต์ A มี flex เท่ากับ 1, คอมโพเนนต์ B มี flex เท่ากับ 2 หมายความว่า B จะใช้พื้นที่มากกว่า A สองเท่า และถ้ามีคอมโพเนนต์ A เพียง คอมโพเนนต์เดียว จะใช้เต็มพื้นที่
- flexDirection – แกนหลักของ Layout ว่าคอมโพเนนต์ลูกควรจัดเรียงแนวนอน (row) และแนวตั้ง (column) โดยค่า default คือ column
- justifyContent – การกระจายตัวของคอมโพเนนต์ลูกว่าควรจะเป็นแบบใด ในแนวนอนเดียวกับ flexDirection
- alignItems – การกระจายตัวของคอมโพเนนต์ลูกว่าควรจะเป็นแบบใด ในแนวแกนกับ flexDirection

ต.ย.

flexDirection: 'row'	flexDirection: 'row'	flexDirection: 'row'	flexDirection: 'row'
justifyContent: 'space-between'	justifyContent: 'center'	justifyContent: 'space-evenly'	justifyContent: 'flex-end'
			
flexDirection: 'column'	flexDirection: 'column'	flexDirection: 'row',	flexDirection: 'column'
justifyContent: 'center'	justifyContent: 'center'	justifyContent: 'center'	justifyContent: 'flex-end'
alignItems: 'center'	alignItems: 'flex-end'	alignItems: 'flex-end'	alignItems: 'flex-start'
			CHECK POINT #2

CHECK POINT #3

จัด Layout โดยใช้ Flex Box และปรับสไตล์ของตัวอักษรและ background
เพิ่มเติม โดยใช้ height, paddingRight, backgroundColor, opacity,
fontSize, color, textAlign

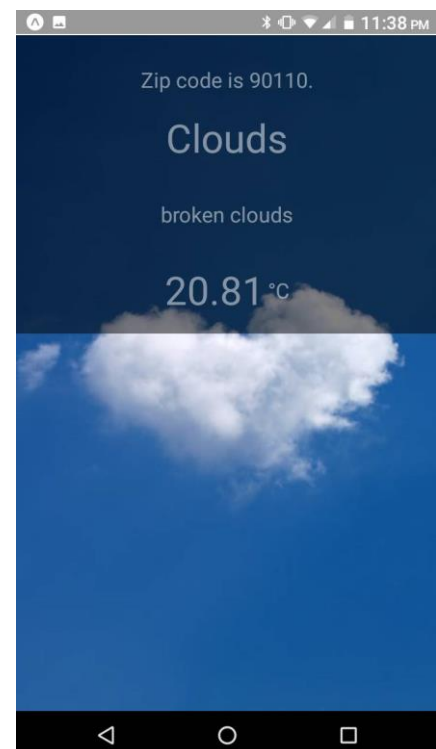


5. Connect

เพิ่ม componentDidMount ลงไปในคอมโพเนนต์ Weather (อาจารย์ควบคุมแล็บจะเป็นผู้แจ้ง APPID)

```
fetchData = () => {
  fetch(`http://api.openweathermap.org/data/2.5/weather?q=${this.props.zipCode},th&units=metric&APPID=...`)
    .then((response) => response.json())
    .then((json) => {
      this.setState({
        forecast: {
          main: json.weather[0].main,
          description: json.weather[0].description,
          temp: json.main.temp
        }
      });
    })
    .catch((error) => {
      console.warn(error);
    });
}

componentDidMount = () => this.fetchData()
```



CHECK POINT #4: แสดงผลการทำงาน

6. Router

Application ที่สมบูรณ์มักมีหน้าจอ UI (สกรีน) มากกว่า 1 หน้าจอ ในการสลับหน้าจอไปมา สามารถทำได้ผ่านการใช้ Navigation Library ซึ่งไลบรารีที่ถือเป็น Official จาก React คือ react-navigation

```
>> yarn add react-navigation
```

