**FlatFinderSG**

**Requirement analysis and design.**

Our mission is to streamline the process of buying and selling home properties, so as to make the transaction for buyers and sellers very efficient and enjoyable.

The main functions for our application include:

- View listing

- Add listings.

- Search for property based on the address,

- we can compare the listing - compare the listing price against other resale properties in the neighborhood and with the same number of rooms, the pricing.

- Filter - We can filter the listing by the number of bedrooms and neighbourhood.

- And we have calculated grants where users can find out what kind of grants they're eligible for and how much they're eligible for.

**Use case diagram**

This is our Use case diagram. Some of our prominent functions that you look out for are comparability, and sending emails, which my group mates will cover later in more detail.

**System design**.

These are the tools we use in our application development. These are the APIs that we use.

For system design, we're using MVC format, in which we have a model, view and controller.

We will be using a total of five controllers. So each of them will be handling their own kind of data.

Next, why do we use MVC? Because MVC will help us to streamline the process of data communication between the two components, and also help us to encapsulate all requests between different components. In this way, it helps us to update the app for the future, and also if there are any errors, the MVC help us to easier check up.

**Live demo**

We'll start off by logging in. When you log in, this is the page that we'll be at.

One of our few main features is the searching feature. For our search feature, you'll be done using the property name. The property names are the ones in red over here as you can see, it automatically filters the properties you're looking for.

For example, if you're looking for property in Yishun. Enter Yishun, then the rest will disappear. We have also added in automatic filtering for the number of bedroom rooms to allow users to more easily search for the kind of property that they want. For example, if you are searching for a 4-room flat, you can just click on 4-rooms and the rest will disappear. Same for the 5-room, three-Room.

This is the Favorite function. If you like this property, and in the future, you want to refer to it in the future. We can click on the favorite button and go to your favorite pitch, and then it'll be over there. Suppose in the future, you don't like the Yishun property because it costs too many problems, then you can just click, and I delete it.

We've covered the feature of your favorite property. Now let's move on to the individual pages. Let's take the example of Yishun. We'll be presenting a few features on this page. First, we're using the Google Maps API to input the postcode and get a pinpoint location. You can click on the pin to open it in Google Maps. We're displaying the Google Map pinpoints.

The second feature on this page is the grant calculator, part of this widget. We'll be having a slide to display the logic and how to get between different grand slots later.

The other feature is the compare feature. This is one of our most innovative features. We make a query to the API data, which is from data.gov, where they show all the flat prices, the neighborhood, the number of rooms, etc. The compare operator makes a query and filters all the data listings by Yishun and three-room flats. The price you see is the price of the selected listing, while the red color is the average price calculator of all the API data listings. This allows users to make a decision in terms of whether they want to buy the property and whether it's a good price, as they can compare the current listing price with the average prices of all available data.

Moving back to the grant calculator, there are a lot of features going on here. The counter will automatically make a call to the user once they log in and fill in some data. The grant amount you're eligible for depends on the information you provide. We call this feature the grant Calculator, which is closely linked to your profile.

If I change my status from married to single, the grant available will be different. This is because single individuals who are below 35 years old can only apply for a specific grant type.

If I change my age to 20, for example, the app will automatically refresh and show

that I'm not eligible for the grant anymore. Conversely, if I turn 35, I can change my profile to reflect my new status, and the app will display the available grants for singles, which is $40,000.

Moving on, this app also has functions for sellers. Aside from browsing properties, you

can also list your property for sale.

Our app also has a postcode feature that automatically generates your address based on Google Maps. For example, if I want to add my property on this app, I can simply input my postcode, and the app will generate my address. From there, I can add more details about my property, such as the number of rooms and the dimensions.

Once I add my property, it will appear on the My Property page, which is closely linked to the Add Property feature. This page only displays properties edited by a specific user, so if you log in with a different account, you won't see any of them.

Finally, there's a Delete Property feature on this page, which allows you to remove any unwanted listings.

