Criterion B

Design

Contents

[Use Cases 3](#_Toc153903276)

[Flowchart Overview 4](#_Toc153903277)

[Versioning 5](#_Toc153903278)

[Prototype Design 6](#_Toc153903279)

[Final Design 10](#_Toc153903280)

[Data Types Used 13](#_Toc153903281)

[Unified Modeling Language (UML) Diagrams 16](#_Toc153903282)

[Hierarchal Chart 19](#_Toc153903283)

[Connection Chart 20](#_Toc153903284)

[Data Flow 21](#_Toc153903285)

[Flowcharts 23](#_Toc153903286)

[Testing Plan 32](#_Toc153903287)

# Use Cases

Based on the success criteria in Criterion A the versioning diagram below represents the client and how all their needs will be serviced by the various use cases of their program.

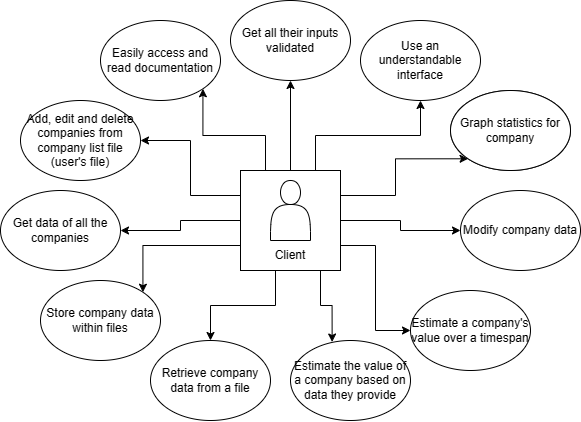


Figure 1 All use cases which the client should be able to do.

# Flowchart Overview

The main dashboard of the program was represented as a flowchart, linking all the processes that happen in the program.

A diagram of a flowchart

Description automatically generated

Flowchart 1 Main dashboard.

# 

# Prototype Design

After the initial meeting with the client, I created prototype designs of some of the main and initial screens that would be used in the program – getting an idea of what the look and feel would be like of the entire program.

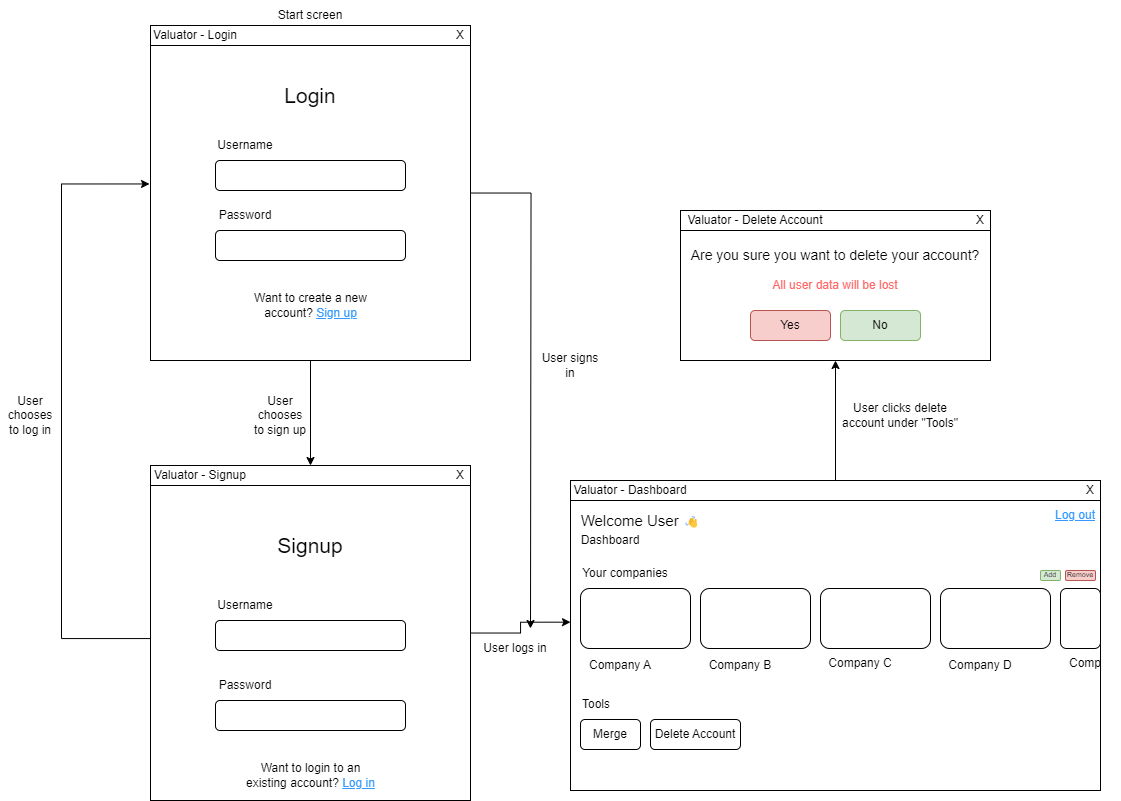


Figure 3 Login, signup, and user dashboard.

