Software Requirements Specification

for

REvaluate

Version 1.6

Prepared by Group Z57

Rishabh Alexander John (Leader) (U2121475H)
Lim Jin Feng, Alexus (Asst. Leader) (U2121689H)
Lee Xuan Wei, Augustine (U2121778E)
Yau Xing Zhe (U2121704L)
Lavanya Sharma(U2120324E)
Tan Jia Ze(U2122410B)

Nanyang Technological University

14th April 2023

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Rishabh	14/04/23	Drafting documentation	1.1
Alexus	14/04/23	Drafting documentation	1.2
Augustine	14/04/23	Drafting documentation	1.3
Jay	14/04/23	Drafting documentation	1.4
Lavanya	14/04/23	Drafting documentation	1.5
Jia Ze	14/04/23	Drafting documentation	1.6

1. Introduction

1.1 Purpose

The purpose of this document is to outline the software requirements and specifications for REvaluate, a web application designed to provide real estate buyers and sellers with relevant information about housing from various types of flats throughout the country. The document will describe the system's purpose and features, its interfaces, functions, operating constraints, and how it will respond to external influences. This document is intended for both the developers and stakeholders involved in the system.

1.2 Document Conventions

Text formats

Font: Arial

Font size: 11 for Body, 17 for Sub-Headings and 21 for Headings

Line height: 1.15

1.3 Intended Audience and Reading Suggestions

This document is meant for everyone involved in the REvaluate project, including stakeholders, customers, designers, coders, testers, and maintainers. The document can be read in any order and readers are encouraged to focus on sections that are relevant to them. Here is a brief summary of what each section covers.

Part 1 (Introduction):

This part gives an overview of the REvaluate project, including its goals, objectives, and project scope.

Part 2 (Overall Description):

This section provides a more detailed description of the REvaluate system, including its functions, user characteristics, constraints, and assumptions.

Part 3 (External Interface Requirements):

This part specifies the system's interactions with external systems, such as other software applications or hardware devices.

Part 4 (System Features):

This section outlines the specific features and functionalities of the REvaluate system, including its inputs, outputs, and processing requirements.

Part 5 (Other Nonfunctional Requirements):

This part covers requirements that are not related to the system's specific features or functions, such as performance, security, and usability.

Part 6 (Other Requirements):

This section includes any additional information that may be helpful to the readers, such as references and glossary of terms.

1.4 Product Scope

REvalutate is a web application that translates open-source housing resale transaction information, from data.gov, into actionable insights for potential home buyers/sellers and allows them to perform ad-hoc queries on resale houses pricing. This project serves to enhance transparency in the housing market and help potential buyers/sellers make more informed pricing decisions.

1.5 References

Source Code (GitHub): https://github.com/RishJohn14/HousingInfo

Libraries, Frameworks and Tools:

MongoDB Atlas: https://www.mongodb.com/

ExpressJS: https://expressjs.com/

React: https://react.dev/

NodeJS: https://nodejs.org/en

Mui: https://mui.com/

Firebase: https://firebase.google.com/

Material UI: https://mui.com/

Bootstrap: https://react-bootstrap.github.io/

Mongoose: https://mongoosejs.com/

Correlation calculator: https://github.com/rubenvar/calculate-correlation

Referenced Articles

Diashkin, A. (2021, July 19). Firebase. facebook sign-in method. Medium. Retrieved April 14, 2023, from https://medium.com/litslink/firebase-facebook-sign-in-method-20571606c889

Ko, D. (2020, September 12). Working with google API. Medium. Retrieved April 14, 2023, from https://medium.com/weekly-webtips/working-with-google-api-38d57d6a23e4

Kumar, P., & Wanyoike, M. (2020, September 30). Create a toggle switch in react as a reusable component. SitePoint. Retrieved April 14, 2023, from https://www.sitepoint.com/react-toggle-switch-reusable-component/

Maboud, I. (2020, October 9). What is the MVC, creating a [Node.js-Express] MVC application. Medium. Retrieved April 14, 2023, from https://medium.com/@ipenywis/what-is-the-mvc-creating-a-node-js-express-mvc-application-da10625a4eda

Pratap, A. (2021, January 18). MongoDB indexes: Deep Dive, understanding indexes. Medium. Retrieved April 14, 2023, from https://medium.com/swlh/mongodb-indexes-deep-dive-understanding-indexes-9bcec6ed7aa6

Rawisglenn. (2021, December 11). How to use onemap API with react, react native app. Medium. Retrieved April 14, 2023, from https://rawisglenn.medium.com/how-to-use-onemap-api-with-react-react-native-app-11e88f70483b

Tham, S. (2021, August 14). Geocoding Singapore coordinates : Onemap API. Medium. Retrieved April 14, 2023, from https://towardsdatascience.com/geocoding-singapore-coordinates-onemap-api-3e1542bf26f7

2. Overall Description

2.1 Product Perspective

The REvaluate web application aims to address the problem of a lack of a centralised platform that provides comprehensive information on different flats across Singapore, and the challenges faced by potential buyers and sellers in making informed decisions and appropriate housing investments. The system provides open-source data on previous real estate transactions, including pricing, location, and other relevant information, to aid users in decision-making. Additionally, the system offers insights into how different factors can influence a property's value, such as highly valued features of a particular neighbourhood, average resale prices over the years, more affordable neighbourhoods, and details of an average house transaction by flat type.

2.2 Product Functions

- **Provide Real-estate price based on House type and Neighbourhood:** The website provides the function to get the estimated pricings of real-estates based on area/postal codes and flat type.
- **Filter houses by Price and Dynamic map**: The details page allows the users to further sort the available houses by price, and provides a dynamic map to allow the user to view and get the information on the houses available with their estimated pricing and additional information on their location.
- **Insights charts and Most valued features:** Displays charts per neighbourhood on the housing transactions over the years, while providing a quantitative analysis on the most valued features per neighbourhood
- **Subscription Service:** Allows the users to subscribe to access more advanced features
- **Neighbourhood Comparator:** Allows the users to perform side-by-side comparison of neighbourhoods with data organised in tables and visualised with charts.
- **RoboAdvisor:** Allows the users to input their household information and received recommendation on suitable neighbourhood and flat type, customised to their needs
- **Forum:** Allows users to connect and discuss with each other the latest housing trends and get advice.

2.3 User Classes and Characteristics

- **Individuals/Families:** Individuals or families looking to relocate and thus buy or sell their houses in their preferred neighbourhoods. New families who are looking to get resale houses may also utilise this service to help them make informed pricing decisions due to their inexperience.

- **Investors/Real-estate agents/Developers:** They are always looking to price and broker houses at the most competitive prices. Hence, REvaluate can act as a benchmarking tool for them to take pricing reference from.

2.4 Operating Environment

Hardware:

- Desktop computers (Windows, Mac OS)
- Laptops (Windows, Mac OS)
- Tablets (iOS, Android)
- Smartphones (iOS, Android)

Operating Systems:

- Windows 11
- macOS Catalins 10.15.7
- iOS 16
- Android 12

Web Browsers:

- Google Chrome 112
- Mozilla Firefox 110
- Apple Safari 16
- Microsoft Edge 112

2.5 Design and Implementation Constraints

- Front-End: ReactJS version 18.2.0
- Back-End: Firebase version 9.19.1
- Navigation: React Router version 6.4.1
- The application must comply with data protection laws and regulations in Singapore.
- The application must have user authentication and authorisation to ensure data security.
- The application must be compatible with modern web browsers such as Google Chrome, Mozilla Firefox, and Safari.
- The application must be able to handle a large number of users concurrently.
- The application must be hosted on a web server capable of handling high traffic.
- The application must be responsive.

2.6 User Documentation

About Us Page

The About Us page on the REvaluate web application offers users an understanding of the web application's mission and the services it offers. This information can help users decide what actions to take on the website.

REvaluate+ Login Page
The REvaluate+ login page provides information on what REvaluate+ is about and the type of services offered within REvluate+, providing a form of guidance to users.

Instant Form Validation
For several functions that require users' input, we provide instant form validation. When the user's inputs are not valid, the submit button will not be enabled and not be clickable. This is seen in the price filter bar and the RoboAdvisor section.

Harnessing Widely-Used Technologies
To provide an ease of use, we utilise Single-Sign-On feature, using Google and Facebook as identity providers. This allows users to easily sign in and not be troubled with how the sign-in process works.

2.7 Assumptions and Dependencies

Assumptions:

- The database of housing transactions will be provided by a third-party vendor.
- The API for retrieving data from the database will be reliable and available at all times.
- The housing data will be up-to-date and accurate.
- The user interface will be designed to work on modern web browsers, such as Google Chrome, Mozilla Firefox, and Microsoft Edge.

Dependencies:

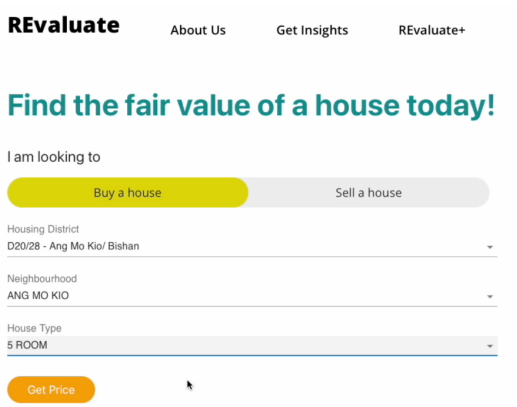
- The system will depend on the availability and reliability of the internet connection.
- The system will rely on the Google Maps API for location-based data.
- The system will use third-party libraries and frameworks, such as React and Node.js.

3. External Interface Requirements

3.1 User Interfaces

1. **Home Screen:**

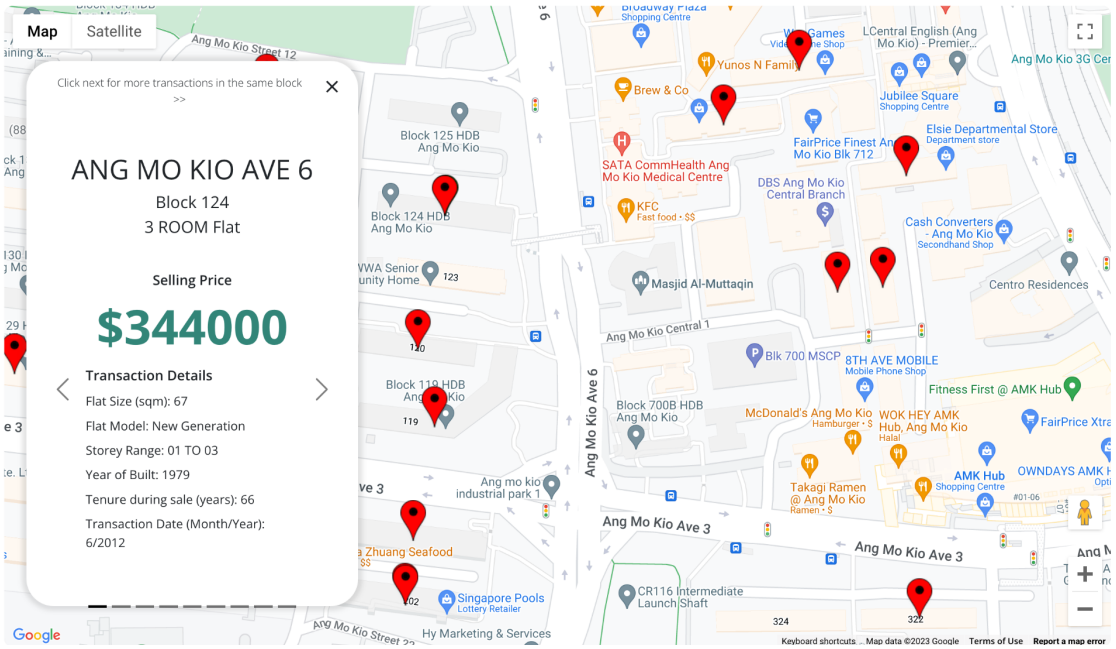
   Provides the initial view to the customer and a toggle option for a user experience as a buyer or a seller. The dropdown boxes allow the customer to indicate parameters before moving ahead.



2. **Search Results Page:**

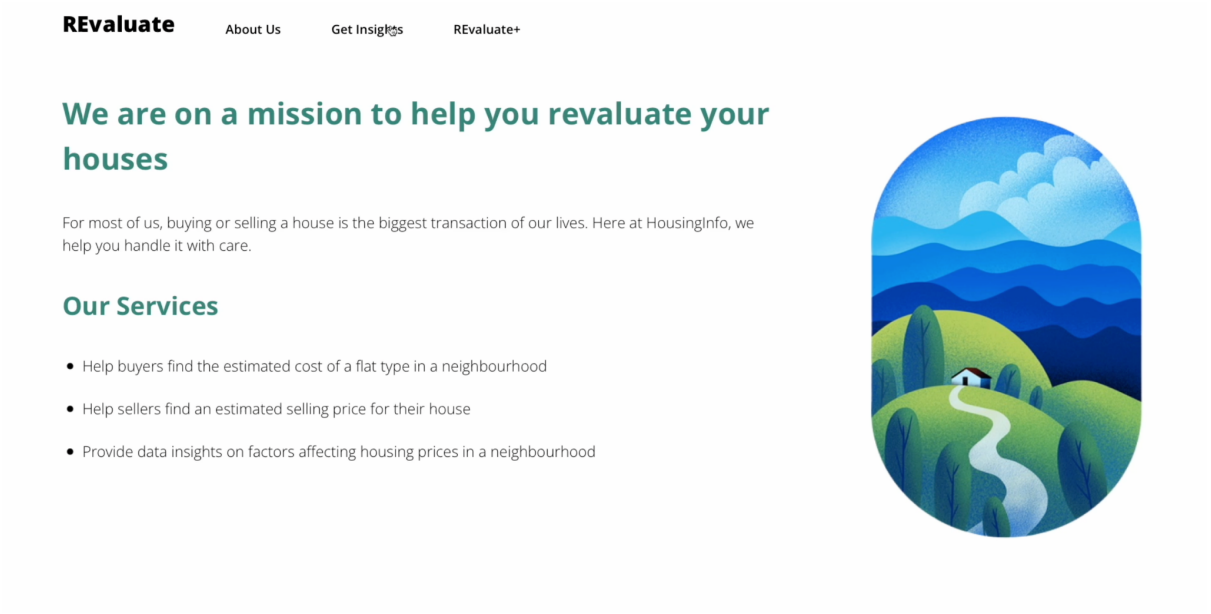
   Based on the customers indicated parameters, REvaluate provides a brief results page. The average price of a retail flat in the area is provided along with the 25th, 50th and 75th percentile of prices. The most valued streets in the neighbourhood are also indicated. The stats card on the right indicates the variables and how they affect the price and also shows prices over the years in the area.

Scrolling down, the customer can use the interactive map where pins indicate buildings with flat information available. The maps display nearby amenities and when a pin is selected, an information card pops up with the exact price of the house and other essential details.



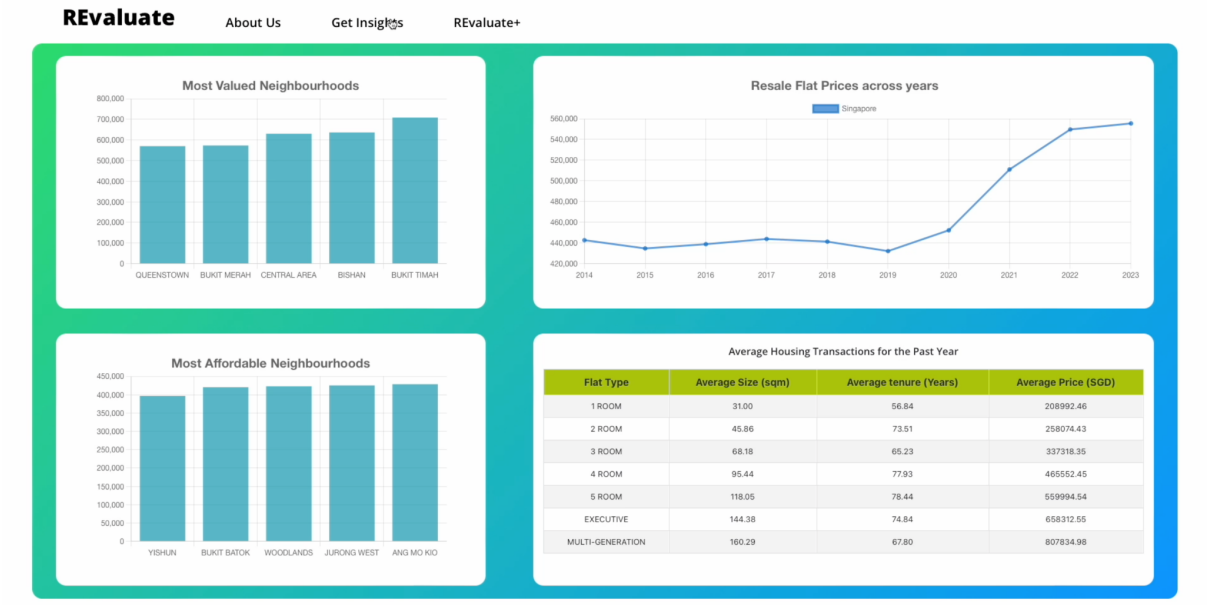
3. **About Us Page**

A brief overview of our mission statements and the services we provide to our users.



4. **Get Insights**

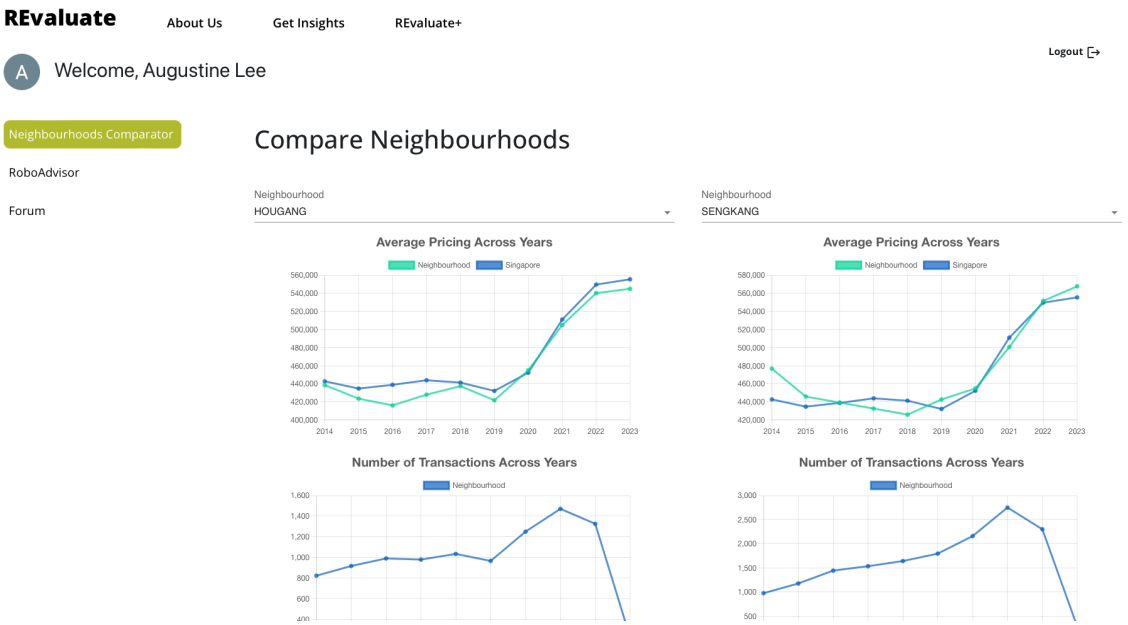
A bird's eye view of important statistics from Singapore's Real Estate market regarding the most expensive and affordable neighbourhoods, price trends on recent sales and average housing transactions



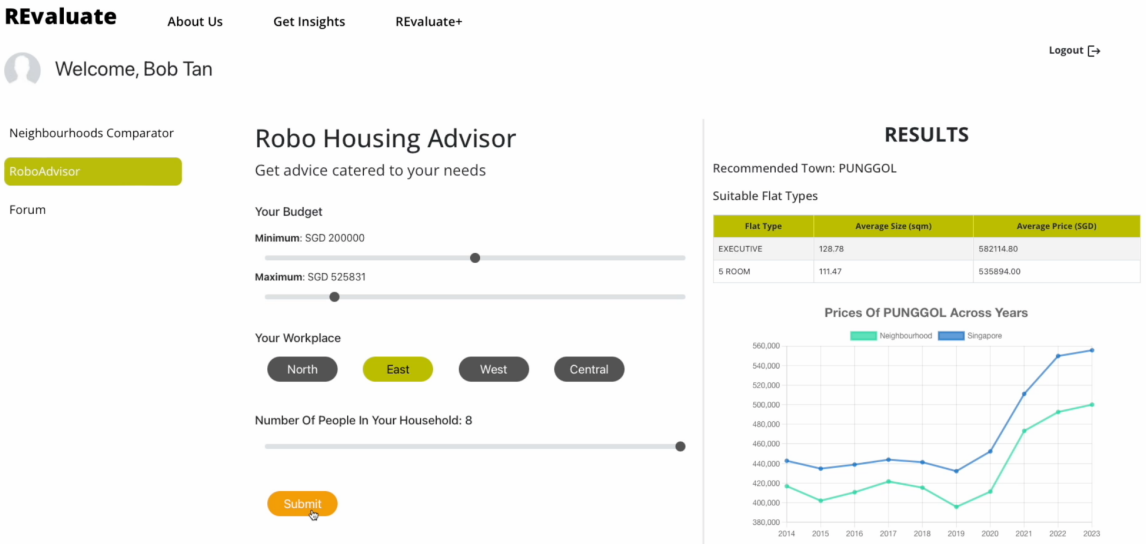
5. REvaluate+

REValuate+ is our subscription based platform that allows users to pay a nominal fee to enjoy a host of additional features. The features include:

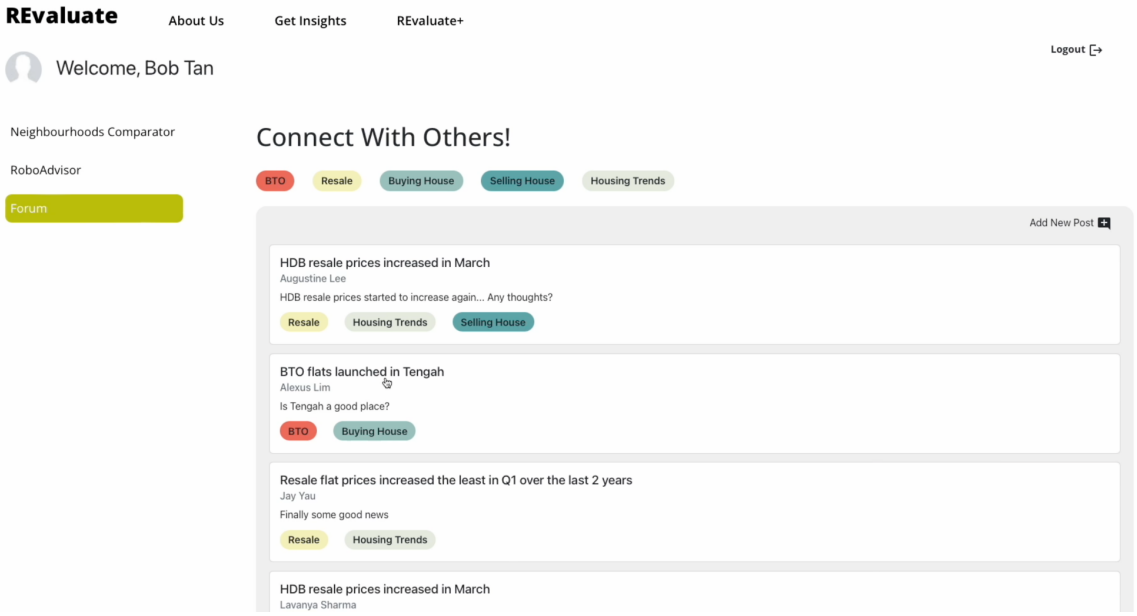
a. A **Neighbourhood Comparator** that allows users to easily compare statistics and trends from different neighbourhoods and see which price range best suits their needs.



b. A **RoboAdvisor** that can take budget, location and size of family parameters and suggest an optimal location that can best fit the needs of the customer as well as provide stats and historical sales of that type of house.



c. An **Open Forum** that allows users to connect with another, discuss views, opinions and allow for open discussion which would help customers make informed choices. The conversations can be filtered by subject and users can make appropriate posts/comments on topics to help others better understand the real estate market in Singapore.



3.2 Hardware Interfaces

Not Applicable

3.3 Software Interfaces

For our application, we utilised the MERN (MongoDB, Express, React, Node.js) stack as tools to develop our application.

1. **React**
   To develop the user interfaces, we utilised the React library. The idea of React is to develop user interfaces using components and putting these components within different pages. This allows greater reusability as we only need to code the same component once. Also, React provides hooks such as UseState and UseEffect which can act as event listeners for changes in dependencies. This allows us to incorporate the Observer Pattern into our code design easily.

2. **Express + Node.js**

   We utilised Node.js in the development of our server and RESTful APIs. This is

further enhanced with the use of Express, a framework for Node.js. Our reason for choosing Express and Node.js for the development of our backend is because they operate in a javascript environment, which is the same as our frontend, hence reducing the learning curve.

3. **MongoDB Atlas**

   We chose MongoDB Atlas as our choice of database for several reasons. Firstly, the document-oriented nature of MongoDB provides a lot of flexibility in defining models as compared to relational databases like SQL. Secondly, it is one of the best-performing databases which is crucial as our database is large and we need the query to perform well. Lastly, MongoDB can be easily integrated with our Express server using a library called Mongoose. This provides a layer of abstraction (Facade Pattern) and simplifies the Create-Read-Update-Delete operations of the database.

We also used external APIs to retrieve and manipulate our data to achieve the desired outcome.

1. **Resale Flat Prices (Data.gov.sg)**

   This API allows us to retrieve past transactions data and save them for further manipulation. We constructed an Extract-Transform-Load (ETL) pipeline where we download the data from this API and transform it to remove unnecessary data and add in additional information like latitude and longitude, and load it to our MongoDB database.

   These is the schema of the information provided by the API:

COLUMNS

No.	Name	Title	Type
1	month	Month	Datetime (Month) "YYYY-MM"
2	town	Town	Text (General)
3	flat_type	Flat type	Text (General)
4	block	Block	Text (General)
5	street_name	Street name	Text (General)
6	storey_range	Storey range	Text (General)
7	floor_area_sqm	Floor area sqm	Numeric (General)
8	flat_model	Flat model	Text (General)
9	lease_commence_date	Lease commence date	Datetime (Year) "YYYY"
10	remaining_lease	Remaining lease	Text (General)
11	resale_price	Resale price	Numeric (General)

ADDITIONAL INFORMATION

Last updated	April 14, 2023
Created	July 28, 2021
Format	CSV
Coverage	January 1, 2017 to April 13, 2023
Licence	Singapore Open Data Licence

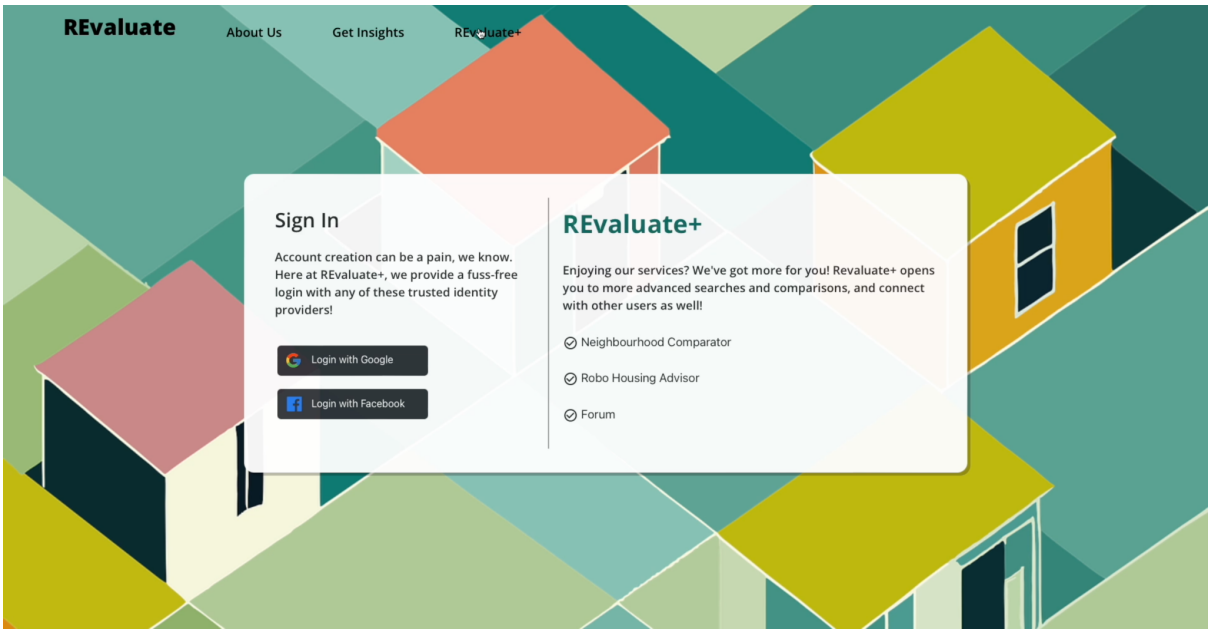
2. **OneMap API**

   We utilised this API for two main purposes. The first is in our ETL pipeline where we need to map a street name to latitude and longitude as the street name is not as meaningful as latitude and longitude in terms of providing the exact location of a place. In the transform stage, we added this extra information to our dataset. The second functionality is to perform validation. When sellers provide a postal code, we will cross check it with the OneMap API to ensure that the postal code is valid, else we will inform the seller that they have inputted an invalid postal code.

3. **Google Maps API**

   Google's Map API allows us to create the interactive map which lets the houses be visualised on the map as individual pins. With this, users can see the pins(houses) as well as neighbourhood amenities which may be factors in their decision process. The users can also scroll through the map and look at other locations as well.

