Software Design Document

Software Technologies Part A

Lachlan Dietrich | Rokhan Khattak | Arsalan Hanif

Table of Contents

[1.0 System Vision 3](#_Toc46748622)

[1.1 Problem Background 3](#_Toc46748623)

[1.2 System Overview 3](#_Toc46748624)

[1.3 Potential Benefits 3](#_Toc46748625)

[2.0 Requirements 4](#_Toc46748626)

[2.1 User Requirements 4](#_Toc46748627)

[2.2 Software Requirements 4](#_Toc46748628)

[2.3 Use Cases 4](#_Toc46748629)

[3.0 System Components and Software Design 5](#_Toc46748630)

[3.1 System Components 5](#_Toc46748631)

[3.2 Software Design 5](#_Toc46748632)

[4.0 User Interface Design 6](#_Toc46748633)

# System Vision

## Problem Background

The project being addressed prescribes an analysis to be conducted on a selected dataset (Sydney Airbnb dataset), allowing both the functions for data visualisation and basic analysis to be demonstrated. Functions to integrate include data updates and selections, both being essential functions for dataset analysis at any complexity level.

Through python-based GUI, the solution is required to display data, formulate a graphical display of such data, complete basic data analysis on given fields and allow for selective searching for specific terms in the data. Each of these requirements are possible through python-based programming paired with sql database analysis tools.

## System Overview

The system prescribed requires the following features to be completed using the Sydney Airbnb dataset:

* reporting information of all listings in a specified suburb from a user-selected period,
* graphical production for the distribution of prices of properties from a user-selected period,
* retrieval of all records containing keywords for a user-selected period,
* analysis for how many customers commented on factors related to cleanliness.

along with one extra analysis tool to be later defined.

Basing the solution using similar preexisting tools such as Microsoft’s Excel, a basic understanding for the GUI and programming requirements can be made.

## Potential Benefits

Potential benefits for the solution include cost-effective analysis and hardware-light analysis. By developing a solution in python, the hardware strain on processing is significantly minimised without disregarding software functionalities. A similar positive outcome is likely for both the economic and time-related costs, as the solution can be developed without any large costs to the development budget and time period. These benefits all allow for the solution to possess a net positive outcome as long as the assigned development requirements and period are closely kept. The benefit for a simplified data analysis tool is also possible, leading the solution into a untapped market for basic analysis tools over the current market taken by solutions possessing numerous functionalities bloating the software and therefore hardware strain.

# Requirements

## User Requirements

Our client target for this usage of this app would be real-estate agents.

User Requirements:

1. Interface:
   1. Users should find the application interface intuitive and easy to navigate.
   2. Bright default colour scheme, clear buttons and menus should easily guide the users through the app.
2. All access to Data:
   1. Users need to access the dataset that includes the property-related information.
3. Visual Data:
   1. Users should be able to visualise property trends, patterns and etc through charts and graphs from the datasets.
4. Input and Keyword Search:
   1. Users should be able to have the option to enter the time periods and suburbs, to shorten down the data analysis.
   2. The app should provide a search option for the users that shows information from the dataset.
5. Comparison:
   1. Users should be able to compare prices of different properties at specific periods from the graph.
6. Testing and Reliability:
   1. Users will expect the app to be free issues and run smoothly.
   2. Thorough testing should be conducted to ensure the app's functionality and accuracy.
7. Compatibility:
   1. The application can be used on both apple and android operating systems.

## Software Requirements

1. Input and Keyword Search:
   1. Users shall be able to input specific time periods and suburbs to refine data analysis.
   2. The app shall include a search feature that allows users to retrieve information from the dataset based on keywords.
2. Data Breakdown:
   1. The app shall provide visualizations to display the distribution of location, price etc.
3. User Interaction:
   1. The app shall enable users to execute keyword searches in the data.
4. Importation:
   1. The program shall allow the importation of data sets in various formats.
5. Analysis:
   1. It shall visualize all the pricing displays through graphs.