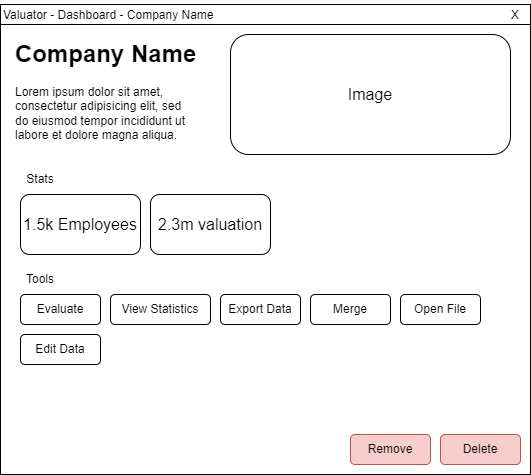


Figure 4 Company window.

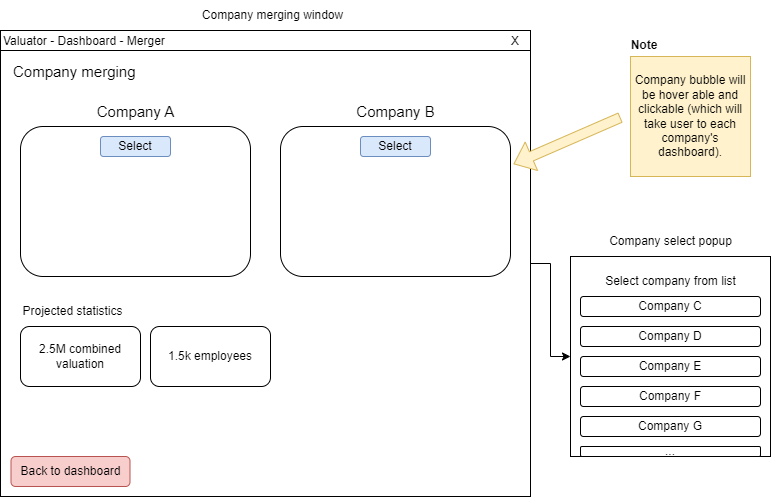


Figure 5 Company merging.

# Final Design

After meeting with the client and showing them the initial prototype design[[1]](#footnote-2), I built the final design concept based on their feedback. Some of their feedback was that the initial interface had redundant components, such as unnecessary information that could be hidden with a simple information button. Additionally, they requested I use the colour blue and create a banner on top of the windows—stating that they felt as if the colour made the program appear trustworthy, feedback I took into consideration when redesigning.

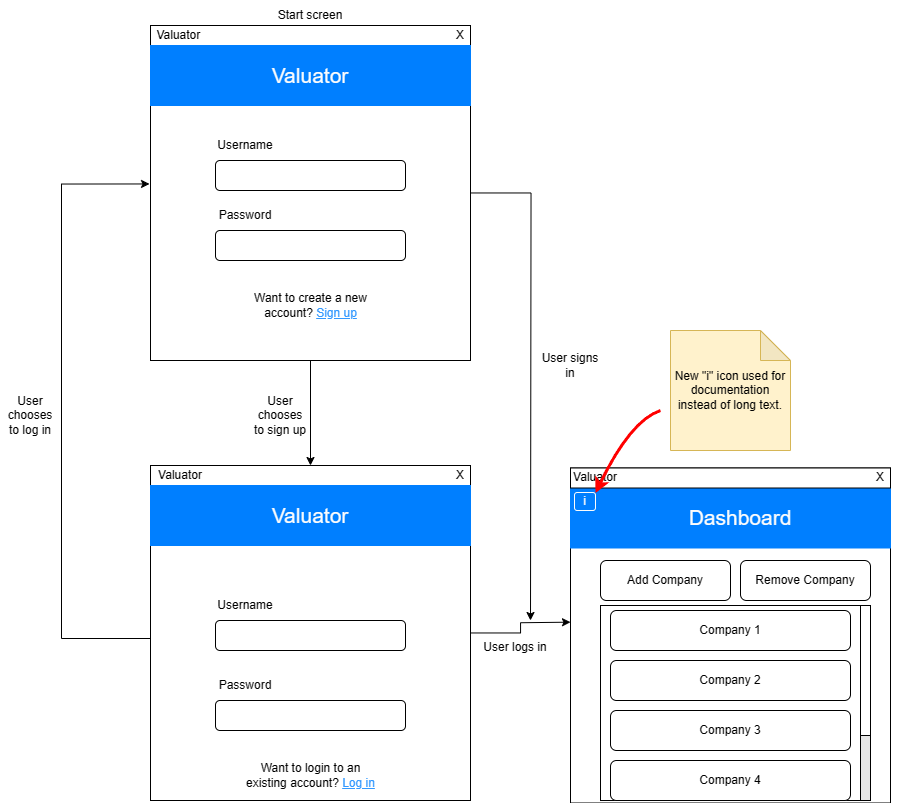


Figure 8 Login, signup, and user dashboard.

