COMP 4905 Honours Project Winter 2022

Rent Wise : A house review app

Author:

Payton Dong Pei

Carleton University, Ottawa

Supervisor:

Dr. Louis D. Nel

Carleton University, Ottawa

An Honours Project submitted in fulfillment of the requirements for the degree of

Bachelor Of Computer Science: Mobile Computing

March 25, 2022

Contents

List of Figures

Abstract

There are around 15 billion mobile devices in use as of 2021.[**numberDevices**] Now, the market share of mobile phone operating systems IOS and Android, where the market share of IOS is about 28.29%, the market share of Android is about 78.94% [**statcounter**]. This also means mobile developers need to learn different programming languages (Java, Kotlin, IOS, Objective-C) for different operating systems. This increases the company's operational cost significantly. Still, if it is a small company or the company does not have enough budget to hire enough Android Developers and IOS developers. Mobile developers will have to learn how to develop IOS software and Android software simultaneously.

This Honours Project used a cross-platform application framework maintained by Facebook(Meta) **React Native**. Over the last two years, Covid-19: The pandemic that swept the world in the previous two years, complicated what was once a simple matter in past. Suppose a student wanted to find a new apartment two years ago. In that case, they could call the Rental offices and schedule an appointment for a visit. However, he could not do it quickly without safety concerns.

Rent Wise is an Android and IOS application that helps students find places to live in a more approachable way. Students can use their email to log in to the app, post reviews of the sites they lived in before and check the rating of the places they will rent in the future.

Acknowledgement

I have used many open-source libraries to complete this project. I am really grateful to all the maintainers and collaborators of these libraries. I would also like to thank Jetbrains for the free student license that allowed me to use Webstorm to develop my application. The five years I spent at Carleton University was a unique and rewarding experience.

I would like to thank Professor Louis D. Nel for his support and guidance. His guidance and advice helped me through the process of writing this project and report. I would also like to thank my family for their support.

1 Introduction

1.1 Brief Introduction

This project is a cross-platform mobile application on Android and IOS, and the name of the project is Rent-Wise. This app is built by TypeScript and uses an Open-sourced framework called React Native. [**github-rn**]. In this application, users can review the apartments they lived in before or find the review of the apartments they intend to rent.

1.2 Motivations

In the past three years, Covid-19 has swept the world. Many people have lost their jobs and homes because of this epidemic. It takes a lot of time to find an apartment, but that apartment is likely to have many minor problems. For example, the refrigerator is broken, too close to the highway, parking is not convenient, etc.

I also had this problem. Last summer, I was in my internship, so I moved to a place close to my company. But I didn't visit the site in advance, so I realized it's boisterous and takes a long time to shovel snow in winter when I signed the contract.

On the technical side, I also need to strengthen my experience in mobile development through projects like this. I developed several Android and IOS applications through my previous studies at Carleton University. However, native development has its limitations, so I learned about cross-platform projects. I learned about React Native, maintained by Facebook (Meta) and started out as a company-wide project. But due to its many features, code

reuse, live to reload, etc., Facebook finally released this enviable project in 2015.

2 Background

2.1 History of Smartphones

On January 9, 2007, the CEO of Apple, Steve Jobs revealed, the iPhone in San Francisco, which has since changed the way users relate to their on-the-go devices.[**applerelease**] At the same time, Google was secretly developing Android. Subsequently, Google released the public beta of Android version 1.0 on November 5, 2017.[**androidrelease**]Since then, an era of smartphones has begun. People no longer need a stylus to control their phones. Instead, they can use their fingers to operate their phones.

2.2 Cross-platform mobile solutions

There are many cross-platform solutions, such as Flutter and Xamarin, but React Native is the choice for this project. Flutter is maintained by Google and seems more popular than React Native, maintained by Facebook (Meta). Star rating is a helpful metric and feature on GitHub, often used to indicate that a user is interested in a project or because it is helpful to him/her. GitHub Issues is a tracking tool that integrates with the GitHub repository. Use GitHub Issues to focus on essential tasks and keep the schedule up to date at the same time. [**github-issue**]

To some extent, if a repository has more stars and issues, it proves that it draws more attention and is more popular. More people get involved in that repository

continuously. From the following two tables, we can see that obviously, Flutter has more stars and issues than React Native.

Github Stars and Issus[**flutter-star**][**rn-star**]:

Flutter	React Native
139k stars	102k stars

Flutter	React Native
5k+ issues	1.9k issues

Dependencies required for React Native and Flutter:

- **React Native** requires npm, node,node_modules, react-native-cli, and other configurations.
- **Flutter** requires flutter SDK and Dart and Flutter plugins on Android Studio or VSCode

Both React Native and Flutter require Android and IOS development environments, i.e. JDK, Android SDK, Xcode and other environment configurations.

In terms of the configurations, Flutter seems to be easier to successfully build the project for the first time compared React Native. The complexity of node_modules dependencies can easily make developers frustrated.

2.2.1 Implementation and Architecture

React Native is a UI framework. By default, React Native loads JS files under Activity, runs in JavaScriptCore to parse the Bundle file layout, and finally stacks a series of native controls for rendering.

Briefly, developers use JavaSccript/TypeScript code to write the User Interface, and then React Native parses the JS/TS scripts and renders it into native components, such as <View> tags for ViewGroup/UIView, <ScrollView> tags for ScrollView/UIScrollView, <Image> tags for ImageView/UIImageView, etc.

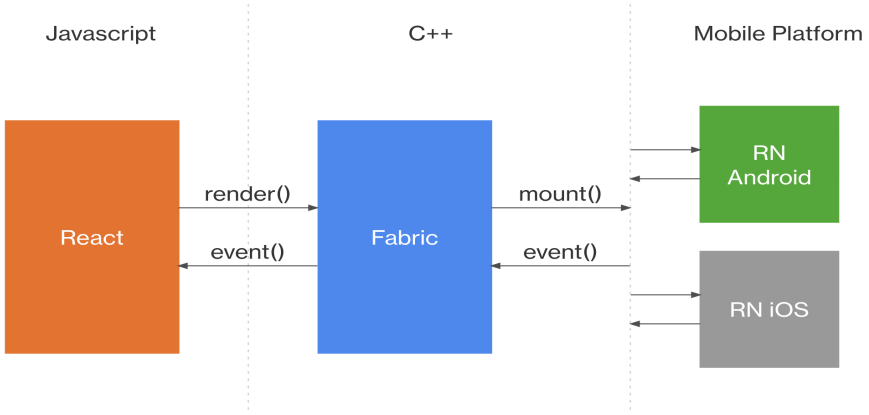


Figure 1: An image of architecture layer of React Native[**rn-arch**]

Flutter has a similar approach to draw the UI. Widget is an essential concept in Flutter and is not being rendered as a native component. Element changes will render the widget and finally transform it into RenderObject for drawing, and the final composition of the RenderObject tree is the actual rendering DOM (React Native's approach). Each Widget tree trigger changes, not necessarily leading to a complete update of the RenderObject tree.[**flutter-tree**] This rendering process gives Flutter more robust performance because Flutter does not call native components but renders Flutter code directly to the screen through the Skia engine. There is no bridge between Flutter and the platform; however,

React Native does.

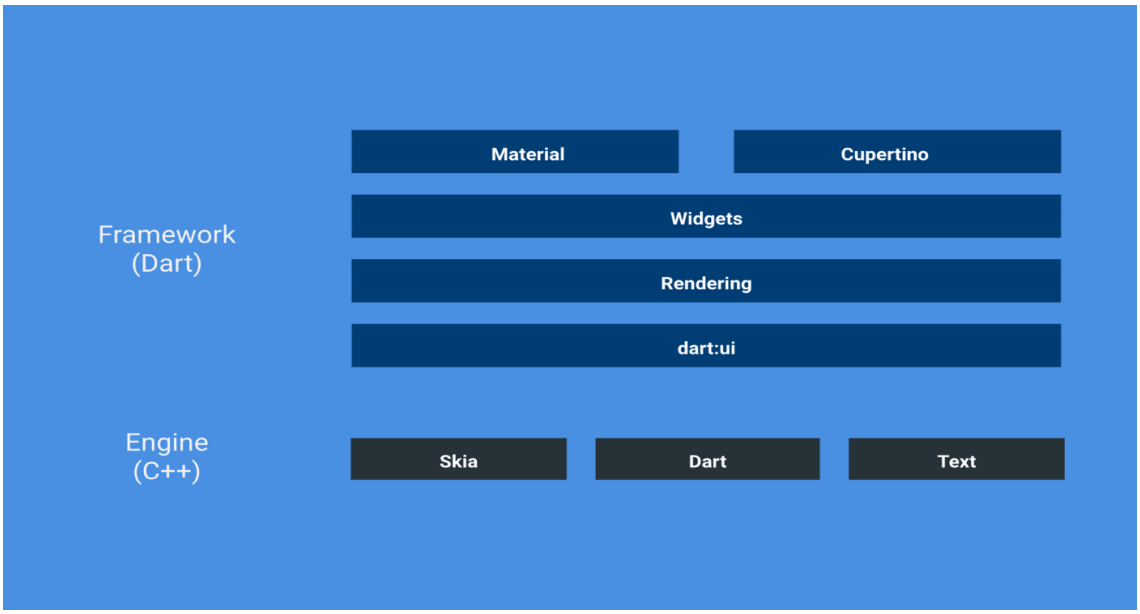


Figure 2: An image of architecture layer of Flutter[**flutter-arch**]

Conceptually, React Native and Flutter have different approaches to achieving the same goal: rendering UI for different mobile devices. Even if they both have a "virtual DOM concept," React Native has vital relevance with the platform(IOS, Android) while Flutter does not.[**comparison**]