3.4 Communications Interfaces



Use of Google's Firebase as part of REvaluate+ - a subscription-based plan. Under this plan, the users would have to login to identify themselves and would then be able to access a host of additional features, designed to further enhance their experience (Forum for discussion, Budgeting Tool as well as Neighbourhood Comparator).

Users will be able to authenticate themselves using single-sign-on with the following identity providers: Google and Facebook. The single-sign-on feature is implemented using Firebase authentication.

Note: Payment feature is currently not implemented as we are trying to attract a larger customer base at the current moment.

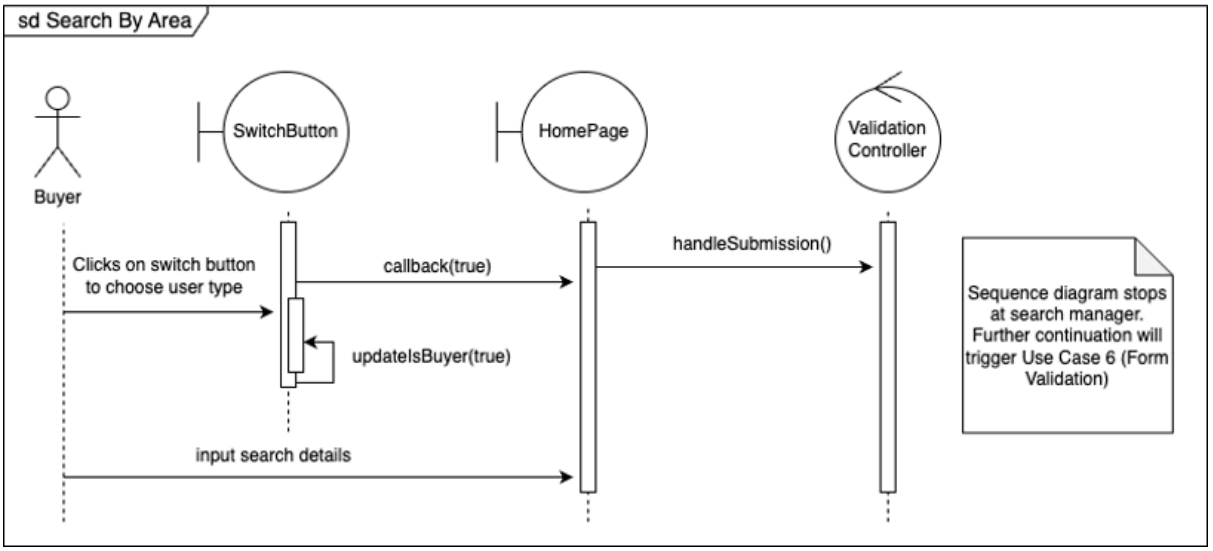
4. System Features

4.1 Use Cases

4.1.1 Buyer Search

Use Case ID:	1		
Use Case Name:	Search By Area		
Created By:	Rishabh	Last Updated By:	Rishabh
Date Created:	28/1/2023	Date Last Updated:	12/4/2023

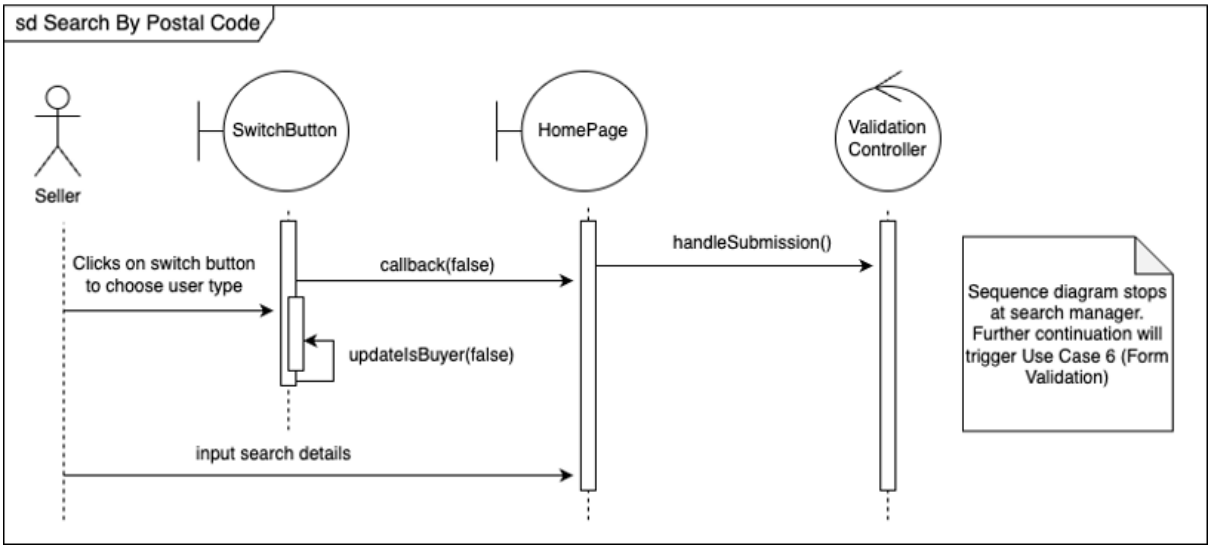
Actor:	User (Buyer)
Description:	At the home page, users will toggle the buyer function and input specific perimeters. From there the app will generate an average housing price, details of property and trends of the property.
Preconditions:	The actor must know the address he/she wishes to query
Postconditions:	Upon selection, the map should display pins of all the available houses in that neighborhood
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. The user launches the website 2. The user selects the Buyer Option and navigates to the drop-down box to select a district. 3. User selects the neighbourhood he/she wishes to query. 4. User selects flat type 5. User click on search button
Alternative Flows:	-
Exceptions:	E1: User did not fill in all the required fields Redirect user to information not found page
Includes:	Generate Search Data Form Validation
Special Requirements:	Users must be able to immediately view all options in the drop-down box and smoothly make a choice
Assumptions:	Assumes that user is aware of the location he is looking for.
Notes and Issues:	-



4.1.2 Seller Search

Use Case ID:	2		
Use Case Name:	Search By Postal Code		
Created By:	Rishabh	Last Updated By:	Rishabh
Date Created:	28/1/2023	Date Last Updated:	12/4/2023

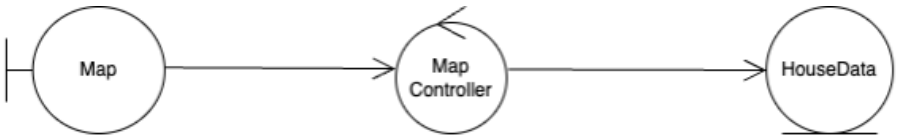
Actor:	User (Seller)
Description:	On the home page, users will toggle the seller function and key in specific perimeters. From there the app will show prices of house sold near the area the user entered.
Preconditions:	The actor must know the postal code he/she wishes to query
Postconditions:	Upon selection, the map will display pins of all the available houses in that neighborhood and their details
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. The user launches the website 2. The selects the Seller Option and navigates to the text Box 3. User inputs postal code he/she wishes to query. 4. User selects flat type 4. User click on search button
Alternative Flows:	-
Exceptions:	E1: No nearby properties in the area No nearby property in the region for the actor to take reference from. The actor can then navigate the map to find nearer alternatives E2: Invalid postal code Redirect user to ‘no information found’ page
Includes:	Generate Search Data Form Validation
Special Requirements:	Users must know the exact postal code location.
Assumptions:	Assumes that user is aware of the location he is looking for.
Notes and Issues:	-

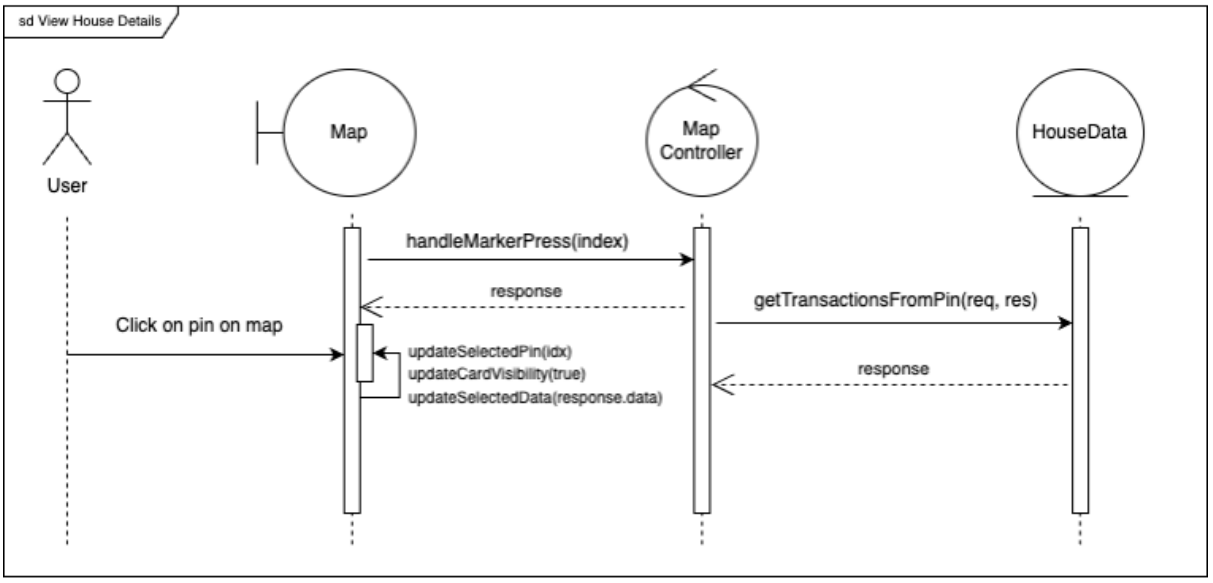


4.1.3 View House Details

Use Case ID:	3		
Use Case Name:	View House Details		
Created By:	Alexus	Alexus	Alexus
Date Created:	26/01/2023	26/01/2023	26/01/2023

Actor:	User (Buyer/Seller)
Description:	Users can interact with the google map and pins. After selecting a pin, the details of the house will be displayed.
Preconditions:	Buyer must have entered Housing District and selected Town and House Type. Seller must have entered Postal Code and House Type. User (Buyer & Seller) must click on an pin on the map to view the house details of the selected house.
Postconditions:	Houses must be displayed on the Google Maps House details must be displayed
Priority:	Very high priority, main feature of the website
Frequency of Use:	1 per Search Area (User may use this feature multiple times depending on the number of searches made)
Flow of Events:	1. User clicks on a pin on the map 2. User views house details
Alternative Flows:	-
Exceptions:	Pins on the map might not display properly or clicking on the pin might not present any housing details if the Google Maps API Network is down.
Includes:	View House Price
Special Requirements:	Users must be able to search for records and get results within 5 seconds.
Assumptions:	Assumes that user is searching for houses to view house details
Notes and Issues:	-

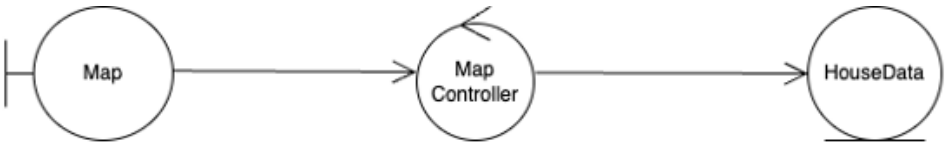


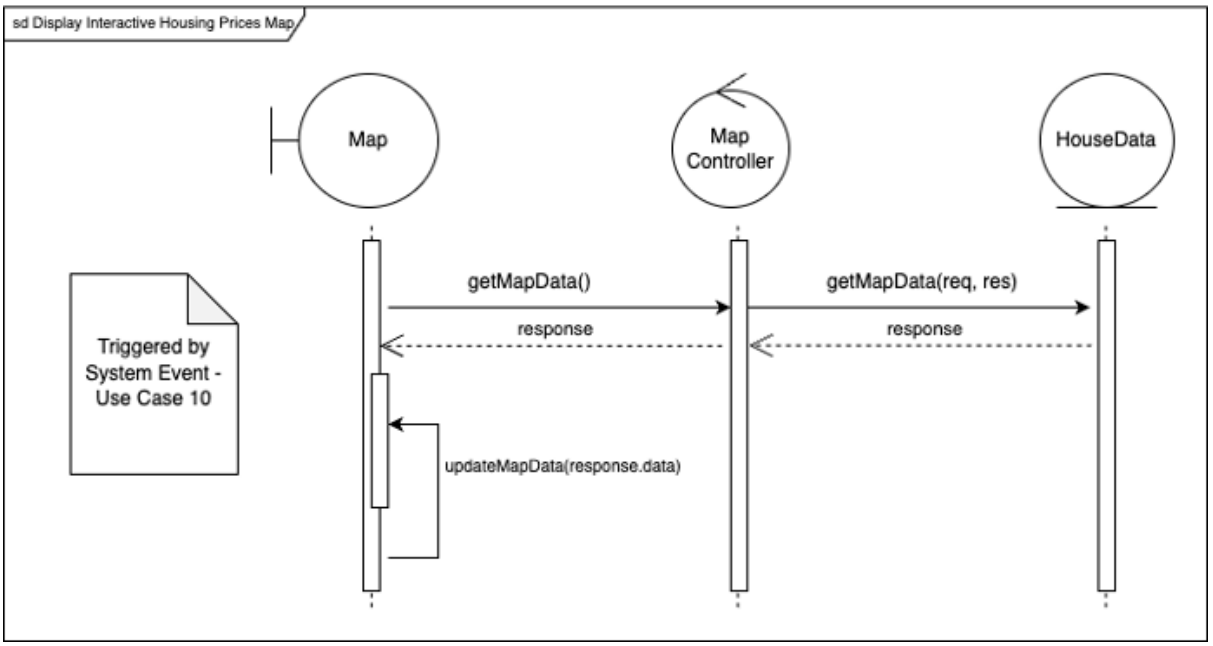


4.1.4 Display Interactive Housing Prices Map

Use Case ID:	4		
Use Case Name:	Display Interactive Housing Prices Map		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	28/1/2023	Date Last Updated:	29/1/2023

Actor:	Google Maps API,
Description:	The platform should display an interactive map with pins to indicate past transactions details. Use case is triggered by a system event where a 'Get Price' form is submitted.
Preconditions:	There is a valid housing neighbourhood or postal code being sent to the Google Maps API.
Postconditions:	An interactive map, with ability to scroll and move, is generated with pins dropped on it
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. Submission of 'Get Price' form triggers backend API (part of system). 2. Backend API to return all unique latitude and longitude according to past transactions. 3. Display map with pins drop according to the unique latitude and longitude received. 4. Send neighbourhood / postal code to Google Maps API so map can be centered at the neighbourhood / postal code 5. 'View House Details' use case when pin is selected
Alternative Flows:	-
Exceptions:	Backend API return error: Display "No Information Found"
Includes:	-
Special Requirements:	-
Assumptions:	Area/ postal code is valid
Notes and Issues:	-

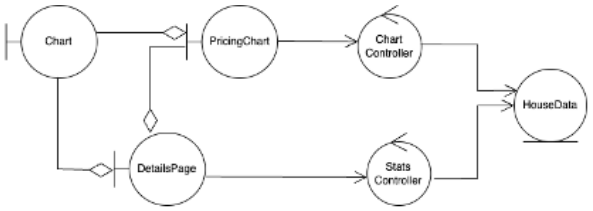


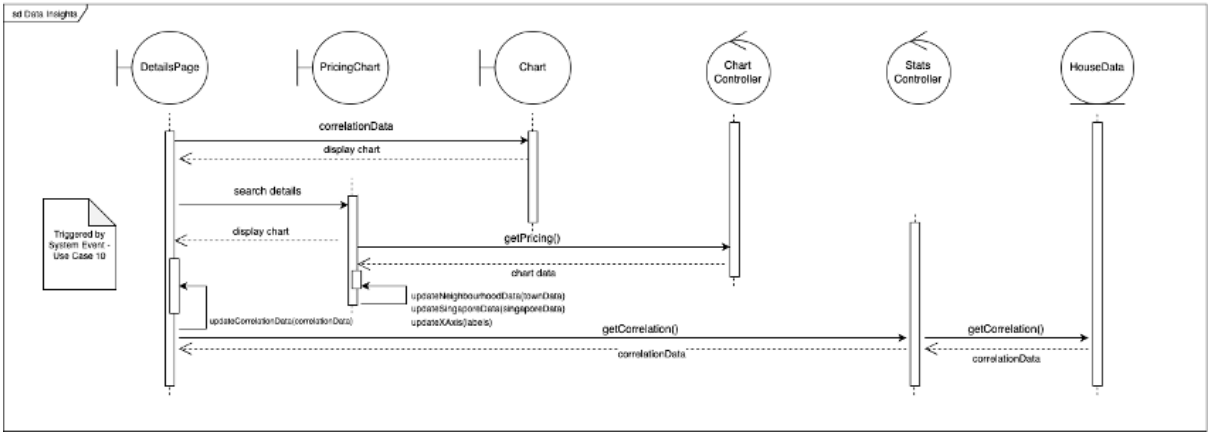


4.1.5 Generate Neighbourhood Data Insights

Use Case ID:	5		
Use Case Name:	Generate Neighbourhood Data Insights		
Created By:	Lavanya	Last Updated By:	Lavanya
Date Created:	27/1/2023	Date Last Updated:	27/1/2023

Actor:	User (Buyer / Seller)
Description:	View advanced data insights of price ranges for different house types
Preconditions:	User must have searched for a particular address or postal code and clicked on advanced data insights for further information
Postconditions:	Statistics for different house types in the chosen area is shown
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. API call to the backend to get data from the database 2. The data is then displayed in the form of charts and graphs
Alternative Flows:	-
Exceptions:	E1: No data found for the region Redirect to 'no information found' page
Includes:	-
Special Requirements:	The data must be logical and consistent and available to the user within a few seconds
Assumptions:	-
Notes and Issues:	-

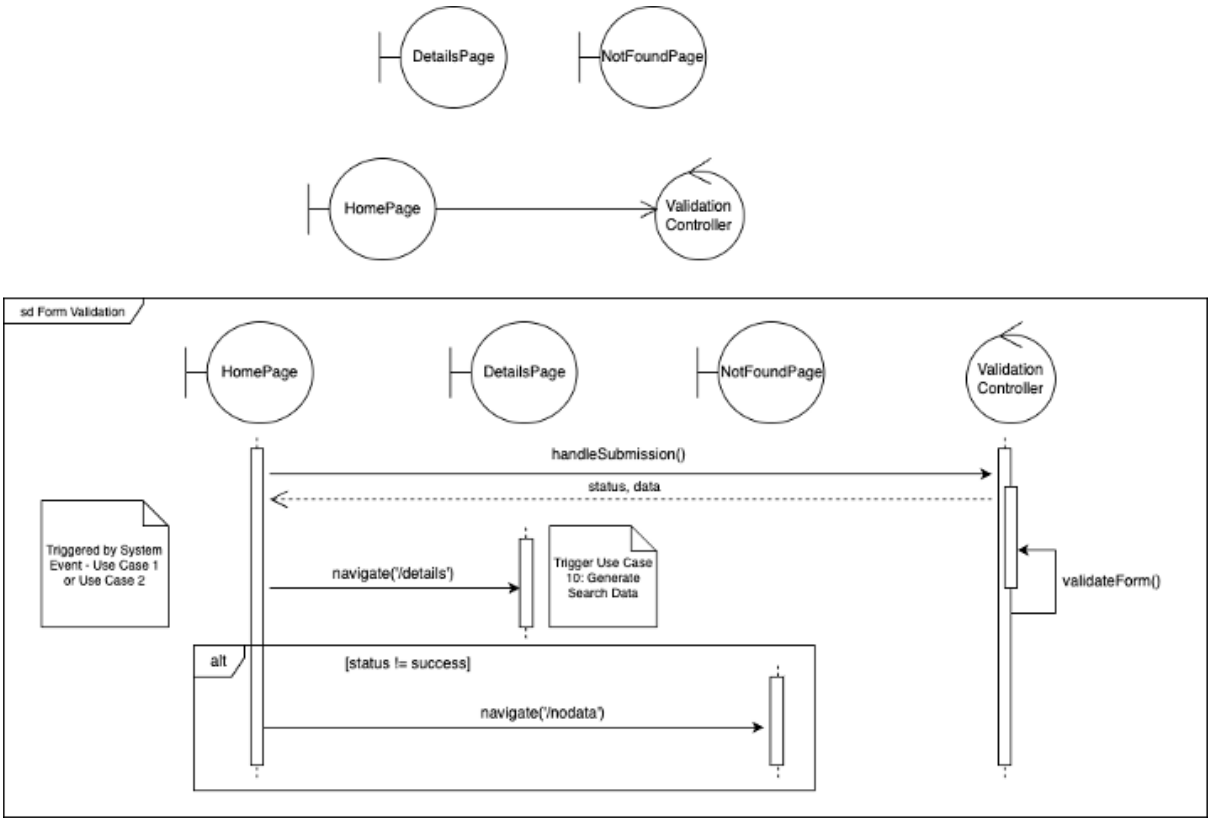




4.1.6 Form Validation

Use Case ID:	6		
Use Case Name:	Form Validation		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	28/1/2023	Date Last Updated:	29/1/2023

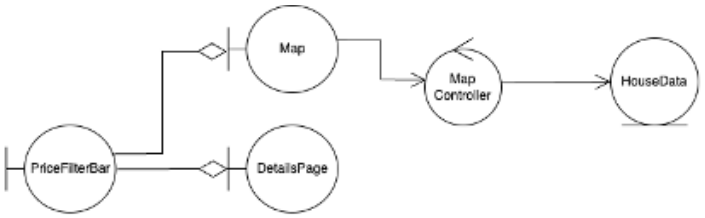
Actor:	User (Both Buyers and Sellers)
Description:	When buyers and sellers submit the form to search for houses, we need to check that: - All fields are filled up - For sellers, the postal code submitted is valid
Preconditions:	Use case triggered when the user clicks on the 'Get Price' button on the landing page
Postconditions:	All field are valid, and user will be redirected to the details page
Priority:	High (will not affect core functionality if there is no malicious actors)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. User clicks on 'Get Price' button 2. Client calls backend API to validate form 3. If all the fields are filled up, system checks if the user is a seller 4. If user is not a seller, system returns success status and the sent data 5. If user is a selling, system validates postal code and return success status if all fields are field up and the latitude and longitude for the postal code
Alternative Flows:	AF1: Backend return status that is not success 1. Redirect user to 'no information found' page
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

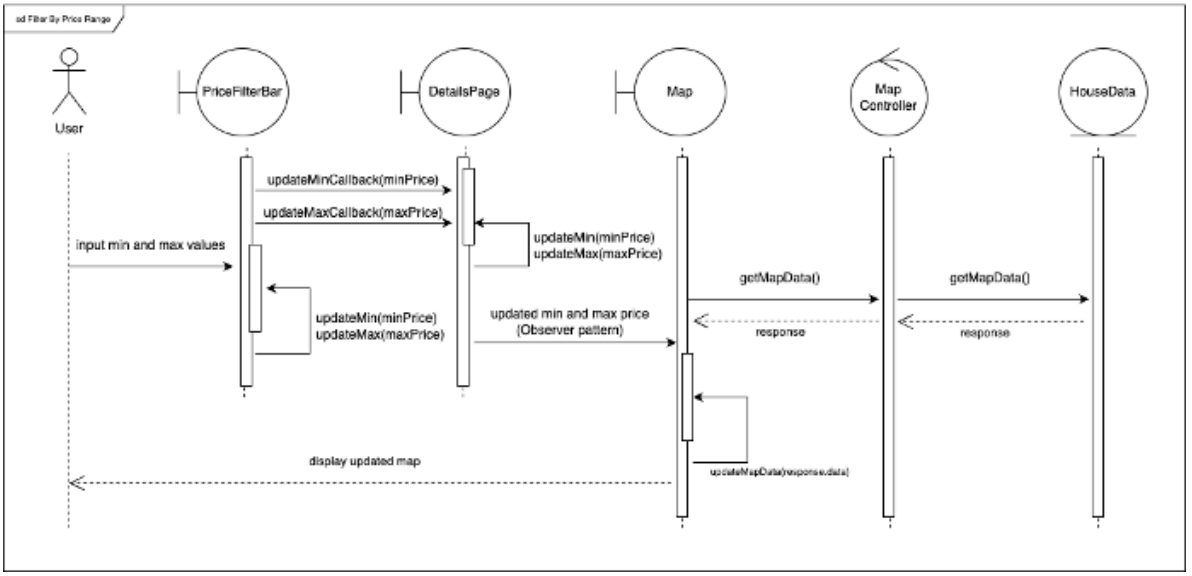


4.1.7 Filter by Price

Use Case ID:	7		
Use Case Name:	Filter by Price Range		
Created By:	Yau Xing Zhe	Last Updated By:	Yau Xing Zhe
Date Created:	30/1/2023	Date Last Updated:	12/4/2023

Actor:	User (Buyer/Seller)
Description:	The actor can specify a price range to be able to view the houses listed within the price range, which will be highlighted from the others
Preconditions:	The actor must know the price range he/she wishes to query
Postconditions:	Upon selection, the map will display pins of all the available houses within the price range
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. The actor input the minimum and maximum price 2. The actor click on the filter button 3. Houses within the price range will be highlighted
Alternative Flows:	-
Exceptions:	E1: Minimum Price > Maximum Price Shows message that says price range is invalid
Includes:	-
Special Requirements:	-
Assumptions:	Assumes that user is searching for houses within a price range
Notes and Issues:	-

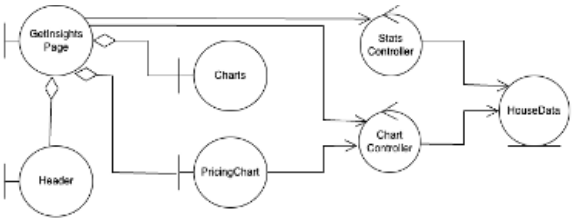


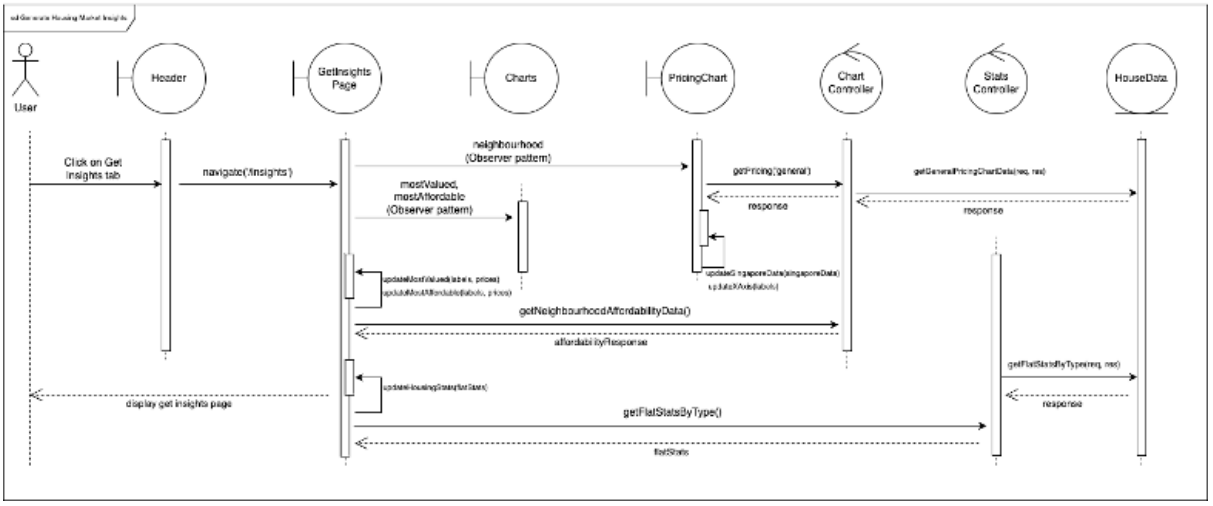


4.1.8 Generate Housing Market Insights

Use Case ID:	8		
Use Case Name:	Generate Housing Market Insights		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	29/1/2023	Date Last Updated:	29/1/2023

Actor:	User (Buyer, Seller)
Description:	Provides insights on Singapore housing market when the user clicks on 'Get Insights' on the navigation bar
Preconditions:	Use case triggered when user clicks on 'Get Insights' on the navigation bar
Postconditions:	Data charts are displayed
Priority:	Medium (secondary functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. User clicks on 'Get Insights' tab on the navigation bar 2. Page displays charts providing data insights on the Singapore housing market
Alternative Flows:	-
Exceptions:	Backend API return error: Display "No Information Found"
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

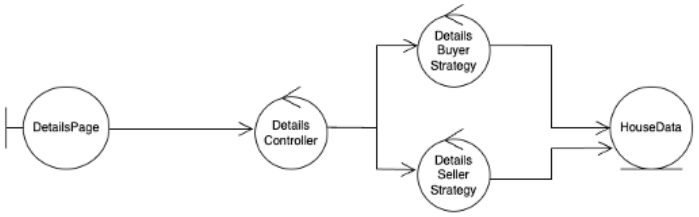


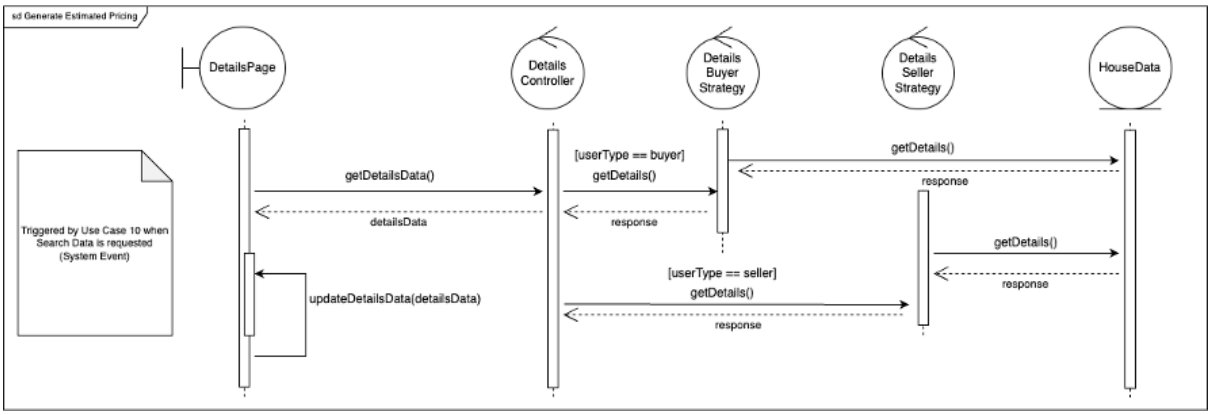


4.1.9 Generate Estimated Pricing

Use Case ID:	9		
Use Case Name:	Generate Estimated Pricing		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	29/1/2023	Date Last Updated:	30/1/2023

