# Code Documentation

This code documentation describes and explains the source code, the user manual with UI screenshots can be found here: https://github.com/lorifranke/RealEstateMobileApp/blob/master/User_Manual.pdf

## Goal

Creating a mobile application that will be a user interface to the existing backend component to connect and extract data from the Realtor.com api. The API is available here (https://rapidapi.com/apidojo/api/realtor). Display data of property, analyze investment and profit. A competing website is DealCheck (https://app.dealcheck.io/!/login/main).

## Setup and Installation Instructions for the mobile App

This Real Estate App is written in TypeScript and JavaScript and built with the framework react native. It can be installed with Node.js using the node package manager. Download this folder and run `npm install` in the command line to get all necessary packages. This will install the packages listed in the package.json file. Alternatively, you can also use `yarn` to install packages.

To run a mobile application, you will need to use `npm install -g expo-cli` for React-Native as well as a mobile phone simulator (unless you run the code as web application). For iOS, please use XCode with its belonging phone simulator to run the app on a virtual phone. Another possibility is to use Expo CLI to run your React Native app on a physical device, therefore, just connect your phone with USB and run the app on your phone.

To initialize an expo project, you will need to run `expo init YourProjectName` in the command line. This will link an expo project to the folder. Then, you can start the application with `npm start` and the browser will open the metro bundler where you are able to access different simulators. Another way to start the app is by directly launching the app on an Android Virtual Device by running `npm run android`, or on the iOS Simulator by running `npm run ios`. The simulator will open directly without using the metro bundler from the browser. Before opening the simulator make sure the backend is running (see link below to backend). The backend can be started in the corresponding folder as well with `npm start`.

A further detailed documentation on how to use and install React Native can be found here: https://reactnative.dev/docs/environment-setup and an introduction how to set up a first mobile app here: https://reactnative.dev/docs/getting-started.

More resources on debugging the app while using React-native can be found here: https://reactnative.dev/docs/troubleshooting

See the Readme.md as well for updates.

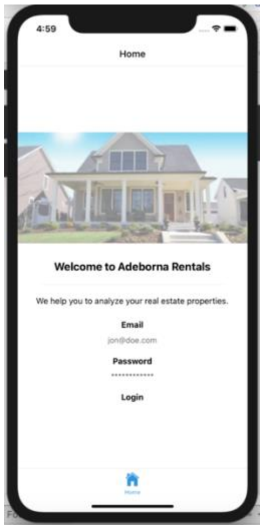
## Source code and folder structure

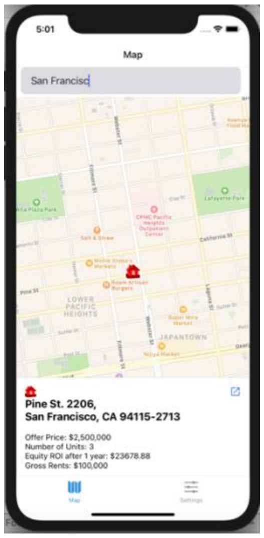
React always requires importing the modules and packages, which is done in the top of each file. The following ones are the most important ones which can be found in almost every file of the code:

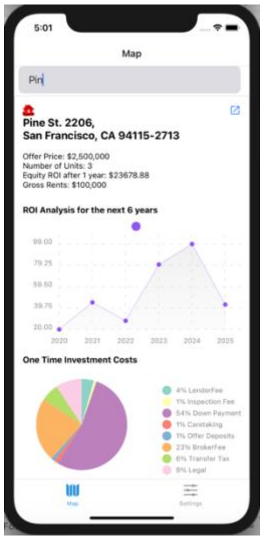
```
import {StyleSheet, ImageBackground, Button, TextInput,
TouchableOpacity, Alert } from "react-native";
```

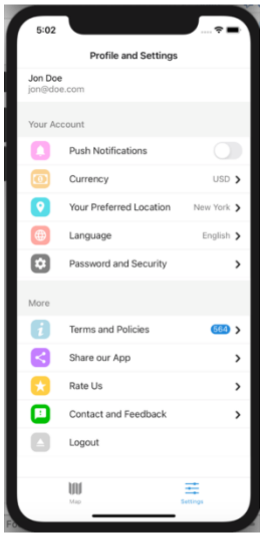
We can also import other files, or components we have written in other files. The description of the source code is described by each of the different folders of this repository. Please also check out the comments in the code as another reference.

### 1. Screens:


Home
Welcome to Adeborna Rentals
We help you to analyze your real estate properties.
Email
jon@doe.com
Password
Login
Home




Map
Pin
Pine St. 2206,
San Francisco, CA 94115-2713
Offer Price: $2,500,000
Number of Units: 3
Equity ROI after 1 year: $23678.88
Gross Rents: $100,000
ROI Analysis for the next 6 years
One Time Investment Costs
Map
Settings



This is the most important folder of the UI. It contains all main screens (or pages) of the mobile application. It contains different files, each belonging to one screen. The three screens we have in the app right now are called in the source code: tabone, tabtwo, tabthree. These are all linked in the navigation bar in the bottom of the UI. The UI screenshots and how the app's page/screen flow looks like can be found in the user manual of this repository. The login or home screen belongs to the TabOneScreen.tsx file. The profiles and settings are constructed in the TabTwoScreen.tsx file and the map is implemented with the TabThreeScreenPage. If we want to build additional pages that should be displayed in the app, we create a new screen file and put it in this Screens folder. In case we have some errors when displaying the app or call a screen is called that does not exist, there is the fallback screen NotFoundScreen.tsx for error handling.

