ECommerce Mobile App Template

Date: 31-January-2022 **By**: Samaritan Technology **Email**:hello@samaritan-technologies.com tahirawan4@gmail.com	Thank you for purchasing my theme. If you have any questions that are beyond the scope of this help file, please feel free to email me. Thanks so much!

Table of Contents

Basic Introduction & Background:

A Quote about React Native is **"Learn Once, Write anywhere"**. It combines the best parts of native development with React, a best-in-class JavaScript library for building user interfaces. You can use React Native today in your existing Android and iOS projects or you can create a whole new app from scratch. It provides a core set of platform agnostic native components like View, Text, and Image that map directly to the platform's native UI building blocks. React-native renders native app view components like <View> , <Text>, <Image>, <ScrollView> etc. Components let you split the UI into independent, reusable pieces, and think about each piece in isolation. To define components, we use "Functional base Components and Class base Components".

Basic Syntax of React Native:

```
import React from 'react';
import { View, Text, Image, StyleSheet } from 'react-native';
const Welcome = () => {
   return(
       <View>
           <Image source={require("welcome.png"),}/>
           <Text>Welcome to React Native Development</Text>
       </View>
   )}
const styles= StyleSheet.create({
   container:{
       flex:1,
   },
})
export default Welcome;
```

This example is based on the Function base component in which we have used different react native UI components for "Welcome Screen" and used inner Styling Method to design layout. We can use inline, inner and external styling methods to design our App screen layout.

Installation Steps:

To start the React Native project you need to install the required tools. The compulsory tools for React Native are:

- Visual Studio Code
- Android Studio
- XCode
- Emulator/ Simulator
- React Native Project Build

Visual Studio Code:
To start the React Native project download the Visual Studio Code. You can use the below link to download the VS Code according to your machine (Windows, Mac).
Visual Studio Code: https://code.visualstudio.com/download.

After downloading the VS code install it on your system with the basic steps. You can follow the youtube tutorial for **Installation Steps** → https://www.youtube.com/watch?v=bN6DE-4uFNo)

Android Studio:

Download the **Android Studio** from google with the latest version or by clicking it https://developer.android.com/studio?gclid=Cj0KCQiAgP6PBhDmARIsAPWMq6n7iTFYi5Wl_tC3MYywA7Bqvrt2GAzMiCzadU5eVTeAxs1-ZSWbATsaApZeEALw_wcB&gclsrc=aw.ds.

After download **Android Studio** install it in your System to run it. It will be used to download Emulator.

XCode:

You can download it from AppleStore. It will be only used with a mac for running a simulator.

Emulator/ Simulator:

These are the Virtual devices which will be used on your system to run React Native applications. Simulators and Emulators are typically used to perform most software tests. Real-device testing tends to be performed only late in the software delivery pipeline, just before releasing software into production. That way, you can take advantage of the speed and flexibility of simulated and emulated test environments for most software tests.

React Native Project Setup:

Follow the different Steps to build your first React-Native project. We will use React Native CLI for this project.
Open your VS code and open new terminal then write below commands in terminal one by one which are:

1. brew install node
2. brew install watchman
3. Then install HomeBrew, the command will be used from this link → https://brew.sh/
4. Then will download the Emulator from android studio by following the different Steps in the bellowing Link.
5. npx react-native init newProject
6. Open Project in VS Code the run command in terminal → **npx react-native run-android**

Some commands will be used only for Mac_os

7. Download the simulator from xcode, follow steps from the bellowing link.
8. Write on Terminal → sudo gem install cocoapods
9. npx react-native run-ios

For **React-Native complete environment setup** you can use the following link for both Windows or Mac, also for Android and mac.

(**Link**→ https://reactnative.dev/docs/environment-setup)

E-Commerce App Introduction:

ECommerce Mobile App is developed with React Native. This App is available with a different theme which can be changed by the desired color selection in the settings screen. All the necessary data/information has been covered in this App which would be an imported factor for the eCommerce Selling point. Complete Login/ SignUp system has built in with complete Authentications/Verifications. The App is completely responsive on both platforms and Quality has tested on different devices which are good and Reliable.