Actor:	- (triggered by system event)
Description:	When a buyer or seller submits the 'Get Price' form, the system needs to provide an estimated buying or selling price to display to the user.
Preconditions:	Use case triggered when the system generates the search data
Postconditions:	The details page displays an estimated buying or selling price
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. System calls backend API to scan through all transactions with the same details as the searched details 2. Backend API returns the 25th, 50th and 75th percentile of the transactions 3. Set estimated buying or selling price as the 50th percentile price
Alternative Flows:	-
Exceptions:	Backend API return error: Display "No Information Found"
Includes:	-
Special Requirements:	-
Assumptions:	The neighbourhood or postal code, and flat type provided are valid
Notes and Issues:	-

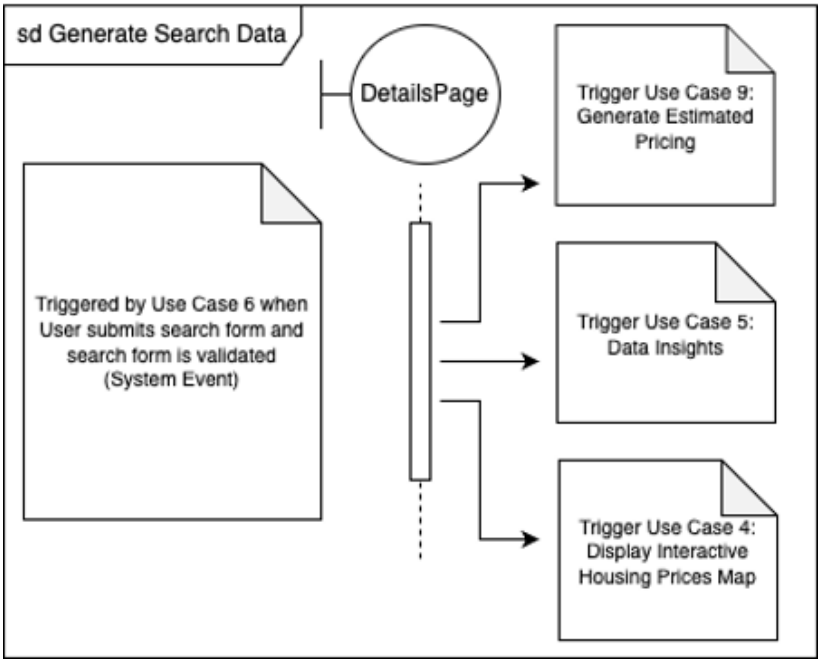




4.1.10 Generate Search Data

Use Case ID:	10		
Use Case Name:	Generate Search Data		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	29/1/2023	Date Last Updated:	30/1/2023

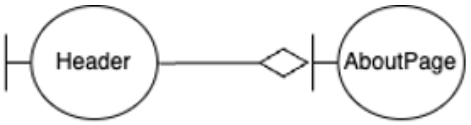
Actor:	- (triggered by system event)
Description:	When the search details are submitted and the form is validated, the search data must be generated for display on the details page
Preconditions:	Use case triggered when search details are submitted and validated
Postconditions:	Pricing, data insights and map details populated on the details page
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. When the user is redirected to the details page, this use case is triggered 2. System to generate pricing details (Case 9) 3. System to generate data insights (Case 5) 4. System to generate interactive housing map (Case 4)
Alternative Flows:	-
Exceptions:	Backend API return error: Display “No Information Found”
Includes:	-
Special Requirements:	-
Assumptions:	The neighbourhood or postal code, and flat type provided are valid
Notes and Issues:	-

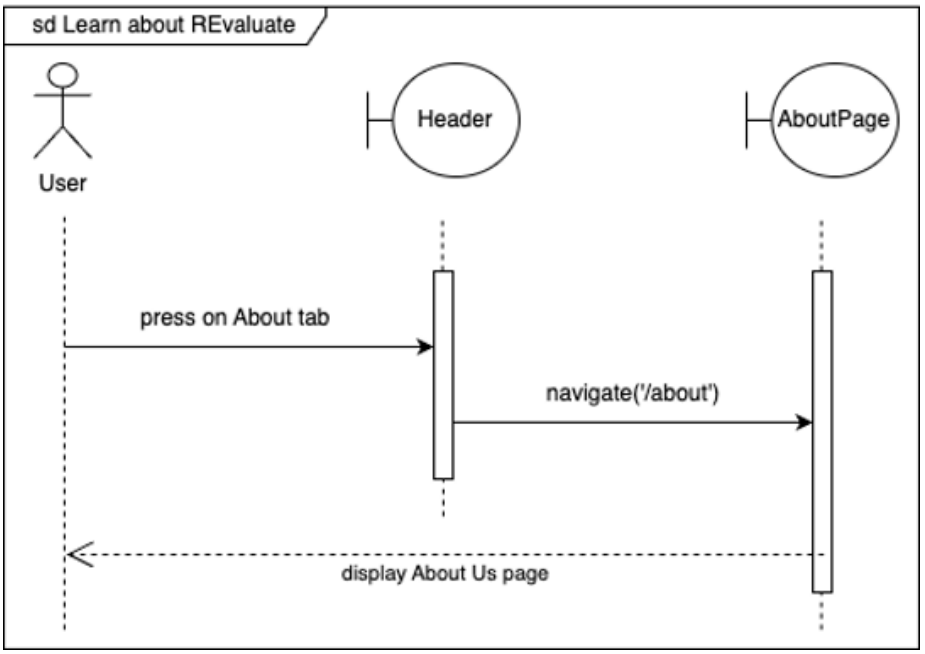


4.1.11 About Us

Use Case ID:	11		
Use Case Name:	Learn about REvaluate		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	30/1/2023	Date Last Updated:	30/1/2023

Actor:	User (Both Buyers and Sellers)
Description:	When buyers and sellers click on the 'About Us' page, they should be directed to the 'About Us' page with the introduction of what REvaluate do
Preconditions:	Use case triggered when the user clicks on the 'About Us' button on the header
Postconditions:	User redirected to the 'About Us' page
Priority:	Low (does not affect core functionality of system)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. User clicks on 'About Us' button on the header 2. User redirected to the 'About Us' page with the write-up of what REvaluate is about
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

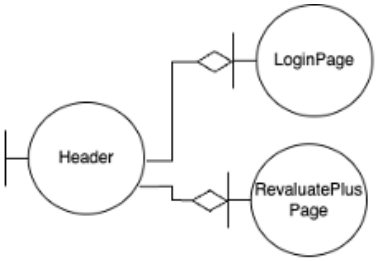


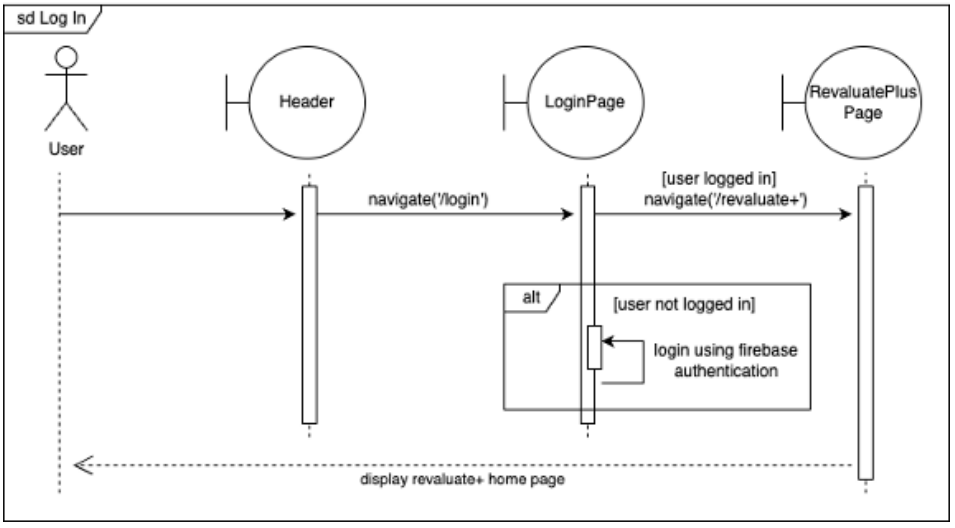


4.1.12 REvaluate+ Account Log In

Use Case ID:	12		
Use Case Name:	Log In		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	1/4/2023	Date Last Updated:	12/4/2023

Actor:	User (Buyer/Seller)
Description:	User must be able to log in using Single-Sign On using an identity provider (Google/Microsoft)
Preconditions:	User has an account with the identity provider
Postconditions:	User will be directed to the REvaluate+ home page
Priority:	Very High (core functionality)
Frequency of Use:	1 per session
Flow of Events:	1. User visits the REvaluate+ tab and click on one of the sign-in button 2. A new tab is opened for the user to sign in with the identity provider 3. Upon successful login, user is directed to the REvaluate+ home page
Alternative Flows:	AF1: Not successful login User will keep trying until they are able to log in through the identity provider
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	Identity provider authentication service will not fail
Notes and Issues:	-

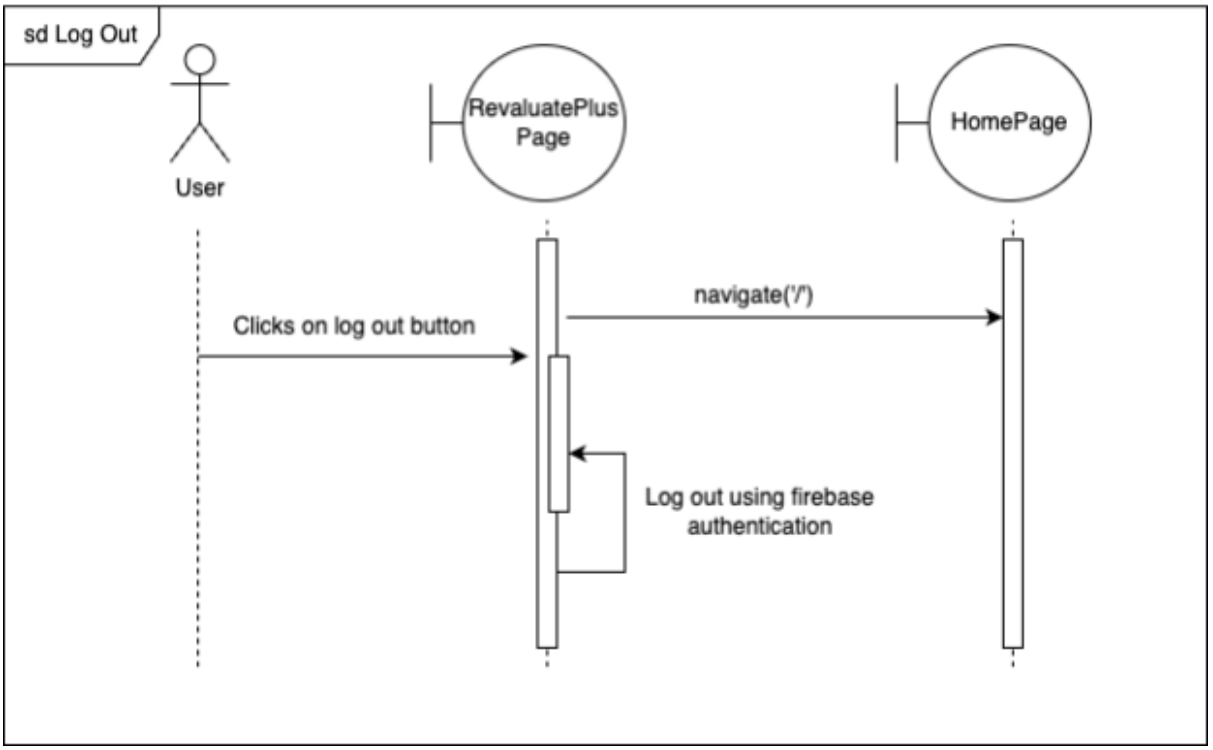




4.1.13 REvaluate+ Account Log Out

Use Case ID:	13		
Use Case Name:	Log out		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	1/4/2023	Date Last Updated:	12/4/2023

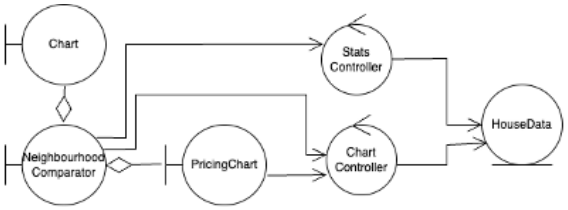
Actor:	User (Buyer/Seller)
Description:	User must be able to sign out of their account after they signed in
Preconditions:	User is signed in to REvaluate+
Postconditions:	User is signed out and redirected to home page
Priority:	Very High (core functionality)
Frequency of Use:	1 per session
Flow of Events:	1. User clicks on the log out button in REvaluate+ 2. User is signed out and redirected to the home page
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	Identity provider authentication service will not fail
Notes and Issues:	-

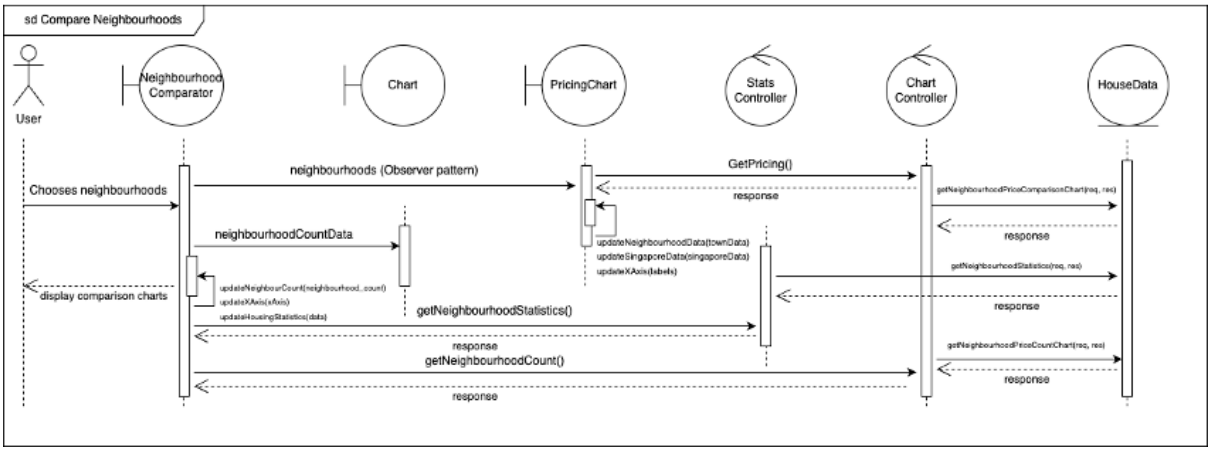


4.1.14 Compare Neighbourhoods

Use Case ID:	14		
Use Case Name:	Compare Neighbourhoods		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	1/4/2023	Date Last Updated:	12/4/2023

Actor:	User (Buyer/Seller)
Description:	User must be able to view charts and tables to compare statistics of two neighbourhoods
Preconditions:	User chooses two neighbourhoods
Postconditions:	Charts and tables displaying the statistics of the two neighbourhoods are shown
Priority:	Very High (core functionality)
Frequency of Use:	1 per minute
Flow of Events:	1. User choose a neighbourhood from the dropdown box 2. Backend API retrieves statistics of the neighbourhood from the database 3. Statistics displayed using charts and tables 4. User chooses another neighbourhood from the dropdown box and steps 2-3 repeat
Alternative Flows:	-
Exceptions:	Backend API return error: Display “No Information Found”
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

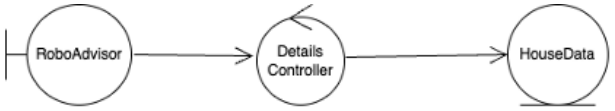


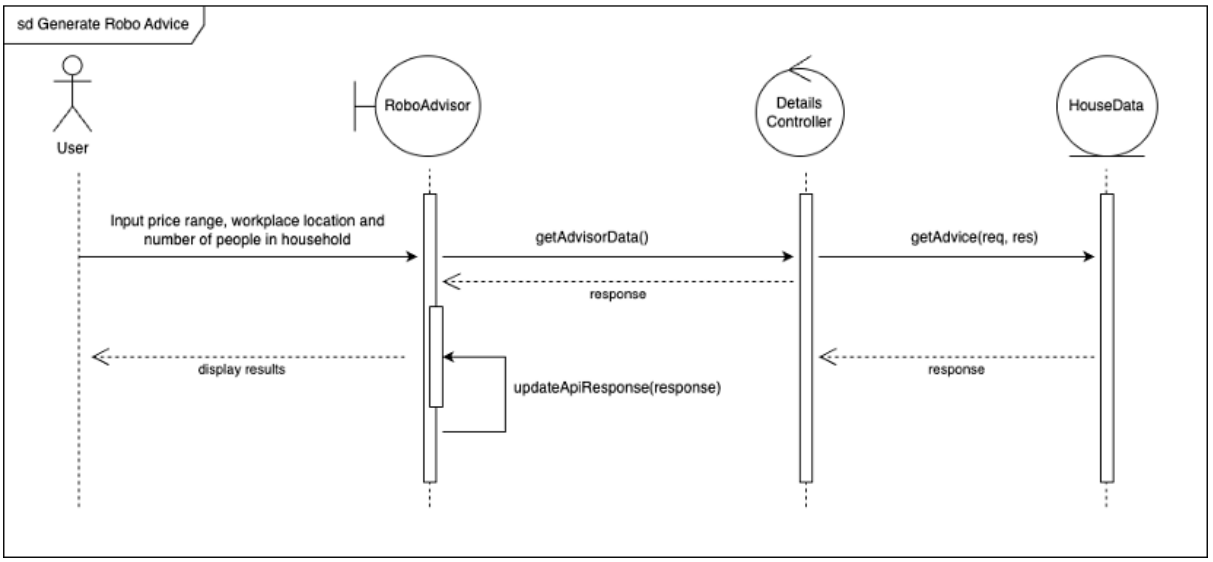


4.1.15 Generate Robo Advice

Use Case ID:	15		
Use Case Name:	Generate Robo Advice		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	1/4/2023	Date Last Updated:	12/4/2023

Actor:	User (Buyer/Seller)
Description:	User must be able to generate advice on a suitable flat type and neighbourhood based on the needs
Preconditions:	User clicks on the Robo Housing Advisor tab
Postconditions:	A side tab of a suitable flat type and neighbourhood is shown and a chart displaying the prices across time is also shown
Priority:	Very High (core functionality)
Frequency of Use:	1 (per form submitted)
Flow of Events:	1. User chooses the minimum and maximum price using the slider 2. User chooses their workplace location 3. User chooses the number of people in their household 4. User clicks on the get advice button 5. Results side tab is displayed
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

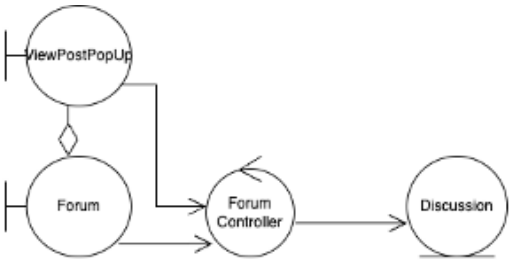


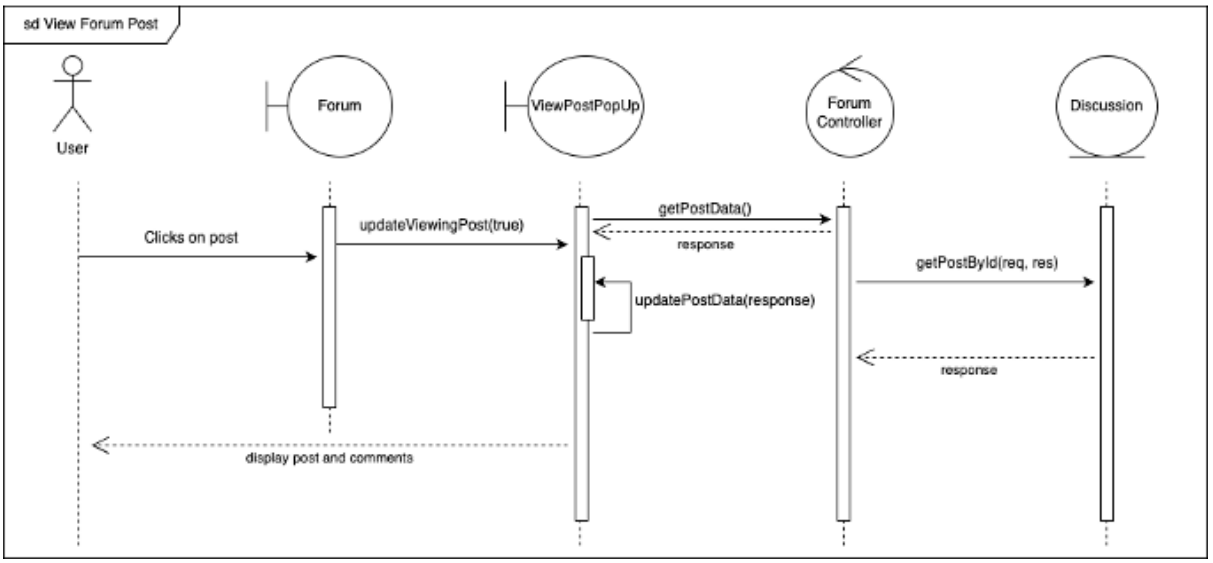


4.1.16 Add Forum Comment

Use Case ID:	16		
Use Case Name:	Add Forum Comment		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	1/4/2023	Date Last Updated:	12/4/2023

Actor:	User (Buyer/Seller)
Description:	User must be able to add a comment to an existing forum post
Preconditions:	User is in the forum post page
Postconditions:	Comment is seen on the forum post page
Priority:	Very High (core functionality)
Frequency of Use:	1 per minute
Flow of Events:	1. User types in the comment input space 2. User submits comment 3. Comment is seen on the forum post page
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

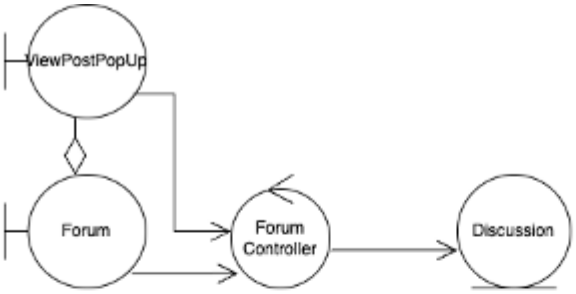


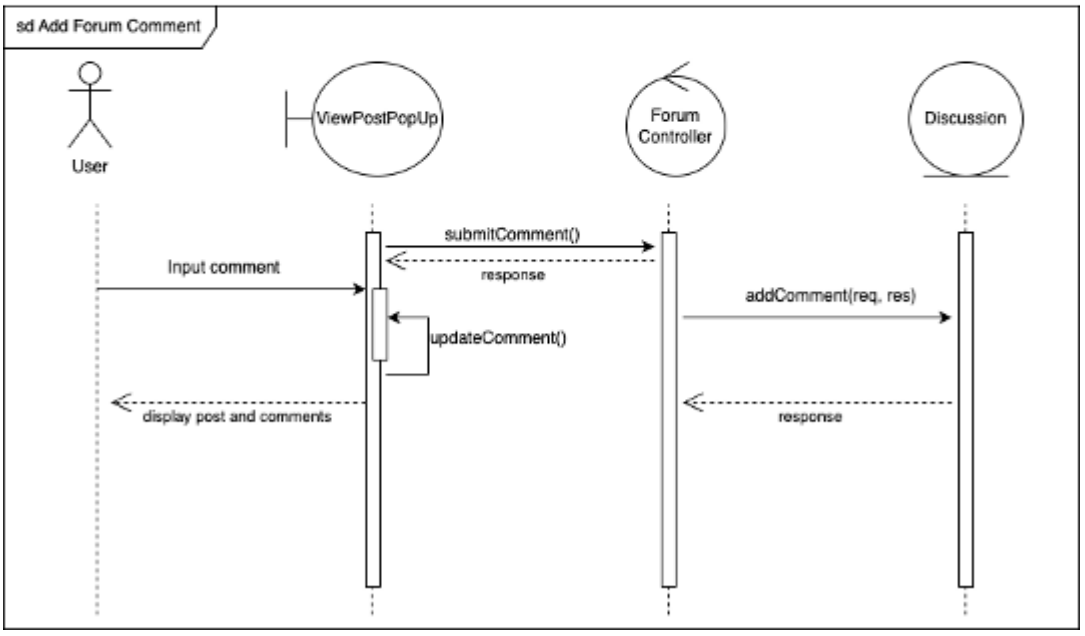


4.1.17 View Forum Post

Use Case ID:	17		
Use Case Name:	View Forum Post		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	1/4/2023	Date Last Updated:	12/4/2023

Actor:	User (Buyer/Seller)
Description:	User must be able to view an existing forum post
Preconditions:	User clicks on the forum post
Postconditions:	User is directed to the forum post page
Priority:	Very High (core functionality)
Frequency of Use:	1 per minute
Flow of Events:	1. User clicks on a forum post 2. Information about the post is retrieved from the backend 3. User is directed to the page displaying the post and all its comments
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-

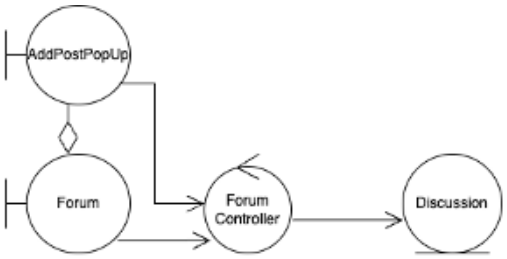


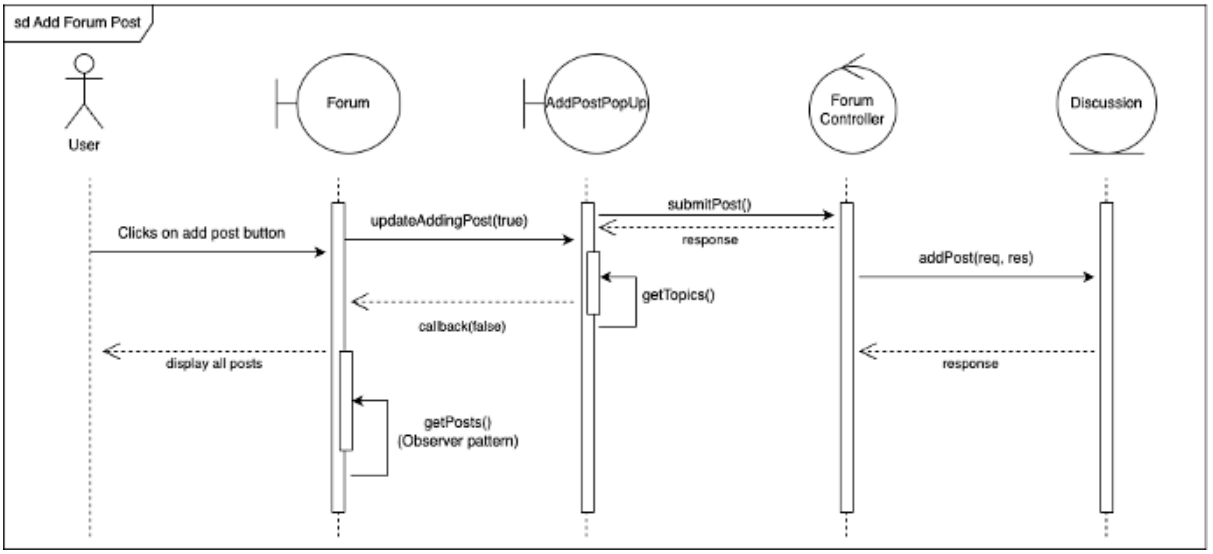


4.1.18 Add Forum Post

Use Case ID:	18		
Use Case Name:	Add Forum Post		
Created By:	Augustine	Last Updated By:	Augustine
Date Created:	1/4/2023	Date Last Updated:	12/4/2023

Actor:	User (Buyer/ Seller)
Description:	User must be able to add a forum post
Preconditions:	User clicks on the add post button
Postconditions:	A new forum post is created and seen on the forum page
Priority:	Very High (core functionality)
Frequency of Use:	1 per hour
Flow of Events:	1. User clicks on the add post button 2. User fills in the information of the post 3. User submits the post 4. User is directed back to the forum page and the new post is seen
Alternative Flows:	-
Exceptions:	-
Includes:	-
Special Requirements:	-
Assumptions:	-
Notes and Issues:	-





4.2 Functional Requirements

1. Users will be able to view the housings on a dynamic map and see pop up pins after making a search.
    1.1. Users will be able to view more detailed information of a housing transaction by clicking on a pin
2. The system must be able to search for housing transactions based on specific flat type and town or postal code
    2.1. The system must be able to prevent a search if there are missing fields in the ‘Get Price’’ form
    2.2. The system must be able to prevent a search if the postal code provided in the ‘Get Price’ form is invalid.
3. The system must show all past housing transactions as per the user’s query
    3.1. The system must be able to filter the houses by price range set by user
4. The system must be able to provide additional data insights as to how different each house is based on neighbourhood, nearby facilities.
5. User must be able to understand services provided by the system
    5.1. There must be an ‘About Us’ page with information on services provided by system
6. User must be able to get insights on the Singapore housing market
    6.1. There must be an ‘Get Insights’ page with charts based on past housing transaction data
        6.1.1. There must be a chart to show the most valued neighbourhoods
        6.1.2. There must be a chart to show the average resale price across the years
        6.1.3. There must be a chart to show the most affordable neighbourhoods
        6.1.4. There must be a chart to show details of an average resale house transaction according to flat type
7. User must be able to log in if they have a valid Google or Facebook account
8. System must prompt the user to enter correct details until they have done so.
9. User must be able to compare neighbourhoods by clicking on the dropdown boxes in the neighbourhood comparator
10. User must be able to receive recommendations based on their housing needs
11. User must be able to utilise forum services
    11.1. User must be able to create a post
    11.2. User must be able to add a comment
    11.3. User must be able to view a post with its comments
    11.4. User must be able to filter posts by topics
12. User must be aware of a network error
    12.1. There must be a error page when there is a network error

5. Other Nonfunctional Requirements

5.1 Performance Requirements

1. Users must be able to search for records and get results within 3 seconds.
2. The system should be able to support 1000 simultaneous users.
3. The database should be able to store more housing records as the Resale Prices API updates with new data.
4. The map should be able to render all locations within Singapore.
5. The average time to load a webpage over a 56Kbps modem connection will not exceed 3 seconds.
6. The application will run on all web browsers
7. The user interface should be readable and easy for the user to navigate.
8. The system shall be available in simple English
9. The selected pin must be highlighted in a different colour to make the user's selection known to him/her.
10. The user interface should be consistent with a colour palette across pages

5.2 Safety Requirements

1. Users can only register for an account using their existing Google or Facebook accounts. As the minimum age required to register for Google and Facebook accounts is 13 years old, this implies that users, at the minimum, must be at least 13 years old to create an account on REvaluate.

