Software Design Document

<Project Name>

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

In this section you detail how a user is supposed to interact with or use your program. What do they ***need*** to be able to do? This should all be from the end users perspective. Can be a combination of narrative text and listing of needs.

**Assignment note: You have not been given a client/user, so you can make one up. Who do you think would be using your software?**

## Software Requirements

In this section you detail what the requirements for the software are. What functionality will it provide? This is usually a formal listing, with requirements often using the word ‘Shall’. IE:

R1.1 The program shall accept multiple file names as arguments from the command line.

R1.2 Each file name can be a simple file name or include the full path of the file with one or more levels.

etc …

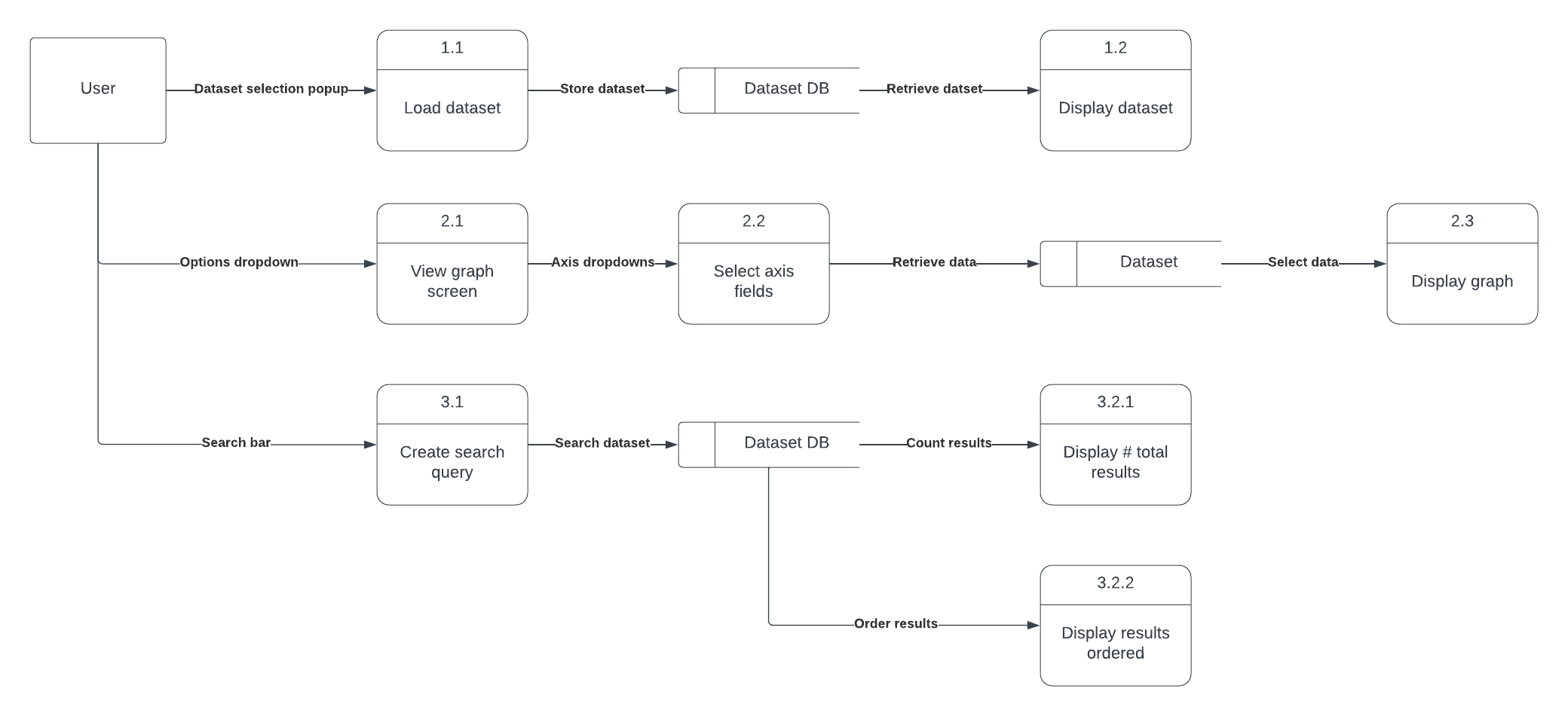
Can be primarily functional requirements, though you may include other types if you think of them.

## Use Cases & Use Case Diagrams

In this section you provide some use cases showing how people may use your software.

# Software Design and System Components

## Software Design



A block diagram/flowchart of how your software might work

## 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.

Preliminary list of all functions in the software. For each function in the list the following information is provided:

* a brief description of what it does (1 or 2 sentences);
* a list of the input parameters, and their data types, and what they are used for;
* a list of any side effects caused by the function (ie change global or member variables, changes data passed by reference from calling function etc)
* a description of the function’s return value

### Data Structures / Data Sources

List of all data structures in the software (eg linked lists, trees, arrays etc) or eternal data sources. For each data structure in the list the following information is provided:

* Type of structure (tree, list etc),
* Description of where and how it is used
* List of data members, and what each one is for do
* List of functions that use it

### Detailed Design

Pseudocode for all non-standard / non-trivial algorithms that operate on data structures

# 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

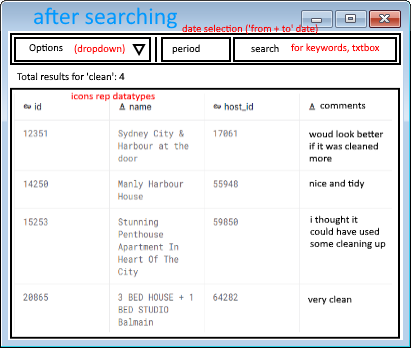
The above hierarchy demonstrates the structure for the solution. The solution depends on the foundation of the backend code, supplying the data analysis tools and processing tools, closely working with the database itself. From the database, the frontend functions including the UI and GUI are conducted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

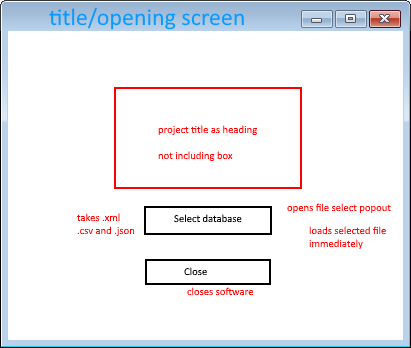
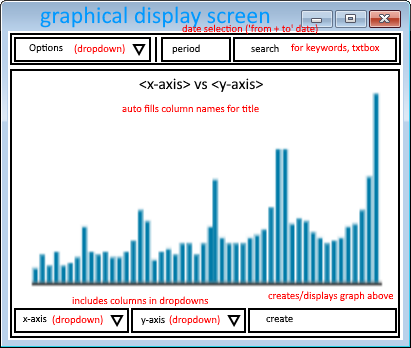
This flow was chosen due to its simplicity and data security, minimizing the possibility of breaches or data leaks by keeping the database in close relation to the backend code. Furthermore, separating the backend from the frontend simplifies error handling, keeping the output/analysis portions separate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

Structural design refers to the navigational and information structure of your product – the structure that supports the interface layout. How will you structure your product? How will you group your information? How will you navigate through your product? Why? This can take the form of a diagram showing structure and hierarchy, supported by a discussion and justification of your choices. Why have you made these design choices? Describe and outline the structure of your interface and of your information.

## 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.