REACT NATIVE

Developed by Alabian Solutions Ltd

CHAPTER 1

Introduction

React Native is a JavaScript framework used for building native mobile application for android and iOS. It uses React framework which is library of JavaScript to build amazing user interfaces.

React Native apps are applications built with react native which involves the use of reactjs and JSX (XML-esque markup) for developing the user interface.

Brief history of React Native

React Native is a relatively new technology and was officially available since March 2015, having been in private beta since the start of that year and internally at facebook.

It certainly seems like React Native has a bright future looking forward. As its community continues to grow and new updates gets released.

Prerequisites to Learning React Native

You should be familiar with the following :

- React js
- Flexbox
- Keen approach to JavaScript

Basic Concept

React Native lets you build mobile apps using only JavaScript. It uses the same design as React, letting you compose a rich mobile UI from declarative components. With React Native, you don't build a mobile web app, an HTML5 app, or a hybrid app; you build a real mobile app that's indistinguishable from an app built using Objective-C or Java. React Native uses the same fundamental UI building blocks as regular iOS and Android apps. You just put those building blocks together using JavaScript and React. *(Official docs)*
React Native could be classified as something from the third category. For the iOS and Android versions, React Native uses JavaScript Core under the hood, which is the default JavaScript engine on iOS.

ADVANTAGES OF REACT NATIVE

- **JavaScript**: Easy transitioning from a JavaScript web developer to a mobile developer. Code are written in JavaScript but rendered as native codes.
- **Cross-Platform Usage:** By its interaction with native code, it is platform agonistic and works well for both Android and iOS devices with the rule of thumb: *“learn once and write anywhere”*.
- **Hot Reloading:** with the concept of virtual Dom in react, a change made in code is immediately rendered during development.

Limitation OF REACT NATIVE

Following are the limitations of React Native:

- **Native Components –** If you want to create native functionality which is not created yet, you will need to write some platform specific code.
- **Managing Modules –** Due to its implementation with dependencies, keeping track of modules that are dependent on the app can be a little overwhelming.

Keeping Them Separate

CHAPTER 2

React Native - Environment Setup

React Native uses Node.js, a JavaScript runtime, to build your JavaScript code. React Native also requires a recent version of the Java SE Development Kit (JDK) to run on Android. Follow the instructions for your system to make sure you install the required versions.

MacOS

First install Homebrew using the instructions on the Homebrew website. Then install Node.js by executing the following in Terminal:

```
brew install node
```

Next, use `homebrew` to install watchman, a file watcher from Facebook, used to figure out when your code changes and rebuild accordingly. It's like having Android Studio do a build each time you save your file. :

```
brew install watchman
```

Finally, download and install JDK 8 or newer if needed.

Windows

Install Node.js if you don't have it. While install Node, check the button that would help install Chocolatey and python2

Linux

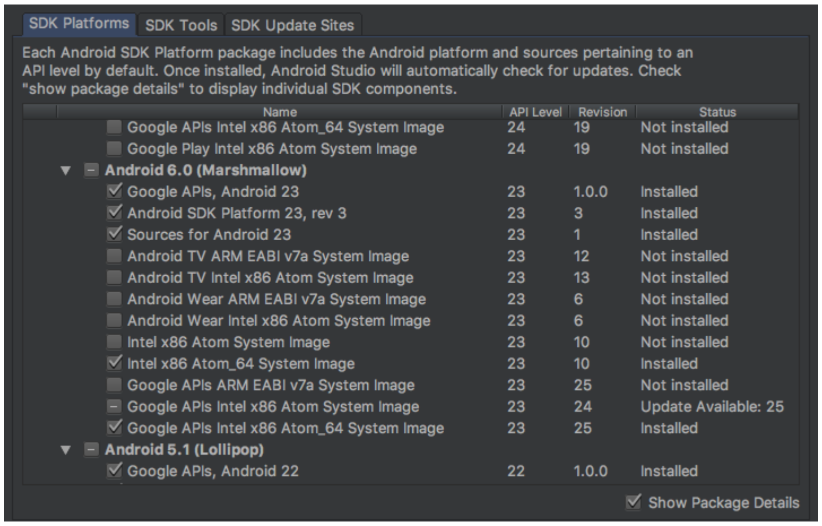
Install Node.js by following the installation instructions for your Linux distribution. You will want to install Node.js version 6 or newer.
Finally, download and install JDK 8 or newer if needed.

Android Studio

Android studio is an IDE used to develop mobile apps for android.
Its SDKs allow you to create **Android Virtual Devices** (AVDs) according to your personal configuration for the purpose of testing your app.
But for IOS we use **XCode** as an environment to develop mobile apps for IOS.