5.3 Security Requirements

1. Users must keep their account details confidential and not share them with others.
2. Users must be civil on the forums and not post derogatory remarks.
3. Users must not share their personal information on the forums.

5.4 Software Quality Attributes

1. Adaptability
    1.1. The system will be able to operate on all web browsers.
    1.2. The software can be easily extended to include other forms of login like Yahoo, GitHub, etc.
    1.3. The software can be easily extended to include housing data of private property, if such data is made available.
2. Maintainability
    2.1. The system will be opened for extension but closed for modification, allowing addition of new features without modifying existing code.
    2.2. The system code will be easy to read and understand.

3. Reliability
    3.1. There will be backup servers in place in the event the database malfunctions.
    3.2. Performed Black Box and White Box testing to improve reliability of the software (More information in Appendix E)
4. Usability
    4.1. The User Interface should be readable and easy for the user to navigate
    4.2. The system shall be available in simple English.
    4.3. The selected pin on the map must be highlighted in a different colour to make the user's selection clear and known to him.

5.5 Business Rules

1. Developer
    1.1. Developers can remove users and posts that contain inflammatory and derogatory remarks on the forums.
        1.1.1. This will cancel the user's monthly subscription when the service is monetised.
2. User
    2.1. Creating an account will allow Users to do the following. These features are currently available to all users but it may be monetised and exclusive to users who pay a monthly subscription fee.
        2.1.1. Users can compare resale price trends and statistics between different neighbourhoods
        2.1.2. Users can input their preferred price, household size and flat-type into a Robo-Advisor. It will then recommend a Town and Flat-Type based on the User's input.
        2.1.3. Users can create posts and comment on other users' posts on the forums section.

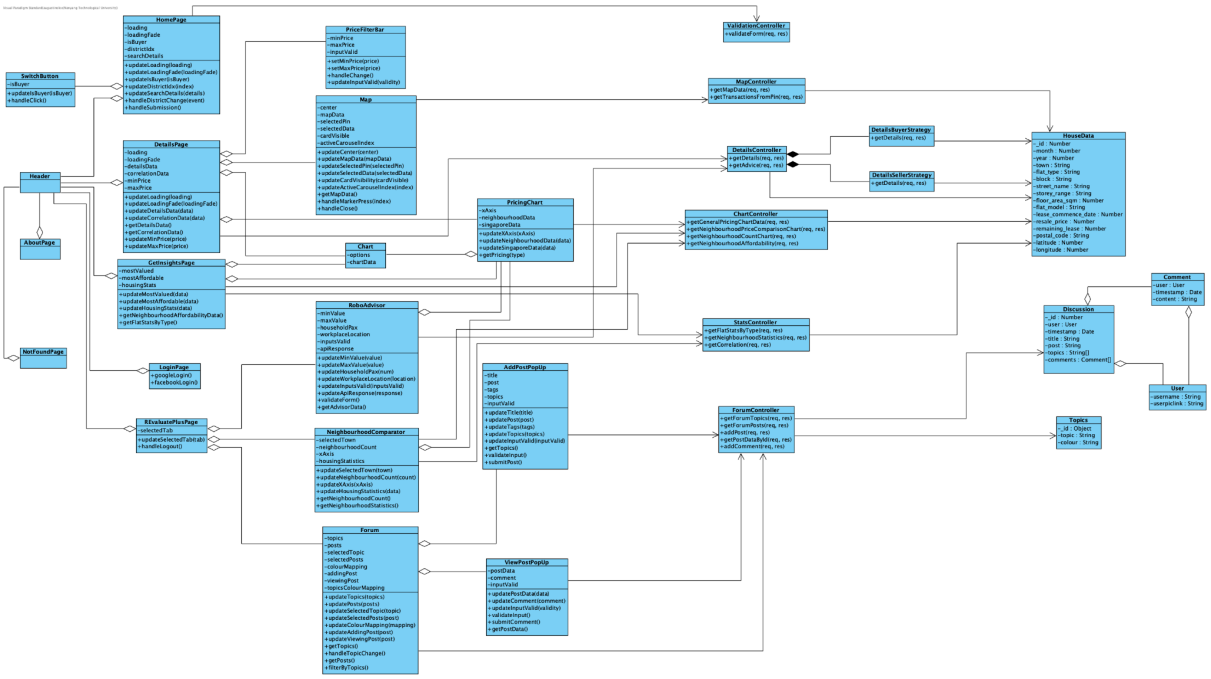
Appendix

Appendix A: Data Directory

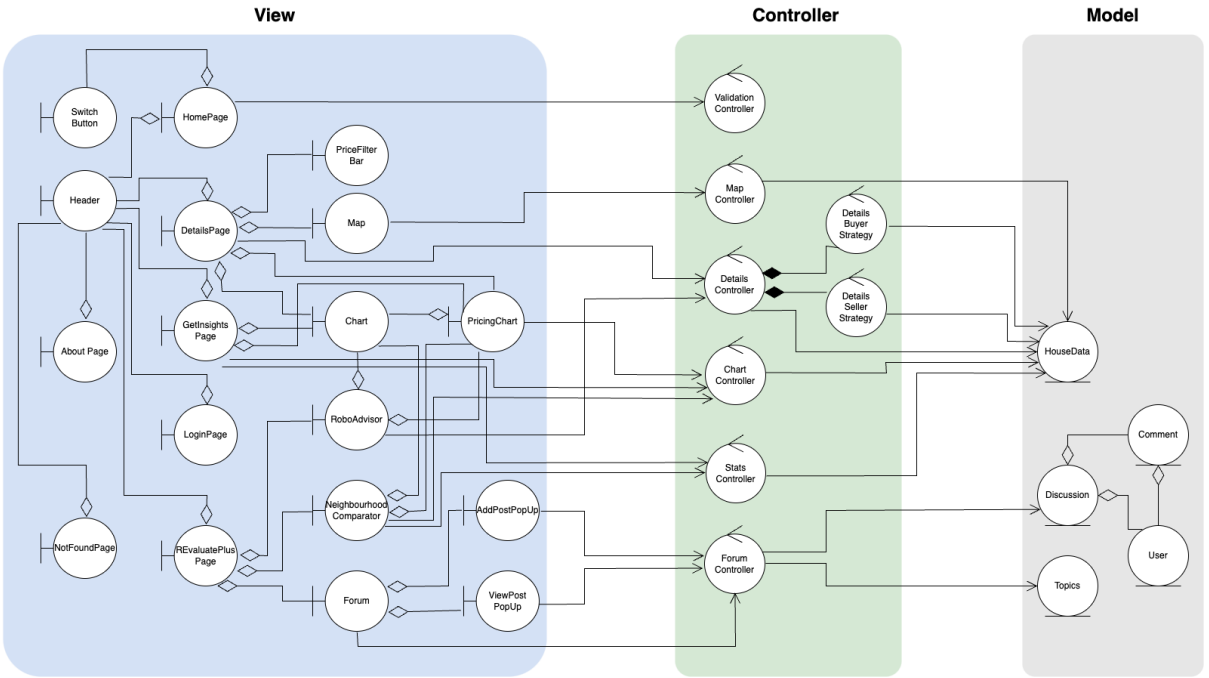
Housing District	Official housing districts of Singapore: D01 - Boat Quay/ Raffles Place/ Marina D02 - Chinatown/ Tanjong Pagar D03 - Alexandra/ Commonwealth D04 - Harbourfront/ Telok Blangah D05 - Buona Vista/ West Coast/ Clementi D06 - City Hall/ Clarke Quay D07 - Beach Road/ Bugis/ Rochor D08 - Farrer Park/ Serangoon Road D09 - Orchard/ River Valley D10 - Tanglin/ Holland/ Bukit Timah D11 - Newton/ Novena D12 - Balestier/ Toa Payoh D13 - Macpherson/ Potong pasir D14 - Eunos/ Geylang /Paya Lebar D15 - East Coast/ Marine Parade D16 - Bedok/ Upper East Coast D17 - Changi Airport/ Changi Village D18 - Pasir Ris. Tampines D19 - Hougang/ Punggol/ Sengkang D20 - Ang Mo Kio/ Bishan/ Thomson D21 - Clementi Park/ Upper Bukit Timah D22 - Boon Lay/ Jurong/ Tuas D23 - Dairy Farm/ Bukit Panjang/ Choa Chu Kang D24 - Lim Chu Kang/ Tengah D25 - Admiralty/ Woodlands D26 - Mandai/ Upper Thomson D27 - Sembawang/ Yishun D28 - Seletar/ Yio Chu Kang Singapore is officially divided into 28 housing districts under the old Postal District System. Though initially developed by the Housing Board, this set of postal code is no longer in use, property and real estate agents still make use of these codes when mentioning the location of the property. These districts include both public and private housing estates along with the community amenities that are available to use for residents.
Neighbourhood	ANG MO KIO, BEDOK, BISHAN, BUKIT BATOK, BUKIT MERAH, BUKIT PANJANG, BUKIT TIMAH,CENTRAL AREA, CHOA CHU KANG, CLEMENTI, GEYLANG, HOUGANG, JURONG EAST, JURONG WEST, KALLANG/WHAMPOA, MARINE PARADE, PASIR RIS, PUNGGOL, QUEENSTOWN, SEMBAWANG, SENGKANG, SERANGOON, TAMPINES, TOA PAYOH, WOODLANDS, YISHUN

Flat Type	Types of flats with different sizes and rooms: - 1-Room Flat - 2-Room Flat - 3-Room Flat - 4-Room Flat - 5-Room Flat - Executive Flat - Multi-Generational Flat
Postal Code	The 6-digit postal code is made up of the sector code and the delivery point. The sector is represented by the first two numbers of the postal code. The remaining four numbers define the delivery point within the sector. e.g. 53 Ang Mo Kio Avenue 3. Valid sector codes: 01, 02, 03, 04, 05, 06, 07, 08, 09, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 75, 76, 77, 78, 79, 80, 81, 82

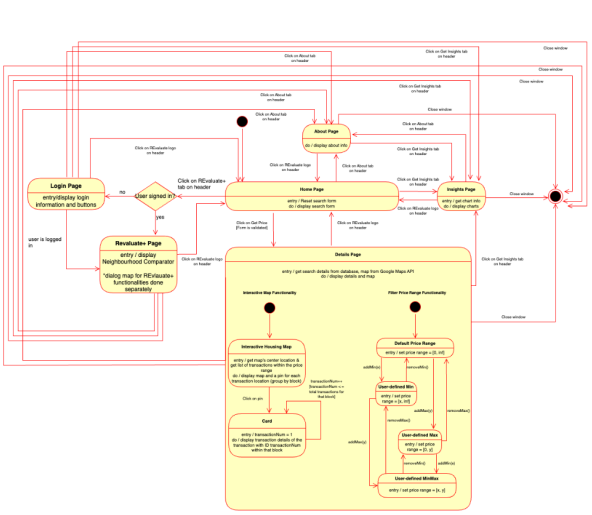
Appendix B: UML Class Diagram

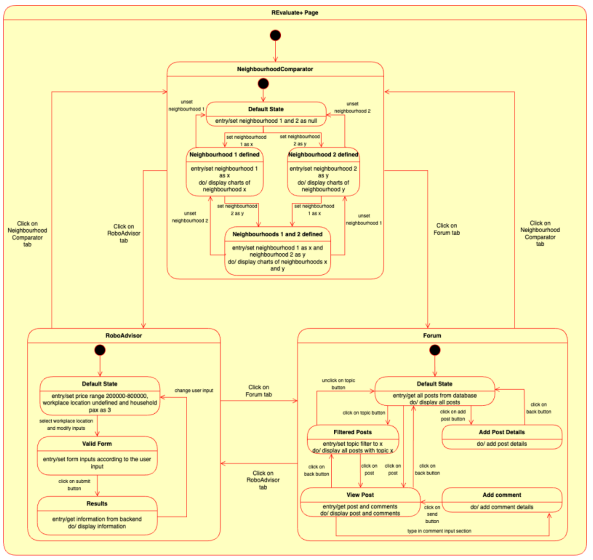


Appendix C: System Architecture Diagram (using Model-View-Controller Design Pattern)



Appendix D: Dialog Map





Appendix E: Software Testing

We performed both Blackbox and Whitebox Testing for our software.

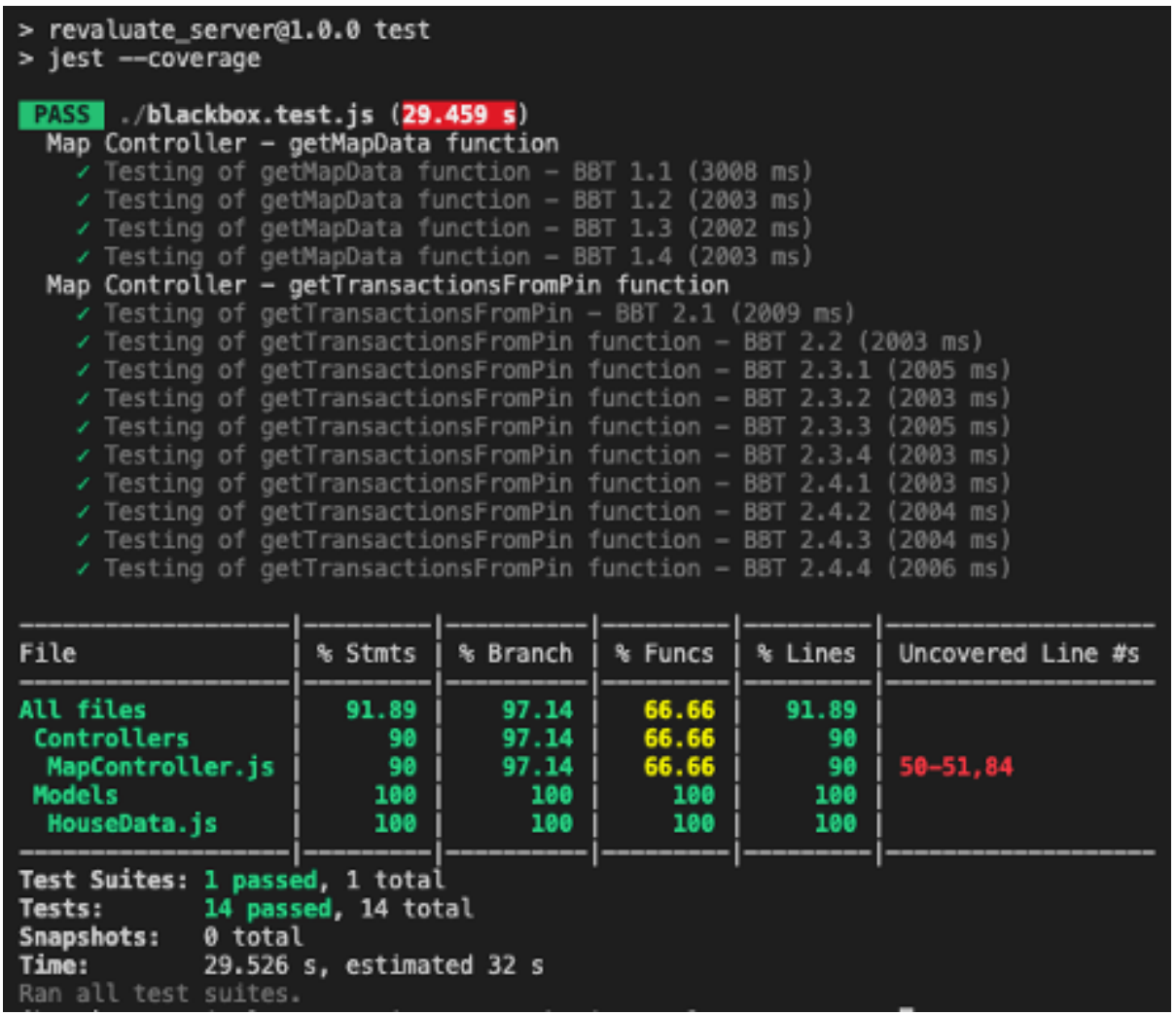
Blackbox Testing

Introduction
Our group conducted black box testing on the Map Controller. The Map Controller provides the business logic for returning information to be displayed on our interactive housing map. The Map Controller has 2 functions – getMapData(req, res) and getTransactionsFromPin(req, res).

Approach
Both functions take in a http request and a http response as their inputs. To perform the testing, we identified the equivalence classes. We then took the boundary values and perform automated testing using Jest. The test cases are detailed in the following pages.

Implementation & Results
The testing code is found in the file blackbox.test.js .

Jest Test Coverage Report
Note: Some of the paths were not able to be tested as they will only occur during network errors.



Function	getMapData(req, res)	
Parameters in query	houseType, minPrice, maxPrice	
Equivalence Classes	houseType	EC1.1 (valid): '1 ROOM', '2 ROOM', '3 ROOM', '4 ROOM', '5 ROOM', 'EXECUTIVE', 'MULTI-GENERATIONAL' EC1.2 (invalid): Any other values
	minPrice and maxPrice	EC1.3 (valid): - Number - minPrice < maxPrice EC1.4 (Invalid): Any other values
Boundary Values	houseType	No boundary values as it is discrete value
	minPrice and maxPrice	BV1.1: minPrice == maxPrice - 1 BV1.2: minPrice == maxPrice

Test Case	Input	Expected Output	Actual Output	Status

BBT 1.1
 	
Valid inputs
	{query: {
	houseType:'EXECUTIVE',
	minPrice: 0,
	maxPrice: 1000000
	}}
		status: 200 data: {
		status: 'success',
		…
		}
			status: 200 data: {
			status: 'success',
			…
			}
				PASS

BBT 1.2
 	
Invalid house type
	{query: {
	houseType:'', minPrice: 0,
	maxPrice: 1000000
	}}
		status: 200 data: {
		status: 'Invalid Inputs - HouseType'
		}
			status: 200 data: {
			status: 'Invalid Inputs – HouseType'
			}
				PASS

BBT 1.3
 	
Testing using BV1.1
	{query: { houseType:'5 ROOM', minPrice: 100000,
	maxPrice: 100000
	}}
		status: 200 data: {
		status: 'Invalid Inputs – Price Range'
		}
			status: 200 data: {
			status: 'Invalid Inputs - HouseType'
			}
				PASS

BBT 1.3
 	
Testing using BV1.2
	{query: { houseType:'5 ROOM', minPrice: 100000,
	maxPrice: 100001
	}}
		status: 200 data: {
		status: 'Invalid Inputs – Price Range'
		}
			status: 200 data: {
			status: 'Invalid Inputs - HouseType'
			}
				PASS

Function	getTransactionsFromPin (req, res)	
Parameters in query	latitude, longitude, flat_type	
Equivalence Classes	latitude	EC2.1 (valid): latitude within range of [1.2258860606437172 - 1.470283185835049] EC2.2 (invalid): latitude < 1.2258860606437172 latitude > 1.470283185835049
	longitude	EC2.3 (valid): longitude within range of [103.62027151387078-104.03369364878364] EC2.4 (invalid): longitude < 103.62027151387078 longitude > 104.03369364878364
	flat_type	EC2.5 (valid): '1 ROOM', '2 ROOM', '3 ROOM', '4 ROOM', '5 ROOM', 'EXECUTIVE', 'MULTI-GENERATIONAL' EC2.6 (invalid): Any other values
Boundary Values	flat_type	No boundary values as it is discrete value
	latitude	BV2.1: latitude = 1.2258860606437172 BV2.2: latitude = 1.225 BV2.3: latitude = 1.470283185835049 BV2.4: latitude = 1.471
	longitude	BV2.5: longitude = 103.62027151387078 BV2.6: longitude = 103.6 BV2.7: longitude = 104.03369364878364 BV2.8: longitude = 104.05

Test Case	Input	Expected Output	Actual Output	Status

BBT 2.1
 	
Valid Inputs
	{query: {
	flat_type: '5 ROOM',
	latitude: 1.25,
	longitude: 103.81
	}}
		status: 200 data: {
		status: 'success',
		…
		}
			status: 200 data: {
			status: 'success',
			…
			}
				PASS

BBT 2.2
 	
Invalid house type
	{query: {
	flat_type: '', latitude: 1.25,
	longitude: 103.81
	}}
		status: 200 data: {
		status: 'Invalid Input - House Type'
		}
			status: 200 data: {
			status: 'Invalid Input - House Type'
			}
				PASS

BBT 2.3.1
 	
Testing using BV2.1
	{query: {
	flat_type:'5 ROOM',
	latitude: 1.2258860606437172,
	longitude: 103.81
	}}
		status: 200 data: {
		status: 'success',
		…
		}
			status: 200 data: {
			status: 'success',
			…
			}
				PASS

BBT 2.3.2
 	
Testing using BV2.2
	{query: {
	flat_type:'5 ROOM',
	latitude: 1.225,
	longitude: 103.81
	}}
		{ status: 200 data: {
		status: 'Invalid Input - Latitude'
		}
			status: 200 data: {
			status: 'Invalid Input - Latitude'
			}
				PASS

BBT 2.3.3
 	
Testing using BV2.3
	{query: {
	flat_type:'5 ROOM',
	latitude: 1.
	1.470283185835049,
	longitude: 103.81
	}}
		status: 200 data: {
		status: 'success'
		}
			status: 200 data: {
			status: 'success'
			}
				PASS

BBT 2.3.4
 	
Testing using BV2.4
	{query: {
	flat_type:'5 ROOM',
	latitude: 1.471,
	longitude: 103.81
	}}
		status: 200 data: {
		status: 'Invalid Input - Latitude'
		}
			status: 200 data: {
			status: 'Invalid Input - Latitude'
			}
				PASS

BBT 2.4.1
 	
Testing using BV2.5
	{query: {
	flat_type: '',
	latitude: 1.25,
	longitude: 103.62027151387078
	}}
		status: 200 data: {
		status: 'success'
		}
			status: 200 data: {
			status: 'success'
			}
				PASS

BBT 2.4.2
 	
Testing using BV2.6
	{query: {
	flat_type: '5 ROOM',
	latitude: 1.25,
	longitude: 103.6
	}}
		status: 200 data: {
		status: 'Invalid Input - Longitude'
		}
			status: 200 data: {
			status: 'Invalid Input - Longitude'
			}
				PASS

BBT 2.4.3
Testing using BV2.7
	{query: {
	flat_type: '5 ROOM',
	latitude: 1.25,
	longitude: 104.03369364878364
	}}
		status: 200
		data: {
		status: 'success',
		…
		}
			status: 200
			data: {
			status: 'success',
			…
			}
				PASS

BBT 2.4.4
 	
Testing using BV2.8
	{query: {
	flat_type: '5 ROOM',
	latitude: 1.25,
	longitude: 104.05
	}}
		status: 200
		data: {
		status: 'Invalid Input – Longitude'
		}
			status: 200
			data: {
			status: 'Invalid Input – Longitude'
			}
				PASS

Whitebox Testing

Introduction

Our group conducted white box testing on 2 functions that implement complex business logic – getNeighbourhoodPriceComparisonChart(req, res) in ChartController and getAdvice(req, res) in DetailsController. The getNeighbourhoodPriceComparisonChart(req, res) takes user input and returns a time-series data of average house pricing. Due to different user types, the way of searching and aggregating the data is different across the different user types. The getAdvice(req, res) function acts as a logic-based agent, generating a suitable house type and neighbourhood according to the user selection.

Approach

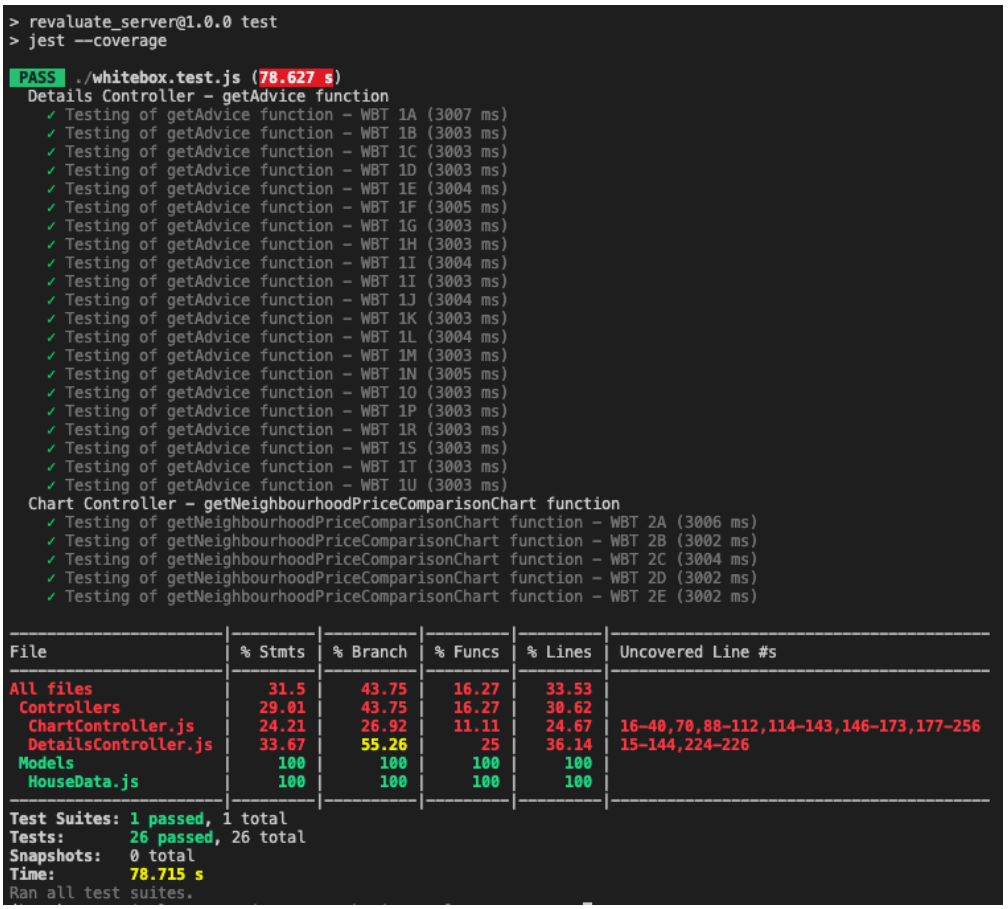
We drew the Control Flow Graphs (CFG) for both functions and sought to perform Level 2 (100% statement testing). We then perform automated testing using Jest. The CFG and test cases are detailed in the following pages.

Implementation & Results

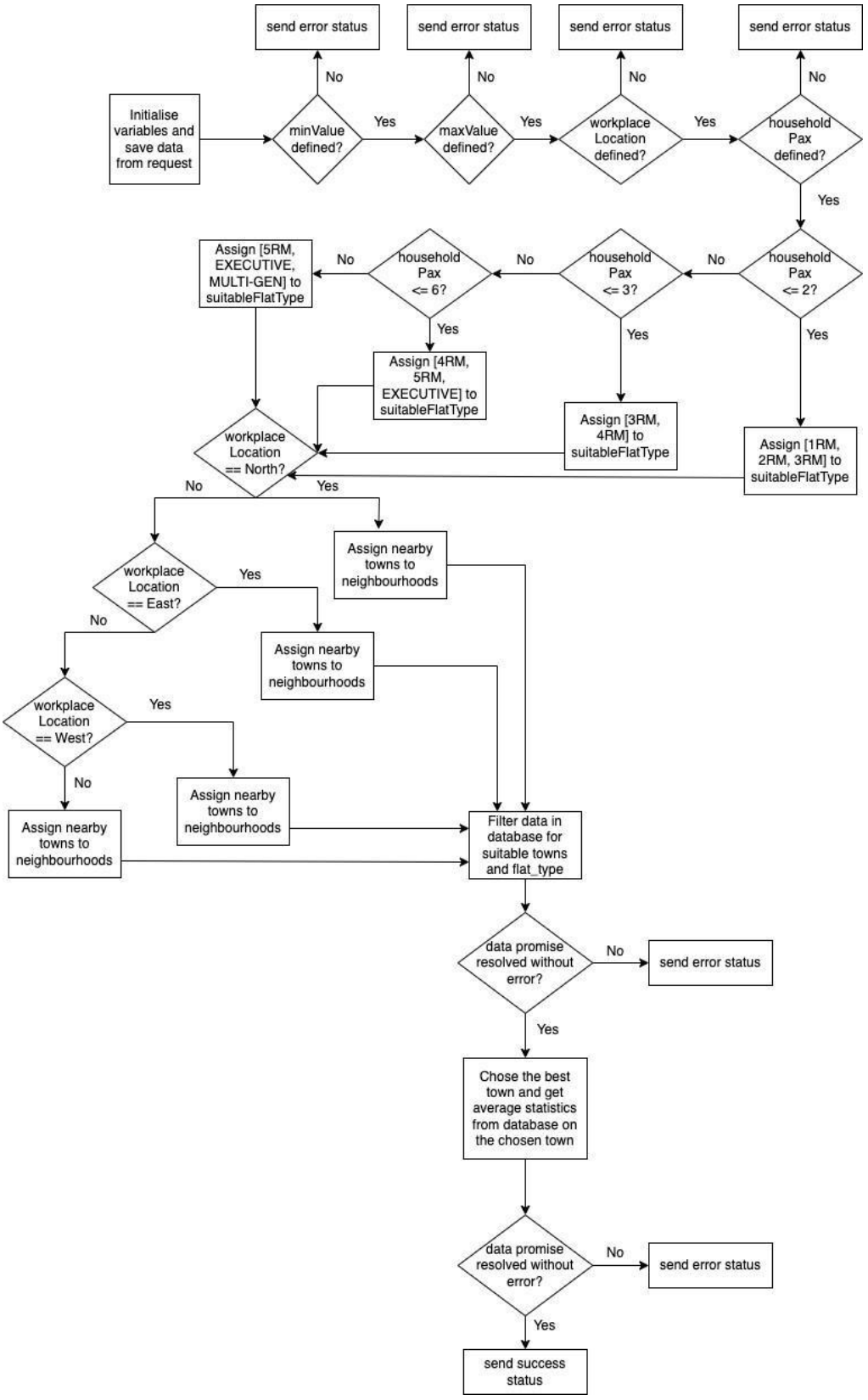
The testing code is found in the file whitebox.test.js .