## Use Cases & Use Case Diagrams

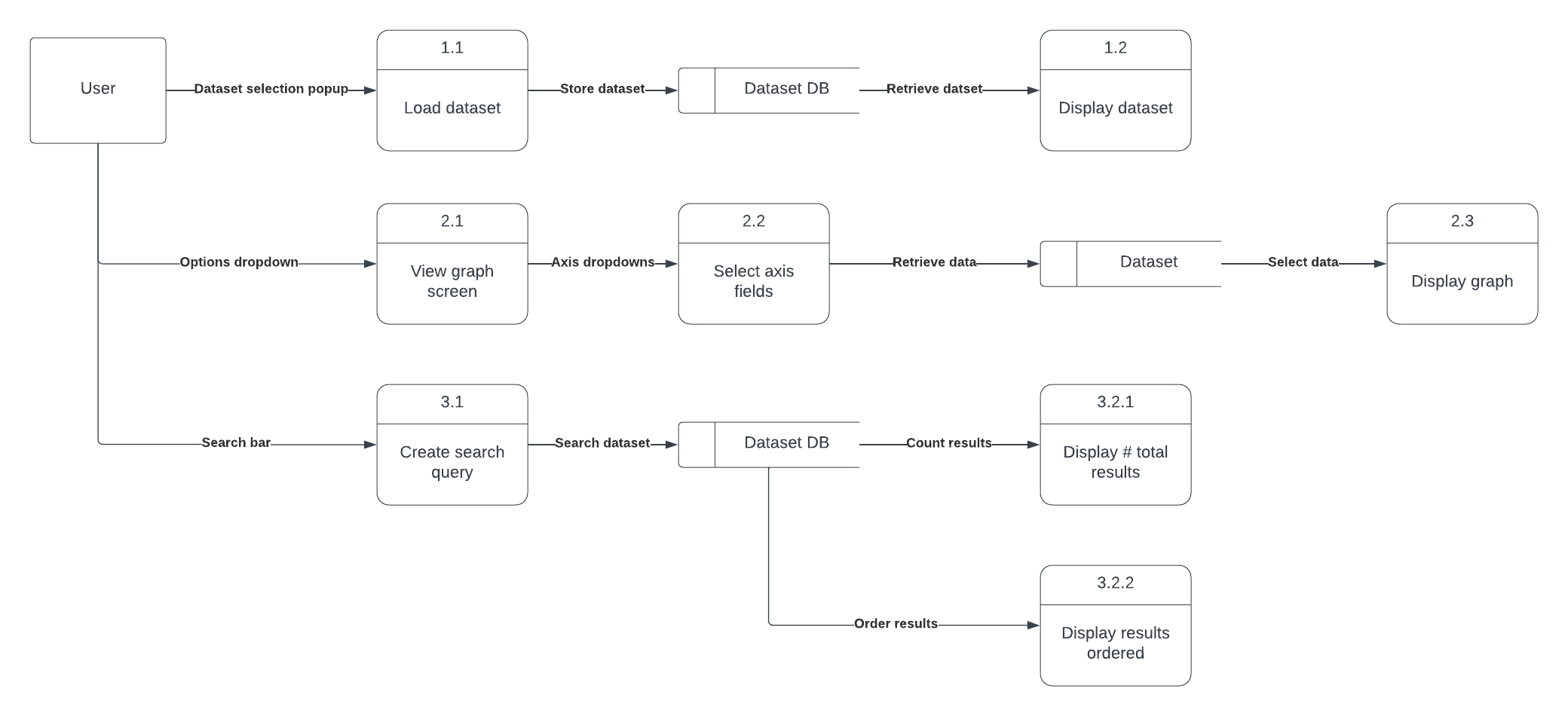
1. Visualising Property Data:
   1. Actor: The Real Estate Agent
   2. Description: Accessing property data.
   3. Event: The user selects the "Display Graph" option from the menu bar located at the top of the screen. The app provides access to the data with the graphical representation.
2. Home Screen:
   1. Actor: The Real Estate Agent
   2. Description: Accessing and opening a dataset to view.
   3. Event: The user opens the app:
      1. The user clicks on the "Select Database" button to select a dataset file.
      2. It takes .xml, .csv and .json.
      3. Once the files open the user will be taken to the next page.
3. Data:
   1. Actor: The Real Estate Agent
   2. Description: Reading the dataset after opening it.
   3. Event: After the data has been opened a dropdown of the whole dataset is present
4. Keyword Search:
   1. Actor: The Real Estate Agent
   2. Description: Searching through the comments using the search button.
   3. Event: Top right there will be a “Search “ button where the user can do keyword searches through the dataset.
5. Closing of the software:
   1. Actor: Real Estate Agent
   2. Description: close button closing the whole software
   3. Event: The user will press the “close “ button and it will close the whole software.