To set up Android Studio for react-native mobile app development visit https://facebook.github.io/react-native/docs/getting-started

Note: Make sure you are able to start your emulator
Make sure that the following items are checked:
Google APIs, Android 23
Android SDK Platform 23
Intel x86 Atom_64 System Image
Google APIs Intel x86 Atom_64 System Image



SDK Platforms | SDK Tools | SDK Update Sites

Each Android SDK Platform package includes the Android platform and sources pertaining to an API level by default. Once installed, Android Studio will automatically check for updates. Check "show package details" to display individual SDK components.

Name	API Level	Revision	Status
Google APIs Intel x86 Atom_64 System Image	24	19	Not installed
Google Play Intel x86 Atom System Image	24	19	Not installed
▼ – **Android 6.0 (Marshmallow)**			
✓ Google APIs, Android 23	23	1.0.0	Installed
✓ Android SDK Platform 23, rev 3	23	3	Installed
✓ Sources for Android 23	23	1	Installed
Android TV ARM EABI v7a System Image	23	12	Not installed
Android TV Intel x86 Atom System Image	23	13	Not installed
Android Wear ARM EABI v7a System Image	23	6	Not installed
Android Wear Intel x86 Atom System Image	23	6	Not installed
Intel x86 Atom System Image	23	10	Not installed
✓ Intel x86 Atom_64 System Image	23	10	Installed
Google APIs ARM EABI v7a System Image	23	25	Not installed
– Google APIs Intel x86 Atom System Image	23	24	Update Available: 25
✓ Google APIs Intel x86 Atom_64 System Image	23	25	Installed
▼ – **Android 5.1 (Lollipop)**			
✓ Google APIs, Android 22	22	1.0.0	Installed

✓ Show Package Details

Next, select **SDK Tools** and check **Show Package Details.** Expand **Android SDK Build-Tools** and make sure `23.0.1 and higher` is selected.
Finally, tap **Apply** to install your selections.
When the Android components are finished installing, create a new emulator running SDK Platform 23.

Create Project

To create a react native project, we can either go through with the bare workflow using React-native-CLI by running **npm i –g react-native-cli** or using expo by running **npm i –g expo-cli** to get a quick setup of react-native.

Expo is a framework and a platform for universal React applications. It is a set of tools and services built around React Native and native platforms that help you develop, build, deploy, and quickly iterate on iOS, Android, and web apps from the same JavaScript/TypeScript codebase. https://docs.expo.io/versions/latest/

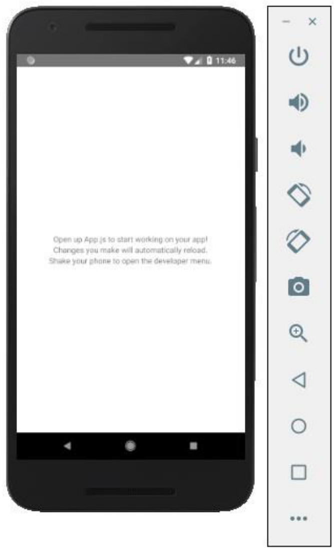
Note: you must have a git account to use expo

To create a project using expo in your command-line, run:

expo init projectName

cd projectName

expo start

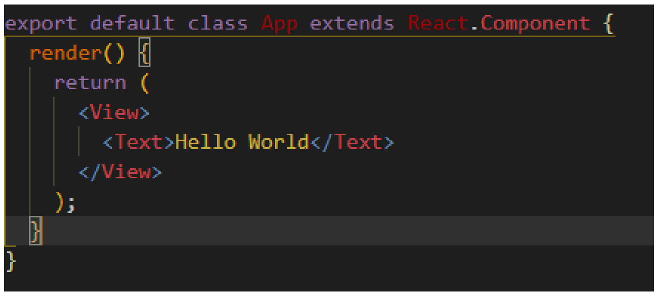


Open up App.js to start working on your app!
Changes you make will automatically reload.
Shake your phone to open the developer menu.

If your emulator is not displaying this, in your command line run the command **a** to view on your emulator. You can also view on your android phone by installing expo from google playstore and using the QR code.

