**Prerequisites:**

JavaScript and basics of React (Check out Doc in <https://github.com/ritveak/Learning-React>) .

**React Native:**

Code is written in React which uses JavaScript and is rendered with native code. Hence facilitates cross platform development.

With all the benefits of React, React Native becomes a powerful cross platform development technology with huge support on online platforms.

**Issues:**

* Styling has to be done separately
* Only basic set of pre-built components
* Responsiveness is achieved by extra efforts.
* Iphone simulator doesn’t work on windows.
* Constant updates has to be made whenever React Native is updated.
* Styling Issues:
  + [Button Styling doesn’t work](https://stackoverflow.com/questions/43585297/react-native-button-style-not-work)
  + [Font weight values from 100 to 900 doesn’t work only bold works.](https://github.com/facebook/react-native/issues/26193)

**Perfect learning resource is the** [**official React Native doc**](https://facebook.github.io/react-native/docs/getting-started) **. So I will be pulling off few examples from there and explaining em. Also most of the contents in this doc will be derived from the course on react Native by Maximilian Schwarzmüller which I am following.**

**Expo CLI vs React Native CLI:**

**React Native init:**

**Advantages:**

* You can add native modules written in Java/Objective-C (probably the only but the strongest one)

**Disadvantages:**

* Needs Android Studio and XCode to run the projects
* You can't develop for iOS without having a mac
* Device has to be connected via USB to use it for testing
* Fonts need to be imported manually in XCode
* If you want to share the app you need to send the whole .apk / .ipa file
* Does not provide JS APIs out of the box, e.g. Push-Notifications, Asset Manager, they need to be manually installed and linked with npm for example
* Setting up a working project properly (inlcuding device configuration) is rather complicated and can take time

**Expo:**

**Advantages:**

* Setting up a project is easy and can be done in minutes
* You (and other people) can open the project while you're working on it
* Sharing the app is easy (via QR-code or link), you don't have to send the whole .apk or .ipa file
* No build necessary to run the app
* Integrates some basic libraries in a standard project (Push Notifications, Asset Manager,...)
* You can eject it to ExpoKit and integrate native code continuing using some of the Expo features, but not all of them
* Expo can build .apk and .ipa files (distribution to stores possible with Expo)

**Disadvantages:**

* You can't add native modules (probably a gamechanger for some)
* You can't use libraries that use native code in Objective-C/Java
* The standard Hello World app is about 25MB big (because of the integrated libraries)
* If you want to use: FaceDetector, ARKit o Payments you need to eject it to ExpoKit
* Ejecting it to ExpoKit has a trade-off of features of Expo, e.g. you cannot share via QR code
* When ejecting to ExpoKit you are limited to the react native version that is supported by ExpoKit at that point in time
* Debugging in ExpoKit (with native modules) is a lot more complicated, since it mixes two languages and different libraries (no official Expo support anymore)

***And you can use any one which satisfies your applications requirement. (source –*** [***StackOverflow***](https://stackoverflow.com/questions/54862388/what-is-the-difference-between-expo-cli-and-react-native-cli)***)***

(I am currently using Expo)

**Steps to get Started:**

1. Install Expo cli : npm install expo-cli –global
2. Create new project : expo init one (ie, proj name)
3. Go to your folder : cd my-new-project
4. Start : npm start

**Installing all dependencies: Whenever we download a project, the node modules are ignored coz they would make the project bulky, so we have to run : npm install so that all the node dependencies mentioned in the package.json file are downloaded and installed.**

**Styling :**

* No CSS!
* inlineStyles AND/OR StyleSheet Objects (although based on CSS, it is written in javascript).
* The properties name has changed into camelCase

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

});

* This is how Style sheet Objects are created and can be implemented in components like this:

View style={styles.container}>

* A bigger and better example:

import React, { Component } from 'react';