Jest Test Coverage Report

Note: The test coverage is low as the whitebox testing is only for the chosen functions instead of the whole controller. Some of the paths were not able to be tested as they will only occur during network errors.

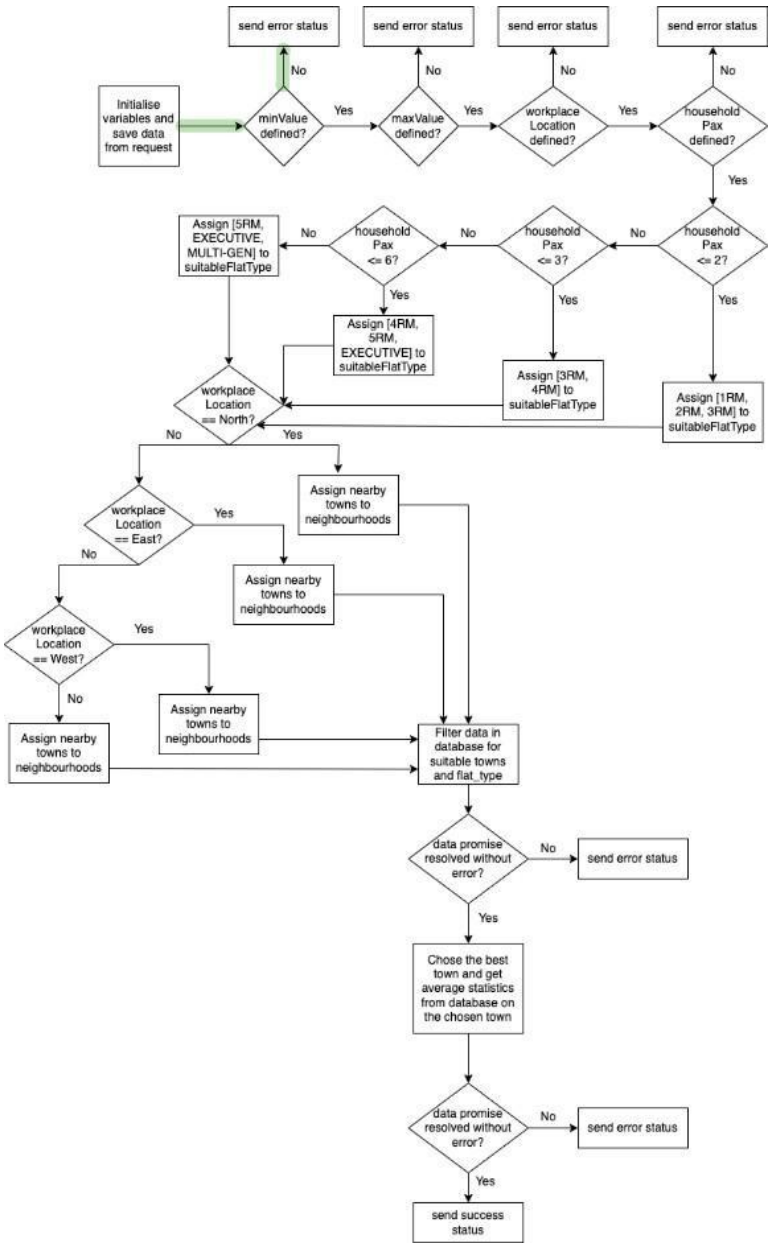


Control Flow Graph for getAdvice (req, res)



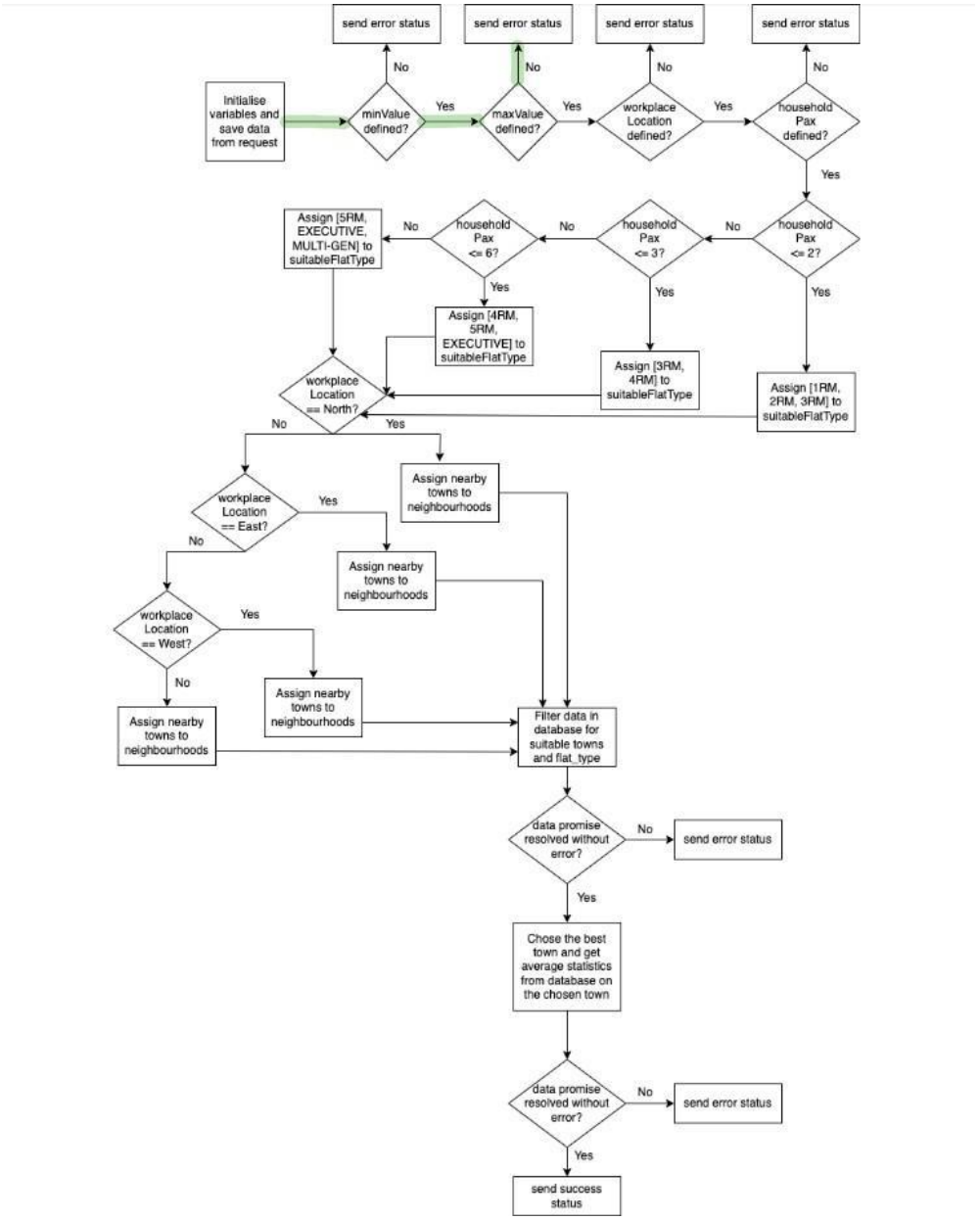
Test Code	Input	Expected Output	Actual Output	Status

WBT 1A	{query: {
	maxValue: 1000000,
	workplaceLocation:
	‘North’,
	householdPax: 5
	}}
		Status: 200
		Data: {
		Status: ‘error’
		}
			Status: 200
			Data: {
			Status: ‘error’
			}	



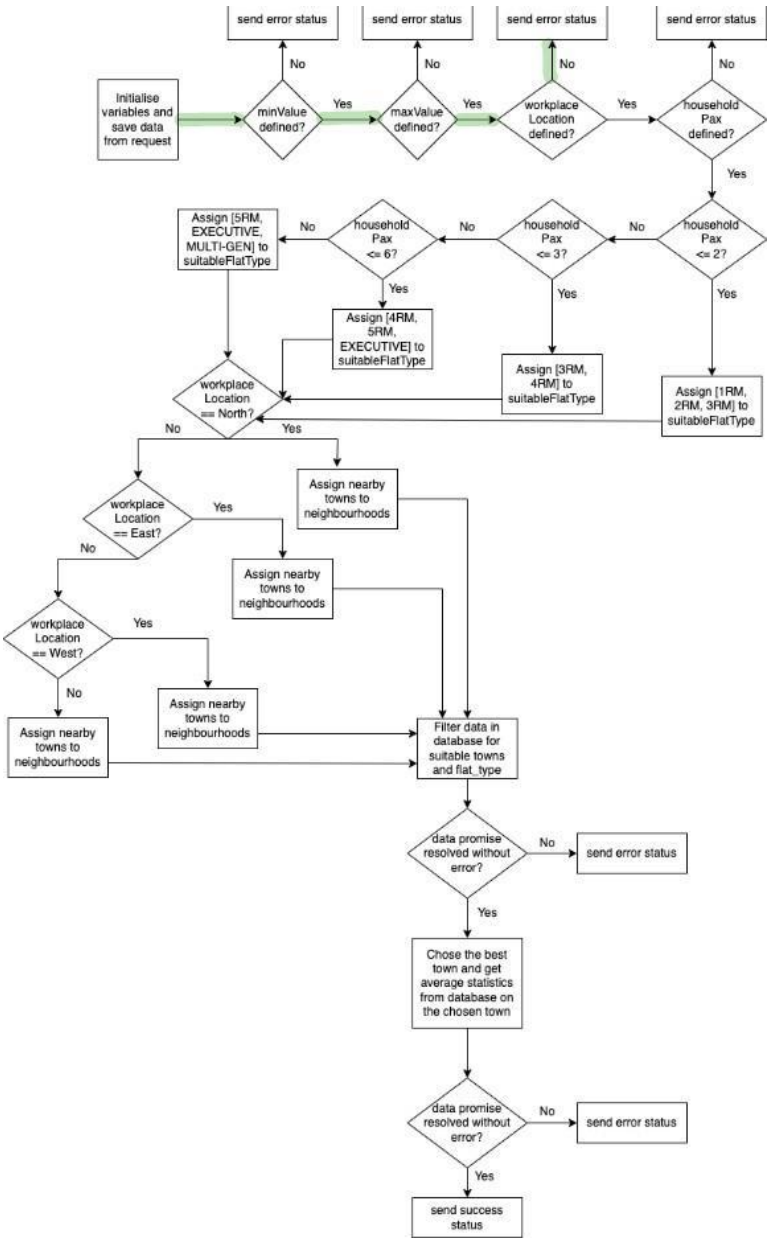
Test Code	Input	Expected Output	Actual Output	Status

WBT 1B	{query: {
	minValue: 100000,
	workplaceLocation:
	‘North’,
	householdPax: 5
	}}
		Status: 200
		Data: {
		Status: ‘error’
		}
			Status: 200
			Data: {
			Status: ‘error’
			}	



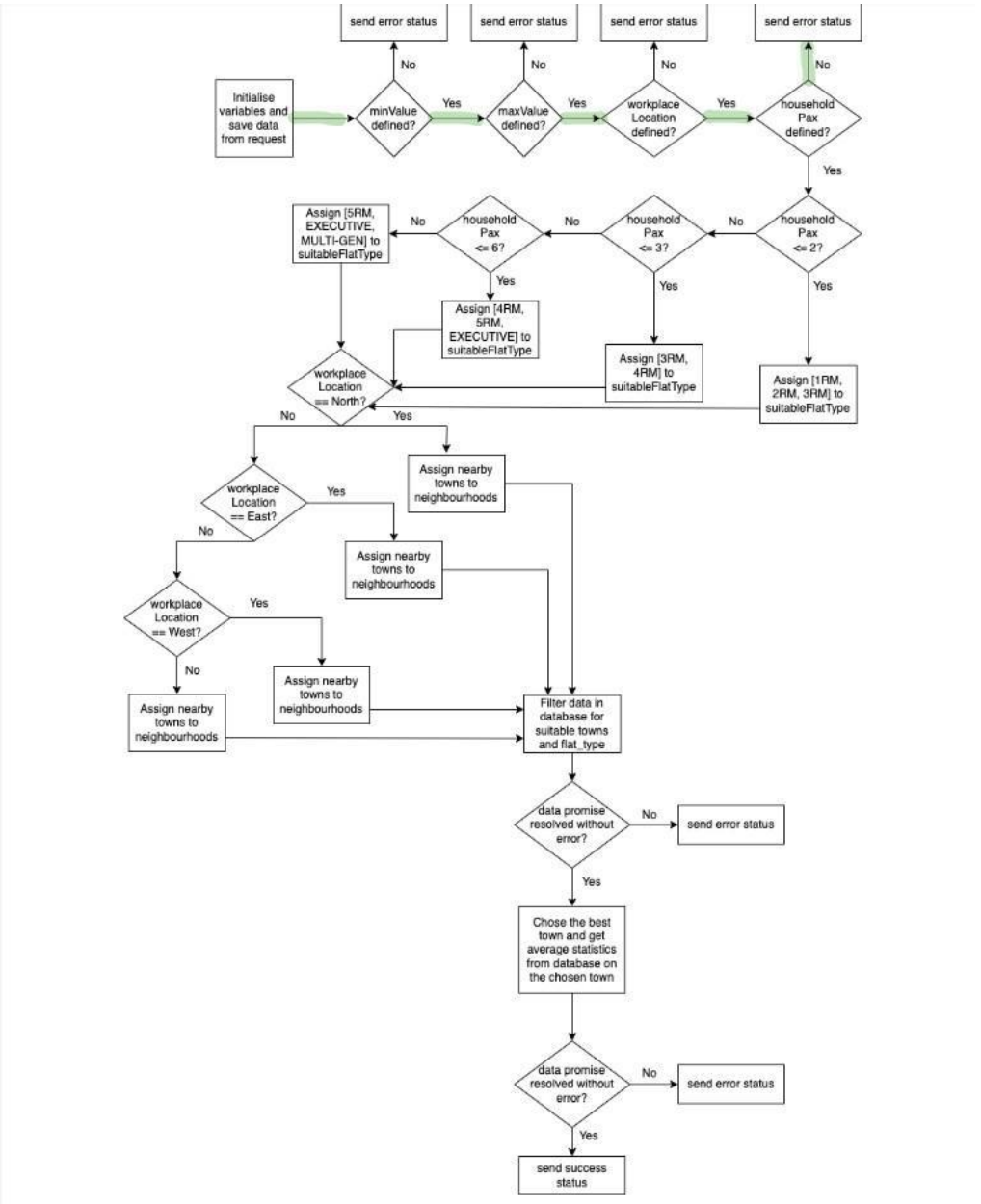
Test Code	Input	Expected Output	Actual Output	Status

WBT 1C	{query: {
	minValue: 100000,
	maxValue: 1000000,
	householdPax: 5
	}}
		Status: 200
		Data: {
		Status: ‘error’
		}
			Status: 200
			Data: {
			Status: ‘error’
			}	



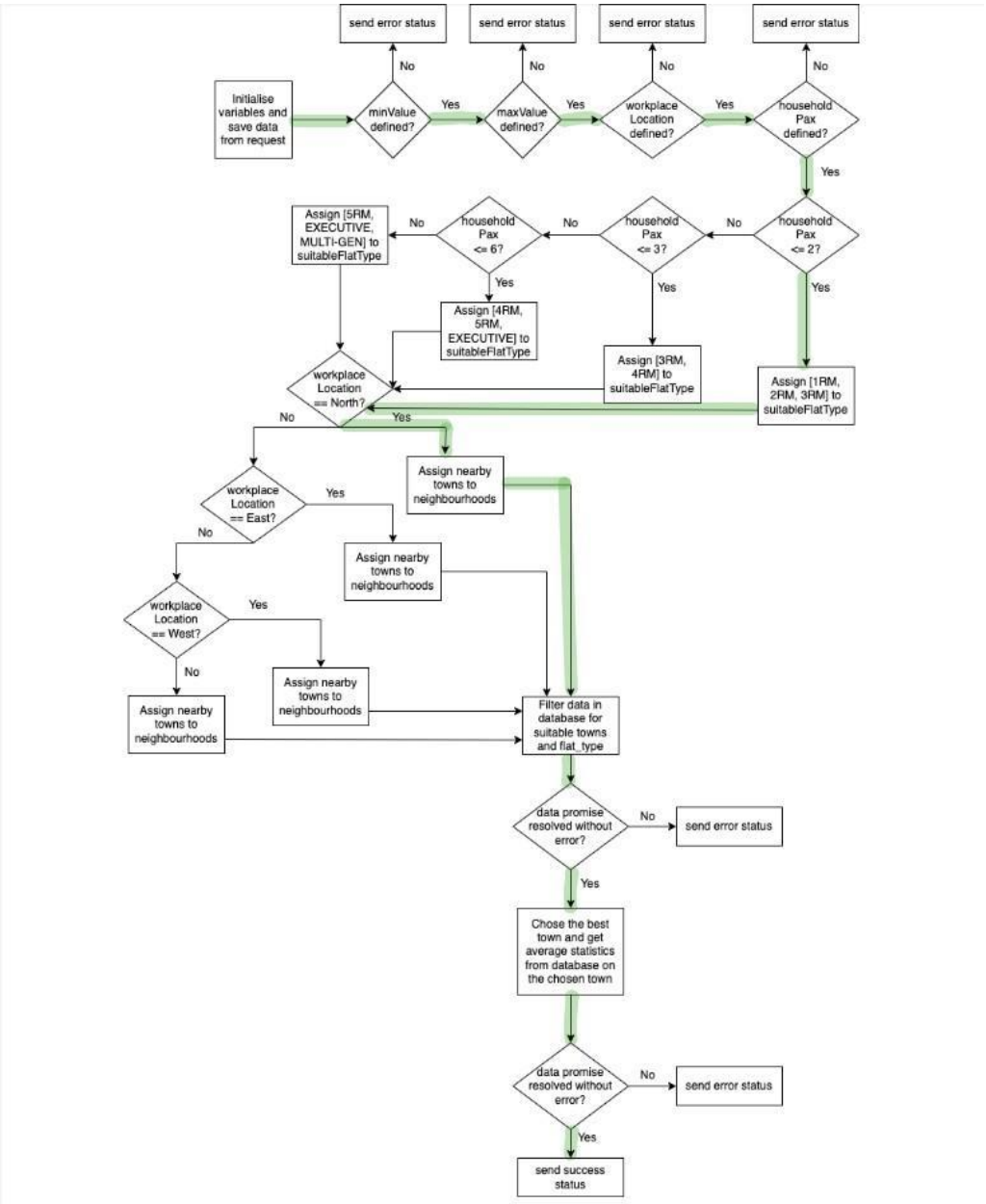
Test Code	Input	Expected Output	Actual Output	Status

WBT 1D	{query: {
	minValue: 0,
	maxValue: 1000000,
	workplaceLocation:
	'North'
	}}
		Status: 200
		Data: {
		Status: 'error'
		}
			Status: 200
			Data: {
			Status: 'error'
			}	



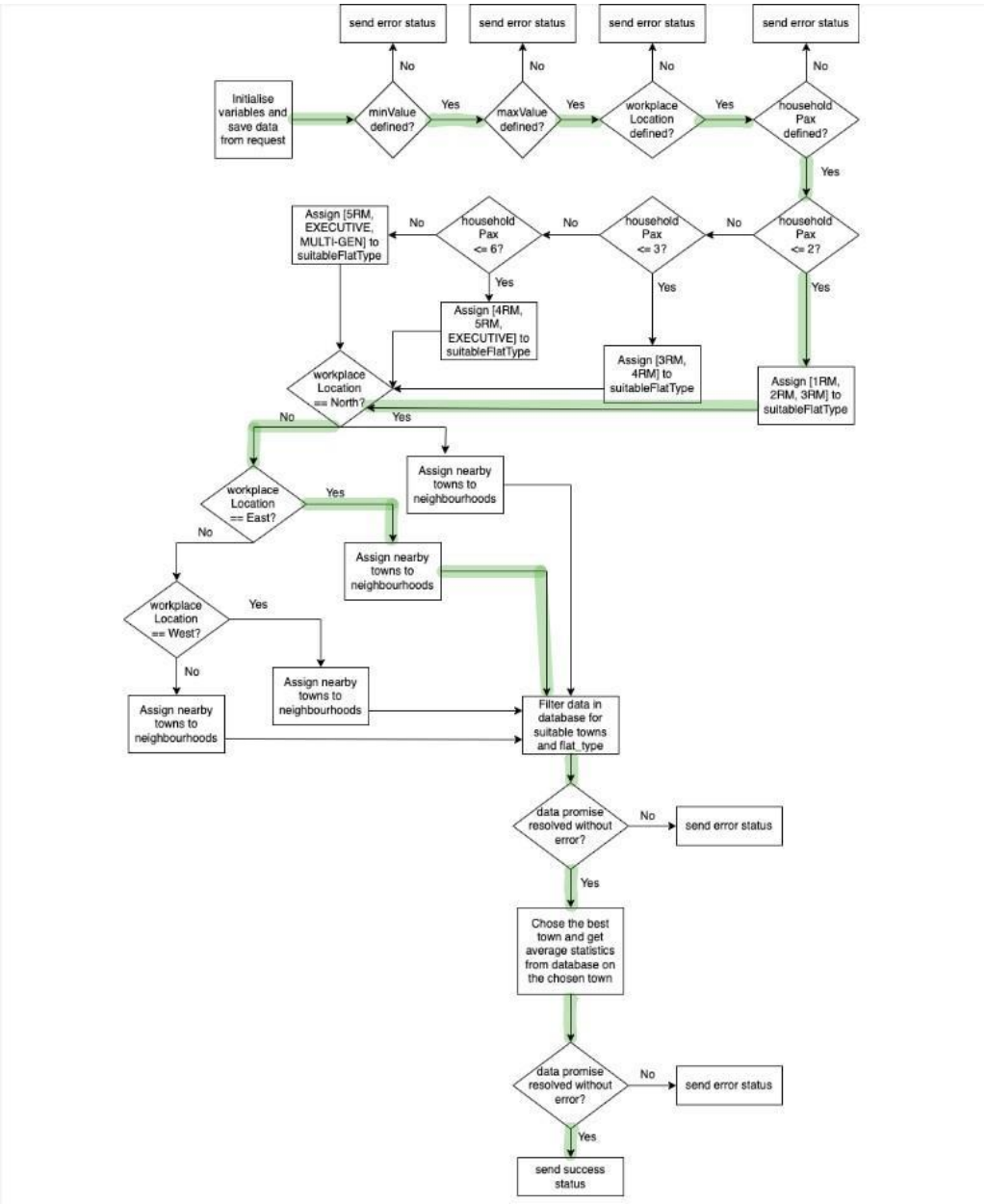
Test Code	Input	Expected Output	Actual Output	Status

WBT 1E	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'North',
	householdPax: 2
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



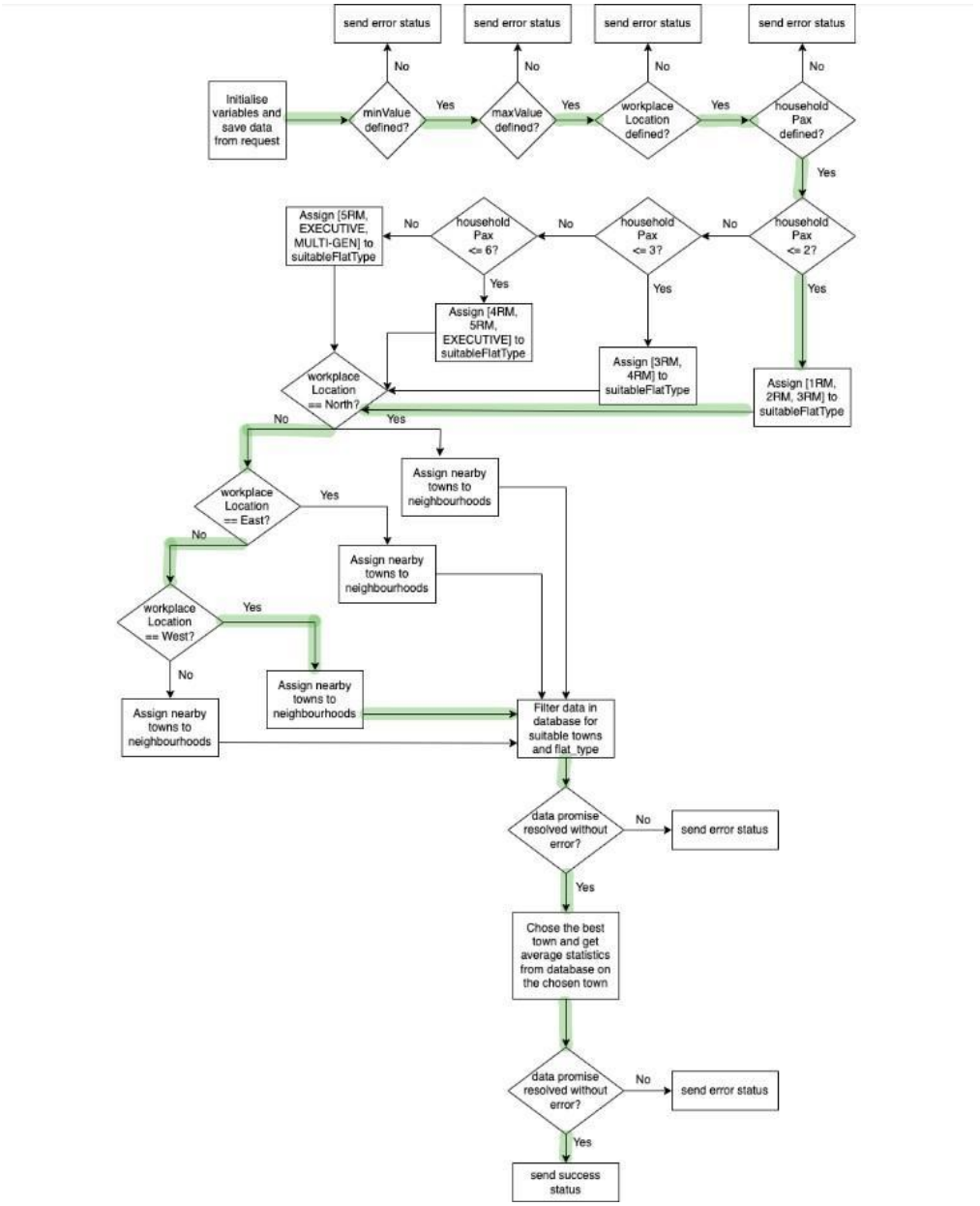
Test Code	Input	Expected Output	Actual Output	Status

WBT 1F	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	‘East’,
	householdPax: 2
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