เพิ่มคอมโพเนนต์ WeatherScreen สำหรับหน้าจอแสดงคอมโพเนนต์ Weather

```
export default class WeatherScreen extends React.Component {
  static navigationOptions = ({navigation}) => {
    return {
      headerTitle: (<Text>Weather</Text>),
    }
  }

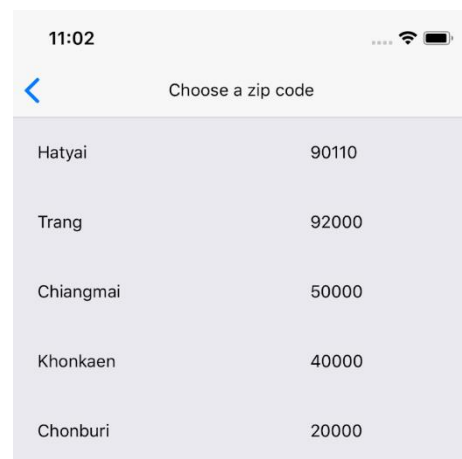
  render() {
    return (
      <Weather zipCode="90110"/>
    );
  }
}
```

เพิ่มคอมโพเนนต์ ZipCodeScreen ซึ่งเป็นหน้าจอสำหรับเลือกรหัสไปรษณีย์ (zip code) จากรายการที่กำหนดไว้ ทั้งนี้ นักศึกษาจะต้องกำหนด Style ให้เหมาะสมด้วยตนเอง

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

const availableZipItems = [
  { place: 'Hatyai', code: '90110' },
  { place: 'Trang', code: '92000' },
  { place: 'Chiangmai', code: '50000' },
  { place: 'Khonkaen', code: '40000' },
  { place: 'Chonburi', code: '20000' },
]

const ZipItem = ({place, code, navigate}) => (
  <View style={styles.zipItem}>
    <Text style={styles.zipPlace}>{place}</Text>
    <Text style={styles.zipCode}>{code}</Text>
  </View>
)
```



```

const _keyExtractor = item => item.code

export default class WeatherScreen extends React.Component {
  static navigationOptions = ({navigation}) => {
    return {
      headerTitle: (<Text>Choose a zip code</Text>),
    }
  }
  render() {
    const { navigate } = this.props.navigation;
    return (
      <View>
        <FlatList
          data={availableZipItems}
          keyExtractor={_keyExtractor}
          renderItem={({item}) => <ZipItem {...item} navigate={navigate}/>}
        />
      </View>
    );
  }
}

```

ปรับ App.js ให้ render ผลลัพธ์จากไลบรารี react-navigation แทนการ render คอมโพเนนต์ Weather โดยตรง

```

import { createStackNavigator } from 'react-navigation';

const RootStack = createStackNavigator({
  Weather: WeatherScreen,
  ZipCode: ZipCodeScreen
},{
  initialRouteName: 'Weather',
})
export default class App extends React.Component {
  render() {
    return (
      <RootStack/>
    );
  }
}

```

หมายเหตุ นักศึกษาสามารถทดลองเปลี่ยน initialRouteName จาก 'Weather' ไปเป็น 'ZipCode' เพื่อทดสอบการแสดงผลหน้า ZipCode

CHECK POINT #5: แสดงผลการทำงาน

Route Parameter

ในการทำงานร่วมกันของแต่ละหน้าจอ UI เราสามารถส่งผ่านค่าการทำงานได้โดยใช้ Route Parameter

ปรับให้ WeatherScreen รับ zip code จาก Route Parameter

แก้ไขไฟล์ App.js ให้ส่งผ่าน 90110 เป็น default route parameter

```
const RootStack = createStackNavigator({
  Weather: WeatherScreen,
  ZipCode: ZipCodeScreen
},{
  initialRouteName: 'Weather',
  initialRouteParams: {zipCode: '90110'}
})
```

แก้ไข WeatherScreen.js เพื่อรับ zip code ผ่านทาง route parameter