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

const styles = StyleSheet.create({

  bigBlue: {

    color: 'blue',

    fontWeight: 'bold',

    fontSize: 30,

  },

  red: {

    color: 'red',

  },

});

export default class LotsOfStyles extends Component {

  render() {

    return (

      <View>

        <Text style={styles.red}>just red</Text>

        <Text style={styles.bigBlue}>just bigBlue</Text>

        <Text style={[styles.bigBlue, styles.red]}>bigBlue, then red</Text>

        <Text style={[styles.red, styles.bigBlue]}>red, then bigBlue</Text>

      </View>

    );

  }

}

Here we can see, that the CSS are over lapped and not overwritten, ie. If same property is changed in the latter mentioned class, then only those properties are updated, the rest stays and doesn’t get removed.

* We can also use inline CSS as follows:

<View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />

* To add slight additions to style which is should also use a defined style object, we can use both inline and style object by wrapping around [ ]:

 <View style={[{height:80},styles.field]}>

**Props (**[**Layout props list**](https://facebook.github.io/react-native/docs/layout-props)**) :**

* Basically the properties that we send inside a tag for example source in below:

<Image source={pic} style={{width: 193, height: 110}}/>

Where pic is:

let pic = {

uri: 'https://upload.wikimedia.org/wikipedia/commons/d/de/Bananavarieties.jpg'

};

* Props can come in handy when we make our own components for eg:

import React, { Component } from 'react';

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

class Greeting extends Component {

  render() {

    return (

      <View style={{alignItems: 'center'}}>

        <Text>Hello {this.props.name}!</Text>

      </View>

    );

  }

}

export default class LotsOfGreetings extends Component {

  render() {

    return (

      <View style={{alignItems: 'center', top: 50}}>

        <Greeting name='Rexxar' />

        <Greeting name='Jaina' />

        <Greeting name='Valeera' />

      </View>

    );

  }

}

* Here, Greeting uses this.prop.name to greet the individual being called. The name is given when the <Greeting> tag is used as a prop(as mentioned inside LotsOfGreetings).
* Props is set by parent and stays same throughout.
* To use something which can be changed by functions, we use State.

**State:**

* State acts as a variable which can be easily changed using setState or React Hook’s useState (React Hooks - let you use state and other React features without writing a class. Key concept – use functions instead of classes, and it also reduces the hassle of writing functions for eg, setState is eased into useState. Check [learning- react](https://github.com/ritveak/Learning-React) doc for more. )
* Example:

import React, { Component } from 'react';

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

class Blink extends Component {

  componentDidMount(){

    // Toggle the state every second

    setInterval(() => (

      this.setState(previousState => (

        { isShowingText: !previousState.isShowingText }

      ))

    ), 1000);

  }

  //state object

  state = { isShowingText: true };

  render() {

    if (!this.state.isShowingText) {

      return null;

    }

    return (

      <Text>{this.props.text}</Text>

    );

  }

}

export default class BlinkApp extends Component {

  render() {

    return (

      <View>

        <Blink text='I love to blink' />

        <Blink text='Yes blinking is so great' />

        <Blink text='Why did they ever take this out of HTML' />

        <Blink text='Look at me look at me look at me' />

      </View>

    );

  }

}

* Here the state is getting updated every second making use of interval and previousState , the state being a Boolean which decides if the render function should render the text from props or not.

**useState ([video](https://www.youtube.com/watch?v=i3n1bkrkUww)):**

useState can be imported with help of react hooks:

import React, {useState} from 'react';

useState returns two parameters, the output text and a function. Whenever state is changed, the whole thing is re-rendered.

For example:

import React, {useState} from 'react';

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

export default function App() {

  const[outputText, setOutputText] = useState('Open up App.js to start working on your app!')

  return (

    <View style={styles.container}>

<Text>{outputText}</Text>

      <Button title="Change Text" onPress={()=>setOutputText('The text changed')} />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

});

This code gives out a screen with a button, tapping which, changes the text.