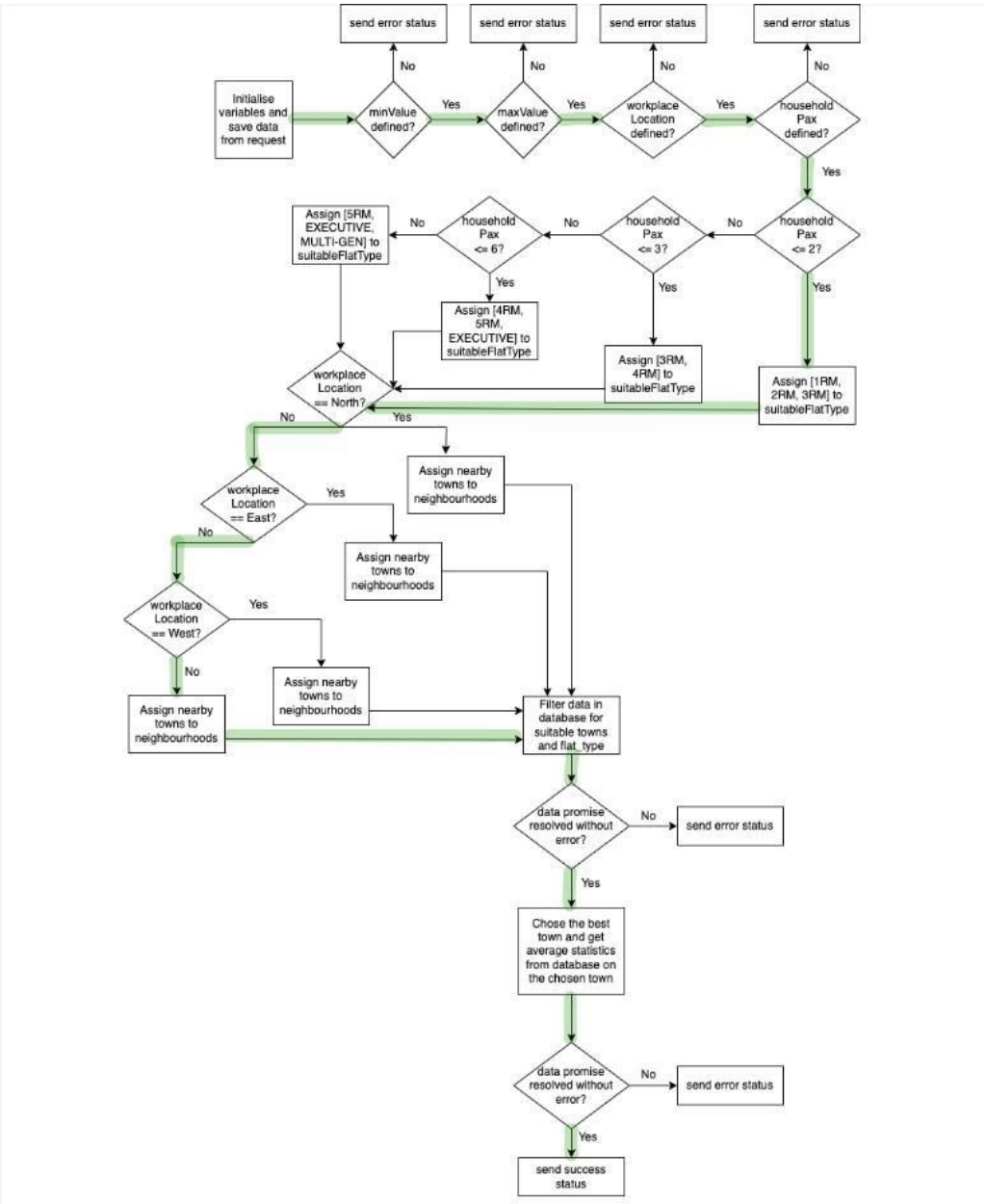
Test Code	Input	Expected Output	Actual Output	Status

WBT 1G	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'West',
	householdPax: 2
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



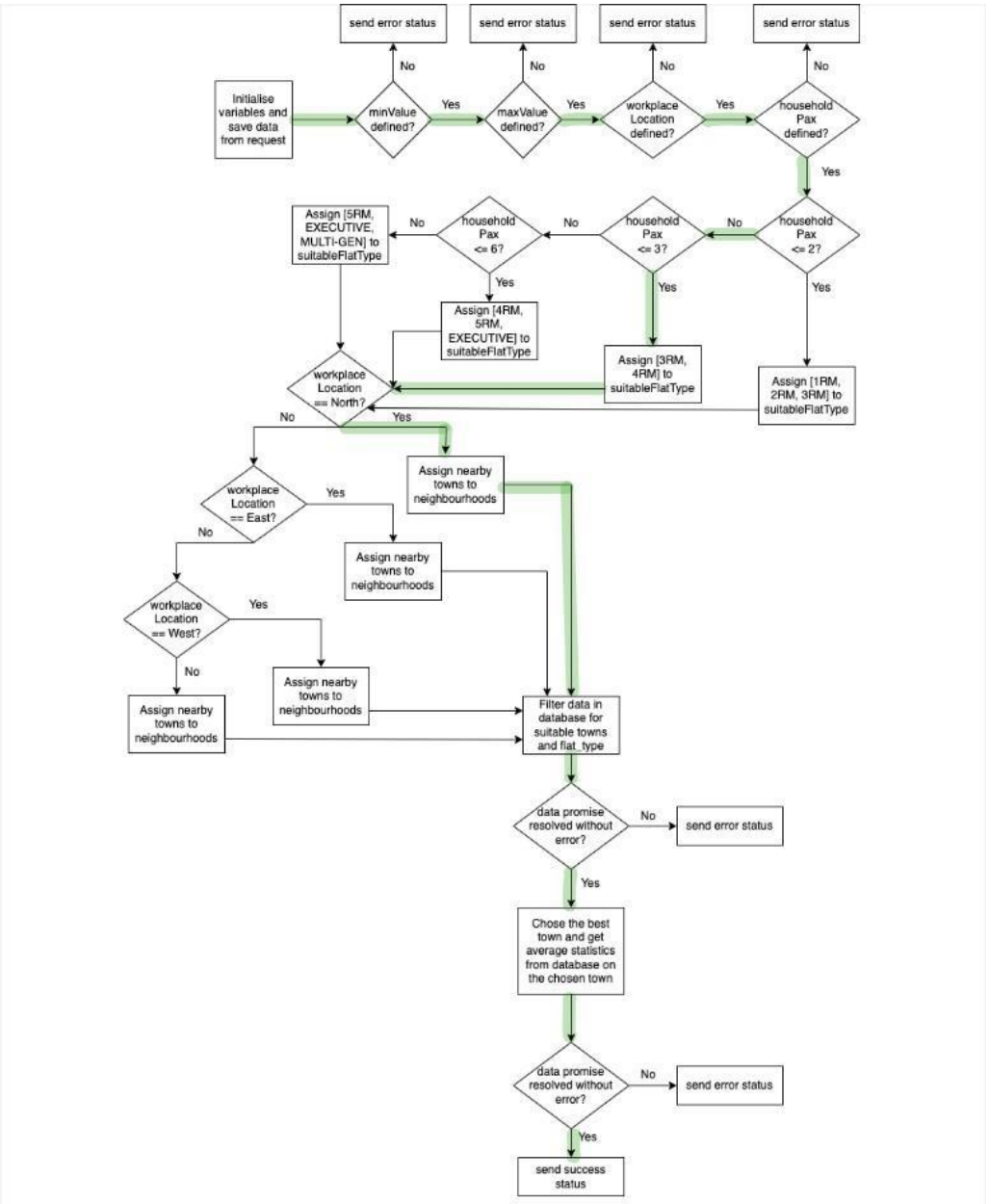
Test Code	Input	Expected Output	Actual Output	Status

WBT 1H	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	‘South’,
	householdPax: 2
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



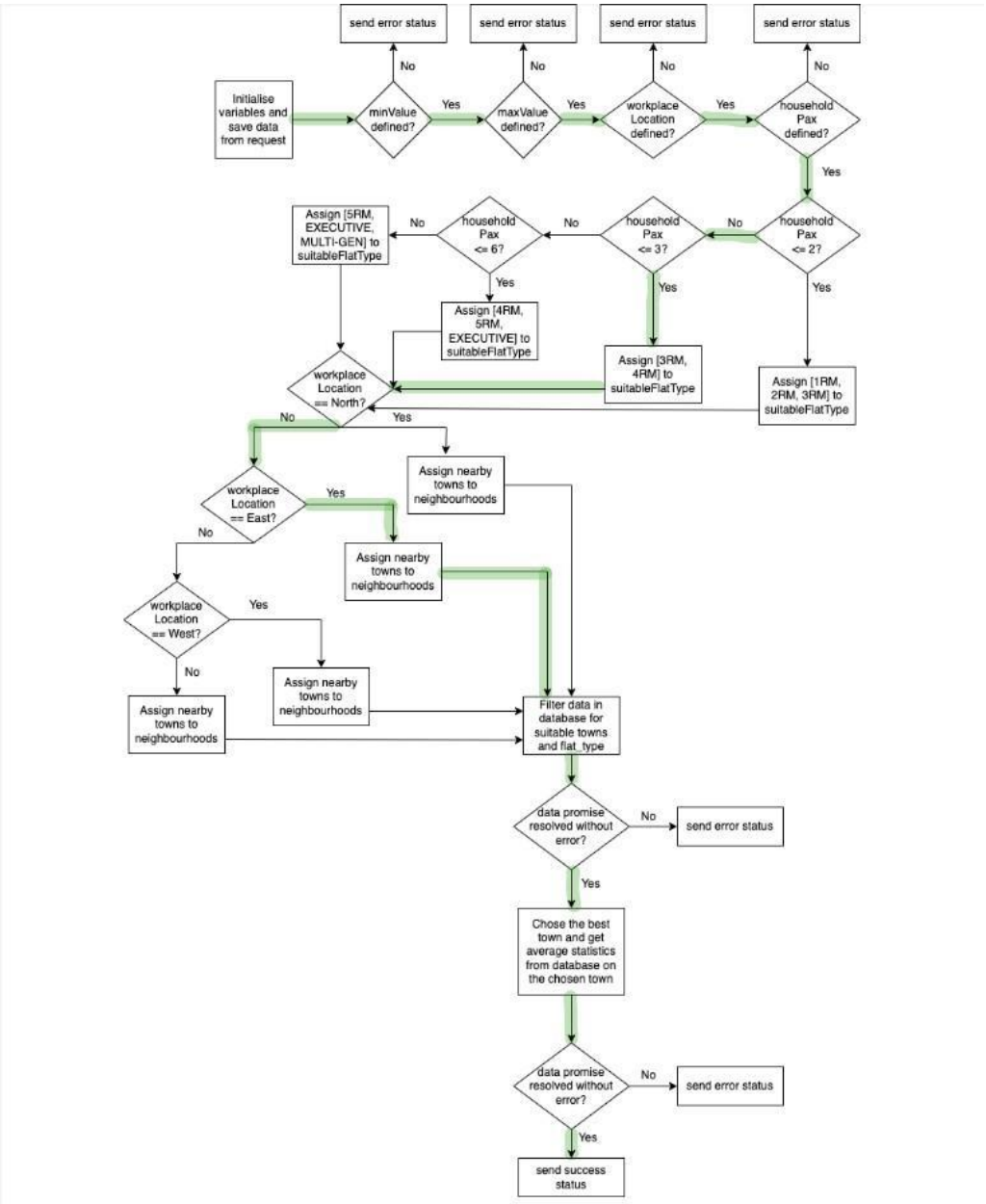
Test Code	Input	Expected Output	Actual Output	Status

WBT 1I	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'North',
	householdPax: 3
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



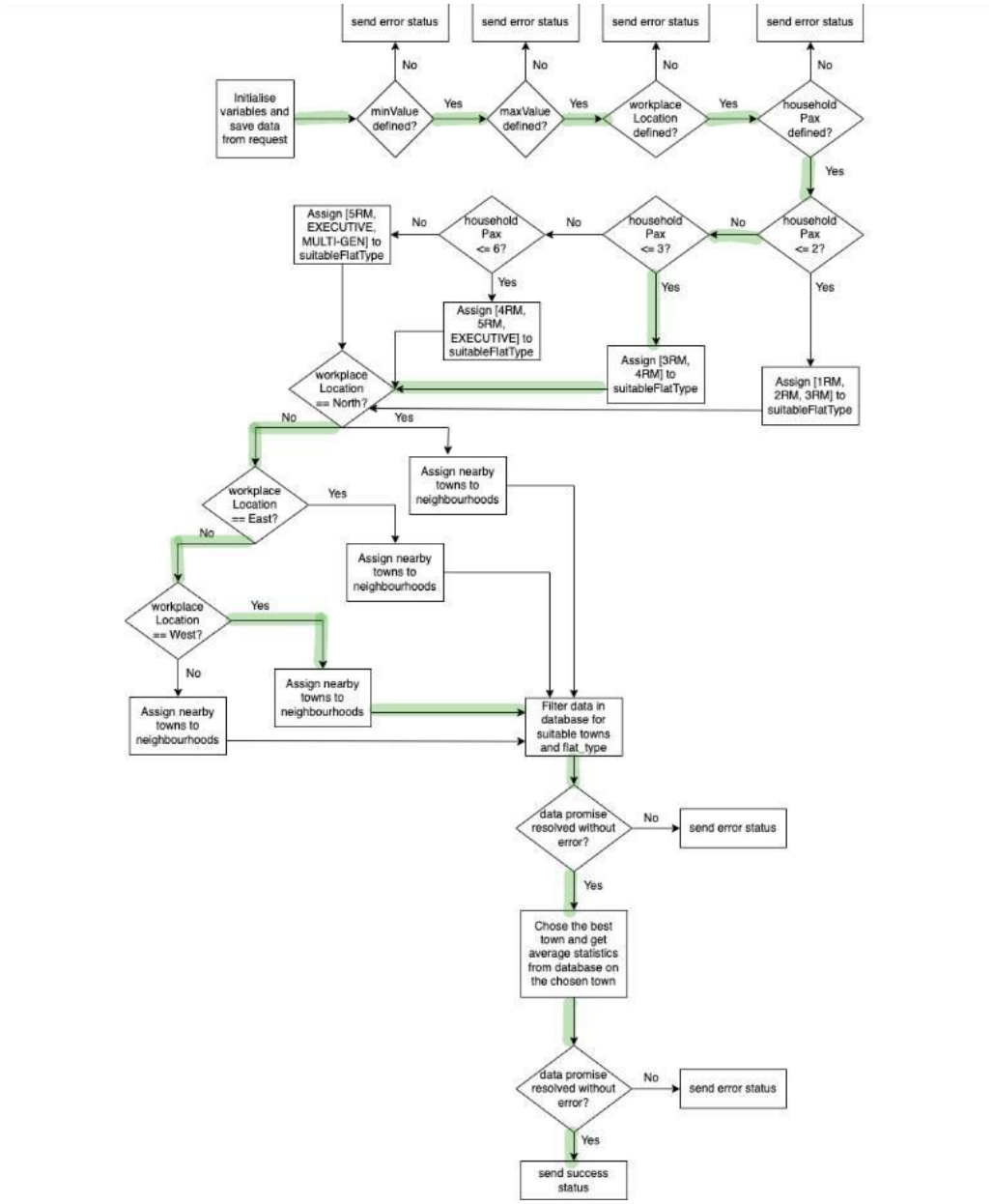
Test Code	Input	Expected Output	Actual Output	Status

WBT 1J	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	‘East’,
	householdPax: 3
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



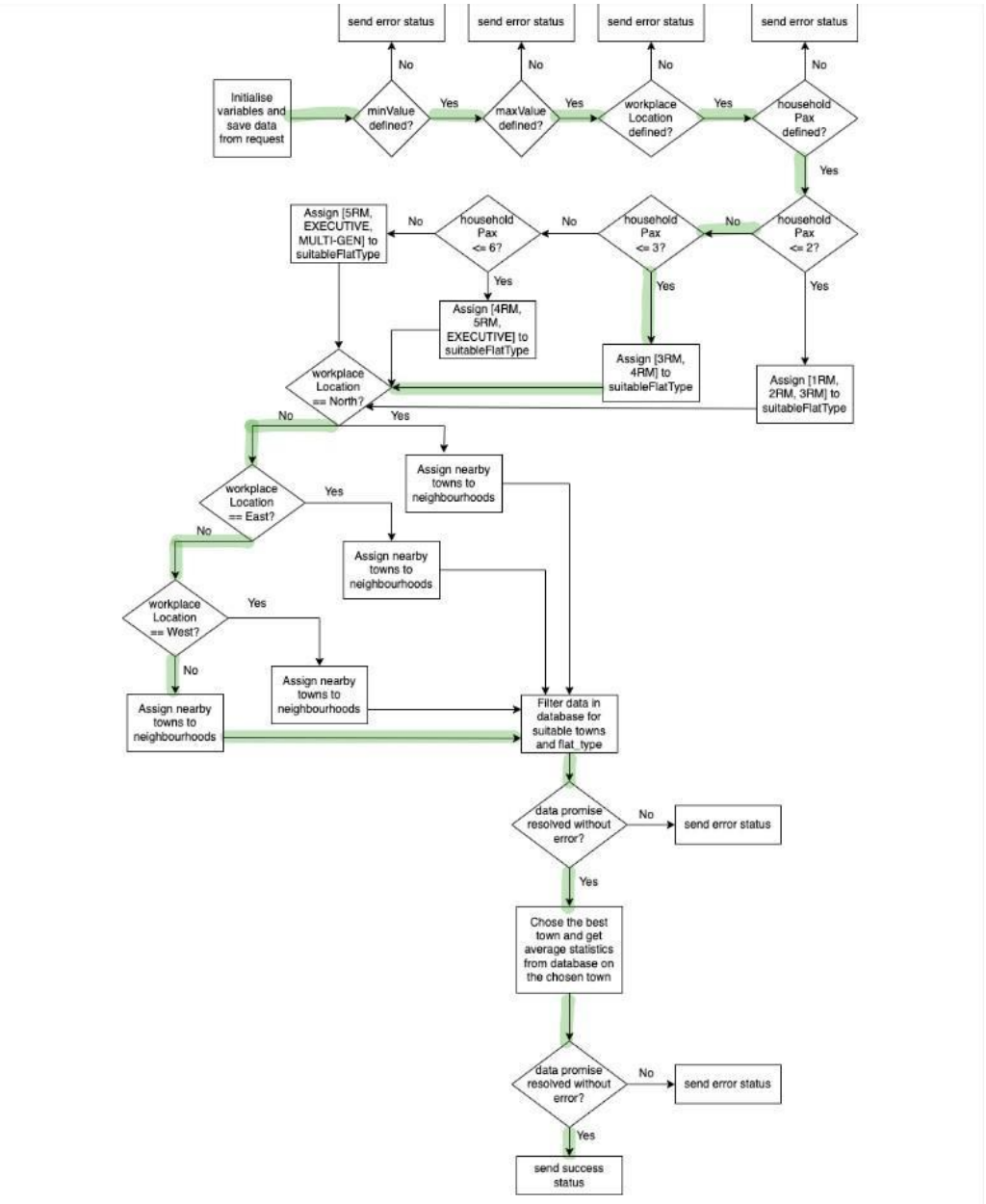
Test Code	Input	Expected Output	Actual Output	Status

WBT 1K	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'West',
	householdPax: 3
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



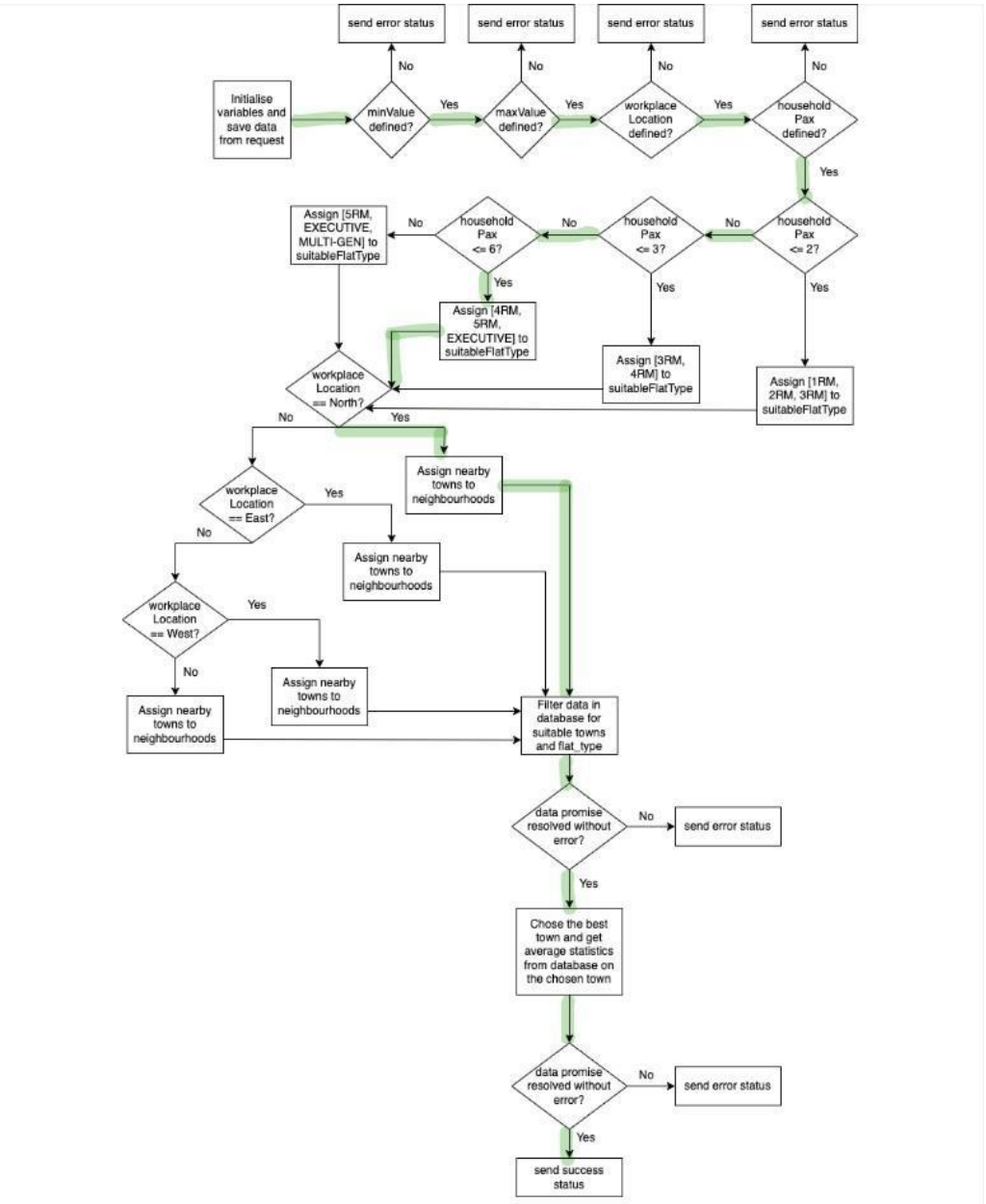
Test Code	Input	Expected Output	Actual Output	Status

WBT 1L	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'South',
	householdPax: 3
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



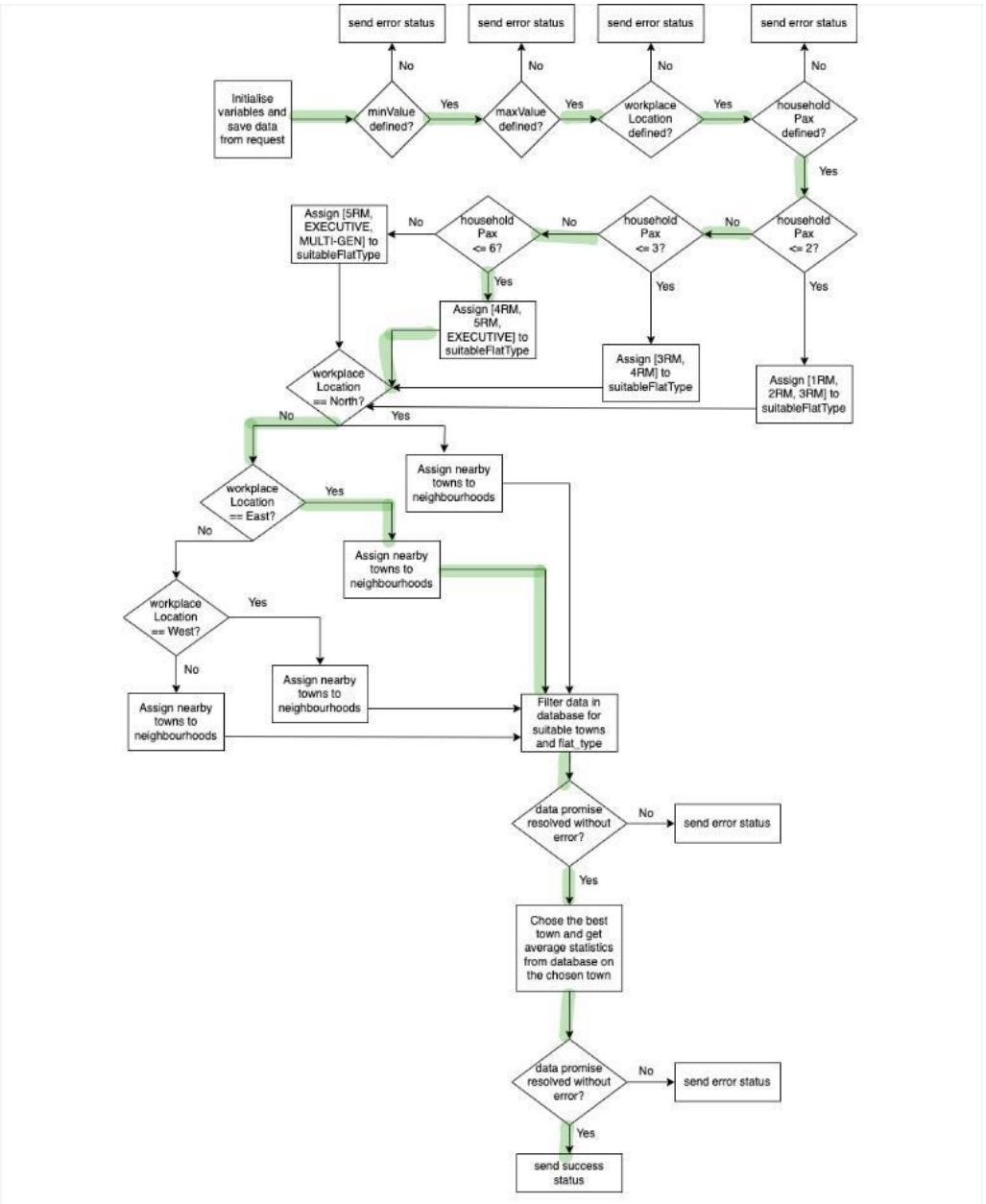
Test Code	Input	Expected Output	Actual Output	Status

WBT 1M	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'North',
	householdPax: 4
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



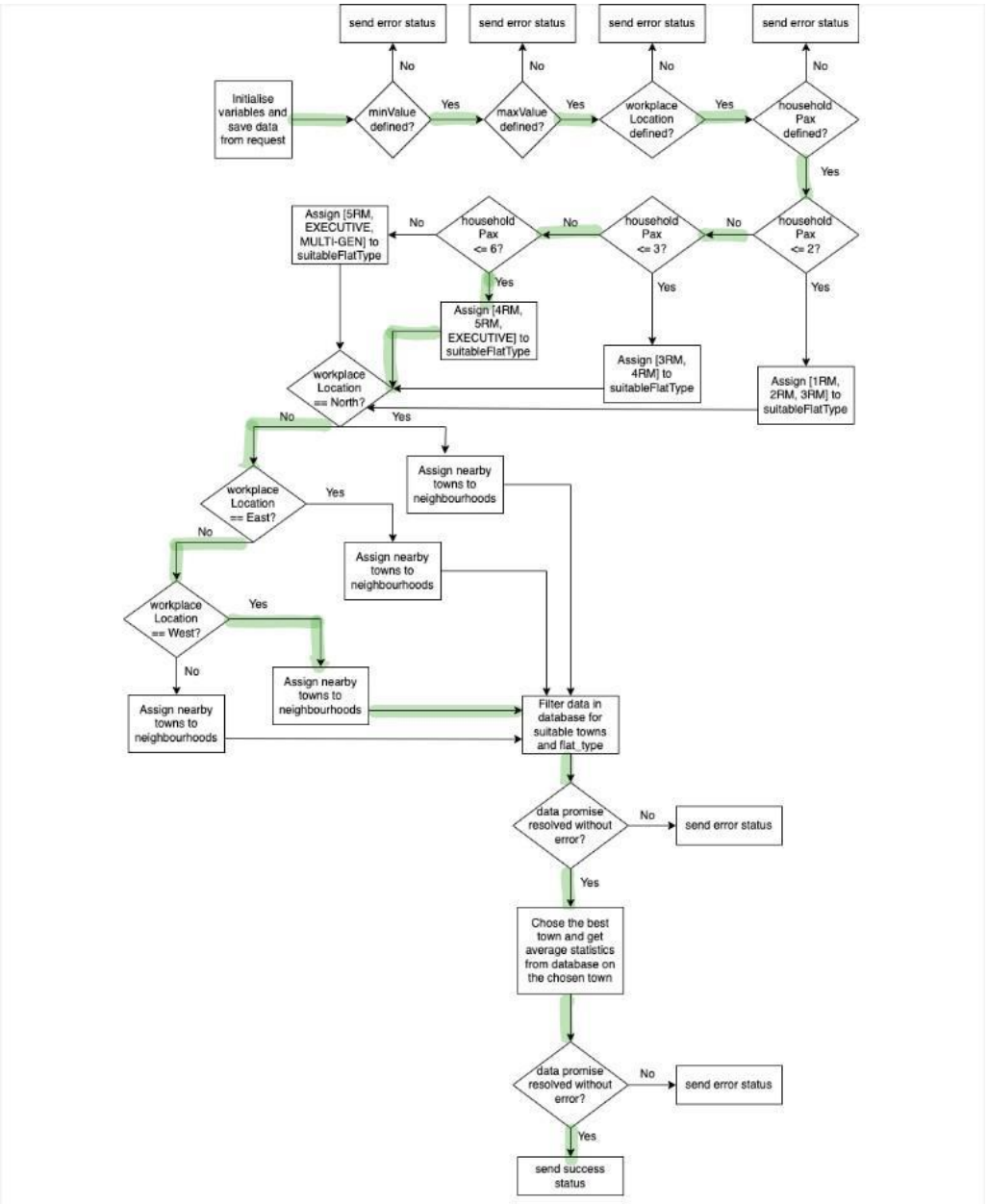
Test Code	Input	Expected Output	Actual Output	Status

WBT 1N	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	‘East’,
	householdPax: 4
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



