Outline of Design

Very early on, it was decided by the team that we would implement as much as possible from first principles. Our main focus from the very start was to learn from this project and we felt that implementing features ourselves was the best way to accomplish this. The task we had at hand can be broken down into:

- Reading input from the file.
- Building the database (and identifying if a database is already present).
- Retrieving information.
- Graphical User Interface system
- More complex GUI features.

Work Distribution

For the first few weeks work was distributed into two pairs: graphical user interface and database work. Tony and Matthew focused on making the original GUI that allowed us to test the database and were the only reason we had something to demonstrate every week. Alex and Ebin worked on the database side of things. Alex made the csv file parser and the database builder, while Ebin linked the GUI to the database by providing easy to use functions to retrieve data.

The next two weeks of the project had a very different work distribution. Database building was no longer a big focus and so Alex moved on to develop a new GUI system on which the final project was built on. Ebin stayed updating the queries and making them faster (as this actually affected the program) and worked with APIs and external libraries to develop the map that plays a big part of the project. Tony and Matthew were tasked with porting over the features from the old system to the new one while also adding new features that the new system allowed them to.

The final week was more of polishing up, bugfixing and designing the final GUI. When a problem arose, the person in charge of the feature that originated it was asked to look into it. Ebin took charge of displaying general statistics on the screen, Matthew and Tony worked on making the graphs more readable (speeding up animations, color schemes, etc.) and Alex designed the final user query screen.

The Program

Reading Input

Sticking to first principles, the different fields of the csv file were all read and parsed without the use of an external library. This was accomplished through analysing the CSV fields and using regular expressions to retrieve the fields from

each line. This part of the program also filtered out unwanted fields such as the unique key identifier and some others from the full data set.

Problems Encountered

- When the dataset was switched to a custom subset of the original data, the format had changed drastically. There were more fields that weren't included before (which had to be filtered), all fields were enclosed in quotation marks and it followed a more consistent format.

Building the Database

Another decision that was made really early was the use of a database to store the data as we were aiming to use a large amount. Alex and Ebin learned SQL as a result of this as they were the two who would be working on the database side of things. It was agreed that the SQLite would be the implementation used as it was easy to set up and was serverless.

The database was built from the filtered data as discussed in the previous section. Additionally to this data, there were three extra columns in the database: ID (a unique number for each property), latitude and longitude (obtained through the Postcode.io API). The last two columns proved invaluable to our project as it unlocked a lot of potential with the map.

Other than that, there wasn't much focus on the database building. As it wasn't a thing that had to be made everytime the program was run (if there was an already built database, the program would just connect to that one). This is one of the things that could be improved immensely, as it's very slow to build a new database.

Problems Encountered

- The only problem encountered while building a database was related to bad reading of input, which resulted in some columns being shifted left.

Retrieving Information

The next step was to retrieve information from the database. This included a lot of methods to retrieve data in different formats, mainly lists of properties. Each graph that was implemented required different data as they displayed different information. Almost everytime a new feature was implemented, it required a different set of data. Ebin was required to do constant work in the queries to provide the program with all the information it needed.

Problems Encountered

- One of the major problems that was encountered in the entire project was the speed and inefficiency of our queries as they took a long time to load. After testing under different conditions, Ebin determined hat the reason that the

queries took so long was due to Java rather than SQL. To resolve this issue, functionality was shifted over to the SQL side of things.
- The queries class is a large file and has a lot of methods, bugs popped up occasionally when certain methods were changed or new features implemented, especially when Ebin was trying to make it more efficient.

The GUI System

Universal Dimensions

A universal dimension is a method of describing pixel dimensions. The classical dimensions of a 2D plane are x and y. However, a universal dimension has two components to it: scale and offset. The scale component is related to the parent object of a GUI object (discussed in the next section) and the offset is an immediate value (think the classical dimensions). So for example if a dimension had 0.5 as the scale and 100 as the offset, that dimension means it's half of the parent's dimension plus 100 pixels. As we're still working in a 2D plane, we still need two universal dimensions, therefore UDim2 (x scale, x offset, y scale and y offset).

Nested Objects

One of the features of the screen, is that it can hold GUI objects. This is not unique to a screen however, and every GUI object can hold other GUI objects. The object that holds other objects is referred to as the parent and the objects that it holds as its children. There is no limit to how many children an object can have and each child of an object can have subsequent children. In conjuction with the universal dimension system, a lot of flexibility is already given: GUI can be made to scale with the user's resolution.

Z-Levels

Z-levels determine what order things are drawn in, the higher the Z-level the sooner it gets drawn (an object with a z-level of 5 will get drawn before one with a z-level of 0, thus the z-level 5 object will be drawn behind the z-level 0). Building on the nested objects system, z-levels are unique to each parent and its children. A screen has a default number of 10 Z-levels, but each of its children will have 10 more Z-levels.

Clipping

Clipping is another feature of the system that works based off the nested object feature. What clipping does it it limits children of an object to be drawn within its borders only. This uses Processing's clip() function and it calculates all required regions and subregions. This is a very useful feature of the system, it made the implementation of scrolling frames, dropdown menus and other features trivial.

Tweening

Tweening is short for "in between'ing" or in other words, finding the position between two points. The sole purpose of it, is to make animations. It's really easy to make an object move from one point to another by just calling a method and not having to worry about how it does that.

Events

Events were an interesting implementation. Alex wasn't aware of Processing's event handler system and came up with events that took advantage of lambda expressions (a functional language feature of Java) to connect blocks of code to certain events. The Event class is very simple, it has a connect method which adds the lambda expression passed to a list. A disconnect method which removes a lambda expression from the list and a trigger method which executes all lambda expressions connected. This proved to be a very flexible system as it allowed events to be created with ease. Events were triggered within respective classes and did not need to be triggered outside (self contained, again).

More Complex GUI Objects

Input Components

Receiving input from the user involved in just implementing new GUI components using the system from the previous section. In the final project, the components that were used include: instant search bar (Advanced Dropdown, takes advantage of Tony's textboxes and Alex's dropdown/scrolling frame), double slider (An extended version of slider by Matthew), a general list, checkboxes and textboxes.

Graphs

Graphs were the main form of data visualisation in our program. Tony and Matthew worked on the graphs to allow the user to examine the house by different fields, such as average price per month, amount of sales, types of property sold, the condition of properties sold and to see correlations between graphs (amount of sales is proportional to price).

Map

Ebin made a map GUI using the UnfoldingMaps API that allowed properties to be represented by a marker in the map, its colour signifying the type of property it is. When a marker is pressed, an info window is opened that shows all the details about the property. Using the Google Street View API, an image of the property was displayed in this window.