Hello world

To display a simple message saying "Hello World" remove the CSS part and insert the message to be printed wrapped by the <Text></Text> tags inside <View></View> as shown below.



```
export default class App extends React.Component {
  render() {
    return (
      <View>
        <Text>Hello World</Text>
      </View>
    );
  }
}
```

Course content

View

Text

Nested Component

State and props

Image

Flexbox

Buttons

BirthdayApp

TextInput/ Animations (Birthday app landing page)

Navigation

ScrollView

List

Listing Cars App

Modal

WebView

Redux – use redux to complete birthday app

HTTP – Weather App

Permissions

Project- NewsApp

CHAPTER 3

Concepts – Native Components Api

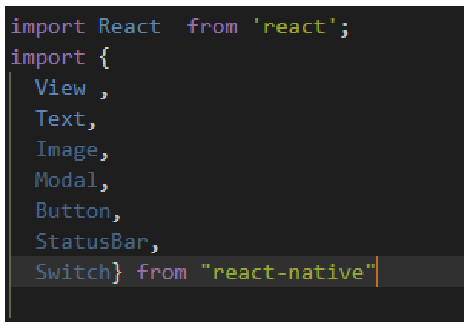
Intro

React-native provides us with native component api (JSX) that would help us connect with the native codes to build awesome UI.
Examples include:

1. View
2. Text
3. Image
4. WebView
5. Button
6. Modal
7. StatusBar
8. Switch
9. ActivityIndicator
10. Alert
11. AsyncStorage
12. Geolocation

Components can be categorized into six categories:
Basic or Core components: View, Text, Image, ScrollView, TextInput, StyleSheet
List components: FlatList and SectionList
User Interface or Form Control components Picker, Slider, Button, Switch
Android Specific components: DatePickerAndroid, TimePickerAndroid, ViewPagerAndroid, ToastAndroid, PermissionAndroid
iOS Specific Component : AlertIOS, pushNotificationsIOS, etc.
Others: Alert, Animated, CameraRoll, Dimensions, Clipboard, StatusBar, Linking, Keyboard, ActivityIndicator, WebView, Modal, e.tc

All native api must be imported from react-native module before implementation.



```
import React  from 'react';
import {
  View ,
  Text,
  Image,
  Modal,
  Button,
  StatusBar,
  Switch} from "react-native"
```

View

View is the most common element in React Native. You can consider it as an equivalent of the **div** element used in web development.

Some few common use cases:

- When you need to wrap your elements inside the container, you can use **View** as a container element.
- When you want to nest more elements inside the parent element, both parent and child can be **View**. It can have as many children as you want.
- When you want to style different elements, you can place them inside **View** since it supports **style** property, **flexbox** etc.
- **View** also supports synthetic touch events, which can be useful for different purposes.

Text

The <Text> component is used to render text in the application. It can be nested and can inherit properties from parent to child. It can be related with the <span> tag on the web.

```
const App = ()=>(
   <View>
      <Text>I am a Text</Text>
   </View>
);
```

Image

Nested Component

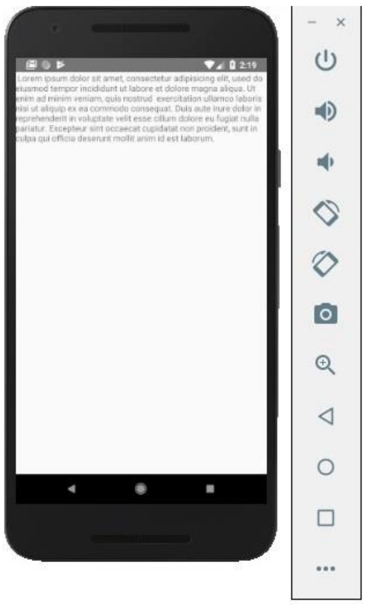
These are custom components stacked within another component which are either imported or created in the same file.

```
const App = ()=>(
    <View>
        <Text>I am a Text</Text>
        <NestedComponent />
    </View>
);

const NestedComponent = ()=>(
    <View>
        <Text>I am a Nested Component</Text>
    </View>
);
```

State And Props

The data inside React Components is managed by state and props. **State** is the data generated and used by a component which can be passed to a nested component as **props.**
Props are immutable properties of a component unlike state which is mutable.

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

export default class App extends Component {
   state = {
      data: "Lorem ipsum dolor sit amet, consectetur adipisicing elit, used do eius
modtempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, qui
snostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat Duis
aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat n
ulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in"
   }
   render() {
      return (
      <View>
         <Text> {this.state.data} </Text>
      </View>
      );
   }
}
```



Lorem ipsum dolor sit amet, consectetur adipisicing elit, used do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Passing State as Props

From our understanding of react, there are two types of container component:

a. Presentational component
b. Stateful component

Presentational components are used only for presenting view to the users. They are known as stateless component; hence they receive all their data and function as **props**

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

export default class App extends React.Component {
   state = {
       name: "My name is Alabian"
   }
   render() {
      return (
      <View>
        <PresentationalComponent data={this.state.name} />
      </View>
      );
   }
}

const PresentationalComponent = (props)=>(
    <Text>{props.data}</Text>
)
```

Styling Components

React native provides an api called StyleSheet which uses a method called create() to create styling in the form of javascript object. It allows the naming of CSS properties and values but in the form of camel casing. Although not all CSS properties are available at the moment.