Three Use Case Diagrams:

|  |  |  |
| --- | --- | --- |
| A diagram of a data set  Description automatically generated | A diagram of a data processing process  Description automatically generated | A black and white rectangular sign with black text  Description automatically generated |

# Software Design and System Components

## Software Design



## System Components

### Functions

The following functions will be provided by the software:

* Searching the dataset for keywords, ordering the data accordingly and displaying the total result per search.
* Using 2 fields from the dataset, displaying select data as a viewable graph.
* Searching the dataset using search parameters such as selected fields and periods.

Search function: When searching the dataset for keywords, the software uses the user’s query as a string variable, and searches the given dataset for matching strings, counting the total matches as a results variable. Once the results are collated, the display screen is ordered ascending by the related results and the results variable is displayed. Depending on the character size of the largest datatype, the search function will be assigned the same limit to account for all possible inputs without causing extra bloat by allowing too many characters. The setback for this function however is the possible hardware strain it may cause due to searching the entire dataset, hence care should be taken to minimise the strain, and an efficient coding structure should be prioritised utilising coding functions such as arrays to store the result ids to call for their matching data.

Field searching: The user is prompted with all the fields given in the dataset to select 2 for searching. The software uses the selected fields are variables and conducts a similar process to the search function, instead using the 2 fields to find appropriate matches. Similar parameters will be applied, however the strain is likely lower, as the function is only searching for matching fields instead of datatypes. Storing the results in an array again will help minimise this strain.

Graphing function: Using the dropdown option to open the graphing screen, the user is prompted to select a field from the dataset for each axis to then use for graphing. The software takes the selected fields, and using imported graph capabilities, assigns the data from the fields to arrays for each axis, then plots the data and outputs to the user. The title given to the graph is synthesised during this process by filling in the blank name ‘<x-axis> vs <y-axis>’ with the respective axis headings. To minimise the software strain, the imports will only be called when the user is in the graphing screen.

### Data Structures / Data Sources

The following are the data sources in the software:

* User-defined dataset *(Database)*
  + On the software’s initialisation, the user is prompted to select a dataset by the file convention .xml, .csv, or .json and this dataset is saved as the database the software will analyse. The file must include at least one row and field of valid data to allow the software to display and must include two fields to allow for the graphical display.
* User-defined query *(String)*
  + If the user prompts the software with any search query in the search bar, a user-defined query is created as a string variable. This variable is used to query the database and find any matches to push to the UI. This is not predetermined and will only be synthesised if the user chooses.
* User-defined axes *(Array)*
  + In the graph display screen, the user is provided with two axes dropdowns to formulate a graph by. The dropdowns are determined by assigning the fields in the database to two arrays per axes, and the user selects the appropriate field they wish to place on either axis. This is done by matching the user input to the relative array index and then pushing that array location to the graph display function.
* User-defined parameters *(Array)*
  + Similar to the ‘User-defined axes’, the user may provide parameters to search the database by. These parameters are restricted to the fields offered by the database, being assigned to an array similar to the axes. The software uses the user-selected field/s in the array to find any relative data.

The following are the UI features in the software:

* Menu Screens
  + Menu screens contain all the GUI elements per menu option such as the display and graph options. This is contained in forms.
* Dropdown menus
  + Dropdown menus contain the menu screens inside a menu array and is called only when the user chooses to change screens. As the screens are stored in an array, this allows the drop to be expandable for future additions.
* Buttons
  + Buttons are used primarily on the title screen, used for closing the program and opening a file popup. A button is also used to initial python to create the user-selected graph in the respective screen.
* Search bar
  + The search bar is a textbox that saves the user input as a string variable to query the database by. The character limit on the box is set to the largest item in the database.
* Parameter box
  + The parameter box uses an array made from the fields in the database. The user may select a field to limit searches to, and the software assigns that array index to search only that field in the database.
* Python graph function
  + Using python’s graph import, the user can opt two fields for the axes and python will use the database data for each field to first formulate the graph, then on the user’s command, display the graph. This two-step process should minimise errors encountered in displaying the graph.