* The problem with use State is that it doesn’t merge the updations, it replaces the whole state with new one.

**Layout:**

* Height and width can be set manually in styles, but it won’t be responsive. To make things responsive we use [Flex](https://facebook.github.io/react-native/docs/flexbox).

<View  style={{flex: 1}}>

        <View style={{flex: 1, backgroundColor: 'powderblue'}} />

        <View style={{flex: 2, backgroundColor: 'skyblue'}} />

        <View style={{flex: 3, backgroundColor: 'steelblue'}} />

    </View>

* We can add flex direction to choose if we want the children to be in a row or in column. Thereby flex can be used for height as well as width, depending upon the flex direction.
* By default it is usually column.
* For Row we do it like this:

<View style={{flex: 1, flexDirection: 'row'}}>

        <View style={{width: 50, height: 50, backgroundColor: 'powderblue'}} />

        <View style={{width: 50, height: 50, backgroundColor: 'skyblue'}} />

        <View style={{width: 50, height: 50, backgroundColor: 'steelblue'}} />

      </View>

**Components:**

* [TextInput](https://facebook.github.io/react-native/docs/handling-text-input)
* [Button and touches](https://facebook.github.io/react-native/docs/handling-touches)
* [ScrollView](https://facebook.github.io/react-native/docs/using-a-scrollview)
* [Lists](https://facebook.github.io/react-native/docs/using-a-listview)

React Native Resources : A Pandora’s box which has everything for React Native [**https://www.awesome-react-native.com/**](https://www.awesome-react-native.com/)

**Screen Navigation:**

* Installing dependencies:

npm install --save react-navigation

npm install --save react-native-gesture-handler

npm install --save react-navigation-stack

* Making Navigation Stack:

import {createAppContainer } from 'react-navigation';

import { createStackNavigator } from 'react-navigation-stack'

//importing all the screens.

import CategoriesScreen from '../screens/CategoriesScreen';

import CategoryMealScreen from '../screens/CategoryMealScreen';

import MealDetailScreen from '../screens/MealDetailScreen';

const MealsNavigator = createStackNavigator({

    Categories : CategoriesScreen,

    CategoryMeals : CategoryMealScreen,

    MealDetail : MealDetailScreen

});

export default createAppContainer (MealsNavigator);

* Import in app.js:

import React,{useState} from 'react';

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

import \* as Font from 'expo-font';

import {AppLoading} from 'expo';

**import MealsNavigator from './navigation/MealsNavigator';**

const fetchFonts =()=>{

  return Font.loadAsync({

    'open-sans':require('./assets/fonts/OpenSans-Regular.ttf'),

    'open-sans-bold':require('./assets/fonts/OpenSans-Bold.ttf')

  });

}

export default function App() {

  const [fontLoaded, setFontLoaded]=useState(false);

  if(!fontLoaded){

    return(

      <AppLoading

       startAsync={fetchFonts}

       onFinish={()=>setFontLoaded(true)}

       />

    )

  }

**return (**

**<MealsNavigator/>**

**);**

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

    alignItems: 'center',

    justifyContent: 'center',

  },

});

*The above code also shows how to import fonts in the program.*

* Now using button to switch between screens. Below is one among the many screens defined:

import React from 'react';

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

const CategoriesScreen = props=>{

    return (

        <View style={styles.screen}>

            <Text>The Categories Screen</Text>

            <Button

title="Go to meals"

onPress={()=>{

**props.navigation.navigate({routeName:'CategoryMeals'});**

}}

/>

        </View>

    )

};

const styles=StyleSheet.create({

    screen:{

        flex:1,

        justifyContent:'center',

        alignItems:'center'

    }

});

export default CategoriesScreen;

* Use .navigate(’screenName’) when navigating to a different screen.
* Use .push(‘screenName’) instead of .navigate if you want to go to the same screen but want to reload that screen with different content (will be useful in making dashboard page).
* Use .goBack() / .pop() if you want to get the last screen on tap (for example on saving something.)
* Use .popToTop() to go back to first page.(use in logout)
* Use .replace(‘screenName’) to replace the stack, so that user cannot go back to the prev screen, (Use it after login screen.)