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

const PresentationalComponent = (props) => {
   return (
      <View>
         <Text style = {styles.myState}>
            {props.name}
         </Text>
      </View>
   )
}
export default PresentationalComponent

const styles = StyleSheet.create ({
   myState: {
      marginTop: 20,
      textAlign: 'center',
      color: 'blue',
      fontWeight: 'bold',
      fontSize: 20
   }
})
```

Image

`Image` is used to render images which may either be bundled with the app as assets, or downloaded from the web.

To bundle an image in the app, `require()` the image file just as you would any other file. When loading bundled images, the same images are used to render on both iOS and Android.

App.js

```
import React from 'react';
import ImagesExample from './ImagesExample.js'

const App = () => {
   return (
      <ImagesExample />
   )
}
export default App
```

For bundled images

imageExample.js

```
import React, { Component } from 'react'
import { Image } from 'react-native'

const ImagesExample = () => (
   <Image source = {require('../assets/logo.png')} style={styles.image}
/>
)
export default ImagesExample
```

```
const styles = StyleSheet.create({
  image: {
    width: 200,
    height: 200
  }
});
```

For Network Images

The dimensions aren't known till they are downloaded, hence you must specify dimensions in the style prop, e.g. style={{width: 100, height: 100}}

imageExample.js

```
import React, { Component } from 'react'
import { View, Image } from 'react-native'

const ImagesExample = () => (
   <Image source =
{{uri:'https://pbs.twimg.com/profile_images/486929358120964097/gNLINY67_
400x400.png'}}
   style = {{ width: 200, height: 200 }}
   />
)
export default ImagesExample
```

Chapter 4

Flexbox

Flexbox lay outing helps to responsively layout our components.
To achieve the desired layout, flexbox offers three main properties – flexDirection justifyContent and alignItems.

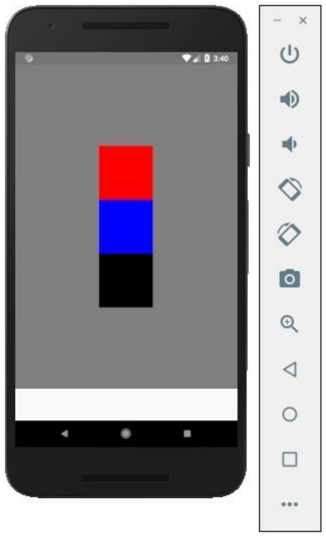
Property	Values	Description
flexDirection	'column', 'row'	Used to specify if elements will be aligned vertically or horizontally. Default is column which is
justifyContent	'center', 'flex-start', 'flex-end', 'space-around', 'space-between'	Used to determine how the child components should be distributed inside the container, along the horizontal plane
alignItems	'center', 'flex-start', 'flex-end', 'stretch'	Used to determine how the child components should be distributed inside the container along the secondary axis (opposite of flexDirection) i.e along the vertical plane

App.js

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

const Home = (props) => {
   return (
      <View style = {styles.container}>
         <View style = {styles.redbox} />
         <View style = {styles.bluebox} />
         <View style = {styles.blackbox} />
      </View>
   )
}

export default Home
```

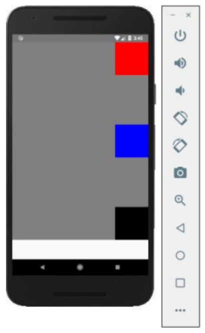
```
const styles = StyleSheet.create ({
   container: {
      flexDirection: 'column',
      justifyContent: 'center',
      alignItems: 'center',
      backgroundColor: 'grey',
      height: 600
   },
   redbox: {
      width: 100,
      height: 100,
      backgroundColor: 'red'
   },
   bluebox: {
      width: 100,
      height: 100,
      backgroundColor: 'blue'
   },
   blackbox: {
      width: 100,
      height: 100,
      backgroundColor: 'black'
   },
})
```

Output



Exercise

Achieve the layout below using flexbox and others(todo)



Chapter 5

Touchable Component & TextInput

React Native provides some touchable components which offer built in animations and props for handling touch event such as onPress.
E.g: Button, TouchableOpacity, TouchableWithoutFeedBack, TouchableHighlight, e.t.c.
These components must be imported from react-native into the app.

```
import React, { Component } from "react";
import { Button, TouchableOpacity } from "react-native";

const App = () => {
  const handlePress = () => false;
  return <Button onPress={handlePress} title="Red button!" color="red" />;
};


const Touch = ()=>{
    const showAlert = ()=>Alert.alert('I am an alert');
    return(
        <TouchableOpacity onPress={()=>showAlert()}>
            <Text>Click Me</Text>
        </TouchableOpacity>
    )
}
export default App;
```

TouchableOpacity - change the opacity of an element when touched.