This framework is, Oh yeah. So actually we've already presented almost everything. Everything, all the features. Yeah. So actually the last part was just to show you the registration. So basically the users, new users can register a new account and they can log in successfully as, as shown by privacy.

When we register, you can see the email text to the account of them.

**Good Software Design Principles**

Let's discuss the software design principles that we have applied in our application.

Firstly, we have utilized the strategy pattern in combination with the factory method to handle different forms of listing queries. For example, in our search feature, we can query all the listings or search based on the number of rooms. Instead of using the "new" keyword to add more types of queries, which can be cumbersome and error-prone, we have implemented the factory method pattern. This pattern allows us to create objects dynamically, making development and scalability much easier. It also enables us to easily add more types of search queries in the future, such as querying based on the listing price.

Additionally, we have implemented the dependency injection principle in our app development, which is a common practice in modern applications. Since we are using Flutter as our platform, we rely on external libraries or packages to add functionality or features to our application. By using import statements to access functions and methods from these external packages, we can easily mock or replace dependencies, ensuring the quality and reliability of our application.

In terms of non-functional design, we have adopted clean UI and ease-of-use principles based on human-computer interaction principles. We have implemented a navigation bar to make user navigation easier, instead of complex navigation menus. We have also utilized the MVC (Model-View-Controller) framework to separate different classes and components in our application. For security, we have used an authentication controller and Firebase's own authentication system for our backend. Additionally, we have used regular expression functions to perform input validations, such as checking whether the input value is within a certain range or contains only letters and numbers.

Now, I will pass the time to Yupei to elaborate on our approach to good traceability in our application.

**Good Traceability**

We prioritize completing a comprehensive design before implementing any features in our application to ensure traceability. This is especially crucial for sequence diagrams, which depict the flow of data and control between components, allowing us to build our system design on class diagrams and other tools.

Let's take a closer look at two of the most critical functions in our application: the compare and grant calculator functions. When a user clicks on the compare button in the UI, the compare component sends a command to the controller to execute the compare function. However, before doing so, it retrieves data from the property, including the location and number of bedrooms, from property B. Then it calculates and displays the comparison in the UI, connecting the compare components with the enumerations and entities across the system.

Moving on to the grant calculator, when a user enters the grant in the individual property UI, the system retrieves profile details such as the user's information, age, and marital status from the grant entity. The grant entity then determines whether the user's profile is complete. If it's incomplete, the system prompts the user to edit their profile. However, if the profile is complete, a command is sent to the grant controller to process the grant.

To ensure efficient development, we follow the scrum methodology, conducting scrum meetings every two weeks to plan our process, document minutes, and design a product backlog with a list of requirements and features to integrate during implementation. We also hold one to two weekly sprints to align with the lab schedule and submissions. During each sprint, we decide on tasks for the current sprint and plan for the next one, and also conduct reviews and product refinement sessions.

For traceability, we utilize Google Drive and GitHub for documentation and version control. During implementation, we create individual branches for each team member, allowing for debugging and merging to upgrade versions. We use UML for design, and tools like VS code and Android Studio for simulation, ensuring a well-organized and traceable development process.

**White box and Black box Testing**

So next, we will introduce the white box and black box testing. Moving on to the black box testing, we utilized manual inputs to test various cases related to the search function. Since our input is discrete, we do not have any boundary values. We determined the class and tested a few inputs on a list of properties to evaluate the search function feature.

As for white box testing, we focused on two features: the resale data and the grant calculator. For the resale data, we performed black box testing to check if it can fetch data from the API via a HTTP get request, filter the data by the number of rooms and neighborhood, and calculate the average price based on the prices of the properties. We also analyzed the control flow chart and identified three basis paths, removing any impossible paths. We formulated test cases and evaluated the real execution path.

Similarly, for the grant calculator, we analyzed the control flow chart, identified basis paths, and formulated test cases to evaluate the real execution path. The grant calculator involves if condition checks based on user profiles, such as marital status, to determine eligibility for grants.

In conclusion, we performed black-box and white-box testing to thoroughly evaluate the functionality and control flow of our system. Thank you.