- **Version**:- 3.0.0
- **Support**:- Android and IOS
- Theme Selection
- Fully Responsive Template
- Well-Organized & Clean Code Layer

Application main file is App.js file when the application starts running on device by default this file is called. In it, we have called the MainNavigation file where all the navigation related to screens are handled.

```
import React from 'react';
import MainNavigation from './src/Navigation/MainNavigation'

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

All the packages which we have used for E-Commerce App you can see in the package.json file. If you want to import any new package in the app you will import through terminal or view it in this file.

Src folder is the Basic folder in this app development lifecycle. This folder consists of 6 folders which we have discussed below in this document deeply.

Quick Start Guide:

Download the source code of E-commerce app from zip file and follow below steps to run App on your system:

1. Save the project source code on desktop
2. Open this project on Visual Studio Code
3. Open the Terminal
4. Run ‘npm i’ command

   Node module will be added in your project directory

(Important Note: Simulator and Emulator should be downloaded on your system)

5. For **Android** —> run ‘npm run android’ command on terminal
6. For **IOS** —> run ‘npm run ios’ command on terminal

(Other commands if face errors/problems related to ios)

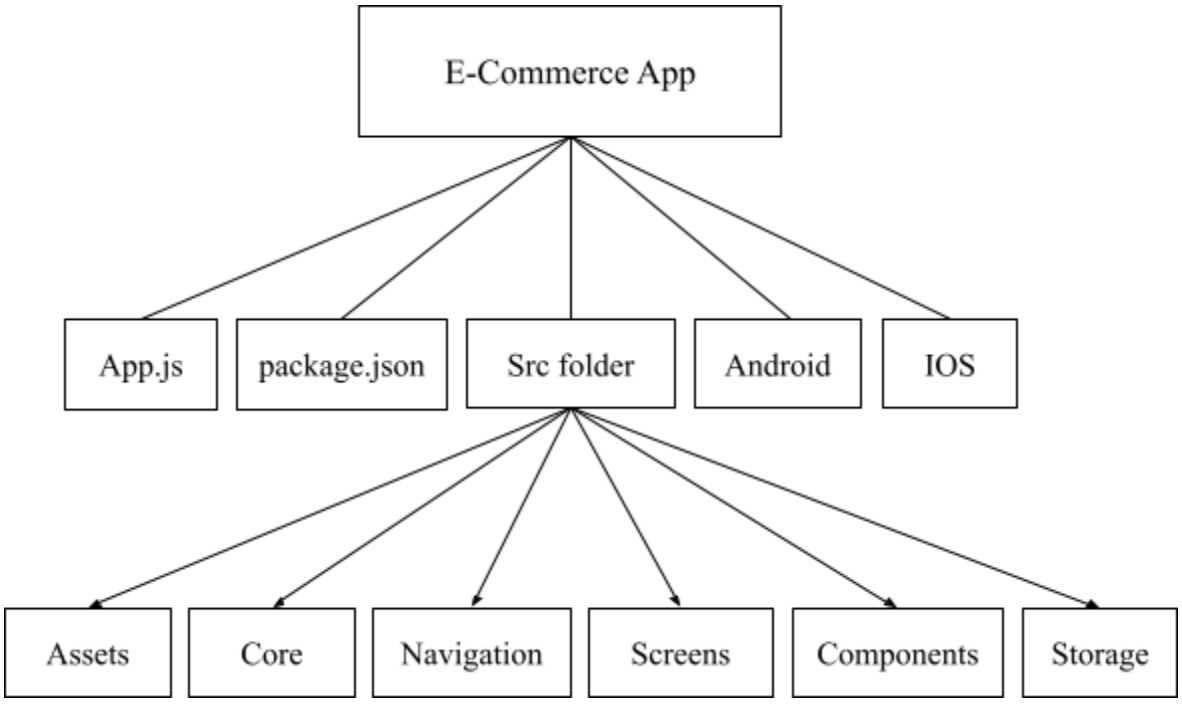
7. Delete POD folder from ios folder
8. Run ‘cd ios && pod install && cd ../ && react-native run-ios’ command on Terminal

(For generating **Android Application Package (**apk**)**)

9. cd android && ./gradlew clean && ./gradlew app:assembleRelease && cd ..

Application Architecture:

E-Commerce app Architecture are:



Ecommerce App is based on src Folder where the whole application factors are placed in it. All layers in the src folder perform a specific role in development.

- **Assets**
  All .png files are saved in this folder which are necessary assets for our application. Assets Folder included fonts, icons, images.

- **Components**
  Default components have been created here which can be used more than one time in the whole application. Components Folder included CheckBox, Font, Headers, Icon, Profile, SearchBar, Social Button, TextInput with .js extension.

- **Core**
  Core file included context, ContextState, theme. We have linked all assets here.in theme file we have used all the strings of Application, colors code, font family, image and icons. In context and contextState we are handling the context state of our application which handles the theme selection method.

- **Navigation**
  A navigation of all Screens are handled in this folder.