TouchableHighlight - When a user presses the element, it will get darker and the underlying color will show through.
TouchableWithoutFeedback - This should be used when you want to handle the touch event without any animation usually, this component is not used much.

TextInput

These are component for putting information into the app such as updating the state using its props. Some major props include:

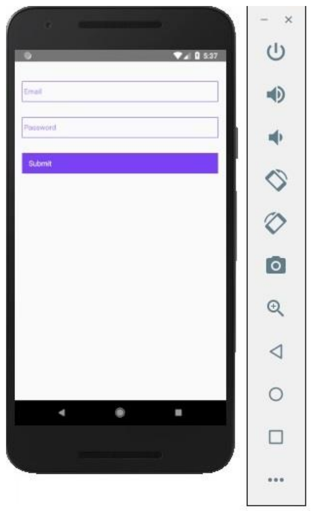
a. onChangeText – used to update a state
b. onFocus – action taken when text input is clicked
c. value – value of the textInput.

App.js

```
import React from 'react';
import Inputs from './inputs.js'

const App = () => {
   return (
      <Inputs />
   )
}
export default App
```

inputs.js

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

class Inputs extends Component {
  state = {
    email: '',
    password: ''
  }
  handleEmail = (text) => {
    this.setState({ email: text })
  }
  handlePassword = (text) => {
    this.setState({ password: text })
  }
  login = (email, pass) => {
    alert('email: ' + email + ' password: ' + pass)
  }
  render() {
    return (
      <View style = {styles.container}>
        <TextInput style = {styles.input}
          underlineColorAndroid = "transparent"
          placeholder = "Email"
          placeholderTextColor = "#9a73ef"
          autoCapitalize = "none"
          onChangeText = {this.handleEmail}/>

        <TextInput style = {styles.input}
          underlineColorAndroid = "transparent"
          placeholder = "Password"
          placeholderTextColor = "#9a73ef"
          autoCapitalize = "none"
          onChangeText = {this.handlePassword}/>

        <TouchableOpacity
          style = {styles.submitButton}
          onPress = {
            () => this.login(this.state.email, this.state.password)
          }>
          <Text style = {styles.submitButtonText}> Submit </Text>
        </TouchableOpacity>
      </View>
    )
  }
}
export default Inputs

const styles = StyleSheet.create({
  container: {
    paddingTop: 23
  },
  input: {
    margin: 15,
    height: 40,
    borderColor: '#7a42f4',
    borderWidth: 1
  },
  submitButton: {
    backgroundColor: '#7a42f4',
    padding: 10,
    margin: 15,
    height: 40,
  },
  submitButtonText:{
    color: 'white'
  }
})
```

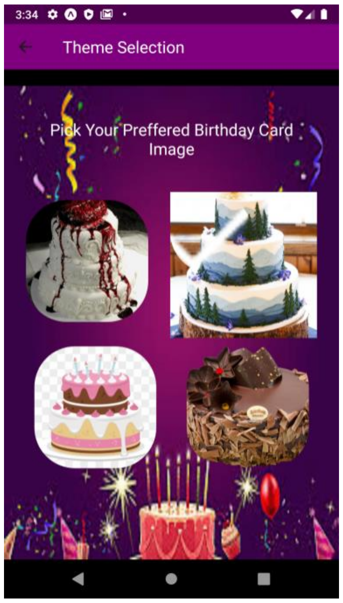


Exercise

Create a Registration screen with name, email, username, phone, date of birth, password

Chapter 6

MINI PROJECT – Birthday App



Module 2

Chapter 1

ScrollView & List

In building a mobile app, there certain times there would be need to scroll through content of a screen, react native provides native components to achieve this :

a. ScrollView
b. List Component
   i. FlatList
   ii. SectionList

Use Cases

ScrollView is used when the amount of content to scrolled through is small
List component is preffered when there is large amount of data to be scrolled through. This is preffered to scrollview because it prevents **memory leakage (**`ScrollView` renders all its react child components at once**).**

`FlatList` is also handy if you want to render separators between your items, multiple columns, infinite scroll loading, etc

Visit https://facebook.github.io/react-native/docs/scrollview for more info on its properties.

App.js