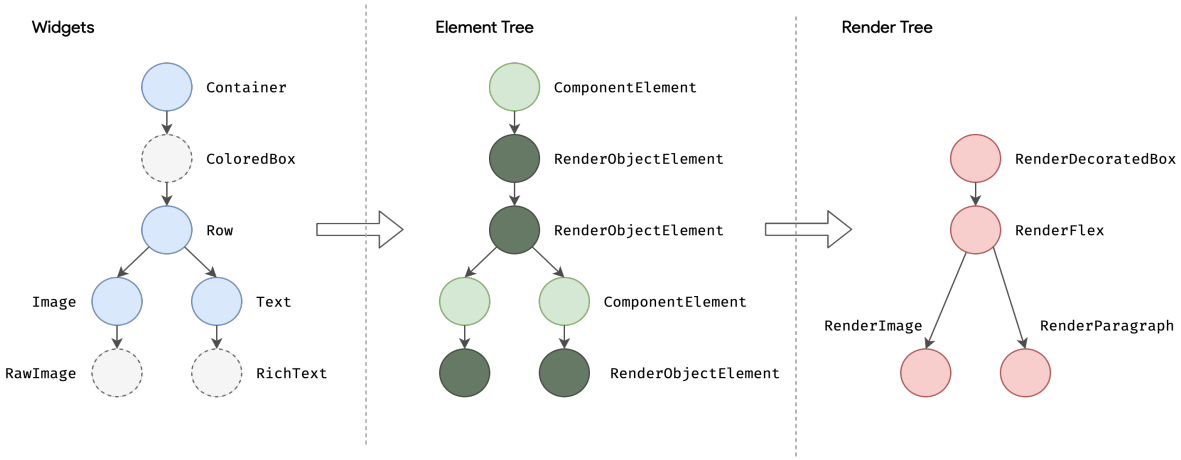


Figure 3: An image of Flutter's layout and rendering[**flutter-tree**]

2.3 Reason of Choosing React Native

Despite the performance advantages of Flutter, Dart is still too young for JavaScript. Learning to develop with Flutter may require time to learn the language features of Dart, and the learning curve becomes steeper compared with JavaScript, and the time may not be sufficient for an Honours Project. Since Brendan Eich invented JavaScript in 1995[**jshistory**], it is now 27 years old. The history of JavaScript is 27 years old. University classes usually offer a course in web development, and most courses use JavaScript as the development language. So it does not take much time to learn the features of JavaScript, and there are plenty of resources on the web. So students/developers can learn the React Native API to develop their project directly and spend most of the time developing the software instead of focusing on the features of the language.

2.4 TypeScript

TypeScript = Type + Script (Standard JavaScript)

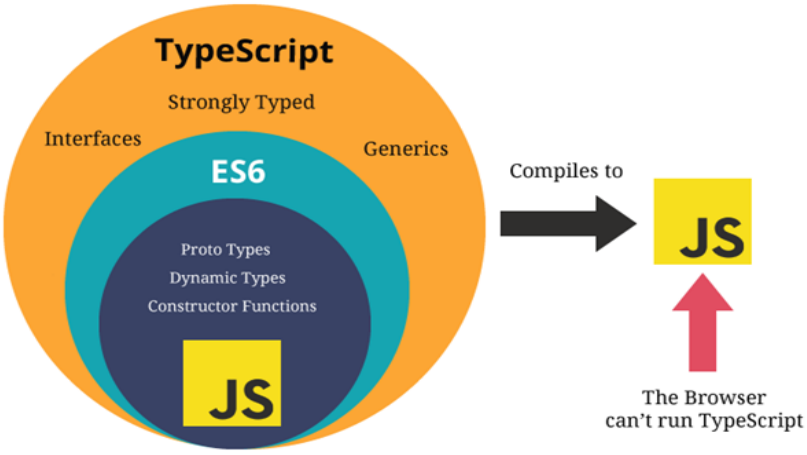


Figure 4: An image of TypeScript[**typescript**]

TypeScript is a programming language that was developed by Microsoft. It was firstly released in October 2012.[**typescript1-0**]. When Microsoft was developing large projects, it discovered the limitations of JavaScript. As time went on, projects created in JavaScript became harder to maintain because JavaScript is not as strict when it comes to syntax. If TypeScript is used, The constraints of Type and Interface can take over the role of some of the unit tests to a certain extent. Meanwhile, the lines of testing code can also quickly find code dependencies through the by the IDE (Webstorm, VSCode).

```
1 //JavaScript
2 function mathOperation(data){
3         return Math.sqrt(data.x ** 2 + data.y ** 3);
4         // This function will return Square Root of data.a^2 + data.b^2;
5     }
6 mathOperation({x:2, y:3}) // No errors for this call and returns 5.5677643628300215
7 mathOperation({x:"Hello World",y:3}) // No errors for this call and returns NaN
```

This math function is executed without any error in JavaScript if the data

are integers, but there will be problems when the data that was passed in is not something expected like a String. It's extremely hard to find errors like this when in development.

```
//TypeScript
function mathOperation(data:{x:number, y:number}){
        return Math.sqrt(data.x ** 2 + data.y ** 3);
        // This function will return Square Root of data.a^2 + data.b^2;
    }
console.log(mathOperation({x:2,y:3})) // NO errors for this call and returns
    2.8284271247461903
mathOperation({x:"Hello World",y:3}) // ERROR Type 'string' is not assignable to type
    'number'.
```

2.4.1 JavaScript: A Dynamically Typed language

When a programming language is **Dynamically Type** means that the 'type' of the variable will not be checked until the runtime executes that line or when the variable is referenced. But statically typed programming languages like Java and C++, the type of the variables will be checked during the compile time. The benefit of dynamically typed programming language is that people do not need to specify the variable's data type. They can just declare the variable and use it; however, situations get complicated when the scale of the project gets larger.

2.4.2 Reasons of Choosing TypeScript

This code demonstrates the power of TypeScript's strict typing, code readability, and ease of maintenance, which means developers can avoid many mistakes during the development phase. At the same time, a JavaScript developer does not need to spend much time getting familiar with the features of the language. Because TypeScript is a superset of JavaScript, JavaScript code can be directly

converted to TypeScript code. Some TypeScript-only features require manual code updates, but they also save much time. This time spent manually updating code also gives programmers more opportunities to learn TypeScript features and functionality.

Traditional JavaScript uses functions and prototype-based inheritance to build reusable components or functions, but this may feel awkward to programmers more comfortable with an object-oriented approach, where classes inherit functionality and objects are built from these classes. Starting with ECMAScript 2015, also known as ECMAScript 6, JavaScript programmers can build their applications using this object-oriented class-based approach. In TypeScript, we allow developers to use these techniques and compile them down to JavaScript that works across all major browsers and platforms without waiting for the next version of JavaScript. [**ts-classes**]

Class and Enumeration are two critical concepts in Object-Oriented Programming. However, the way JavaScript implements Class cannot make the most of the Object-Oriented ProgrammingS concept and Rent Wise requires these two concepts. As a result, TypeScript is selected for this project.

3 Installation

3.1 Project Setup

Suppose the project already has ios/ and android/ directories in the codebase. In that case, the reader may skip the Android, IOS setup in this chapter and continue to configure Firebase Chapter 3.6 Firebase Setup if the reader wants to see how the Firebase Database interacts with the app.

To build both Android and IOS app, you need to install following tools:

- XCode
- Android Studio
- Visual Studio Code
- JDK, JRE
- Node.JS
- cocoapods

3.2 Run Rent-Wise on the Emulators

- iPhone 13 IOS 15.0
- Google Pixel 3 API 28

3.3 Project Directory Structure

```
/Rent-Wise-Main
├── Gamefile
├── Gamefile.lock
└── README.md
```

```
├── __tests__
│   └── App-test.js
├── android
├── app.json
├── ios
├── metro.config.js
├── package.json
├── src
│   ├── App.tsx
│   ├── assets
│   │   ├── apartment.jpg
│   │   ├── avatar.png
│   │   ├── house.jpg
│   │   ├── house2.jpg
│   │   └── house3.jpg
│   ├── common
│   │   ├── constantsDefine.ts
│   │   ├── dateUtils.ts
│   │   └── localStorage.ts
│   ├── components
│   │   └── toast.ts
│   ├── firebase
│   │   ├── firebase-auth.ts
│   │   └── firebase-house.ts
│   ├── screens
│   │   ├── HouseCollectionScreen.tsx
│   │   ├── HouseDetailsScreen.tsx
│   │   ├── HouseSearchResult.tsx
│   │   ├── LoginScreen.tsx
│   │   ├── MarketScreen.tsx
│   │   ├── PostScreen.tsx
│   │   ├── ProfileNavWidget.tsx
│   │   ├── ProfileScreen.tsx
│   │   ├── TabViewWidgets
│   │   │   └── ListingWidget.tsx
│   │   └── UpdateProfileScreen.tsx
├── tsconfig.json
├── yarn-error.log
└── yarn.lock
```