### Detailed Design

**User interacting with the menu screen:**

START

SET database\_select\_button = visible

SET close\_button = visible

SET title = visible

IF user presses close\_button THEN

Close software

ELSE IF user presses database\_select\_button THEN

Open file popup

IF file is selected THEN

Database = selected database

Open display\_screen\_form

END IF

END IF

END

**User changes from display\_screen to graph\_screen:**

START

IF user opens menu\_dropdown THEN

SHOW menu\_screens arrays

IF user selects graph\_screen THEN

Open graph\_screen

END IF

ELSE

Close menu\_dropdown

END ELSE

END IF

END

# User Interface Design

To synthesis the user interface (UI) for each process prescribed, I researched into current data analysis tools such as Microsoft’s Excel and selected the key design components to transfer into my own design. The components I selected include the top navigation bar, the cell-based data display and the primary graph display functions.

Once selecting the components, I used a basic window template to imitate the final solution, then using paint.net, I laid out basic components using box shapes similar to what would be the likely design for the solution. Due to the limitations of python graphic user interfaces (GUI), I opted to use only boxes in the final UI design, as this greatly simplifies the software generation process. I also kept colours and major graphical features minimal to further this philosophy.

To integrate the search function’s design, I used Google’s result’s output (‘Total results for ….’) as reference to ensure the user experience (UX) from using Google could be transferred to using this software. I selectively used dropdown menus for particular functions that could be expanded, as this ensures the software’s longevity if further features were later implemented.

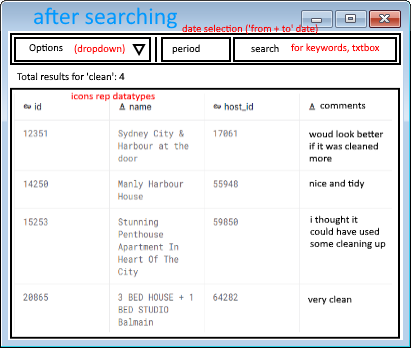
## Structural Design

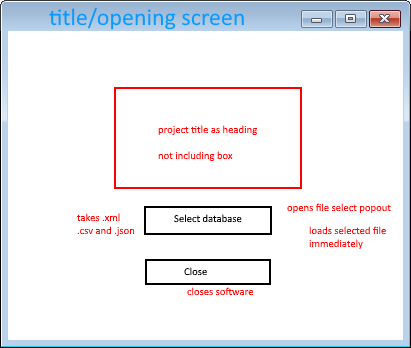
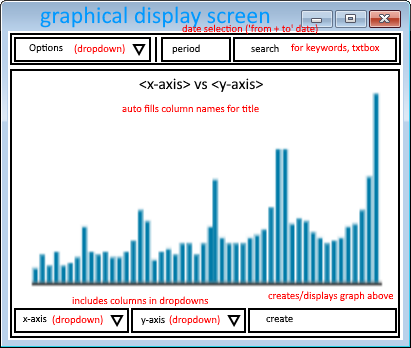
The above hierarchy demonstrates the structure for the solution. The solution depends on the foundation of the backend code, supplying the data analysis tools and processing tools, closely working with the database itself. From the database, the frontend functions including the UI and GUI are conducted and outputted to the user. This basic hierarchy provides the process that proceeds whenever a function is called by the user.

This flow was chosen due to its simplicity and data security, minimizing the possibility of breaches or data leaks by keeping the database in close relation to the backend code. Furthermore, separating the backend from the frontend simplifies error handling, keeping the output/analysis portions separate for diagnosis safely and efficiency.

This standard hierarchy should keep the integrity of the product high and allow for a structured solution that can be well defined for diagnostic purposes or for future additions to be synthesised.

## Visual Design

A screenshot of a computer

Description automatically generated

The above designs utilise a basic box style format to primarily simplify the development process and to prevent overwhelming the user for initial impressions. The solution uses no major colours in the design to further simplify the solution and to avoid alienating users with colour impairments that may impact UX.

A minimal variation of screens was opted to not only further minimise the development period, but more importantly unify the UI under an almost single screen design, enhancing UX. For this reason, the heading involving the options dropdown, search bar and period box was kept throughout all screens to ensure UI cohesion. The basic design for UI should ensure hardware strain is as low as possible without impacting the software requirements.