Coursework 4: London Property Marketplace

Nina Peire Bhavik Gilbert , Lavish Kamal Kumar ,
Heman Seegolam
Word Count: 2071 words

1 Application GUI Description

The property viewer application displays 4 distinct panels which each give insight into the London Property Marketplac Firstly, the Welcome panel introduces and welcomes the user to the application. To access the other panels, the user nee to select their preferred price range using the rangeslider on the right, and by clicking the 'Search Properties' button. Secondly, the Map allows the user to select a borough and view its available properties. This displays the property viewer window which displays each property. Thirdly, the statistics panel gives numerical value to information related to the properties. Fourthly, the favourites panel displays the properties which have been favourited by the user. The user can favourite a property by clicking the heart button next to the property. Lastly, our GUI has various additional features such as a menubar, an overall price calculator, a property selection scroll pane, a button to display the location of each property on OpenStreetMaps, a search bar for distinct values of each property, as well as a darkmode-lightmode implementation, and a Weather API.

2 Application Functionalities

2.1 Base Tasks

– **Application window**

The application window, as displayed in figure 1 in the Appendix, contains a menubar, panel selection buttons, an information pane to vary the displays, and a `centerPane` which changes depending on which panel is displayed. We implemented both arrow buttons (as seen on the middle/bottom right), and panel selection buttons, as we believe the latter are more commonly found in similar applications. Despite this, we still implemented the arrow buttons as the user may prefer using these and they demonstrates certain skills required in this assignment. In addition to the previous and next panel arrow buttons, the `informationPane`, also displays the user's name, a rangeSlider, the currently selected price range, search and reset buttons. Within the informationPane, AnchorPanes have been used to align the elements, this allows for the elements to remain structured in line with our design plan even after the window has been resized. Furthermore, we evaluated both using drop-down boxes and using a rangeSlider, and came to the conclusion that a rangeSlider would make the application more user-friendly. Through research, we also noted that a slider is more commonly found in applications and websites when a price-range is required. Hence, we opted for the rangeSlider over drop-down boxes. In order to decrease the number of displayed panels to those within the selected pricerange, we made use of lambdas to create a stream of the propertyList, filter the propertyList to those within the selected pricerange, and collect them back into a list called filteredPropertyList. This `filterProperties()` method can be seen in Listing 1 in the Appendix. Finally, the centerPane uses inheritance to display each distinct panel. The centerpane is thus capable of displaying the Welcome panel, Map panel, Statistics panel, and Favourites panel.

– **Panel 1: Welcome**

The welcome panel introduces the user to the London Property Marketplace application. We introduced a String variable `username`, which displays the user's name. This is achieved by assigning `System.getProperty("user.name")` to the `username` variable, allowing for the application to appear more personalised and user-friendly. If the user has not yet selected a price range, "Please select a price range to continue." is shown, else, the chosen prixe range is displayed.

– **Panel 2: The Map**

The Map panel consists of a collection of buttons, styled into the shapes of the London borough by SVG Paths. Radio buttons allow for selected information to be displayed on the map. This includes the: number of properties per borough, the current weather (through an API), the amount of crime, and the number of pubs per borough. Each arrangement is logically carried out by the `BoroughRank` class and the colour scheme is handled by the `ColourScheme` class. This works by returning an array of `Colour` objects that carry a hex value to be applied to the associated borough buttons. At the bottom of the pane there is a gradient keycode with statistics on either end to denote the values carried by the different shades of the colour. Additionally, the map makes significant use of enums to determine the actions following a Radiobutton click which result in a different colourcode/colourscheme to be displayed in the map. The RankingType enum for instance, shows the different types of sorting systems that can be applied to the Boroughs. In turn this is

then connected on the map via the ColourScheme enum, which holds a reference to rgb values and a change factor (to show a gradient).

- **Panel 3: Statistics** (additional Statistics Description)

  The user can redirect themselves to the statistics panel. This panel gives the user an overview on a various number of statistics. When the user accesses this panel, they see four distinct boxes, each of which contains a series of statistics. All of the statistics displayed within these boxes are relevant and calculated only as per the properties that are included in the price range that the user has searched for. There are a total of ten statistics, four of which are displayed at a time. No one statistic can be seen multiple times over the different statistic boxes. The user can press the buttons on the left and right of each statistic box to change a statistic. When statistics appear on screen and are taken off screen, they change between two lists that represent whether a statistic is being displayed or not and thus are prevented from appearing twice accordingly. The statistics offered are as follows: average number of reviews per property, total number of available properties, the number of entire home and apartments, the most expensive borough, the number of favourite properties, the average price per night, the number of pubs in the most common borough, the most common borough, the number of crimes per year in the most common borough, the percentage of people who walk three times a week compared to the London average in the most common borough, and the percentage of people who cycle three times a week compared to the London average in the most common borough.