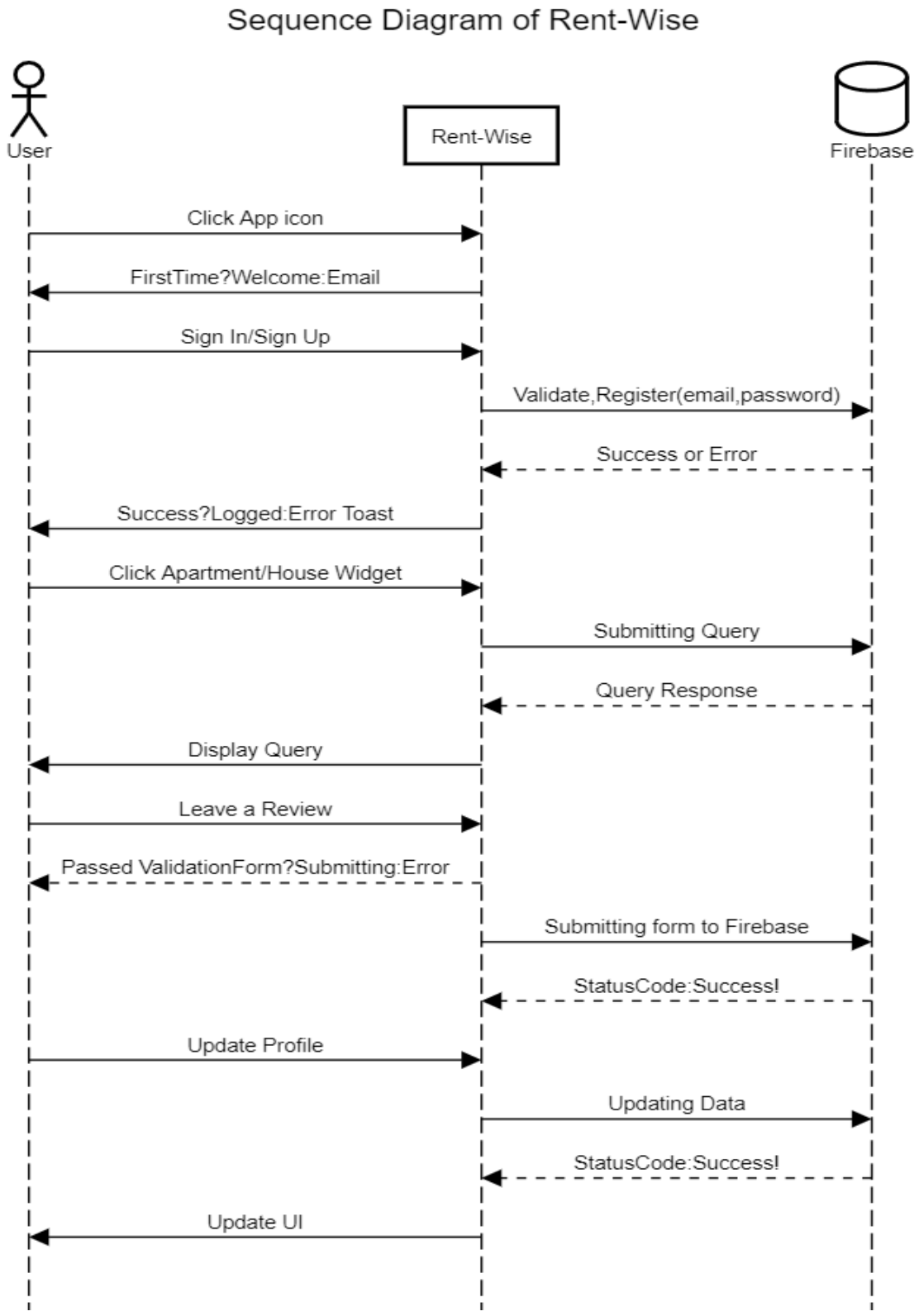


Figure 5: An image of Sequence Diagram of Rent-Wise[**sequenceDiagram**]

3.4 Java Environment

Navigate to Oracle's Downloading Page to download JDK and JRE respectively. You also need to set up the environment variables.

```
//use command "open ~/.zshrc" to open zsh config file
export JAVA_HOME=/Applications/Android\ Studio.app/Contents/jre/Contents/Home
export ANDROID_HOME=/Users/{USERNAME}/Library/Android/sdk
export PATH=$ANDROID_HOME/emulator:$PATH
export PATH=$ANDROID_HOME/platform-tools:$PATH
export PATH=$ANDROID_HOME/tools:$PATH
export PATH=$ANDROID_HOME/tools/bin:$PATH
```

Save the changes and use "source /.zshrc" to reads and executes commands from the file specified as its argument in the current shell environment.

3.5 Android Setup

3.5.1 React-Native-Vector-icons

React-Native-Vector-Icons is a React Native library for buttons, logos and navigation/tab bars. This library requires manual linking.

Navigate to `android/app/build.gradle` and add the following:`apply from: "../../node_modules/react-native-vector-icons/fonts.gradle"`

3.5.2 Firebase Setup

On the Firebase [**fbrn**]console, add a new Android application and enter your projects details. The "Android package name" must match your local projects package name which can be found inside of the manifest tag within the `apply from: /android/app/src/main/AndroidManifest.xml` file within your project.

To generate a certificate run `cd android && ./gradlew signingReport`. Download the google-services.json file and place it inside of your project at the following location: `/android/app/google-services.json`.

Replace parameters below with your own credentials in the google-services.json in your project.

```
/**
 * This const variable is a Firebase for Android device
 */
const credentials_android = {
  clientId: '445424652962-sdva3sv02520c3kp1epaq0259gjq4719.apps.googleusercontent.com',
  appId: '1:445424652962:android:56c78b909eae5d78793dfb',
  apiKey: 'AIzaSyCVE-An-KxL_6y4OfYRn7_xSXhKCt9sUEw',
  databaseURL: '',
  storageBucket: 'gs://rent-wise-4eaf7.appspot.com',
  messagingSenderId: '',
  projectId: 'rent-wise-4eaf7',
};
```

Figure 6: An image of a Firebase credential variable-Android

3.6 IOS Setup

react-native-vector-icons requires manual linking

Execute:

```
pod 'RNVectorIcons', :path => '../node_modules/react-native-vector-icons'
```

Add the following to Info.plist:

```
// Info.plist
// These are Fonts packages that require manual linking.
<key>UIAppFonts</key>
  <array>
    <string>AntDesign.ttf</string>
    <string>Entypo.ttf</string>
    <string>EvilIcons.ttf</string>
    <string>Feather.ttf</string>
    <string>FontAwesome.ttf</string>
```

```
    <string>FontAwesome5_Brands.ttf</string>
    <string>FontAwesome5_Regular.ttf</string>
    <string>FontAwesome5_Solid.ttf</string>
    <string>Foundation.ttf</string>
    <string>Ionicons.ttf</string>
    <string>MaterialIcons.ttf</string>
    <string>MaterialCommunityIcons.ttf</string>
    <string>SimpleLineIcons.ttf</string>
    <string>Octicons.ttf</string>
    <string>Zocial.ttf</string>
    <string>Fontisto.ttf</string>
  </array>
```

3.6.1 Firebase Setup

On the Firebase console, add a new iOS application and enter your projects details. The ”iOS bundle ID” must match your local project bundle ID. The bundle ID can be found within the ”General” tab when opening the project with Xcode. Download the GoogleService-Info.plist file. Using Xcode, open the projects /ios/projectName.xcodeproj file (or /ios/projectName.xcworkspace if using Pods). Replace parameters below with your own credentials in the GoogleService-Info.plist in your project.

```
--
/**
 * This const variable is a Firebase Key for IOS device
 */
const credentials_ios = {
  clientId: '445424652962-tupd11o34fcmj0rpvjpk9e35il79nc7d.apps.googleusercontent.com',
  appId: '1:445424652962:ios:43f15023f3058f9e793dfb',
  apiKey: 'AIzaSyCJ0y2OEZ_JY7YBFZmXrJo8aO6s2TFGUaY',
  databaseURL: '',
  storageBucket: 'rent-wise-4eaf7.appspot.com',
  messagingSenderId: '',
  projectId: 'rent-wise-4eaf7',
};

```

Figure 7: An image of a Firebase credential variable-IOS

3.7 Build the project

The first build time can take about 300s or longer. After that, the build time is rapid, and the hot update feature allows developers to see the changes instantly Build IOS and Android app.

```
// IOS:
cd ios //Navigate to IOS director
pod install //Install required packages for IOS
pod update //Update the configurations.
cd .. // Navigate back to the project directory
yarn ios // Build IOS project
// Android
yarn android //build application - Android
```

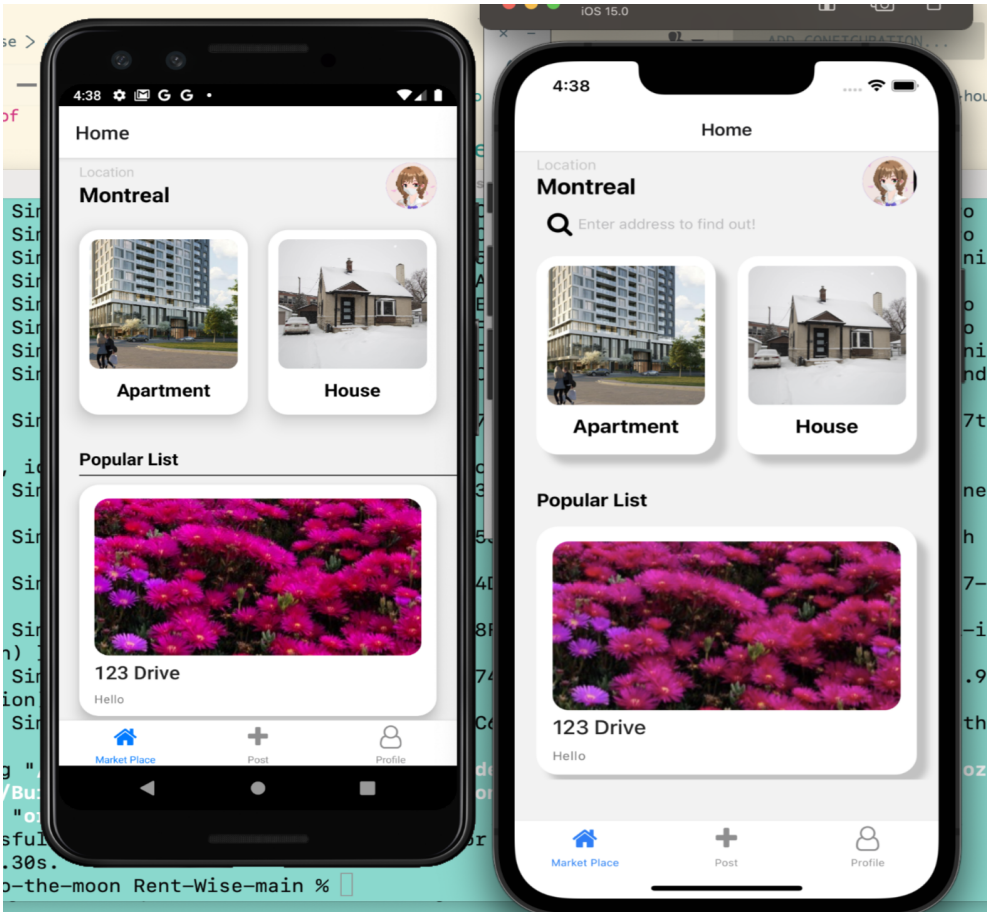


Figure 8: An image of a APP rent-wise on IOS and Android

4 Screens and Navigation

In 2022, the demand for mobile applications is rising, and people rely on them. People tend to order takeaway, online shopping, game on their mobile devices. Mobile Applications rarely consist of a single screen, as this does not satisfy users' needs. Then managing the presentation and transition of multiple screens is usually handled by so-called navigators.[**screens-navigation**] This project used React-Navigation to navigate to screens.[**react-navigation**]