```
import React from 'react';
import ScrollView from './scrollview.js';

const App = () => {
  return (
    <ScrollView  />
  )
}
export default App
```

```
ScrollView.js:
import React, { Component } from 'react';
import { Text, Image, View, StyleSheet, ScrollView } from 'react-native';
 import styles from  './styles'
class ScrollViewExample extends Component {
  state = {
    names: [
      {'name': 'Ben', 'id': 1},
      {'name': 'Susan', 'id': 2},
      {'name': 'Robert', 'id': 3},
      {'name': 'Mary', 'id': 4},
      {'name': 'Daniel', 'id': 5},
      {'name': 'Laura', 'id': 6},
      {'name': 'John', 'id': 7},
      {'name': 'Debra', 'id': 8},
      ]
}
  render() {
    return (
      <View>
        <ScrollView>
          {             this.state.names.map((item, index) => (
            <View key = {item.id} style = {styles.item}>
              <Text>{item.name}</Text>
            </View>
          ))
          }
        </ScrollView>
      </View>
    )
  }
}
export default ScrollViewExample
```

styles.js

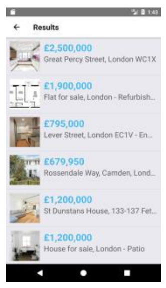
```
const styles = StyleSheet.create ({
  item: {
    flexDirection: 'row',
    justifyContent: 'space-between',
    alignItems: 'center',
    padding: 30,
    margin: 2,
    borderColor: '#2a4944',
    borderWidth: 1,
    backgroundColor: '#d2f7f1'
  }
})
export default styles
```

USING FLATLIST

```
<FlatList
   data = {this.state.names}
   renderItem={({item})=>(
            <View style={styles.item} key={item.id}>
              <Text>{item.name}</Text>
            </View>
            )}
   keyExtractor = {(id)=>id}
 />
```

EXERCISE

Create a list of cars and its description



£2,500,000
Great Percy Street, London WC1X

£1,900,000
Flat for sale, London - Refurbish...

£795,000
Lever Street, London EC1V - En...

£679,950
Rossendale Way, Camden, Lond...

£1,200,000
St Dunstans House, 133-137 Fet...

£1,200,000
House for sale, London - Patio

Modal & WebView

Modal

The Modal component is a basic way to present content above an enclosing view.

Visit https://facebook.github.io/react-native/docs/modal to see more props available for use

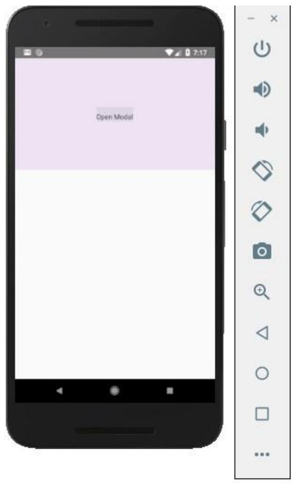
App.js

```
import React, { Component } from 'react'
import ModalExample from './modalExample.js'

const Home = () => {
   return (
      <ModalExample/>
   )
}
export default Home;
```

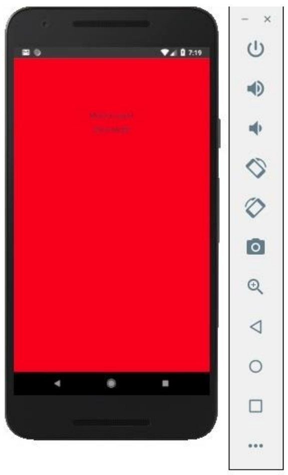
modalExample.js

```
import React, { Component } from 'react';
import { Modal, Text, TouchableHighlight, View, StyleSheet}

from 'react-native'
class ModalExample extends Component {
   state = {
      modalVisible: false,
   }
   toggleModal(visible) {
      this.setState({ modalVisible: visible });
   }
   render() {
      return (
         <View style = {styles.container}>
            <Modal animationType = {"slide"} transparent = {false}
               visible = {this.state.modalVisible}
               onRequestClose = {() => { console.log("Modal has been
closed.") } }>

               <View style = {styles.modal}>
                  <Text style = {styles.text}>Modal is open!</Text>

                  <TouchableHighlight onPress = {() => {
                     this.toggleModal(!this.state.modalVisible)}}>

                     <Text style = {styles.text}>Close Modal</Text>
                  </TouchableHighlight>
               </View>
            </Modal>

            <TouchableHighlight onPress = {() =>
{this.toggleModal(true)}}>
               <Text style = {styles.text}>Open Modal</Text>
            </TouchableHighlight>
         </View>
      )
   }
}
export default ModalExample

const styles = StyleSheet.create ({
   container: {
      alignItems: 'center',
      backgroundColor: '#ede3f2',
      padding: 100
   },
   modal: {
      flex: 1,
      alignItems: 'center',
      backgroundColor: '#f7021a',
      padding: 100
   },
   text: {
      color: '#3f2949',
      marginTop: 10
   }
})
```

Initial screen



opened modal



WebView
The WebView component is used when you want to render web page to your mobile app inline. It has a property of source that receives a url link to be loaded

App.js