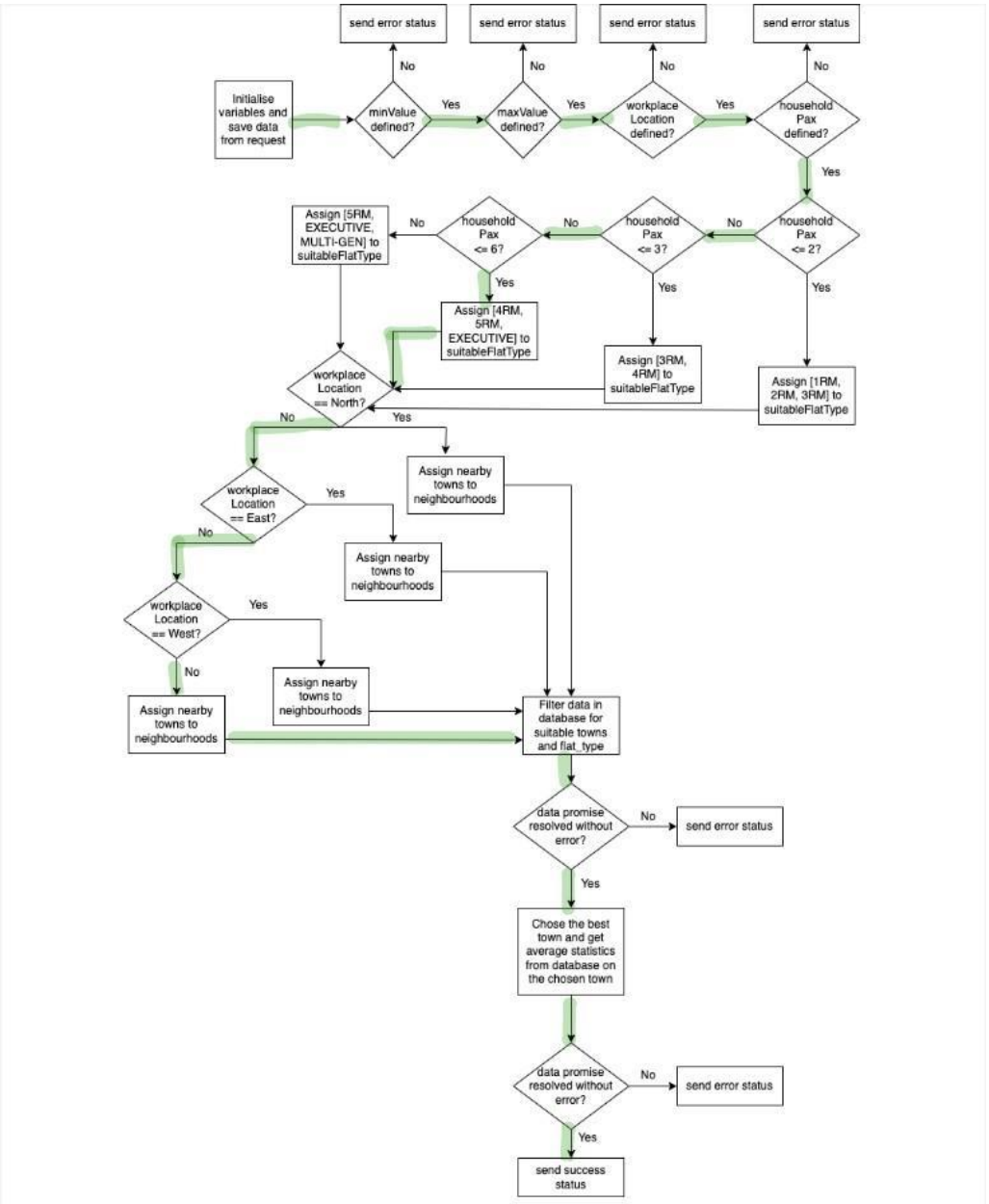
Test Code	Input	Expected Output	Actual Output	Status

WBT 1O	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'West',
	householdPax: 4
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



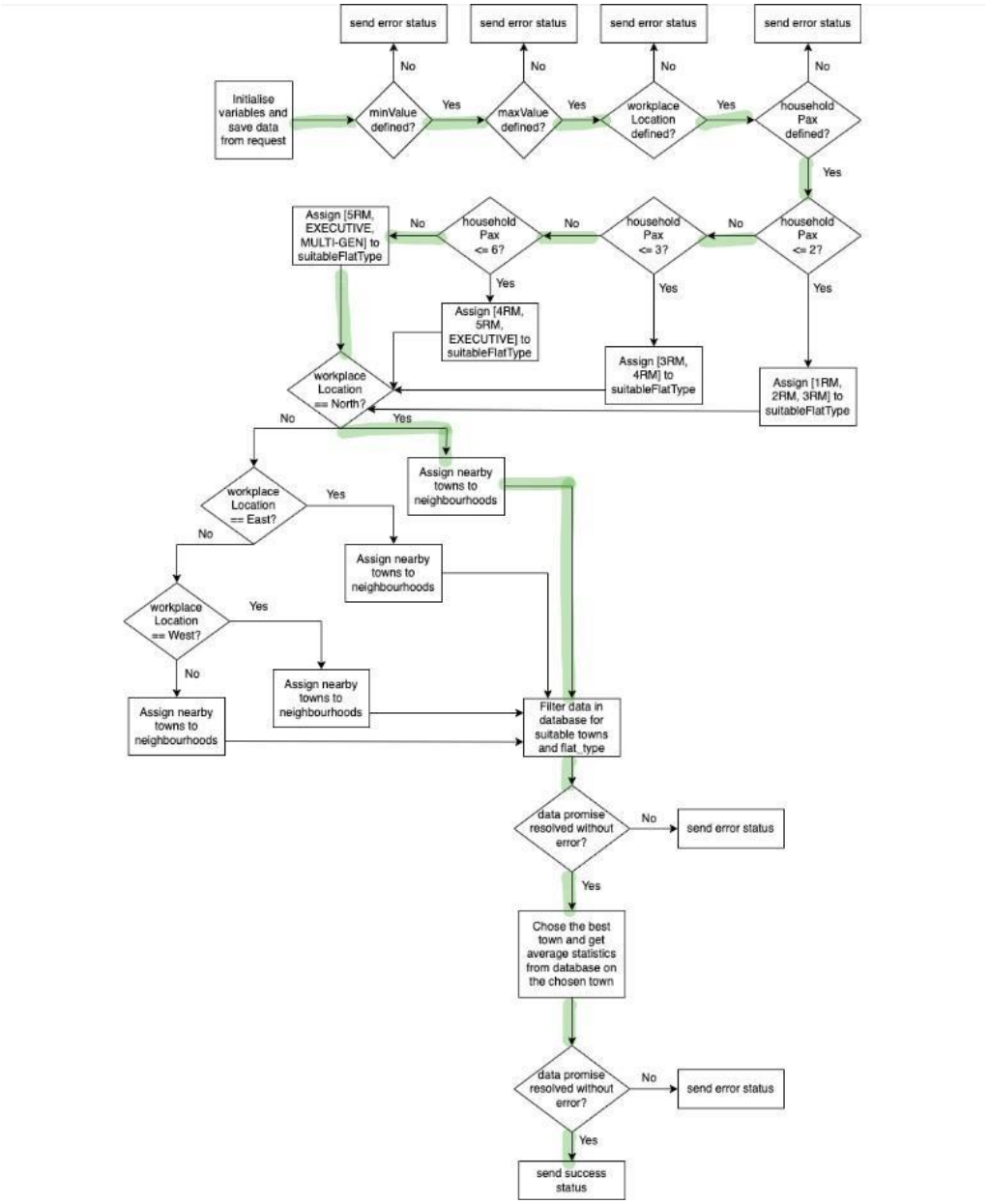
Test Code	Input	Expected Output	Actual Output	Status

WBT 1P	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'South',
	householdPax: 4
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



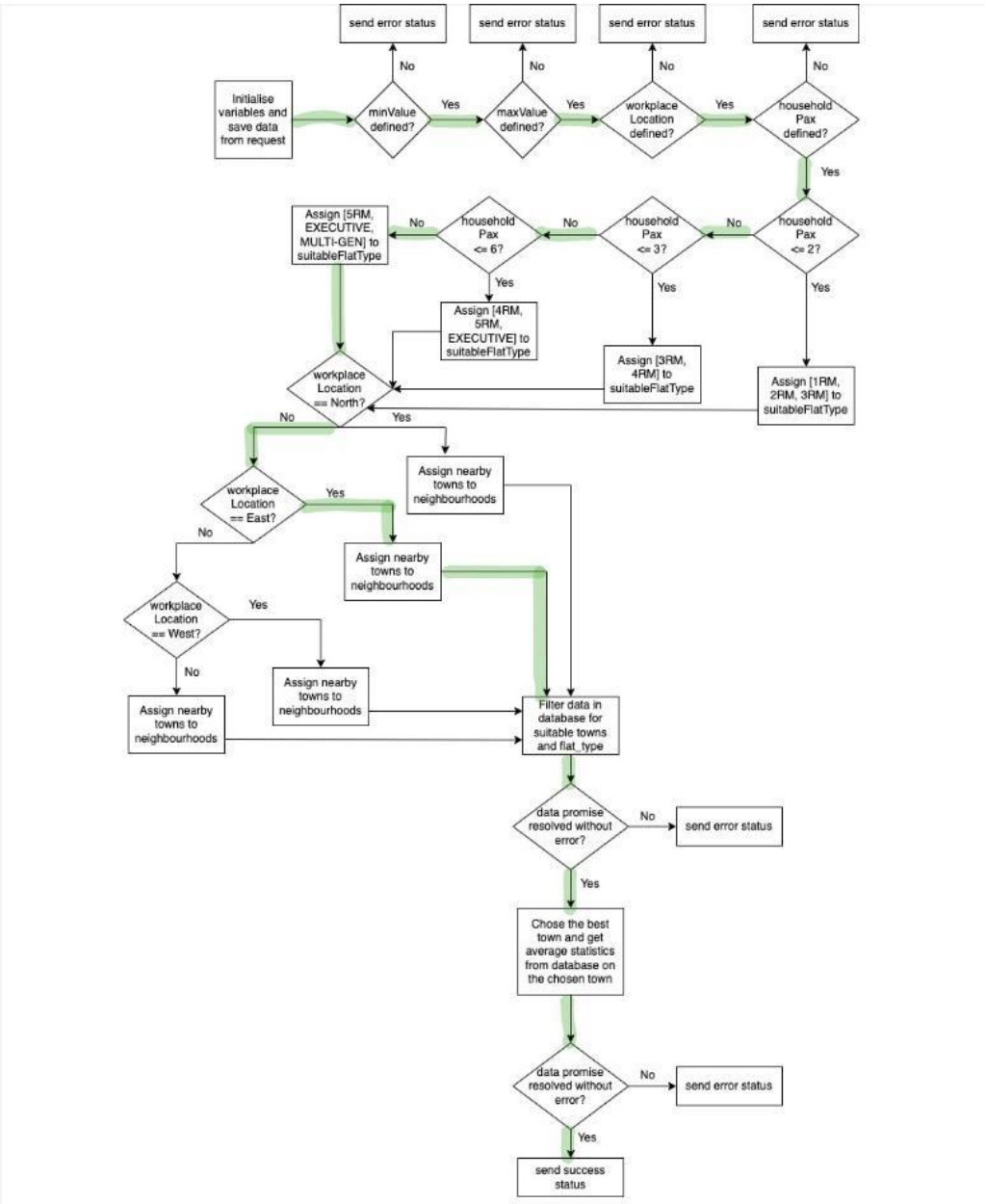
Test Code	Input	Expected Output	Actual Output	Status

WBT 1R	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'North',
	householdPax: 8
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	



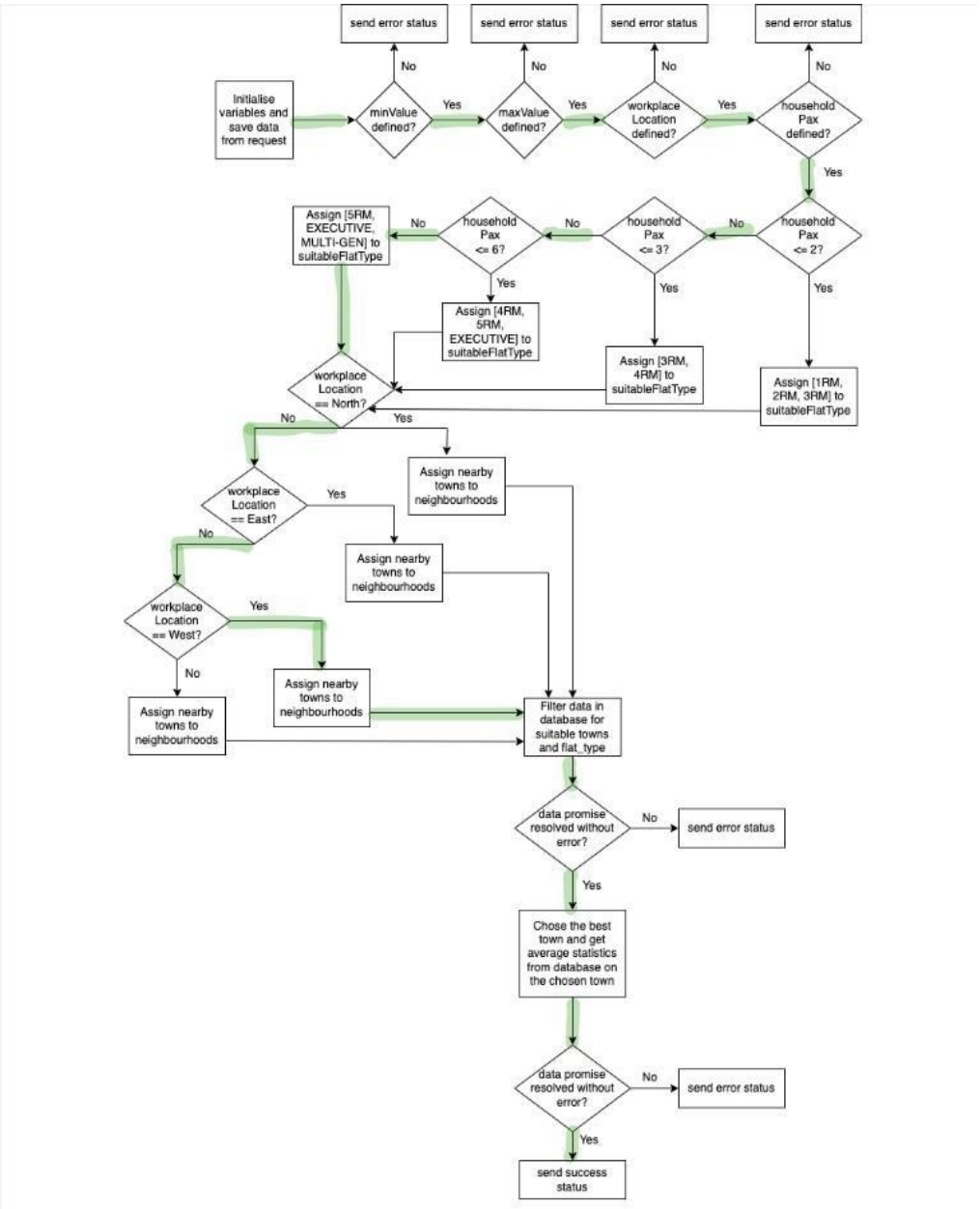
Test Code	Input	Expected Output	Actual Output	Status

WBT 1S	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	‘East’,
	householdPax: 8
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



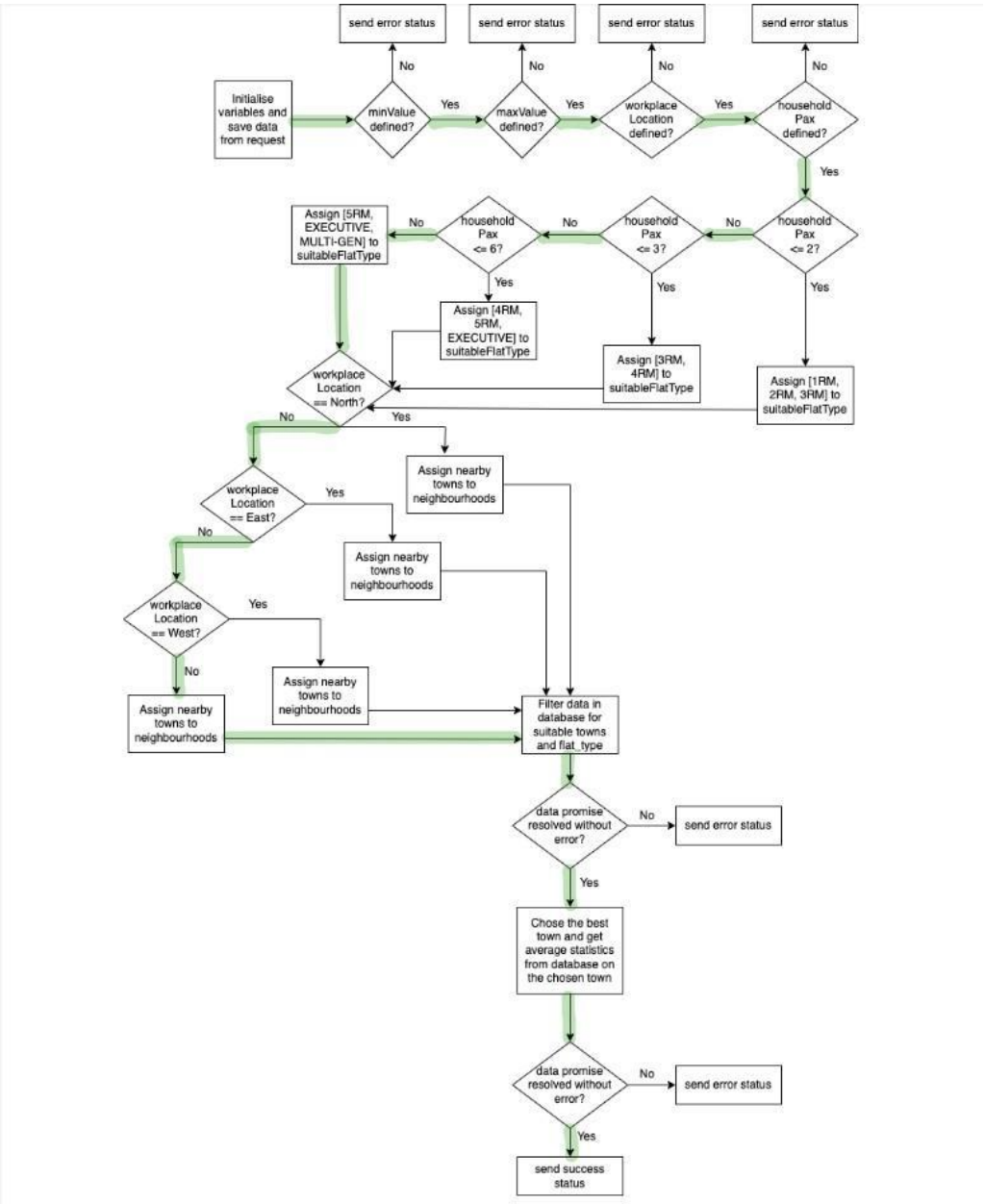
Test Code	Input	Expected Output	Actual Output	Status

WBT 1T	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'West',
	householdPax: 8
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	

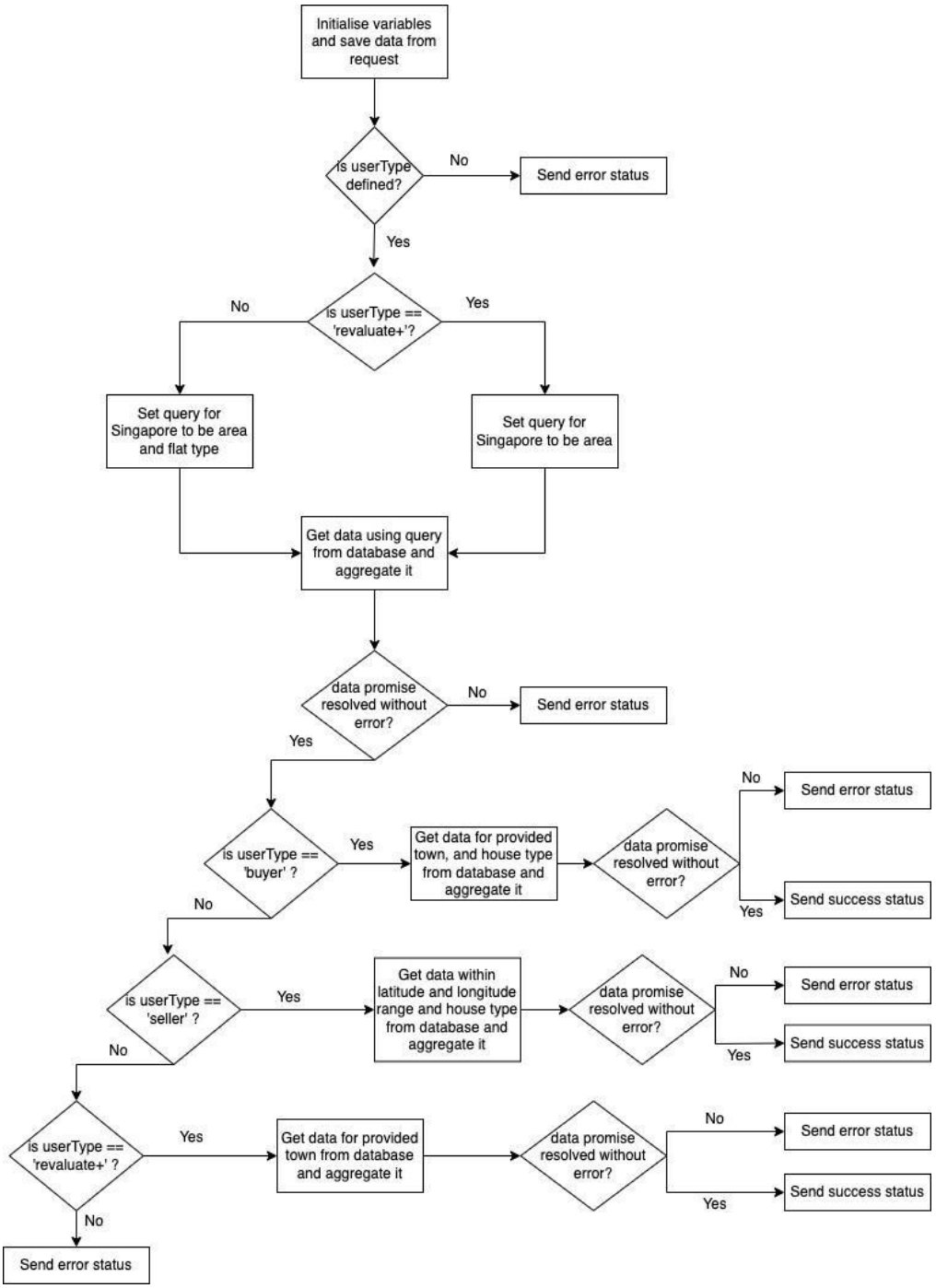


Test Code	Input	Expected Output	Actual Output	Status

WBT 1U	{query: {
	minValue: 100000,
	maxValue: 1000000,
	workplaceLocation:
	'South',
	householdPax: 8
	}}
		Status: 200
		Data: {
		Status: 'success',
		…
		}
			Status: 200
			Data: {
			Status: 'success',
			…
			}	

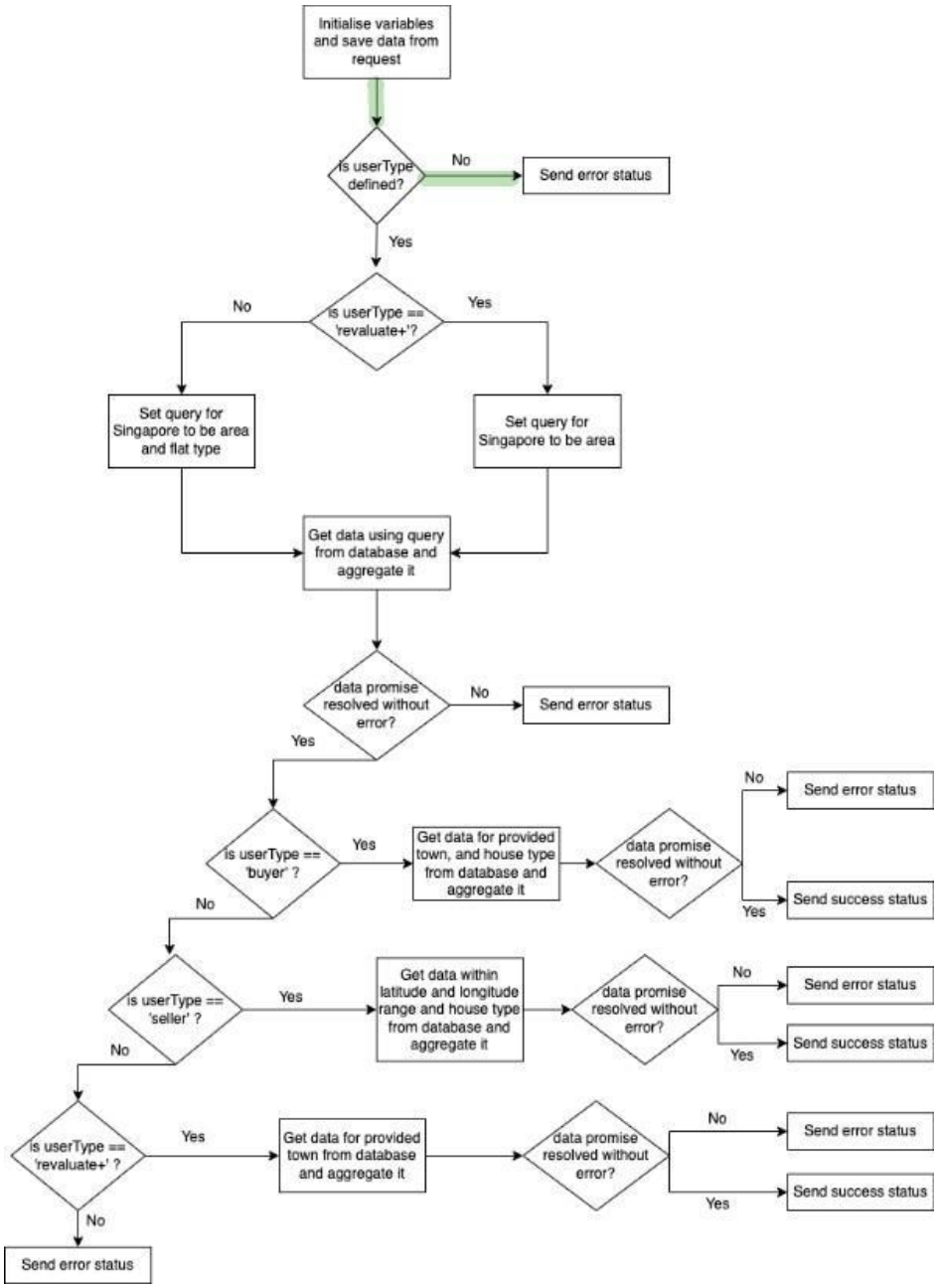


Control Flow Graph for getNeighbourhoodPriceComparisonChart(req, res)



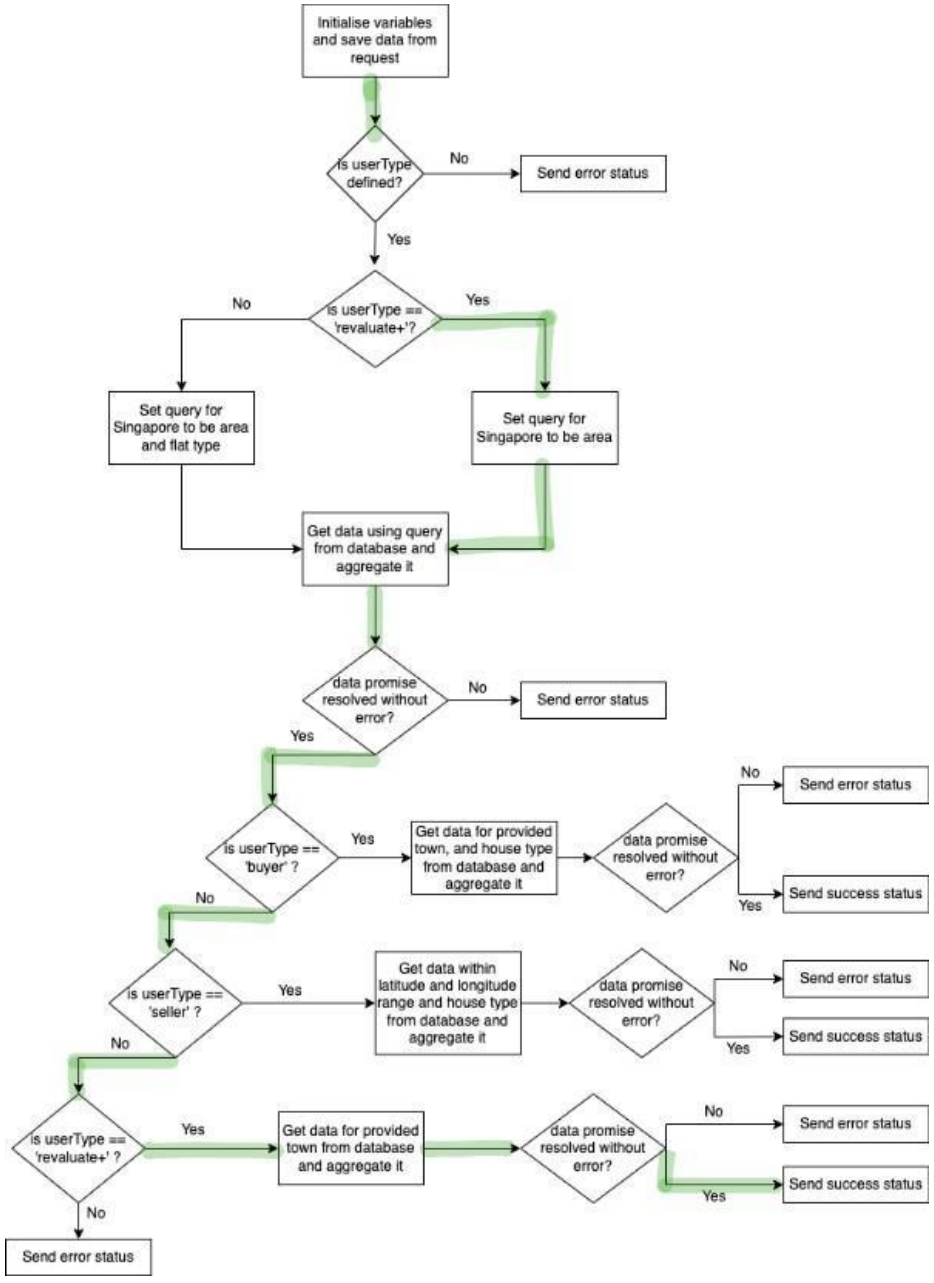
Test Code	Input	Expected Output	Actual Output	Status