Tips and Tricks:

* **Button customization :** [Button Styling doesn’t work](https://stackoverflow.com/questions/43585297/react-native-button-style-not-work), ie it doesn’t accept style as a prop
  + Wrap it inside a View
  + Use TouchOpacity and make your own button
* <TouchableOpacity style={styles.button}>
* <Text> LOGIN </Text>
* </TouchableOpacity>
* const styles = StyleSheet.create({
* button: {
* alignItems: 'center',
* backgroundColor: '#DDDDDD',
* padding: 10
* }
* })
  + And don’t forget to import these:
* import { StyleSheet, Text, View,TouchableOpacity } from 'react-native';
* A nice [TextInput](https://callstack.github.io/react-native-paper/text-input.html#onFocus) – first install npm i react-native-paper

Follow below code to customize easily.

<TextInput label="Outstanding Amount" mode="outlined" theme={{ colors: { placeholder: "#b94253",placeholderTextColor:"grey", text: 'black', primary: "#b94253",underlineColor:'transparent'}}}style={{flex:.5,fontSize:15,marginRight:10}} ></TextInput>

* **Multiple components:**

App.js:

import React from 'react';

import { StyleSheet, Text, View, Container, Button, Image, TextInput,TouchableOpacity,ScrollView } from 'react-native';

import { LinearGradient } from 'expo-linear-gradient';

import Login from './screens/login';

export default function App() {

return (

<View style={styles.container}>

<Login style={{width:'100%'}}/>

</View>

);

}

const styles = StyleSheet.create({

container: {

flex: 1,

// alignItems:'center',

// justifyContent: 'center',

backgroundColor: 'transparent',

}

});

Login.js:

import React, { Component } from 'react';

import { StyleSheet, Text, View, Button, Image, TextInput,TouchableOpacity,ScrollView } from 'react-native';

import { LinearGradient } from 'expo-linear-gradient';

// import login from './screens/login';

class Login extends Component {

render(){

return (

<View style={styles.container}>

<LinearGradient

colors={['#cc2f44','#450c14']}

style={{width:'100%',height:'100%', alignItems: 'center'}}>

<Image source={require('../assets/logo.png')} style={{marginTop:30}} />

<Text  style={{color:'white'}}>Welcome! Please Login </Text>

<TextInput placeholder="Username" style={styles.input1} ></TextInput>

<TextInput placeholder="Password" secureTextEntry={true} style={styles.input2} ></TextInput>

<Text style={{marginTop:40,color:'white'}}>Forgot Password</Text>

<TouchableOpacity

style={styles.button}

>

<Text> LOGIN </Text>

</TouchableOpacity>

</LinearGradient>

</View>

);

}

}

const styles = StyleSheet.create({

container: {

flex: 1,

alignItems:'center',

justifyContent: 'center',

backgroundColor: 'transparent',

},

button: {

alignItems: 'center',

backgroundColor:'white',

color:'red',

borderRadius:5,

padding:10,

marginTop:40,

},

input1: {

width:'70%',

padding:5,

fontSize:20,

backgroundColor:'transparent',

borderColor:'transparent',

borderBottomColor:'white',

borderWidth:1,

marginTop:50

},

input2: {

width:'70%',

padding:5,

fontSize:20,

backgroundColor:'transparent',

borderColor:'transparent',

borderBottomColor:'white',

borderWidth:1,

marginTop:15

}

});

export default Login;