4.1 Welcome Screen

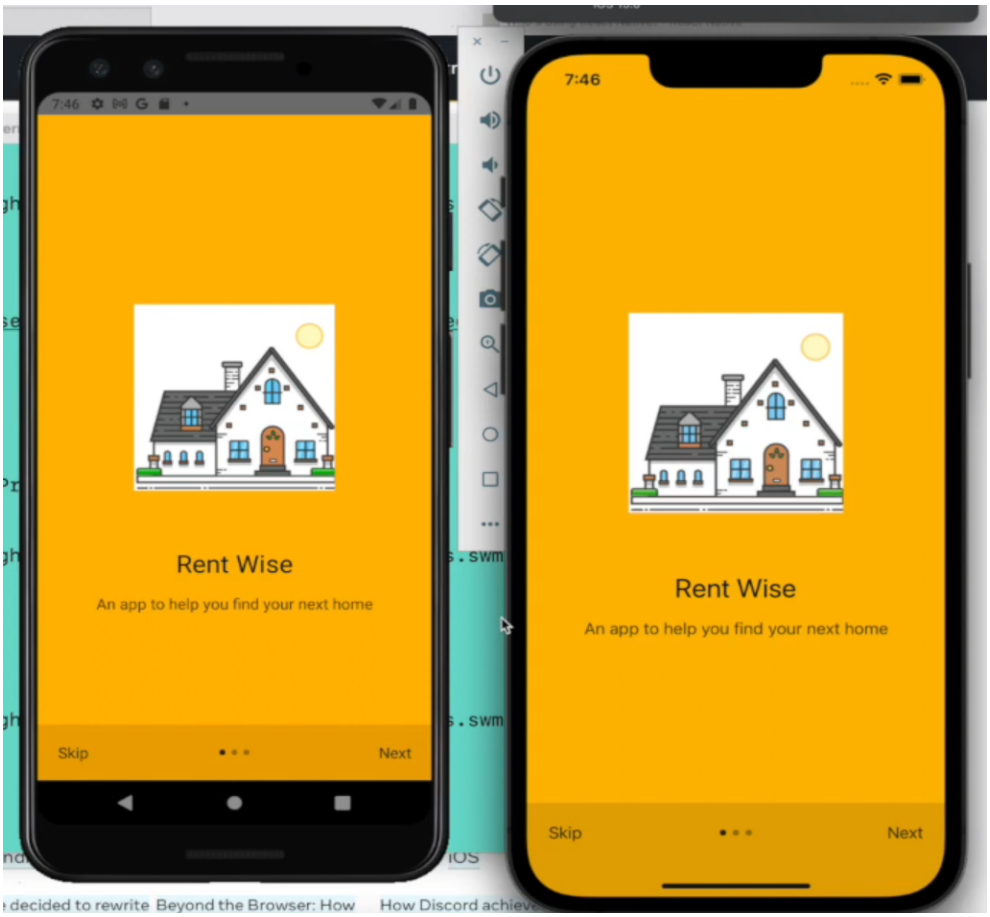


Figure 9: An image of a Welcome Screen rent-wise on IOS and Android

WelcomeScreen is a screen where users can briefly slide to the right or left to know the app. This screen will only appear when the user loads the software for

the first time. The specific implementation method is to query the key(”launched before”) stored in AsyncStorage(Local Storage). If it is the first time to load, the key should return a null value. When the key is a null value, a WelcomeScreen will be displayed and assign the key ”initial load” to true. So that next time app knows it is not the first time to launch and according to display login screen by using a ternary operator.

```
//App.tsx
       <RootStack.Navigator initialRouteName={initialLoad ? 'Welcome' : 'Login'}
```

```
//App.tsx
  useEffect(() => {
    const launchedBefore = AsyncStorage.getItem('launchedBefore');
    if (launchedBefore) {
      // do nothing
      console.log('launched before');
    } else if (launchedBefore === null) {
      AsyncStorage.setItem('launchedBefore', 'true').then((r) => setInitialLoad(true))
    ;
    }
  }, []);
```

4.2 Login Screen

File Path: `Rent-Wise/src/screens/LoginScreen.tsx/`

Rent Wise needs to know the identity of a user. Knowing a user’s identity allows Rent Wise to securely save user data in the cloud and provide the same personalized experience across all users’ devices. Firebase Authentication provides backend services, easy-to-use SDKs, and ready-made UI libraries to authenticate users to Rent Wise. It also supports authentication using passwords, phone numbers, and popular federated identity providers like Google, Facebook,

and Twitter. In this scenario, Rent Wise will use Password to authenticate users. [**firebase-auth**]

This section explains the concept and ideas of the design. There are two `TextInput` fields and two `Buttons`. The initial state of two fields are set to empty.

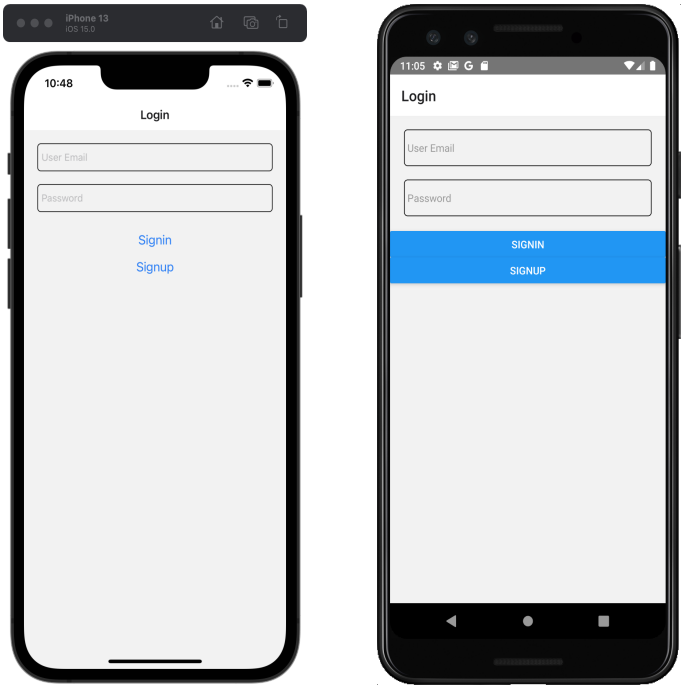
If a user needs to register an account or log in to an account, they must enter their credentials before continuing. When they click on register account, they can continue with asynchronous register_Account if the email is not in the Firebase database. An instance of User_Entity will be instantiated and added to the Firebase doc. A toast message of "Registered Successfully" will be right after the success. If the email is already in the database, an error toast message will be displayed.

When the user has entered credentials, clicked the login in button, and passed the form validation test, an asynchronous Firebase sign function `fireAuth .signInWithEmailAndPassword(userEmail, password)` will be called with credentials the user just entered.

Since the user has not yet used the UpdateProfile function, only the uid and email parameters will be temporarily added to the database. `avatar_url` and `display_name` will be updated to the database once user updates their profile.

When the user enters an account that does not exist in Firebase Authentication Service, the app will display an error toast message that tells the user that the account does not exist. If the account email matches; however, the password is wrong, the app will display an error message that tells the user the password is wrong. When the user registers successfully for the first time or logs in successfully, there is a success toast message.

```
1 // /src/store/entity.ts
```



(a) label 1 (b) label 2

Figure 10: 2 Figures side by side

```
export class UserEntity {
    avatar_url: string
    email: string
    display_name: string
    / * Corresponding to the google uid, uid will be assigned after login
     * @type {(any)}
     * @memberof UserEntity*/
    uid: any
}
```

4.3 Market Screen

File path: `Rent-Wise/src/screens/MarketScreen.tsx`

To understand the design of Rent Wise, it is better first to understand the product positioning. Rent Wise is software that helps to find a place to live.

The target users are college students and young people who have just started

working. A straightforward User Interface is required to attract their attention. Rent Wise's marketplace is divided into three parts. The first part is the search bar in the header, the second part is the two widgets on the top with the types of houses, and the bottom is a popular sliding list.

This thing wraps three components: Searchbar, House List Widgets, and Popular Items. The search bar horizontally aligns with an input text field. The user's strings in the input search field are the final search content that jumps to the search result screen later when he presses Enter key.

Componentizing code increases code repeatability and reduces code coupling.

Therefore, we use the lambda function to construct House List Widgets and Popular lists. This way, we can reduce the difficulty of maintaining the code in the future, as only the arguments passed into these lambda functions need to be taken care of.