WBT 2A	{query: {}}	Status: 200
		Data: {
		Status: ‘error’,
		…
		}
			Status: 200
			Data: {
			Status: ‘error’,
			…
			}	



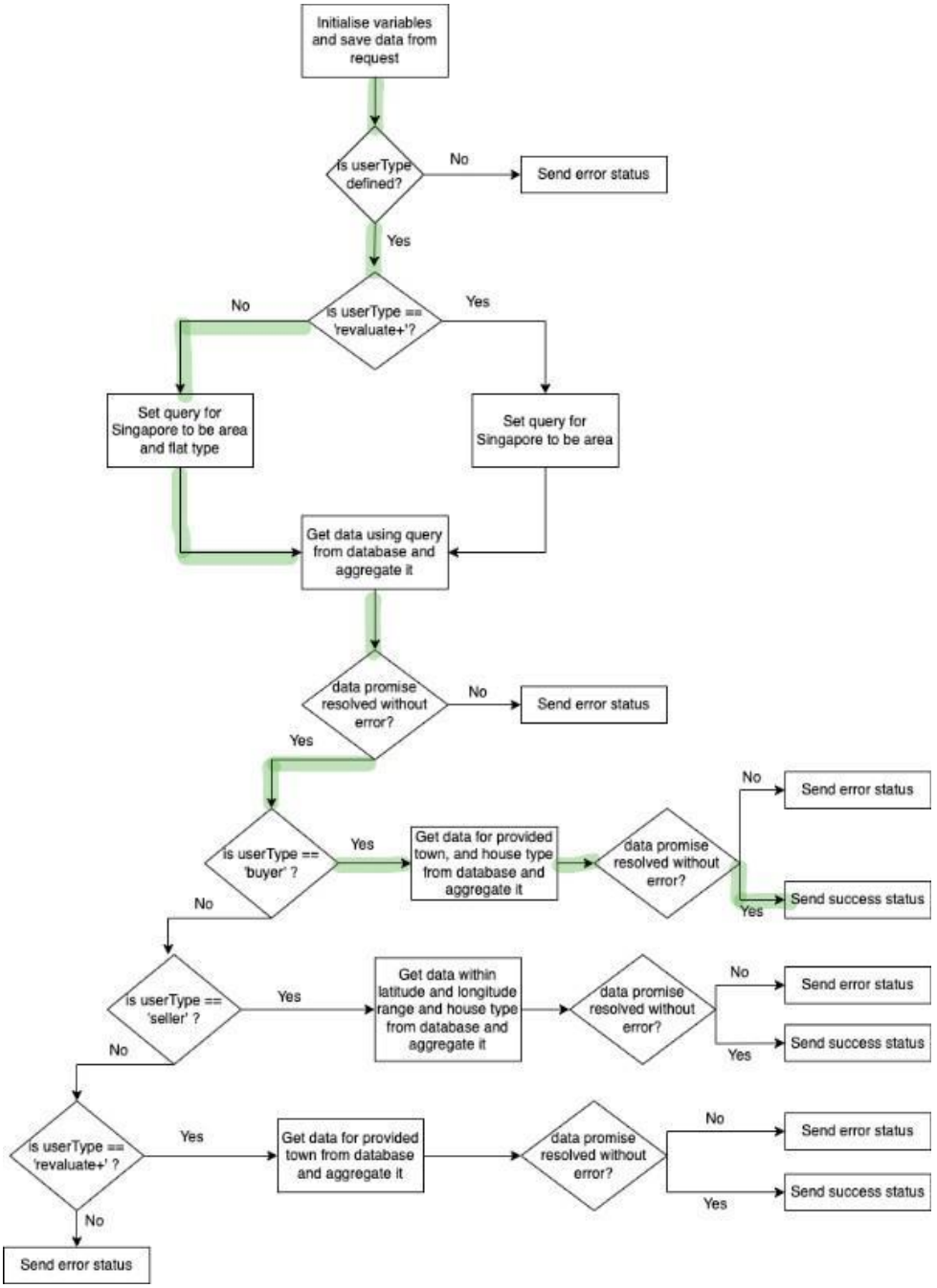
Test Code	Input	Expected Output	Actual Output	Status

WBT 2B	{query: {
	userType:
	‘revaluate+’
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



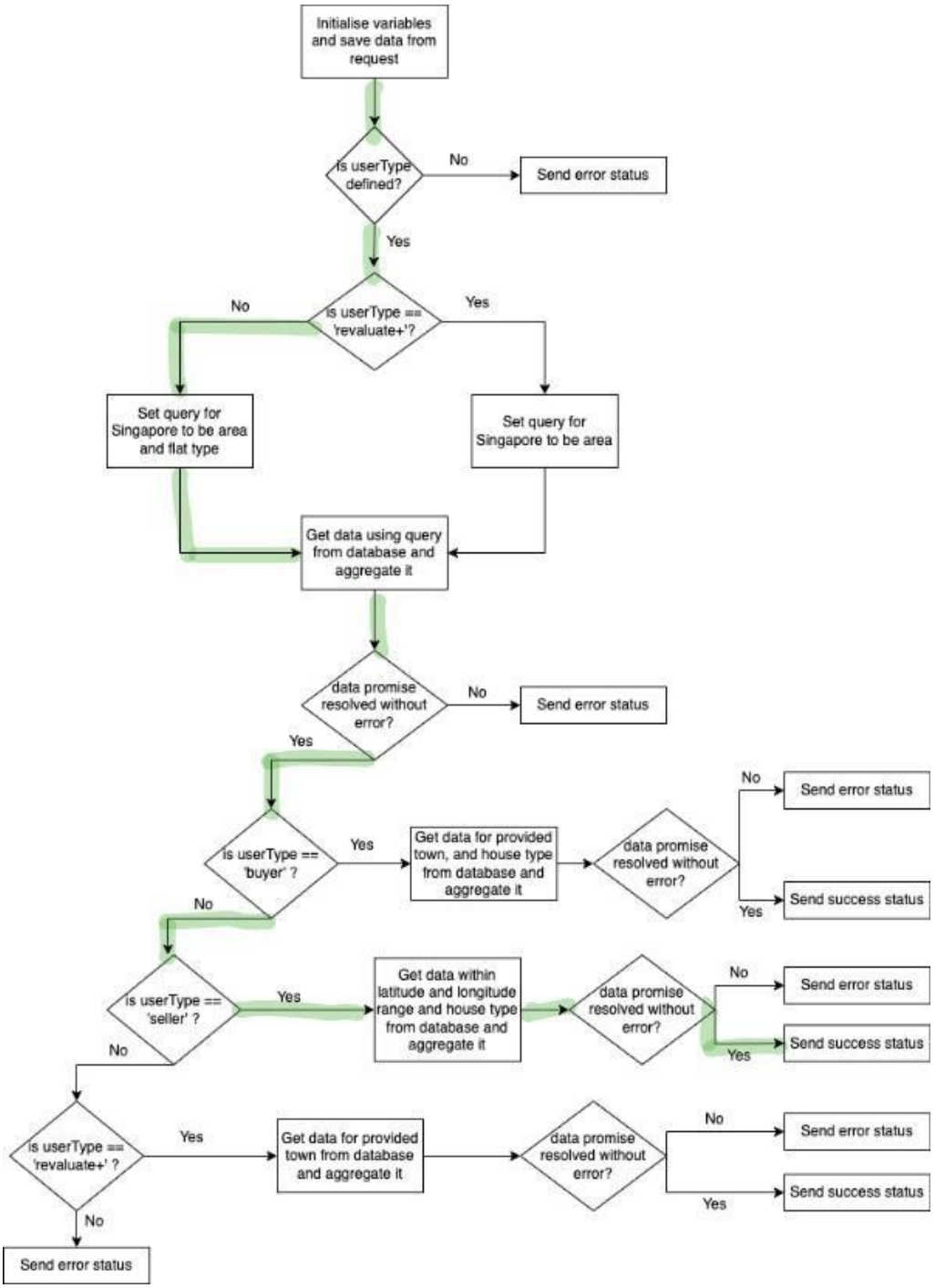
Test Code	Input	Expected Output	Actual Output	Status

WBT 2C	{query: {
	userType: ‘buyer’,
	flat_type: ‘5 ROOM’,
	town: ‘BEDOK’
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



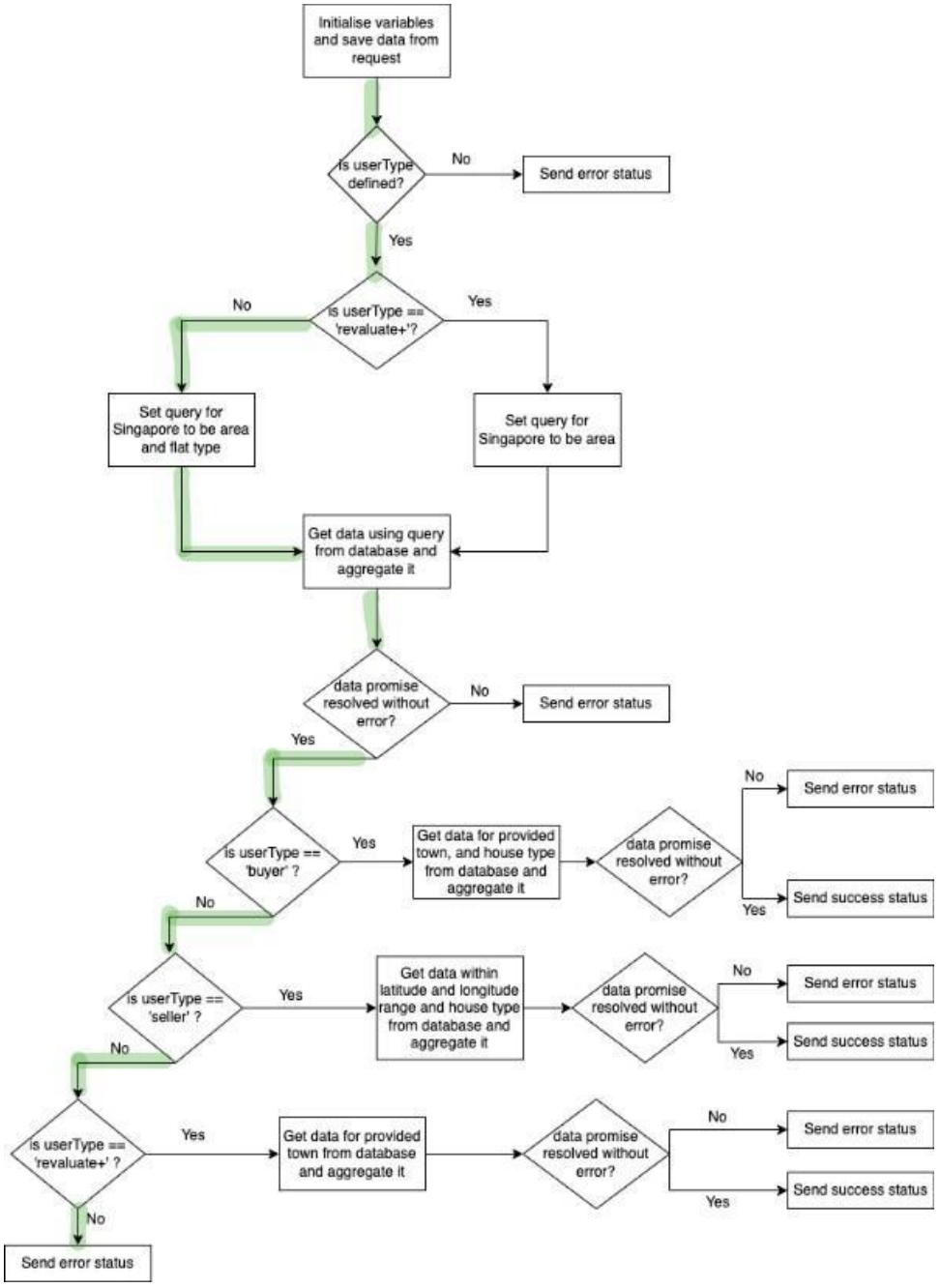
Test Code	Input	Expected Output	Actual Output	Status

WBT 2D	{query: {
	userType: ‘seller’,
	flat_type: ‘5 ROOM’,
	latitude: 1.25,
	longitude: 103.81
	}}
		Status: 200
		Data: {
		Status: ‘success’,
		…
		}
			Status: 200
			Data: {
			Status: ‘success’,
			…
			}	



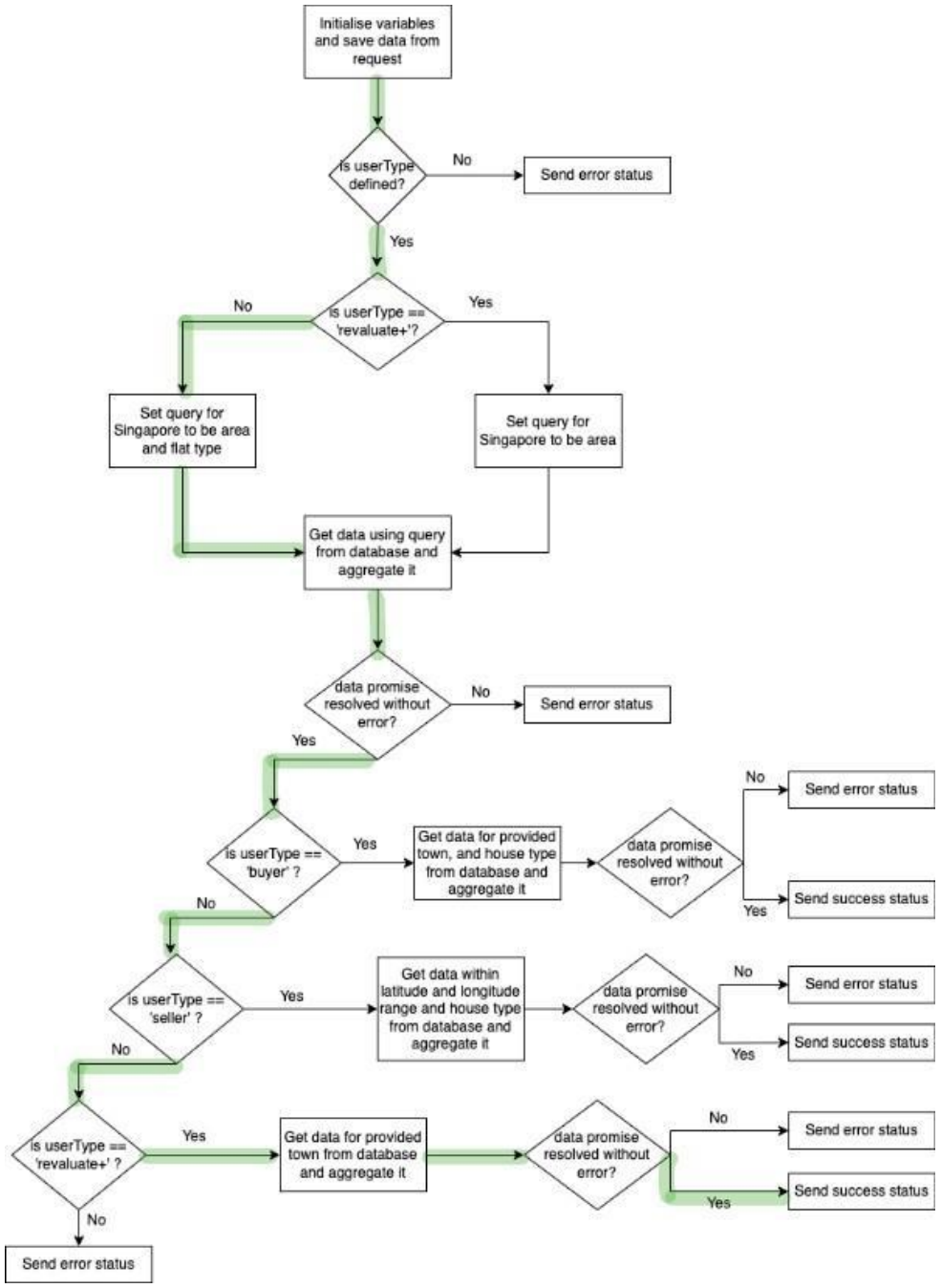
Test Code	Input	Expected Output	Actual Output	Status

WBT 2E	{query: {
	userType: ‘random’,
	flat_type: ‘5 ROOM’
	}}
		Status: 200
		Data: {
		Status: ‘error’,
		…
		}
			Status: 200
			Data: {
			Status: ‘error’,
			…
			}	

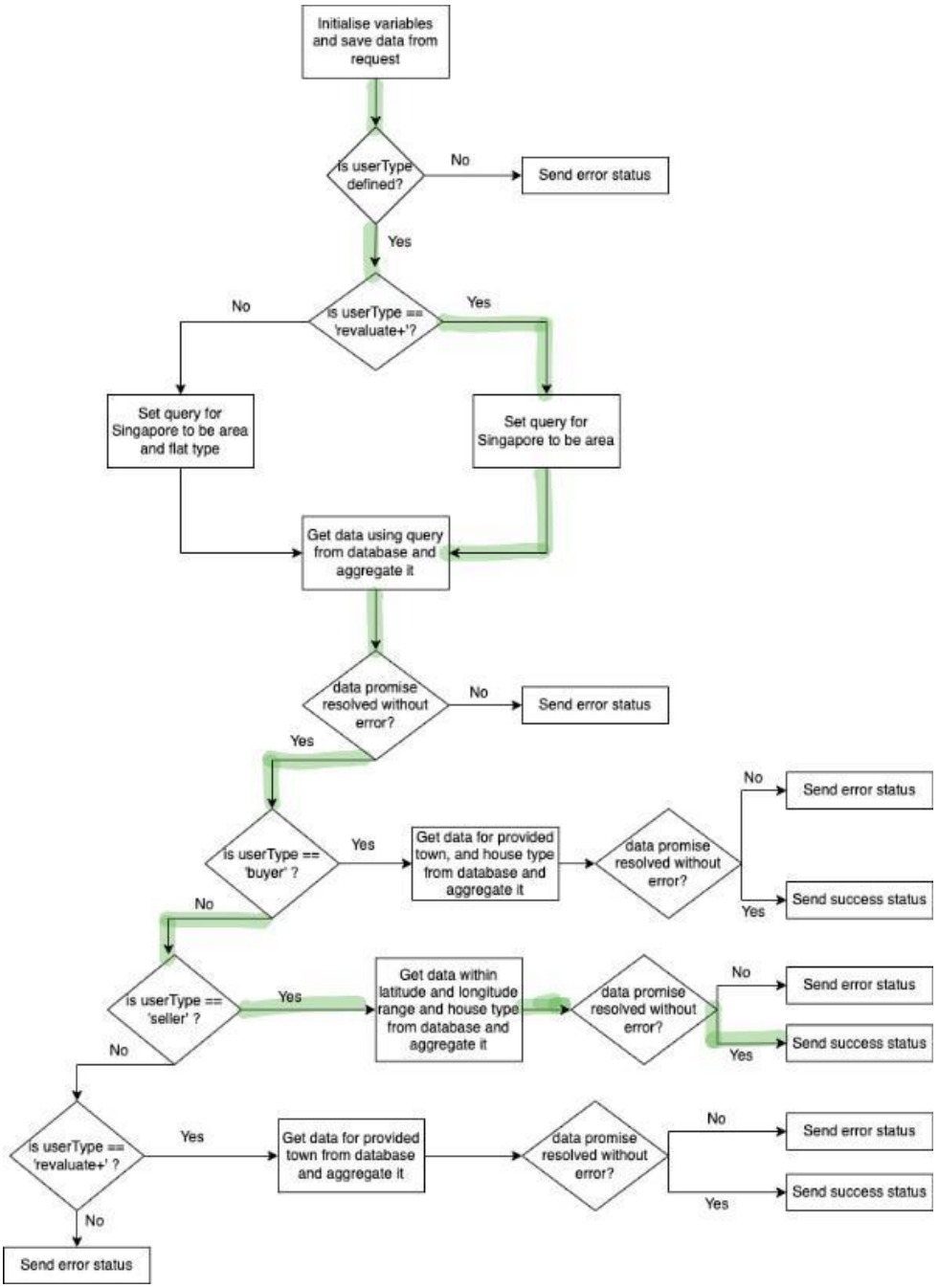


Unreachable paths:

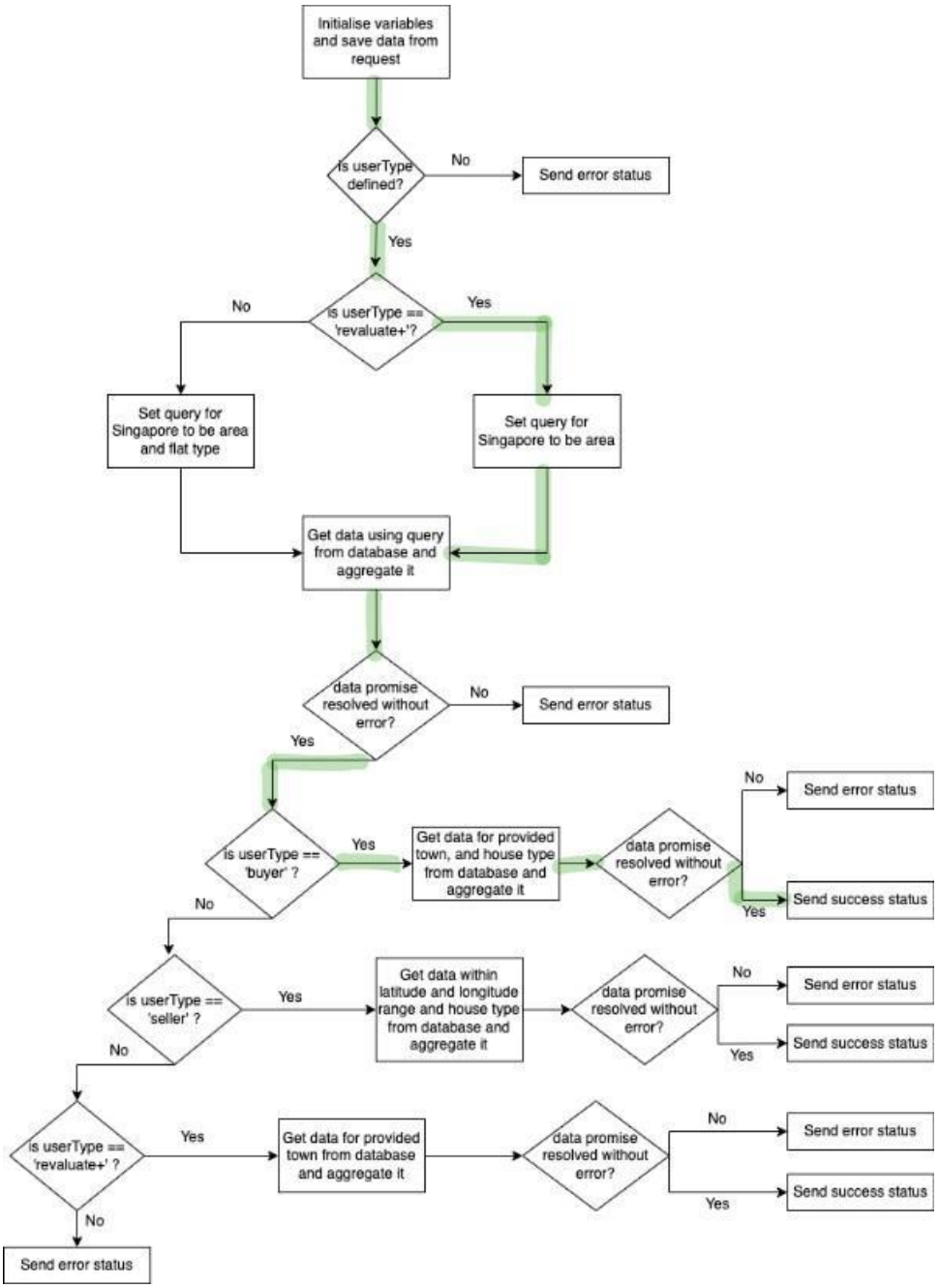
User type cannot be both not ‘revaluate+’ and ‘revaluate+’



User type cannot be both ‘revaluate+’ and ‘seller’



User type cannot be both ‘revaluate+’ and ‘buyer’



Appendix F: Design Patterns

Key Design Issues

1. Data Persistence

Introduction
In our design, we have identified three models to be stored in persistence storage: the Topics, Discussion and Transaction models. The Topics model contains the thematic topics the administrator wishes the forum to have. The Discussion model contains the forum posts and its associated comments. The Transaction model contains all information regarding past transactions that were extracted from open-source government data. The data is persistent as users are able to extract the data even after the webpage has been closed and restarted.

Design Issue
We have decided to use MongoDB Atlas. MongoDB is a document-oriented, noSQL database that stores data in documents similar to JSON. Due to the large data volume of our persistent data, MongoDB is suitable for handling our dataset due to its high scalability and fault-tolerant design. However, extracting and manipulating data from MongoDB Atlas can be complex as it requires knowledge of the database schema, it forms a design issue.

Design Pattern (Facade Pattern)
To abstract the complicated extraction from MongoDB, we have used the Facade design pattern to provide a layer of abstraction between the Views (client-side) and the database with our Controllers. This means that when the client makes an API call to retrieve data, the process is abstracted by a function within the Controller, reducing the need for the client to know the detailed logic behind the data extraction and manipulation.

These are the list of RESTful APIs developed and we see each API is handled by a controller function:

```
app.get("/", (req, res) => {
  console.log(req.query);
  res.send("Welcome to REvaluate Server");
});

app.get('/getTransactionsFromPin', MapController.getTransactionsFromPin)

app.get("/neighbourhoodAffordability", ChartController.getNeighbourhoodAffordability);

app.get("/flatStatsByType", StatsController.getFlatStatsByType);

app.get("/validateform", ValidationController.validateForm);

app.get("/generalPricingChart", ChartController.getGeneralPricingChartData);

app.get("/neighbourhoodPriceComparisonChart", ChartController.getNeighbourhoodPriceComparisonChart);

app.get("/details", DetailsController.getDetails);

app.get("/map", MapController.getMapData);

app.get("/neighbourhoodCountChart", ChartController.getNeighbourhoodCountChart);

app.get("/neighbourhoodStatistics", StatsController.getNeighbourhoodStatistics);

app.get("/roboadvisor", DetailsController.getAdvice);

app.get('/forumTopics', ForumController.getForumTopics);

app.get('/forumPosts', ForumController.getForumPosts);

app.post('/addpost', ForumController.addPost);

app.get('/getpostdata', ForumController.getPostDataById);

app.post('/addcomment', ForumController.addComment);

app.get('/correlation', StatsController.getCorrelation);
```

2. Access Control

Introduction

Our application uses Single-Sign-On service where users can log in using their accounts with various identity providers. This login service is handled by Firebase authentication which we pass the authenticator for each identity provider to.

Design Issue and Pattern (Facade Pattern)

This forms an implementation of the Facade pattern as it forms an abstraction from the Views (client) on the logic to authenticate using various identity providers.

```
//function that performs google SSO login using google identity provider
function googleLogin() {
    signInWithPopup(auth, googleProvider)
    .then((response) => console.log(response))
    .catch((err) => console.log(err))
}

//function that performs facebook SSO login using facebook identity provider
function facebookLogin() {
    signInWithPopup(auth, facebookProvider)
    .then((response) => console.log(response))
    .catch((err) => console.log(err))
}
```

3. Search Results

Introduction

As we have different types of users: buyers and sellers, the search results for each will be different.

Design Issue

We want to get the search results based on the user input type and this will only be known at run time.

Design Pattern (Strategy Pattern)

To return different search results at run time, we adopted the Strategy pattern where we have a strategy for buyer and seller each. We can then choose which strategy to adopt depending on the user input type. The code snippet is seen below:

```
async function getDetails(req, res) {
    const userType = req.query.type;
    const strategies = {
      'buyer': DetailsBuyerStrategy,
      'seller': DetailsSellerStrategy
    }
    strategies[userType].getDetails(req, res);
};
```

4. Charts

Introduction

As our application involves data visualisation using charts, we imported an external library Chart.js which offers different types of chart templates.

Design Issue

As we have a lot of charts, we do not want to have to import multiple charts for each file as this will reduce code readability and offer less flexibility.

Design Pattern (Factory Pattern)

We adopted the Factory pattern by creating a Chart component. The component will take in a type parameter and create a chart of that type. Hence, for every file where we need to have charts, we only need one import of the Chart component instead of importing multiple charts.

```
function Chart(props) {
    //data passed in from parent component
    const { type, data, title, labels, hideLegends, width, height } = props;

    //standardised some options for all charts used in the app
    const options = {
        responsive: true,
        plugins: {
            legend: {display: !hideLegends},
            title: {
                display: true,
                text: title,
                font: {
                    fontFamily: "Open Sans",
                    size: "20%",
                }
            }
        },
        maintainAspectRatio: false,
    };

    const chartData = {
        labels,
        datasets: data,
    };

    return <div className="chart" style={{width: width, height: height}} >
        {type === "bar" && <Bar options={options} data={chartData} />}
        {type === "line" && <Line options={options} data={chartData} />}
    </div>
}

export default Chart;
```

5. Model-View Interactions

Introduction

Whenever our model has an update in information, our views will need to update so it will display the latest information.

Design Issue

This requires the views to listen for changes and updates of the information. Hence, we need a way for the views to be automatically updated whenever there are changes to the information.

Design Pattern (Observer Pattern)

We can adopt the Observer pattern for this. As we use React as our frontend library, we can use one of its hooks, useEffect. The useEffect hook takes in a function and a list of dependencies and will call the function whenever any of the dependencies in the dependencies list changes. We have adopted the use of useEffect extensively and have included some code snippets below:

```
useEffect(() => {
    getMapData();
}, [minPrice, maxPrice]);
```

```
useEffect(() => {
    getPricing(type);
}, [flat_type, latitude, longitude, navigate, town, type]);
```

```
//filter the posts to get posts that matches the selected topic whenever the topic is changed
useEffect(() => {
    filterByTopics();
}, [selectedTopic]);
```