- **Storage**
  It worked like a database which stores the information of the user and generates a key according to this information to handle accessibility of applications screens.

- **Screens**
  This folder holds all Screens of our application. A separate folder is created for each screen which includes 2 .js files, one is based on react native components and second one is based on react native styling. Screens Folder included:

  - Auth Screen
    - SignIn
    - SignUp

  - Carousel Screen
    - Corousel1
    - Carousel2
    - Corousel3

  - Main
    - Cart
    - Coupon
      - Coupon
      - CouponDetail
    - CustomerSupport
    - Delivery
      - Delivery
      - DeliveryForm
      - ShippingAddress
    - Favourite
    - Filter
    - Home
    - Message
    - Notification
    - Order
      - Order
      - ConfirmOrder
      - PlaceOrder
    - PaymentMethod
    - Reviews
      - Reviews

        - WriteReview
    - ❖ Setting
    - ❖ SingleItem
    - ❖ User
        - User
        - EditUser
    - ❖ Wallet

- ➔ SplashScreen
    - ❖ Splash

Updating existing module:

1. If you want to add or update a new module you will work in this **SRC Folder**.
2. If you want to add any new images, icons or fonts in application you will use the **assets** folder where you will place the item then use its source in **CoreFile** to link it with application.
3. **CoreFile** is based on different arrays which are font, images, icons, Strings and colors. You can add any new text in the String array which can be used more than one time in the app.
4. The whole **application's colors** are written in the **colors array** which is used in the app if you want to add a new color then update the color array by the way the other color is added.

Add new screen:

If you want to add new screen design in this app you can create a new **.js extension file** in Screens Folder and navigate this screen by import Screen path at the top of Navigation File and link it in the function body by the same way other screens are linked.

Hooks:

The E-Commerce app is built with functional base Components. In the functional base, we used different hooks to manage app internal data/states. The 4 types of hook are used in applications useState, useEffect, useContext, useRef.

- ➔ useState has been used to store the data related to a specific field.
- ➔ useEffect has been used to immediately render the data after loading Screen.
- ➔ useContext has been used for managing the theme Selection.
- ➔ useRef has been used for managing autoFocus in OTP modal.

Styling:

We have used the external styling Method in the whole app. React Native stylesheet component is used to design the screens layout.