House List Widgets:

The constant variable is a JSON file-like object containing the item's title and the path to the image and will be passed into the map function below. The map function will iterate over the itemList object and render the data to the screen. When the user needs to change the cover or title, the developer can update the itemList directly. The problems that direct code changes can cause are significantly reduced, and the user only needs to change the data in the itemList to make changes to the Cover Page and title.

```
1  const ListOptions = () => {
2    const itemList = [
3      { title: 'Apartment', img: require('../assets/apartment.jpg') },
4      { title: 'House', img: require('../assets/house.jpg') },];
5    return (
6      <View style={styles.listContainer}>
```

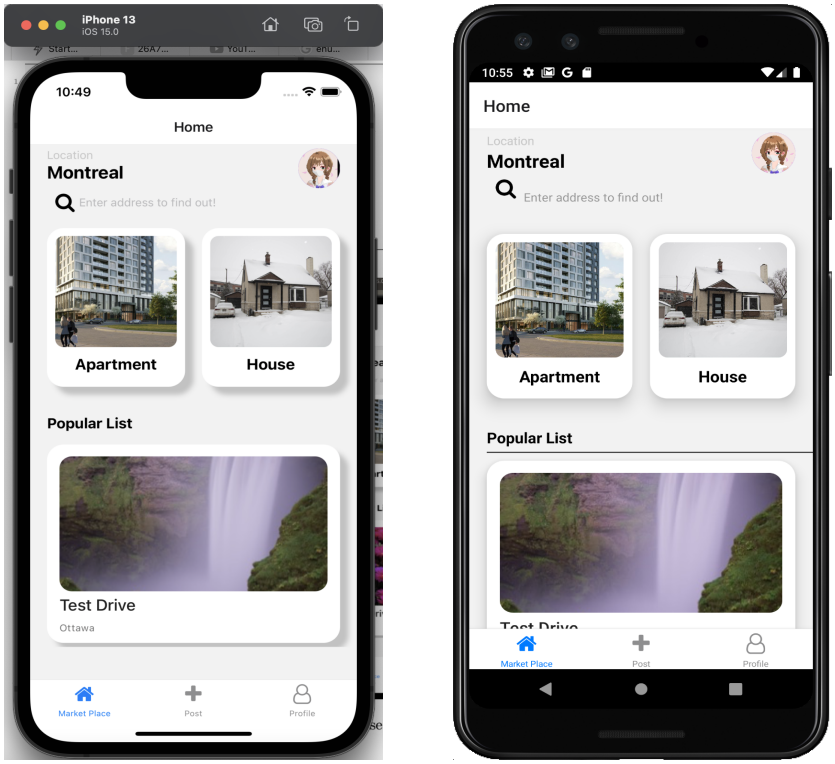
```
      {itemList.map((item, index) => (
        <Pressable key={index.toString()} onPress={() => navigation.navigate('
  Property List', { item, index })}>
          <View style={styles.optionCard} >
            <Image source={item.img} style={styles.optionCardImg} />
            <Text style={styles.optionText}>{item.title}</Text>
          </View>
        </Pressable>
      ))}
    </View>
  );
};
```

Popular List:

The idea of writing this house component is similar to that of listOption. This House Component needs to accept a house object as a prop. And then, React Native renders the house data to the screen. When the user clicks on the component, it jumps to a separate screen displaying the item in detail.

```
const HouseComponent = (props: { house: HouseEntity }) => {
  let { house } = props
  return (
    <Pressable onPress={() => navigation.navigate('HouseDetails', { id: house?.
  doc_id })}>
      <View style={styles.houseCard}>
        <Image source={{ uri: house.cover }} style={styles.houseCardImag} />
        <View>
          <Title style={styles.houseCardAddress}>{house.address}</Title>
          <Caption style={styles.houseCardCity}>{house.city}</Caption>
        </View>
      </View>
    </Pressable>);};
```

At first, only the order of the popular list is the order they are put in, but when the user clicks on the heart icon, there will be a code that compares the



(a) IOS (b) Android

Figure 11: Market Place Screen

two people like count to compare the items in the popular list. If an item's "like count" is bigger, its order will be higher than the other item.

```
// Sort the list of houses entities based on likes number
ret_house_list = ret_house_list.sort((val1, val2) => {
  return val1.like_count - val2.like_count
})
```

4.4 House Deatils Screen

There are two components to this screen. The first component has a card component and two sections. The first section is the background image and a heart icon. Users can hit the heart icon to like the item or hit it to dislike the item. Moreover, the second section is the city, address and average rating. The whole screen has two main components.

The second component is the review list, a flat list of reviews. Each item has the user's email, rating and comments.

```
1    //Assign average rate
2    let rate_count = store_house_rate_docs.filter(rate => rate.data().house_id == item
     .id).length > 0 ?
3    store_house_rate_docs.filter(rate => rate.data().house_id == item.id).length
4    : 1
5    let score_total = 0
6    store_house_rate_docs.filter(item => item.data().house_id == item.id).forEach(rate
      => {
7        score_total += rate.data().score
8      })
9    house_entity.avg_rate_score = score_total / rate_count
10   return house_entity
```

The average score of this item is based on the rating of this property by all users. This value is achieved by firstly using the ternary operator to get the number of all reviews. Meanwhile, using the for-each loop to get that item's total score when the house_id ==item_id.

Divide total scores by total number of reviews for that house and assign this variable to average rate score of house entity

```
1    liked ? await disLike() : await addLike()
```

When a user clicks on the heart icon in the upper right corner of the houseDetailScreen, the heart icon turns red, and when the user clicks again, the colour turns black.

Whenever the user clicks on the heart icon, the software calls async like() and async dislike(). These two methods are written along the same lines. The purpose is to add the Firebase database and delete the like database.

```
const disLike = async () => {
  toast.showLoading("cancel house like")
  let user_id = await LocalStorage.getUser_id()
  let house_id = house.doc_id
  //get the doc from Firebase
  let store_house_like_docs = (await FIRE_STORE_HOUSE_USER_LIKE_COLLECTION
    .where("user_id", "==", user_id)
    .where("house_id", "==", house_id)
    .get()).docs
  //delete like
  store_house_like_docs.forEach(async (doc, index) => {
    await FIRE_STORE_HOUSE_USER_LIKE_COLLECTION.doc(doc.id).delete()
    console.log("documents deleted ->", doc.id)
    if (index == store_house_like_docs.length - 1) {
      //update state
      setLiked(false)
      toast.hideLoading()
    }
  })
}
```

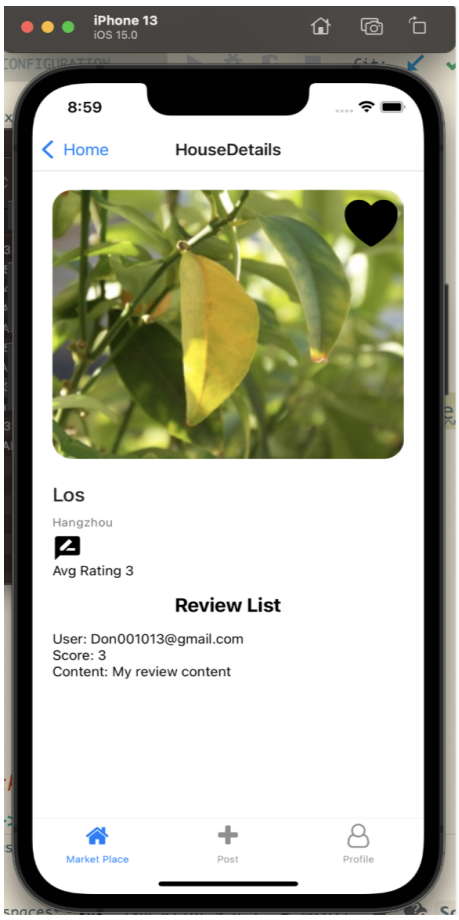
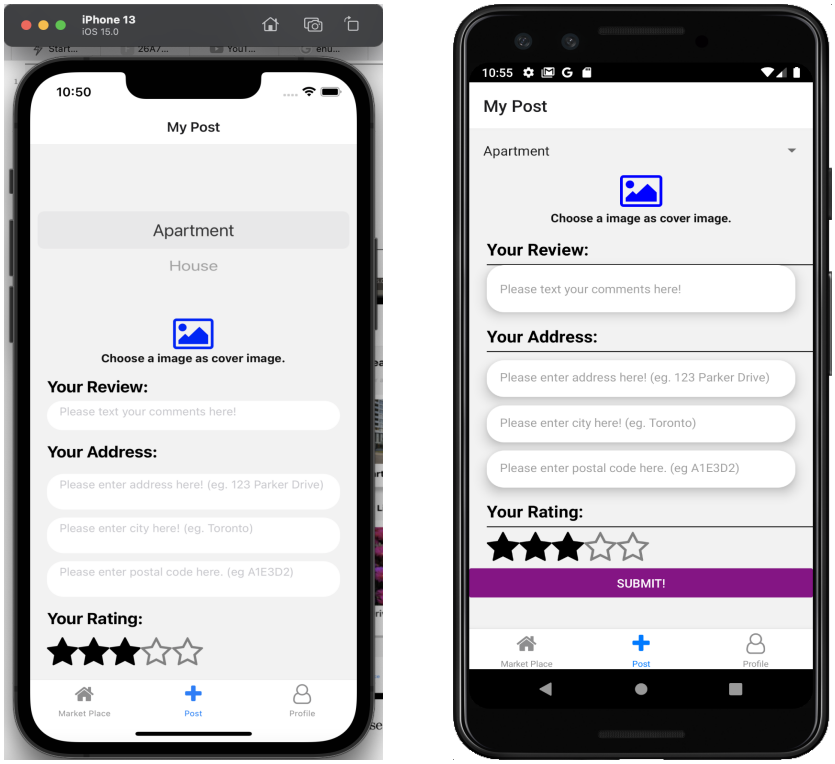


Figure 12: An image of a HouseDetailsScreen



(a) label 1 (b) label 2

Figure 13: Post Screen

4.5 Post Screen

File Path: File path: `Rent-Wise/src/screens/PostScreen.tsx`

This screen is where the user interacts with the software. The user first needs to choose whether it is a house or an apartment through a drop-down menu. This choice determines the properties of the item in the Enumeration.

```
1  //Enum
2 export enum HouseType {
3     apartment = "apartment",
4     house = "house"
5 }
```

ChooseFromLibrary used the package 'react-native-image-crop-picker'. It selects an image from Camera Roll and updates the Select image in the Post Screen. When the user has entered all the fields, ValidateInput() checks if the

fields are empty or not. If the validation test passes, he/she clicks the submit button to send data to Firebase.

```
    const validateInput = () => {
       if (title.val.length == 0) {
         toast.show({ message: "comment is empty" })
         return false
       }
       if (address.val.length == 0) {
         toast.show({ message: "address is empty" })
         return false
       }
   ...
```

Upload all the fields that User just entered to Firebase is divided into two parts. The first part is to upload images to the Firestore. This step requires using the previous step, ChooseFromLibrary images first, getting the image in the phone address and uploading it to the Cloud Firesotre. Then the user input data to generate the corresponding instance and then upload these entity objects to the Firebase database.