```
import React, { Component } from 'react'
import WebViewExample from './webviewExample.js'

const App = () => {
   return (
      <WebViewExample/>
   )
}
export default App;
```

webviewExample.js

```
import React, { Component } from 'react'
import { View, WebView, StyleSheet }

from 'react-native'
const WebViewExample = () => {
   return (
      <View style = {styles.container}>
         <WebView
         source = {{ uri:
         'https://www.alabiansolutions.com’ }}
         />
      </View>
   )
}
export default WebViewExample;

const styles = StyleSheet.create({
   container: {
      height: 350,
   }
})
```

Chapter 2

React Navigation & Redux

React Navigation is responsible for the navigation through different screens of our app Due to it being frequently updated, we would be going through the docs

Redux

Redux is a state manager which provides a single source of truth for the state of the app.

Composition

a. Store: houses the state of the app.
b. Reducers : modifies the store based on the action dispatched to the store
c. Actions and Action creators : interactions by users on the app interface e.g C.R.U.D

Folder Setup

- Create a folder called redux
- In the redux folder create store.js , reducers folder and actions folder
- In the actions folder create index.js and types.js
- In the reducers folder create index.js
- In your command line run ***expo install redux react-redux logger redux-thunk***

Redux Component

- **<Provider> a component from react-redux that makes available the store to connected component**
- **Connect() an higher order component that connects a component to the store.**
- **Connect() receives two main parameters mapStateToProps and mapDispatchToProps**

Store

The store is created by a method from redux called createStore(), which accepts a root reducer as a parameter and any store enhancers called middleware.
E.g Thunk from redux-thunk, enhances the store by halting the dispatch action to allow any asynchronous call in the process of creating an action by action creators

Reducers

The reducers are pure functions that responsible for receiving actions to manipulate the state and update the store. They serve as the middle man between the store and actions created. It receives two parameters: **the initial state and actions.**

Once there are several reducers updating the store, the reducers have to be combined together as one before it is sent into the store using **combineReducers** from redux;

This turns an object whose values are different reducer functions, into a single reducer function. It will call every child reducer, and gather their results into a single state object, whose keys correspond to the keys of the passed reducer functions.

i.e

combineReducers({reducer1, reducer2, ... })

Actions & ActionCreators

Actions are object with keys: type and payload, which are dispatched to specific reducers to update the store. ActionCreators are a function that dispatches an object of type and payload.

i.e

```
const anAction = {
  type: "SIGN_IN",
  payload: userDetails
}


const ActionCreator = (userDetails)=>{
```

```
  return (dispatch)=>dispatch({
    type:"SIGN_IN",
    payload: userDetails
  })
}
```

Store.js

```
import { createStore, applyMiddleware } from "redux";
import reducers from '../reducers';
import logger from 'redux-logger';
import thunk from ‘redux-thunk’;

let middleware = [thunk];
if(process.env.NODE_ENV == 'development')middleware.push(logger);
const store = createStore(reducers, applyMiddleware(...middleware));

export default store;
```

reducers/ index.js

```
import { combineReducers } from "redux";
import ThemeReducer from "./ThemeReducer";
import UserReducer from "./UserReducer";

export default combineReducers({
  Theme: ThemeReducer,
  Users: UserReducer
});
```

actions/types.js

```
export const BIRTHDAY_USER = "BIRTHDAY_USER";
export const BACKGROUND_UPDATE = "BACKGROUND_UPDATE";
```

actions/UserActions.js

```
import { BIRTHDAY_USER } from "./type"

export const birthDayUser = (name)=>{

    return dispatch=>dispatch({
        type:BIRTHDAY_USER,
        payload:name
    })
}
```

reducers/UserReducers.js

```
import { BIRTHDAY_USER } from "../actions/type";

let init = {
  sender: false,
  reciever:false,
  error:false
}
const UserReducer = (state = init, action) => {
  const {payload, type} = action;
  switch (type) {
    case BIRTHDAY_USER:
      // console.log(payload)
      return {
        ...state,
        ...payload
      }
    default:
      return state;
  }
  return state;
};
export default UserReducer;
```

screens/Home.js

```
import React, { Component } from 'react'
import { Text, View, Image, ImageBackground, Button, Alert } from 'react-native'
import Container from '../components/Container'
import TextInput from "../components/TextInput/TextInput";