**CODE SNIPPETS WITH SCREENSHOTS:**

* 

<View style={styles.field}>

          <View style={{flex:.5}}>

            <TextInput label="Balance Due" mode="outlined" theme={{ colors: { placeholder: "#b94253",placeholderTextColor:"grey", text: 'black', primary: "#b94253",underlineColor:'transparent'}}}style={{flex:1,fontSize:15,marginRight:10}} ></TextInput>

          </View>

            <View style={{flex:.5 }}>

              <TouchableOpacity

              style={{flex:1,alignItems:"center",justifyContent:"center"}}>

                  <Image style={{flex:1, width:75, resizeMode: 'contain'}} source={require("../../assets/upload.png")}/>

              </TouchableOpacity>

            </View>

          </View>

  const styles = StyleSheet.create({

field:{

    flex:1,

    flexDirection:"row",

    alignSelf:"center",

    width:"100%",

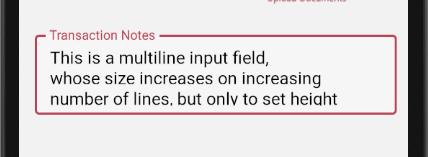
    paddingLeft:15,

    paddingRight:10,

    marginTop:10,

  },

});

* A multiline field whose size increases only to a limit

 <View style={[{height:80},styles.field]}>

     <TextInput label="Transaction Notes" multiline={true} mode="outlined" theme={{ colors: { placeholder: "#b94253",placeholderTextColor:"grey", text: 'black', primary: "#b94253",underlineColor:'transparent'}}}style={{flex:1,}} ></TextInput>

</View>

Style:

  const styles = StyleSheet.create({

field:{

    flex:1,

    flexDirection:"row",

    alignSelf:"center",

    width:"100%",

    paddingLeft:15,

    paddingRight:10,

    marginTop:10,

  },

});

* Login and Logout with Auth - <https://www.youtube.com/watch?v=fWXk2YC8gXI&t=26s>

Troubleshooting which were tough to find:

<https://medium.com/@vnbnews.vn/reactnative-react-navigation-error-createstacknavigator-has-been-moved-to-f90deacbdba9>

**Unable to resolve "@react-native-community/masked-view" from "node\_modules\react-navigation-stack\lib\module\vendor\views\MaskedView.js"**

**Failed building JavaScript bundle.**

**npm i @react-native-community/masked-view**

**Unable to resolve "react-native-safe-area-context" from "node\_modules\rStack\StackView.js"**

**Failed building JavaScript bundle.**

**expo install react-native-safe-area-view react-native-safe-area-context**

**External Resources:**

* Check this out, it is a good resource for understanding life cycle of components- <https://medium.com/@baphemot/understanding-reactjs-component-life-cycle-823a640b3e8d>
* Re- rendering Screens - <https://lucybain.com/blog/2017/react-js-when-to-rerender/>
* Side Menu

<https://stackoverflow.com/questions/52469835/how-to-add-a-drawer-inside-stack-navigation-in-react-navigation>,

<https://stackoverflow.com/questions/45451119/react-native-action-bar-and-react-native-menu>,

<https://codeburst.io/ricky-figures-it-out-simple-authorization-with-react-native-stacknavigator-using-react-navigation-579028b01a92>

* Populating dropdown with API data - [https://stackoverflow.com/questions/59999316/how-to-call-render-in-react-native-after-api-successfully-fetches-data/](https://stackoverflow.com/questions/59999316/how-to-call-render-in-react-native-after-api-successfully-fetches-data/60033599#60033599)

<http://www.mattmorgante.com/technology/dropdown-with-react>

* Popup - <https://reactnativeexample.com/a-simple-react-native-component-that-implements-a-popup-ui/>
* Secure Storage - <https://medium.com/@talut/react-native-secure-storage-rn-secure-storage-f7c84dbcb811>
* Fetch API - <https://codeburst.io/integrating-react-native-apps-with-back-end-code-using-fetch-api-8aeb83dfb428>
* Promises - <https://blog.domenic.me/youre-missing-the-point-of-promises/>