When a user successfully uploads information to Firebase, there will be a toast showing the success, and all the fields entered by the user will be cleared for his subsequent posts.

4.6 Profile Screen

File path: `Rent-Wise/src/screens/ProfileScreen.tsx`

Profile Screen has two parts. The first part is the profile screen which will be discussed in the next section, and the second part is a nested tab view.

The app initializes two entities array:

```
const [reviewhouseList, setReviewhouseList] = useState<Array<HouseEntity>>([])
const [favouriteHouseList, setFavouriteHouseList] = useState<Array<HouseEntity>>([])
```

Both of these two flat lists will fetch HouseEntity from Firebase. However, the filter to get the desired result is different. The nested tab view uses a ”react-native-tab-view” package with two tabs. The first tab is a flat list with the current user’s viewing history. The second tab is a flat list with items that the user likes. Each time the user likes, dislikes, and views an item, React Native re-renders the screen by calling the FetchHouseList function and updating the items in the flat list.

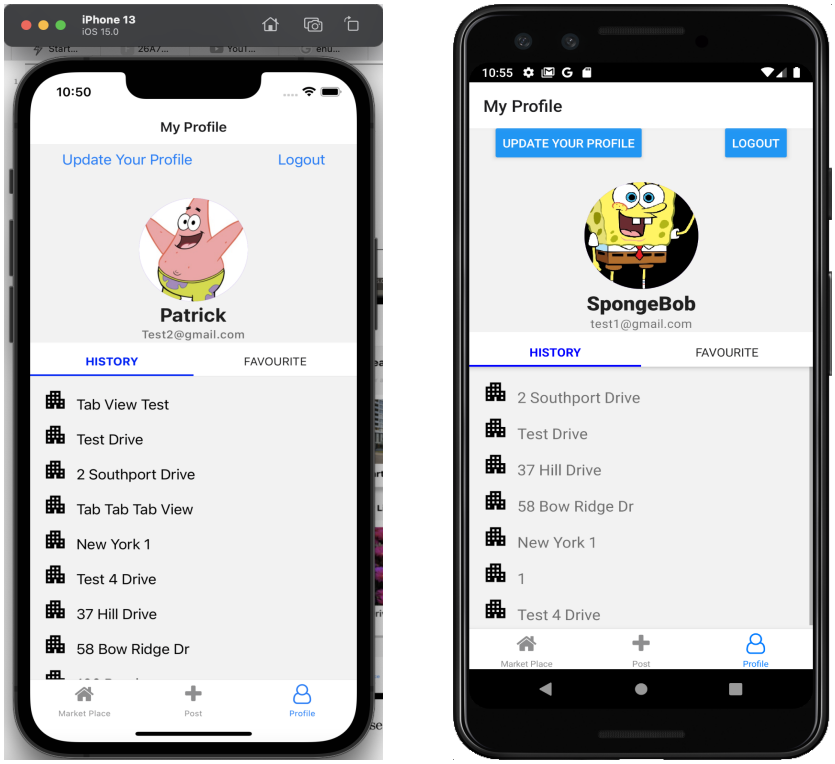
```
    //PART OF FETCHHOUSELIST
    const fetchHouseList = async () => {
    let user_id = await LocalStorage.getUser_id()

    let store_house_like_docs = (await FIRE_STORE_HOUSE_USER_LIKE_COLLECTION
        .where("user_id", "==", user_id)
        .get()).docs

    let store_house_views_docs = (await FIRE_STORE_HOUSE_USER_VIEWS_COLLECTION
        .where("user_id", "==", user_id)
        .get()).docs

    let house_id_list_views = store_house_views_docs.map(item => {
      return item.data().house_id
    })

    let house_id_list_like = store_house_like_docs.map(item => {
      return item.data().house_id
    })
    ...
}
```



(a) IOS (b) Android

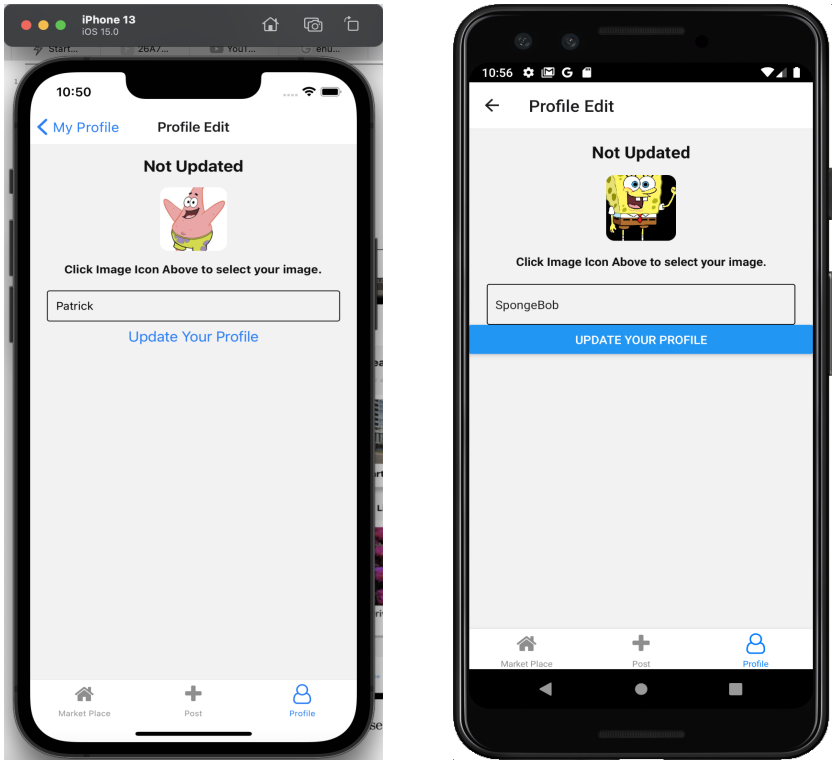
Figure 14: Profile Screen

4.7 UpdateProfile Screen

Initially, Profile Screen has two components: avatar and name. However, the profile will update his/her user name when the user successfully updates his/her profile by clicking Update Profile at the top left corner.

This validateEmail is an anonymous function that takes an email as parameter and use regular expression to validate the email address.

```
1       const validateEmail = (email) => {
2       return String(email)
3           .toLowerCase()
4           .match(
5               /^(([^<>()[\]\\.,;:\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,}))$/
6           );
7   };
```



(a) IOS (b) Android

Figure 15: Update Profile Screen

Once the user enters the new user name and passes validation checks, the application calls this submitUpdate() function. The app will push the image to the local storage and Firebase. The reason for uploading the same images to Firebase and local storage is that local storage on the phone avoids many Firebase calls, saving resources. When the upload succeeds, a toast message of success will be displayed and return to be previous screen.

getUser() will get the user's information from Firebase and update the user's user update message with the Firebase data

```
1   const getUser = async () => {
2       let user = await fetchUserFromRemote()
3       setName(user.display_name)
4       if (user.avatar_url?.length > 0) {
5           setImage_select(user.avatar_url)
6           setSource({ uri: user.avatar_url })
7       }
```

5 Data Persistence

5.1 Advantages and Disadvantages of Relational Database

Persistence means saving data (e.g. data objects in memory) to a persistent device, i.e. to a storage device (e.g. hard disk, Cloud Server) forever. The main application of persistence is to store in-memory data in a relational database, but it can also be stored in complex disk files, XML data files. This project uses a No-SQL database, which is also meant to store data permanently but with a different approach.

There are many advantages of using a relational database. Shared standard SQL query syntax makes it very convenient to operate the relational database, supports complex queries such as join, SQL + 2D relationship is the most incomparable advantage of the relational database, ease of use is very close to the developer—data persistence to disk, no risk of losing data, support for massive data storage. The most commonly used relational database product MySql, Oracle server performance is excellent, stable service, usually minimal downtime exceptions. ACID feature(Atomicity, Consistency, Isolation, and Durability) is an essential feature that a relational database supports, which means the database can maintain consistency between the data, which is one of the significant reasons to use this kind of database. For example, Tom transfers 200 Dollars to Jim and Tom's bank account should deduct 200 dollars immediately. Meanwhile, Jim's account should have that 200 dollars deducted from Tom's bank account and must be successful or fail simultaneously; otherwise, it will cause the user's capital loss.[**sql-diff**]

Nevertheless, in some ways, relational databases also have some disadvan-

tages. Relational databases store data by row. Even if only for one of the columns, the entire row of data will be read from the storage device into memory, resulting in a high IO burden. Data consistency is the core of relational databases, but the cost of maintaining data consistency is also very high. We all know that the SQL standard defines different isolation levels for transactions, from low to high is, read uncommitted, read committed, repeatable, serialized. The lower the transaction isolation level and more concurrent exceptions that may occur, generally speaking, the more concurrency that can be provided. Then in order to ensure transaction consistency, the database needs to provide concurrency control and fault recovery two techniques. The former is used to reduce concurrency exceptions. The latter can ensure that the transaction and database state will not be destroyed when the system is abnormal. For concurrency control, the core idea is to add locks, whether optimistic locks or pessimistic locks, as long as the higher the isolation level provided, the worse the read and write performance is bound to be.[**sql-proAndcon**]

5.2 What is No-SQL database?

NoSQL databases are also called ”non-relational,” ”NoSQL DB,” or ”non-SQL” to emphasize the fact that they can handle large amounts of fast-changing unstructured data in a different way than relational (SQL) databases with rows and tables. (SQL) databases with rows and tables emphasize that they can handle large amounts of rapidly changing unstructured data differently than relational (SQL) databases.

NoSQL technologies have been around since the 1960s under various names; their popularity has proliferated as the data landscape has changed and developers

have needed to adapt to handle the massive amounts of data generated from the cloud, mobile devices, social media, and big data.[**nosql-intro**]

5.2.1 Different typs of NoSQL database

Key-Value NoSql:

Redis and MemCache are the representatives of the Key-Value Stores NoSQL database. Moreover, Redis is the most widely used NoSQL in KeyValue type NoSql. The most significant advantages are down to two points.