- **Unit Testing**

  Unit testing is performed on the functions class, which is comprised of a number of anomalous functions which are required across multiple classes but but are particularly essential for the functionality of the system as a whole. Examples of this include searching and sorting a given list based on predefined public enums, alongside creating a hard copy of lists and looping through a list while incrementing and decrementing. The unit tests are comprised of assertions including: `assertSame`, `assertArrayEquals`, `assertFalse`, `assertTrue` and `assertNotNull`. The tests consist of common use cases alongside edge cases to ensure the functionality of the class works as intended in all situations. As these methods are used throughout the program, it is paramount that the code performs it's expected function in all available scenarios.

2.2 Challenge Tasks

- **Panel 4: Surprise us! Favourites Panel**

  For the fourth panel, we implemented a favourites panel. This allows for the user to select their favourite properties and review such in this distinct panel. This is achieved by a favourites button which sets the property's `isFavourite` boolean variable to true. The favourites panel is similar in GUI structure to the propertyviewer. Properties may also be unfavourited by clicking another time on the favourites button. The `informationPane` is purposefully removed from the Favourites panel as there is no use in selecting a price range in this Favourites Panel because the properties have already been selected based on the user's preferences.

To challenge ourselves considerably, we also implemented the following:

- **Searchbar**

  A search bar has been implemented in the property viewer and favourites panel, which can be used to search the property list fed into it via: (property) `id`, `host_id`, `host_name`, `minimumNights` and `room_type`. It does this by filtering original property into a secondary list, keeping the original list for reference. Throughout the code, the secondary list is used to display the current sorted/searched properties, and as such can be freely manipulated alongside resetting it to the same as the original list as needed.

- **Price Calculator**

  The price calculator consists of + and - buttons, which increment or decrement the user's desired duration of stay. If this is less than the host's given minimum stay, an error message will be displayed. Else, when the calculate button is clicked, the total price is displayed. This value is calculated by multiplying the user's desired duration of stay, by the given price per night.

- **Property Location Display in WebView**

  Users can view the location of their property within the application, given they are connected to the internet. This is done using WebView and the OpenSteetMap website. By passing through the latitude and longitude into the link used to generate the map, the location can be loaded, with additions including a marker on that current location, alongside input buttons on the side outside of WebView that allow the user to get a visual representation of cycle roads and bus stops in the area.

- **Darkmode Lightmode**

  A dark and light mode has been applied throughout the whole application. This is making use of css variables and swapping the css files used for colour to redefine and reset the colours used in the application.

– **WeatherAPI**

  The current weather is retrieved using the `WeatherAPI` class. If there is a stable internet connection, the weather API will retrieve values for the current temperature for each borough. This is then displayed on the map, whereby the darker the colour, the hotter the temperature. If there is not an active internet connection, the colours used will be based on the property density.

3 Code Quality Considerations

3.1 Coupling

The application has been designed with loose coupling in mind, with all the panels being almost completely unrelated to the the implementation of the application they're used in. Each panel is in their own classes with minimal input required for its functionality to be complete, thereby enabling large changes in implementation with minimal updates required. The existence of an external functions class for anomalous functions allows for methods to be completely unrelated to the implementation of the given functions. Changes in data between classes are kept to a minimal, with changes being accessed only doing so when required. Examples of this include: on panel change and updates in the user input data.

3.2 Cohesion

Cohesion has been considered in many places throughout this project. From the ImageReader class to the FileReader class to the separated panel classes, each class made has it's own distinct function within the program and aims to perform that specific function. The best example of this would be the map implementation, being made up of 4 different classes, being: MapController, MapPanel, Boroughs and PropertyDensity, which together are used to discern the colour set for each borough and sending through a filtered list to the PropertyViewer. Each class has a single set function which it is set and perform. Within classes, methods are split up best showcased by the uses of the Functions class, where each method performs exactly 1 function, with these methods being paired up with other methods in other classes to perform the required set of instructions.

3.3 Responsibility-driven Design

Ensuring all methods and data are implemented in the appropriate classes results in responsibility-driven design. The Map panel required thorough attention to not deviate from responsibility-driven design, as it has various aspects such as the individual boroughs, the colour and respective colourschemes, as well as a map controller, BoroughRank and RankingType, which each contribute to the map panel. By using the Model View Controller design pattern, the code for the Map panel is separated out into controller and driver logic, hence permitting encapsulation and adding to the maintainability.

3.4 Maintainability

We heavily focused on maintainability by defining clear method and class names, as well as continuously keeping in mind that it should be possible for the application to be extended without creating significant bugs. Firstly, the method names used to create the main components of the Application Window clearly align: `generateMenuBar()`, `generateInformationPane()`, `generateMainPane()`. This again applies to the `informationPane` components, which have the following method names: `generateUserPane()`, `generatePriceSlider()`, `generateButtonPane()`. A consistent use of words such as "generate", "set", "get", and "display" add consistency to the source code and thus improves maintainability. Secondly, the arrow buttons to navigate between panels and those for navigating between properties, are defined to consider the `list.size()-1` as the final element in the list. Compared to considering element 3 in the list of panels, considering the final element based on size improves maintainability as it allows for more panels to be added to the list without requiring the button roll-over methods to be changed too.