**1.1 TabOneScreen.tsx:** This is the Login Screen, where the user enter his data and we call the backend database (MongoDB) within a fetch method and use the POST method for checking the users input (`fetch("http://localhost:5000/api/users/login", {method: "POST",…..`). The server will send JSON content back as well as a token that we will use to identify the user that should exist in our database. We also catch possible errors with `.then((response) => {`… For example if there is a server error or if the user enter wrong input.

In react native it is important to understand how to work with *states and props,* these can be defined and given to single components. More details on this can be found here: https://reactnative.dev/docs/state: 'Props are set by the parent and they are fixed throughout the lifetime of a component. For data that is going to change, we have to use state. In general, you should initialize state in the constructor, and then call setState when you want to change it'. In multiple files of the project, we define states and props. In the TabOneScreen.tsx file the function `onPressLogin = async () => {const { email, password } = this.state;}` passes the email and password from the user input as state when the user click the login button. This function is called inside the `<TouchableOpacity>` tag. `TouchableOpacity` is a react-native element that is used for buttons or texts that should call when functions when clicking them.

On the bottom of this file, a few CSS-like styles are defined as a const. These are called in each component tag with `style={styles.container}` and enable styling for each of the screens individually.

**1.2 TabTwoScreen:** TabTwoScreen.tsx contains the code to render the settings page. In the constructor in the top of the file, we pass data from the currently logged in user as state to display them. Another important state is defined in the `onChangePushNotifications()` method. The user can switch between enabled and disabled Push notifications, so the state needs to change in React. After this function we call `render()` to display the components of this screen. The settings page consists of multiple `<Listitem>` tags. These tags contain different features, we can remove and add more of these tags to create additional features for the settings page. Each list item has an `onPressSetting()` function that is called where we could define additional actions that are executed. We create different icons for each list item that will be displayed on the left with the tag `<BaseIcon>` and add styling to them with the style property.

**1.3 TabThreeScreen:** This is the Map of the mobile application. In this file we again have different states. For example, the variable `detailsOpened` determines if the details to a property are currently opened or not. We use an `updateSearch()` function to display the new results while typing but not having opened the details to the property. The `getResults()` function verifies the results that we fetch from the property.json files from the API/backend. The search term is matched with data from these files. If the function matches a search term with parts of an address or the full address, then it returns the property. The `markerClick()` method is setting the state for the details opened or not. So if the user clicks the house marker on the ap, the state will change to opened and display details vice versa. In the render function we define some constants that are necessary for the charts (for example which parts of the property.json received by the backend is shown). In the return function we define the map.

Maps in react native always require the `<MapView>` tag. If we have an API key we can switch to different providers (eg. Apple Maps, Google Maps etc.). After importing the provider, we just add: `<MapView provider={PROVIDER_GOOGLE}>` to use Google maps as API. If no provider is defined, the app will use the default map that is on the phone (depending on the OS). More information and how to configure the map can be found here: https://github.com/react-native-maps/react-native-maps. We also add the default location in the MapView tag with region={{…}} in longitude and latitude values. Otherwise, we can also catch the users geographical location in here if he allows location tracking on his/her phone. We render the SearchBar tag and call the search functions and style with the bottom StyleSheet so that it will stick to the top of the Screen. We also add the clickable Button to open the chart and property analysis inside a <ScrollView style={styles.scroll}>. The scrollview allows the user to scroll down if the content of the page is very long. Inside this scroll view we render our charts and data we have available.

## 2. Assets:

In this folder we save all images and icons that we need. See image folder: We have for example the favicon, marker icons for the map, an icon for the waiting screen etc. The second folder in Assets contains files defining the fonts we are using to create a nice design. These fonts can be found online as templates and exist for regular websites as well (e.g., Google Fonts), or they can also be written by oneself.

## 3. Components:

This folder contains helper files that create little components of the app. Chevron.js, InfoText.js, index.js which are helper components for the Profile/Settings screen. Chevron.js, for instance, creates the right clickable icon that is displayed on each list element in the Settings screen.

## 4. Constants:

Here are files containing configurations for the color schemes and the overall layout of the app. React allows similar structured style sheets as CSS for HTML to style elements (see Colors.ts file).

## 5. Navigation:

This folder contains all files that are necessary for the UI navigation in the mobile app. For example, the bottom nav bar that navigates through the app (See BottomTabNavigator.tsx). All the screens (pages) we create *must* be linked in the navigation to be visible and usable. In BottomTabNavigator.tsx they have to be defined in a tag <BottomTab.Screen> in the render() function otherwise they won't be displayed in the app.

6. **Node-modules:** This folder contains all packages necessary for running the app. These modules are all listed in the package.json file as well with their corresponding current version, and will be auto installed when running the setup part with `npm install` of this documentation. Also, make sure to use `npm update` to always install the latest version of the modules and packages!

**property.json file:** This file is created by the backend part of this project and is picked up by this UI. As soon as the backend is able to save multiple properties in a database, we will need to change this part to fetch the files from the databse. The UI can render this provided information from the property.json files. Depending on which information we have available from fetching data from the API connected to the backend, the UI will show the received information in the different graphs we have.

**Backend**:

This application is the User Interface (UI) part of a full software project. The backend belonging to this real estate mobile UI can be found here: https://github.com/MounikaChava304/CS682

The backend consists of different property calculators and connects to the realtor API. For user management the user data is saved with MongoDB.

**Package.json, package-lock.json and yarn.lock**: These are files that are necessary for installing and using the packages required for the app.