Advantages:

- Memory-based data, high read/write efficiency.
- KV type data, time complexity O(1), fast query speed.

Disadvantages:

- It can only look up V based on K, but not K based on V.
- The only way to query is to have data redundancy, but this is a massive waste of storage space.
- Memory is limited and cannot support massive data storage.
- Similarly, because the storage of KV type NoSql is based on memory, there will be the risk of losing data.[**redis**] [**redis-1**]

Document-based NoSql:

Document-based NoSQL refers to NoSQL that stores semi-structured data as documents. Document-based NoSQL usually stores data in JSON or XML

format, so document-based NoSQL has no Schema, and because of the characteristic of no Schema, we can store and read data at will, so the emergence of document-based NoSQL is a solution to the problem of inconvenient expansion of relational database table structure. Google Firebase and MongoDB are examples of document-based NoSQL.

Advantages:

- No predefined fields, easy to extend fields.
- Compared to relational databases, read and write performance is superior. Queries that hit secondary indexes are no slower than relational databases and faster for queries for non-indexed fields.

Disadvantages:

- Realtime synchronization issues.
- Realtime synchronization issues.
- Data migration problems.
- Limited querying capabilities. [**firebase-downside**]

Compared to relational databases, read and write performance is superior. Queries that hit secondary indexes are no slower than relational databases and faster for queries for non-indexed fields.

5.2.2 Firebase

Firebase is a Google product that helps developers easily build, manage and grow their applications. It helps developers to build their applications in a faster

and more secure way. There is no programming required on the Firebase side, which makes it easy to use its features more efficiently. It provides services for android, ios, web and unity. It provides cloud storage. It uses NoSQL database to store data.[**firebase001**]

5.3 User Entity

The behaviour of user entities is to ensure that all user information is carefully stored. When logging in for the first time, firebase only stores the uid and email information, but when the user sends a request to change the email address, the avatar_url is updated along with the email by calling updateUserToRemote().

```
export class UserEntity {
    /**
     * User Entity, since the google auth module is limited, so we provide a entity to
     extend it
     */
    avatar_url: string
    email: string
    display_name: string
    uid: any
}
```

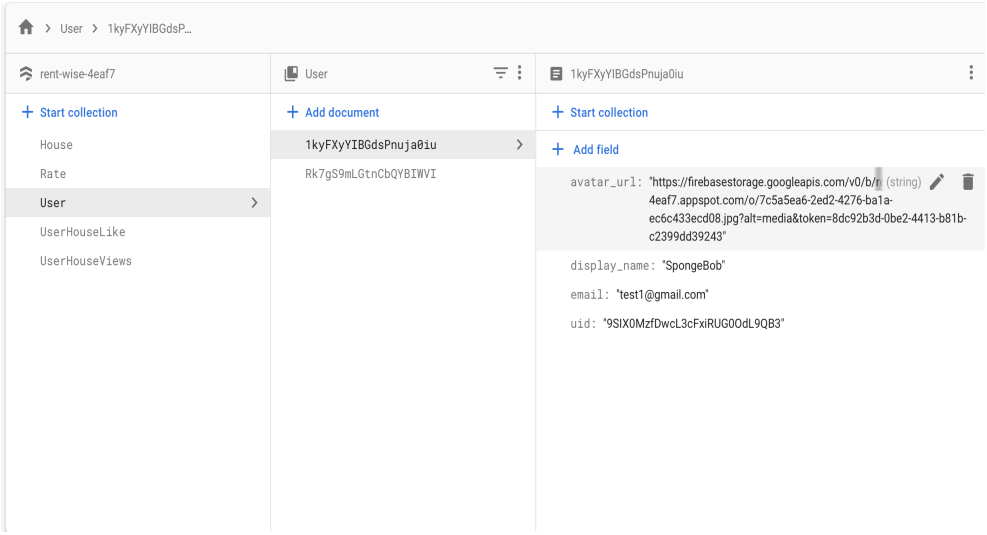


Figure 16: An image of user entity in the console

5.4 Rate Entity

When the user posts a new review, the rated entity is linked to a logged-in user by uid. The user's uid can be considered as the primary key in the User Entity if it uses the term from the relational database. When Rate Entity references this uid in its table, it becomes a foreign key connecting two tables.

```
/**
 * RateEntity
 */
export class RateEntiry {
    user_id: any
    score: number
    content: string
    house_id: string
    email: string

}
```

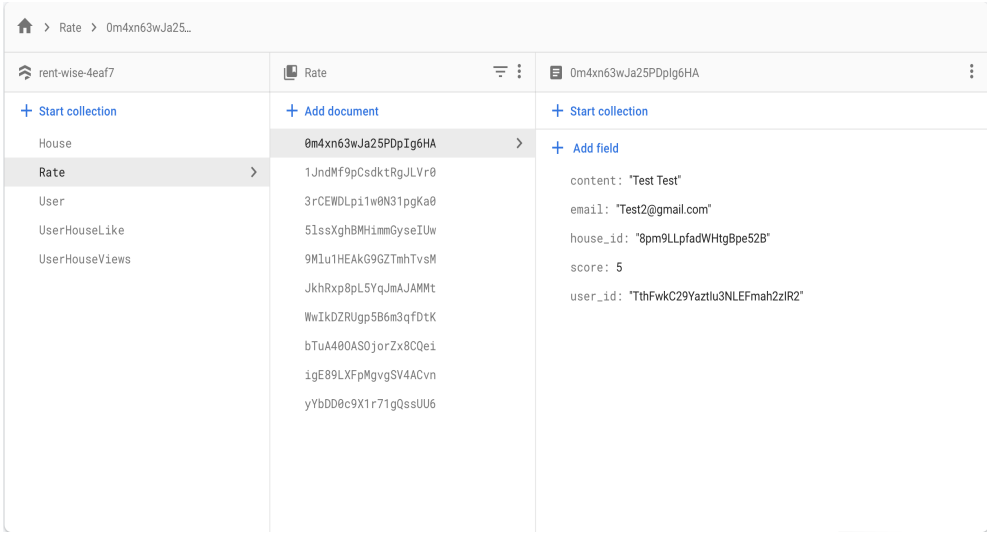


Figure 17: An image of rate entity in the console

5.5 House Entity

HouseEntity is the most crucial entity in Rent Rise, and almost all other entities operate on this entity. The specific implementation of the function has been explained in the previous chapter.

```
/**
 * House(Apartment or House) Entity
 */
export class HouseEntity {
    doc_id: any
    address: string
    city: string
    post_code: string
    type: HouseType
    cover: string
    like_count: number
    avg_rate_score: number
    like: boolean
}
```

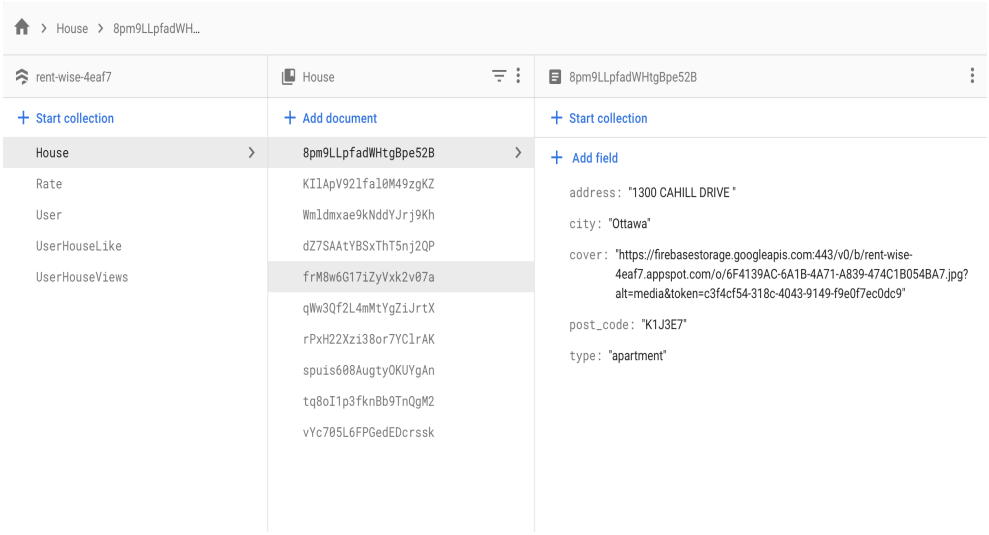


Figure 18: An image of house entity in the console

5.6 User House View Entity

The user screen keeps track of a user's browsing history, and whenever the user clicks on an item that has never been viewed before, the app appends the item to the list and updates the UI state immediately by calling usingFocusEffect() function.

```
export class UserHouseViews {
    user_id: string
    house_id: string
}
```

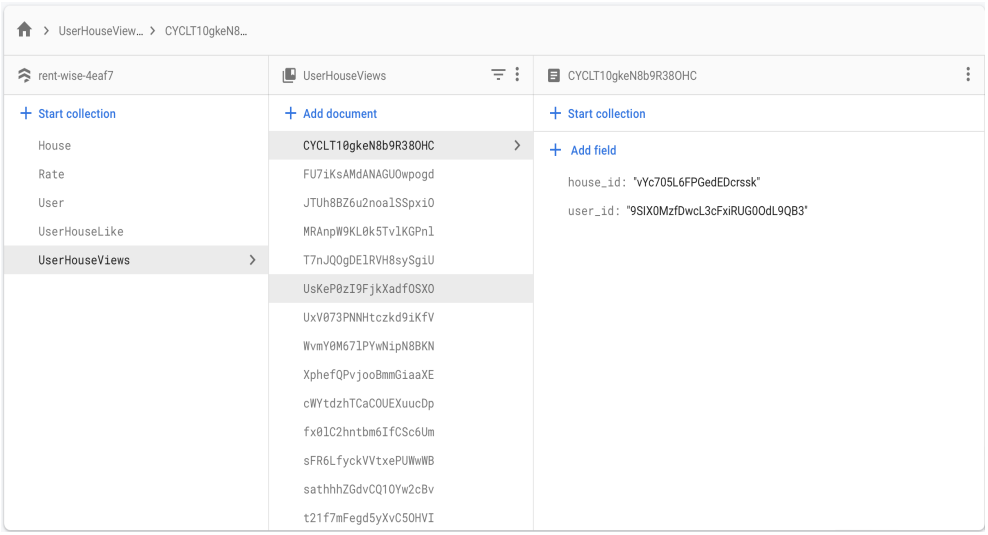


Figure 19: An image of User House entity in the console

5.7 User House Like Entity

User-House-Like-Entity keeps track of like counts a user has on his/her likes and displays this data through a flatlist. Whenever a user likes or dislikes a new item, the screen will immediately refresh the UI and update the screen by calling the function useFocusEffect().

```
/**
 * UserHouseLikeEntity
 */
export class UserHouseLikeEntity {
    user_id: any
    house_id: any
}
```