//redux
import { connect } from 'react-redux'
import { birthDayUser } from '../redux/actions/UserActions';

class Home extends Component {
  state = {
    sender: "",
    reciever:"",
    error:false
  }
  handleCard = async()=>{
    if(!this.state.sender || !this.state.reciever){
      this.setState({
        ...this.state,
        error:"The name fields are required"
      })
      return;
    }
    console.log(this.state)
    this.props.createCeleb(this.state);
    this.setState({
      sender:"",
      reciever:"",
      error:false
    })
    this.props.navigation.navigate("Theme");
  }
    render() {
      // console.log(this.state, this.props.users);

        return (
          <Container src={require("../../assets/back.png")}>
```

```
            <View
              style={{
                // backgroundColor: "#ffff",
                width: 200,
                height: 200,
                borderRadius: 50,
                alignItems: "center",
                justifyContent: "center"
              }}
            >
              <Image
                source={require("../../assets/back2.png")}
                style={{ width: 150, height: 150, borderRadius: 20 }}
              />
            </View>
            <View
              style={{
                alignItems: "center",
                justifyContent: "center",
                width: 300
              }}
            >
              <Text
                style={{
                  textAlign: "center",
                  fontSize: 20,
                  color: "brown",
                  fontWeight: "bold"
                }}
              >
                Create A Birthday Card for a loved one
              </Text>
              <TextInput
                value={this.state.sender}
                placeholder="Enter sender name"
                onChangeText={sender => this.setState({ sender, error: false })}
              />
              <TextInput
                value={this.state.reciever}
                placeholder="Enter Birthday Name"
                onChangeText={reciever =>
                  this.setState({ reciever, error: false })
                }
              />

              {this.state.error ? (
                <Text
                  style={{ fontStyle: "italic", fontSize: 20, color: "red" }}
                >
                  {this.state.error}
                </Text>
              ) : null}
            </View>
            <View>
              <Button
                color="green"
                title="Create Card"
                onPress={() => this.handleCard()}
              />
            </View>
          </Container>
        );
    }
}
const mapDispatchToProps = dispatch => {
  return{
    createCeleb: (users) => dispatch(birthDayUser(users))
  }
};
const mapStateToProps= state=>({
  users: state.Users

})
export default connect(mapStateToProps, mapDispatchToProps)(Home);
```

App.js

```
import React from "react";

import Home from './screens/Home';

//redux
import store from './redux/store'
import {Provider} from 'react-redux'

export default () => (
    <Provider store={store}>
        <Home />
    </Provider>
);
```

Exercise

Integrate redux into your birthday App

Chapter 3

Networking and Permissions

In this chapter we will be learning how to consume api using axios.
In your command line run *expo install axios*
We will use the **componentDidMount lifecycle** method to load the data from server as soon as the component is mounted. This function will send a request to the server, return JSON data, log output to console and update our state.
Visit https://api.darksky.net/forecast/ to get an api key.

```
import Axios from "axios";
import { Alert } from "react-native";

const getWeather = async(latitude =  LAT, longitude=LONG) => {
    // Alert.alert('api call')
    try {
        let result = "";
        let data = await Axios.get(
          `https://api.darksky.net/forecast/${API_KEY}/${latitude},${longitude}`
        );
       result = data.data.daily;
       return result;
    } catch (error) {
        throw error;



    }

};

export default getWeather;
```

```
export class Weather extends Component {
  constructor (props){
    super(props);
    this.state = {
      isLoading: true,
    };
    if(this.props.weather){
      this.setState({
        ...this.state,
        isLoading:false
      })

    }
  }
  componentDidMount() {
     getWeather()
    .then(res=>{
      console.log(res.data);
      this.props.updateBg(weatherConditions[res.data[this.state.pos].icon].bg);
      if(this.props.updateWeather(res.data, this.state.pos)){
        this.setState({
          ...this.state,
          isLoading:false
        })
      }

    })
    .catch((err)=>console.log(err));
  }

  render() {
    // console.log(this.props.weather, this.state)
    const {isLoading} = this.state;
    return (
      <View style={{ marginTop: 20 }}>
        {isLoading ? (
          <Text style={{ color: "#00a", fontSize: 25, fontWeight: "bold" }}>
            Loading Weather Report ...
          </Text>
        ) : (
          <View style={{ flex: 1 }}>
            <View style={styles.container}>
              <MaterialCommunityIcons
                size={80}
                name={
                  weatherConditions[this.props.weather[this.state.pos].icon]
                    .icon
                }
                color={"#fff"}
              />
              <Text style={styles.tempText}>
                {this.props.weather[this.state.pos].temperatureLow}&deg;
              </Text>
            </View>
            <View style={styles.bodyCont}>
              <Text style={styles.title}>
                {this.props.weather[this.state.pos].summary}
              </Text>

              <Text style={styles.subtitle}>
                {
                  weatherConditions[this.props.weather[this.state.pos].icon]
                    .subtitle
                }{" "}
                !
              </Text>
            </View>
          </View>
        )}
      </View>
    );
  }
}
```

PERMISSIONS