```
import { StyleSheet, Dimensions, Platform } from 'react-native';
import theme from '../../Core/theme'
const { colors, font } = theme;
import Text_Size_Type from '../../Components/Font'

const styles= StyleSheet.create({
   container:{
       flex:1,
   },
   touchButton:{
       marginHorizontal:'5%',
       marginTop:Platform.OS === 'ios'? '10%': '4%',
       marginBottom:'28%',
   },
   icon:{
       margin:'5%'
   },
   formContainer:{
       backgroundColor:'white',
       marginVertical:'2%',
       marginHorizontal:'5%',
       paddingHorizontal:'6%',
       paddingVertical:'1%',
       borderRadius:Dimensions.get('window').width*0.06,
   },
   underlineText:{
       textDecorationLine:'underline',
   },
   modalWindow:{
       backgroundColor:'white',
       borderRadius:Dimensions.get('window').width*0.08,
       padding:'5%'
   },
export default styles;
```

Theme Selection:

The main feature of this application is that you can change the theme color of the whole application by clicking on one button. The buttons are shown in the settings screen at the top. The theme selection method is handled by the useContext hook in the app.useContext functionality related to theme manage are used inside the core folder by the name of **context, contextState** than change context state in the setting screen with the following code:

```
const ThemeSet = async (payload) => {
       const color = JSON.stringify(payload)
       try{
           await AsyncStorage.setItem('theme',color)
           getTheme()
       }
       catch(error){
           console.log("Token Set Error",error)
       }
   }

const getTheme = async () => {
       try {
           const color1 = await AsyncStorage.getItem('theme')
           let color = JSON.parse(color1)
           themeColor.setTheme(color)
           return color
       }
       catch (error) {
           console.log("Theme Get Error", error)
       }
   }
```

We have used two colors for the theme Selection. You can add more options but offered colors code are:

```
      orangetheme:'#FF7466',
      bluetheme:'#009DFF',
```

Validation/Verification:

The user validation/verification Method is handled in SignUp or SignIn Screens, Where inputs are fully restricted for registered users or login users.

```
const passwordMatch = (text) => {
       if (text === password) {
         return true
       } else {
         return false
       }
};
const emailValidation = (email) => {
                                                                    var         re         =
/^(([^<>()\[\]\\.,;:\s@”]+(\.[^<>()\[\]\\.,;:\s@”]+)*)|(“.+”))@((\[[0-9]{1,3}\.[0-9]{1
,3}\.[0-9]{1,3}\.[0-9]{1,3}])|(([a-zA-Z\-0-9]+\.)+[a-zA-Z]{2,}))$/;
       return re.test(email); // this will return true of false
};
const passwordValidation = (password) => {
       var re = /^(?=.*\d)(?=.*[!@#$%^&*])(?=.*[a-z])(?=.*[A-Z]).{8,}$/;
       return re.test(password); // this will return true of false
};
```

These Functions are used to validate the email or password.

```
LoginData = async (payload)=>{
       try{
           await AsyncStorage.getItem('auth')
           .then(req =>JSON.parse(req))
                   .then(JSON => payload.email == JSON.email && payload.password ==
JSON.password
           ? this.TokenSet(payload)
           : payload.modal(payload.error))
       }
       catch(error){
           payload.modal("Please First SignUp than Login")
       }
   }
```

This is the asyncStorage Method to verify the user by comparing their credentials.

AsyncStorage:

AsyncStorage is an unencrypted, asynchronous, persistent, key-value storage system that is global to the app. It should be used instead of LocalStorage. AsyncStorage is backed by native code that stores small values in a serialized dictionary and larger values in separate files. On Android, AsyncStorage will use either RocksDB or SQLite based on what is available. The AsyncStorage JavaScript code is a facade that provides a clear JavaScript API, real Error objects, and non-multi functions. Each method in the API returns a Promise object.

In the E-Commerce App, we have used async Storage. We don't need to install any package for it, just **import the library** and use it.

import { AsyncStorage } from 'react-native';

Persisting data:

```
_storeData = async () => {
try {
  await AsyncStorage.setItem('@MySuperStore:key', 'I like to save it.');
} catch (error) {
  // Error saving data
}
};
```

Retrieving data:

```
_retrieveData = async () => {
 try {
   const value = await AsyncStorage.getItem('TASKS');
   if (value !== null) {
     // We have data!!
     console.log(value);
   }
 } catch (error) {
   // Error retrieving data
 }
};
```

UseCase:

Users can interact with E-Commerce App with some defined module.