4 Bugs or Problems

– The favourites panel button needs to be clicked again after favouriting a property such that a refresh happens, resulting in the panel displaying a list of favourited properties.

– Starting the applicatino takes a while due to the api

– Text does not wrap in statistics on windows, but it does for mac.

A Appendix

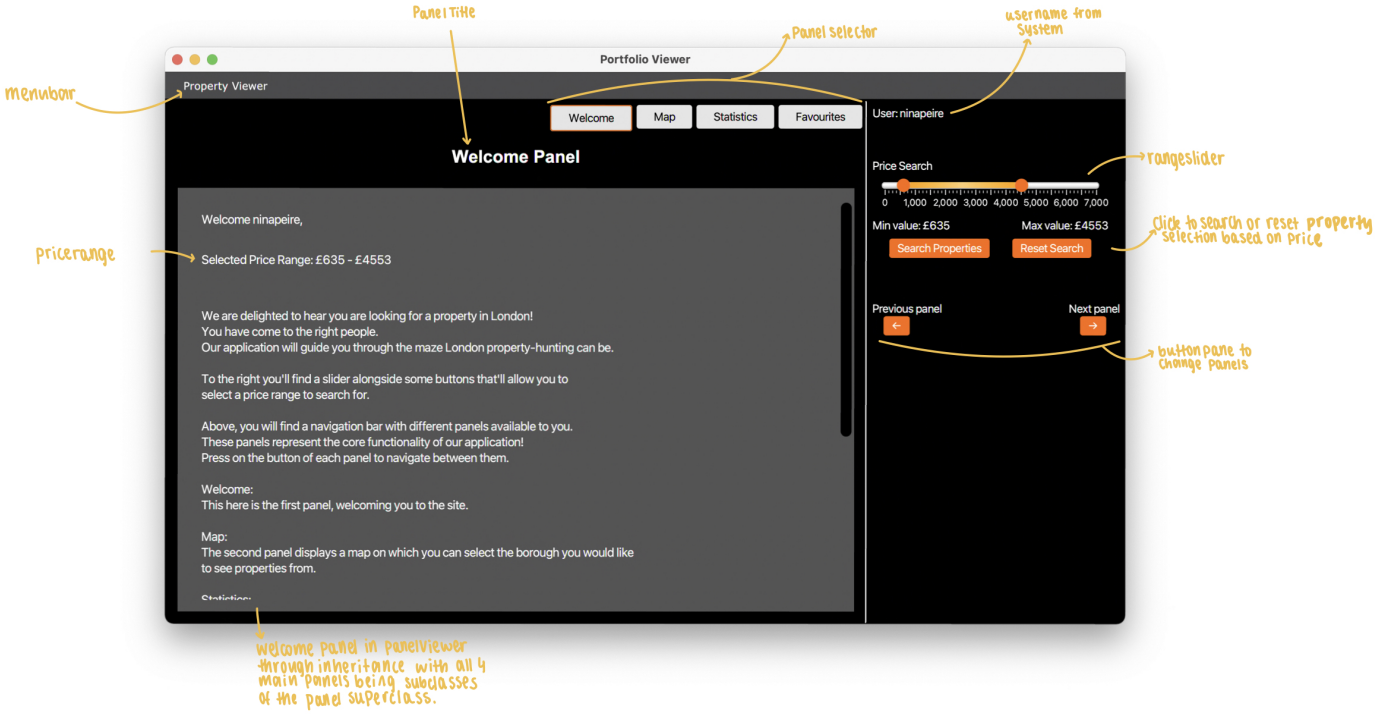

Figure 1: Annotated Application Window GUI

```java
/**
  * Retrieves a list of properties via search.
  *
  * @return The list of properties within the search range.
  */
private void filterProperties()
{
    List<AirbnbListing> priceListings = propertyList
        .stream()
        .filter(property -> property.getPrice() >= (int)priceSlider.getLowValue() &&
            property.getPrice() <= (int)priceSlider.getHighValue())
        .collect(Collectors.toList());

        filteredPropertyList = (ArrayList<AirbnbListing>) priceListings;
}
```

Listing 1: `filterProperties( ) method outline`

References

[1] Mhshams. (2010, August 31). Convert a RGB Color Value to a Hexadecimal String. Stack Overflow. Retrieved 29 March 2022, from https://stackoverflow.com/questions/3607858/convert-a-rgb-color-value-to-a-hexadecimal-string/36079423607942

[2] Alex. (2012, August 30). Node.getTextContent() is there a way to get text content of the current node, not the descendant's text. Stack Overflow. Retrieved 29 March 2022, from https://stackoverflow.com/questions/12191414/node-gettextcontent-is-there-a-way-to-get-text-content-of-the-current-node-no/12192072

[3] SVGATOR. (n.d.). SVGator: Free SVG Animation Creator Online - No Coding. Retrieved 20 March 2022, from https://www.svgator.com/

[4] © OpenStreetMap contributors. (n.d.). OpenStreetMap. Https://Www.Openstreetmap.Org/. Retrieved 24 March 2022, from https://www.openstreetmap.org/