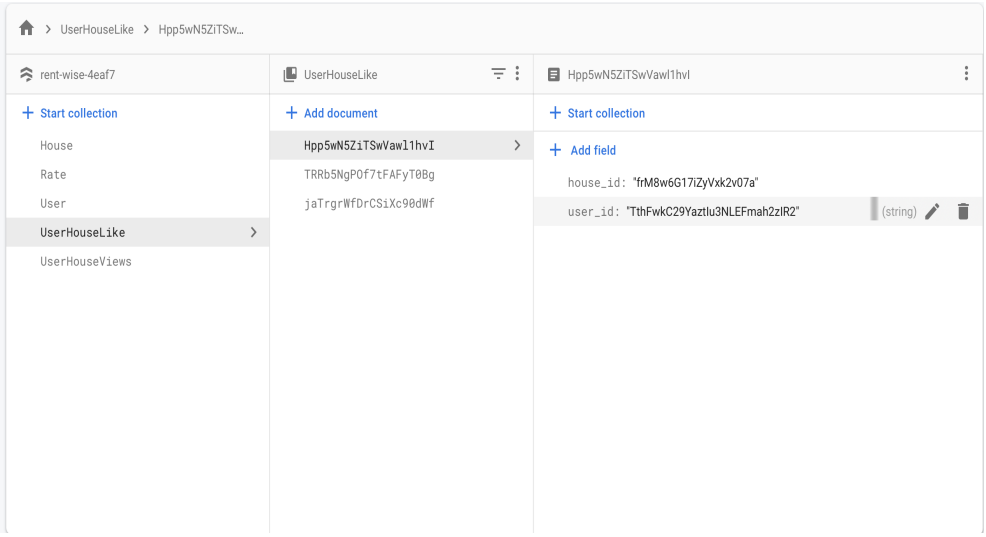


Figure 20: An image of User Like count entity in the console

6 Collaborative Recommendations

6.1 What is Covariance?

According to this short paragraph in this website[**covariance**], "In mathematics and statistics, covariance is a measure of the relationship between two random variables. The metric evaluates how much – to what extent – the variables change together. In other words, it is essentially a measure of the variance between two variables. However, the metric does not assess the dependency between variables."

The range of the correlation value is from -1 to 1. The closer the value is to 1, the stronger this correlation is. Adjusted Cosine Similarity,

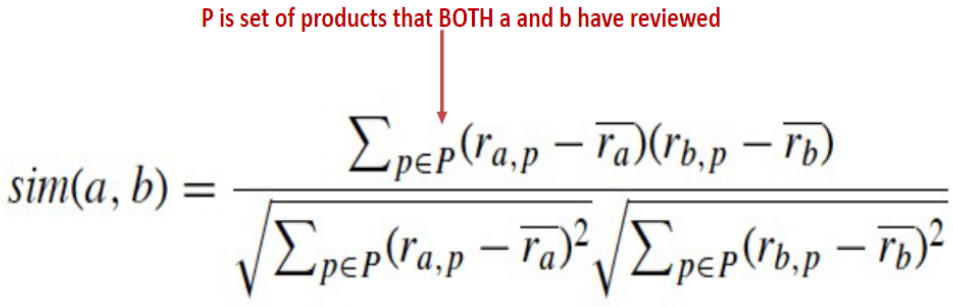


Figure 21: An image of Pearson's Correlation Coefficient Fomula [**4601**] [**recommenderSystem**]

How do we predict Patrick's Item5?

An Example Table:

	Item1	Item2	Item3	Item4	Item5
Patrick	5	3	4	4	?
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

There is a table of ratings, and we want to predict the missing value, we should firstly find the similarity score between two columns.

$$\overline{Patrick} = \frac{5+3+4+4}{4} = 4$$

$$\overline{User1} = \frac{3+1+2+3+3}{5} = 2.4$$

$$\overline{User2} = \frac{4+3+4+3+5}{5} = 3.8$$

$$\overline{User3} = \frac{3+3+1+5+4}{5} = 3.2$$

$$\overline{User5} = \frac{1+5+5+2+1}{5} = 2.8$$

There are 5 rows and neighbourhood size of 2. Initially, $\overline{Patrick}$ is $\overline{r_a}$ and $\overline{User1}$ is $\overline{r_b}$. The rest of Users will play as r_a and r_b later. Substitute values of row 1 and row 2 to the formula above

$$\frac{(5-r_a)*(3-r_b)+(3-r_a)*(1-r_b)\ldots(4-r_a)*(3-r_b)}{\sqrt{(5-r_a)^2+(3-r_b)^2\ldots}*\sqrt{(3-r_b)^2+(1-r_b^2)\ldots}} \approx 0.83918136$$

Apply the formula to the rest of the columns:

$$PCC_P, User1 \approx 0.83918136$$

$$PCC_P, User2 \approx 0.60633906$$

$$PCC_P, User3 = 0.0$$

$$PCC_P, User4 \approx -0.7680954$$

$$pred(a,\ p) = \overline{r_a} + \frac{\sum_{b \in N} sim(a,b) * (r_{b,p} - \overline{r_b})}{\sum_{b \in N} sim(a,b)}$$

Figure 22: An image of Prediction Fomula [**4601**] [**recommenderSystem**]

Since the neighbourhood size is 2, the first 2 items will be selected for Prediction calculation.

$$4 + \frac{(0.83918136 * (3 - 2.4) + 0.60633906 * (5 - 3.8))}{0.83918136 + 0.60633906} \approx 4.85$$

Hence the predicted value for Patrick's Item 5 is 4.85

	Item1	Item2	Item3	Item4	Item5
Patrick	5	3	4	4	4.85
User1	3	1	2	3	3
User2	4	3	4	3	5
User3	3	3	1	5	4
User4	1	5	5	2	1

6.2 Integration with the Application

Since the algorithm happens to be based on estimating each item's rating, Rent Wise is also operating based on the reviews posted by users. Potentially the app could integrate with this User-Base recommending algorithm, but because of the time constraints, there is no better way to find an efficient way to collect accurate data from simulating the environment, and there is no way to apply the algorithm to this software.

6.3 Code Implementation:

Github Repository

7 Conclusion

7.1 Outcome

According to the project proposal, all Expected Features were implemented. However, due to time constraints, additional features were not completed.

Rent Wise is a platform that connects neighbourhoods. Users can post reviews and help the neighbourhood find 'the one' dream home on IOS and Android Platforms.

Users could use their email to register accounts and update their in-app profiles with the help of Firebase Authentication API. After logging in, users could post reviews or get reviews of their dream home. Rent Wise also keeps track of the browsing history under the history-tab-view so that users can still find it after working on something else.

After testing on different devices, Rent Wise accomplished the expected goals and gave good feedback to users.

7.2 Limitations

The software itself is not a limitation in this type of social software, but the core of this kind of social app is the recommendation algorithm. Since only the successful recommendation of houses/apartments to the user would be the key to keeping the users stay at this app and recommending this app to his/her friends.

Due to the time constraints and course load, the User-Base Recommendation algorithm was studied yet implemented

7.3 Future Improvements

The additional plan is to complete user chat and integration of the recommendation algorithm when there is enough time. However, I could not complete it on time.

Nevertheless, all the required requirements have been completed and are very similar to the prototype of the project proposal.

Extra User chat and integration of recommendation algorithms have not been completed. Hopefully, these two features will be completed in the future.

Some components do not look good enough, but it is a long process that takes time, so some fundamental components are used now. For example, <Button> is a minimal component in React Native. Developers usually use <Pressable> wrapper to detect the various stage of press interactions on its defined children components.[**react-pressable**]

A Appendix

Lists are easy to create:

- /rent-wise-app/blob/main/index.js
- /rent-wise-app/blob/main/src/App.tsx
- /rent-wise-app/blob/main/src/store/entity.ts
- /rent-wise-app/blob/main/src/screens/HouseCollectionScreen.tsx
- /rent-wise-app/blob/main/src/screens/HouseDetailsScreen.tsx
- /rent-wise-app/blob/main/src/screens/HouseSearchResult.tsx
- /rent-wise-app/blob/main/src/screens/LoginScreen.tsx
- /rent-wise-app/blob/main/src/screens/MarketScreen.tsx
- /rent-wise-app/blob/main/src/screens/PostScreen.tsx
- /rent-wise-app/blob/main/src/screens/ProfileNavWidget.tsx
- /rent-wise-app/blob/main/src/screens/ProfileScreen.tsx
- /rent-wise-app/blob/main/src/screens/UpdateProfileScreen.tsx
- /rent-wise-app/blob/main/src/screens/TabViewWidgets/ListingWidget.tsx
- /rent-wise-app/blob/main/src/firebase/firebase-auth.ts
- /rent-wise-app/blob/main/src/firebase/firebase-house.ts
- /rent-wise-app/blob/main/src/components/toast.tsx
- /rent-wise-app/blob/main/src/common/constantsDefine.ts

- /rent-wise-app/blob/main/src/common/dateUtils.ts
- /rent-wise-app/blob/main/src/common/localStorage.ts