```
render() {
  const zipCode = this.props.navigation.getParam('zipCode')
  return (<Weather zipCode={zipCode}/>);
}
```

ปรับ Weather ให้รองรับการเปลี่ยน props โดยการเพิ่ม

ComponentDidUpdate

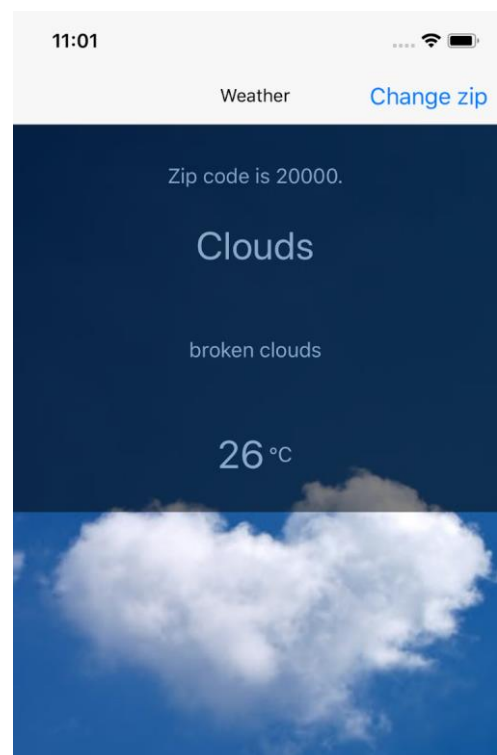
```
componentDidUpdate = (prevProps) => {
  if (prevProps.zipCode !== this.props.zipCode) {
    this.fetchData()
  }
}
```

Navigation

เพิ่มปุ่มในหน้า WeatherScreen เพื่อให้ลิงค์ไปยังหน้า ZipCodeScreen

โดยการปรับเพิ่ม navigationOptions ในไฟล์ WeatherScreen.js

```
static navigationOptions = ({navigation}) => {
  return {
    headerTitle: (<Text>Weather</Text>),
    headerRight: (
      <Button title="Change zip" onPress={() => navigation.navigate('ZipCode')} />
    )
  }
}
```



ในหน้า ZipCodeScreen เพิ่มฟังก์ชันให้สามารถเลือก zip code เพื่อเปลี่ยนรหัสไปรษณีย์ที่จะแสดงสภาพภูมิอากาศของรหัสไปรษณีย์ที่เลือก

```
const ZipItem = ({place, code, navigate}) => (
  <TouchableHighlight onPress={() => navigate('Weather', {zipCode: code})}>
    <View style={styles.zipItem}>
      <Text style={styles.zipPlace}>{place}</Text>
      <Text style={styles.zipCode}>{code}</Text>
    </View>
  </TouchableHighlight>
)
```

CHECK POINT #6

แสดงผลการทำงาน และตอบคำถามของ TA เพื่อทวนสอบความเข้าใจในเนื้อหา

7. Eject

ในการทดลองตอนก่อนหน้า เราพัฒนาฟังก์ชันต่างๆ แล้วทดลองการทำงานด้วย Expo อย่างไรก็ตาม หากมีความต้องการในการแก้ไขส่วนที่เป็น Native หรือใช้ไลบรารีส่วนอื่นที่เป็น Native เราจะต้องใช้โครงสร้างโปรเจกต์ที่เป็น Native ของ Android และ iOS โดยใช้คำสั่ง (กระบวนการนี้ เมื่อทำแล้ว จะไม่สามารถรันโปรแกรมบน Expo ได้อีก และย้อนกลับไม่ได้)

>> yarn run eject

```
Ejecting is permanent! Please be careful with your selection.
? How would you like to eject from create-react-native-app? (Use arrow keys)
> React Native: I'd like a regular React Native project.
```

ทดลองรันโปรแกรมบนสมาร์ตโฟน Android โดยการรันโปรแกรมด้วย Android Studio

ทั้งนี้ในระหว่างการรันโปรแกรมผ่าน Android Studio เรายังคงจำเป็นต้องเปิด react native server ทิ้งไว้ (ด้วยคำสั่ง yarn start เหมือนเดิม)

เลือก Open an existing Android Studio Project แล้วเลือกไฟล์เดอร์ชื่อ Android ในโปรเจกต์ แล้วกดรัน (อาจจะรันบนสมาร์ตโฟน Android จริง หรือใช้ Android Virtual Device)