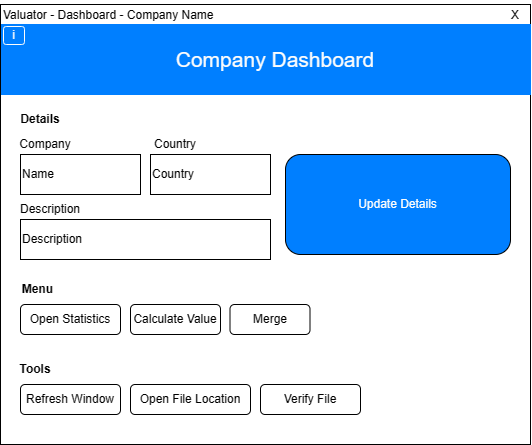


Figure 9 Company dashboard.

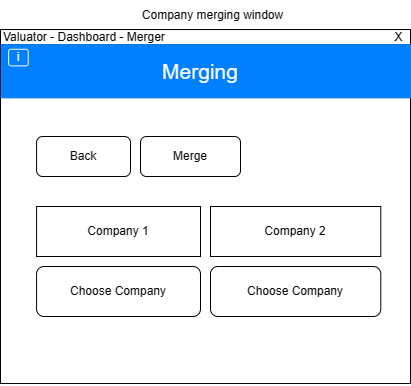


Figure 10 Merging interface.

# Data Types Used

A table holding various variables that are seen throughout the program, accompanied by the data type used to represent this and the reasoning behind why this was chosen.

|  |  |  |
| --- | --- | --- |
| **Data Type** | **Variable** | **Reasoning** |
| String | username | I used Strings for the usernames someone might choose when using the program as a name is a series of characters. |
| Password | I used a password string since simply using a numerical password would have allowed for brute forcing and would not have been very secure. |
| Boolean | isAdmin | Determining whether the user has administrator permissions or not is a true or false, there are only two conditions – therefore, I used a Boolean. |
| debug | Determining whether the program should be in state of debug is a yes or no situation. Since I am not using configurable debugging where there is a level to which debugging exists (more complicated or simpler) it suffices to simply use a boolean to indicate to the program if it should output debug messages. |
| Scanner | scanner | I used a Scanner object from the java.util library because it gives me access to methods such as “nextInt()” which allow me to gather input from the user with ease. This is compared to if I were to create a ‘scanning’ object or class of my own. This class is also part of a library which is very well known and documented, hence another reason why I used it. |
| File | File | I used a File object from the java.io.File library because it would mean that every time the user’s file needs to be used it does not need to be re-read, as it would be if file was a string of the filepath for example. |
| User | currentUser | I chose to represent the current user as an object of the class User as it is not only descriptive but serves the purpose of being able to construct a user as an object to which we can add specific methods. The current user in the program will have data that needs to be accessed specific to them so that is why I chose to use the specific data type. |
| Company | head | Used in the CompanyList class primarily this is used to set the first Company in the linked list. I decided to go with this data type as the first item in the list of companies needed to be a company. |
| current | Current is the company which is ‘current’ in the linked list. This is useful when looking through the list as it allows us to check for the current company and then proceed to the next one. I decided to use the company object because I wanted the CompanyList object to be a list of companies. |
| Company[] | companyArray | I chose to keep this array with actual Company objects because it meant that, instead of using a name for example, I could use the Company object to retrieve any data I needed from the company in the array. |

# Unified Modeling Language (UML) Diagrams

Diagrams representing each proposed main class used for the functionality of the program, containing their proposed methods and attributes.

|  |  |
| --- | --- |
| **Authentication** | |
| -  -  - | usersFile: String  currentUser: User  debug: boolean |
| +  -  -  -  -  +  - | Authentication()  signUp(username: String, password: String): User  logIn(username: String, password: String): User  writeString(file: RandomAccessFile, string: String): void  readString(file: RandomAccessFile, string: String): void  getUser(): User  toggleDebug(toggle: boolean): void |

UML Diagram 1 Authentication class

|  |  |
| --- | --- |
| **User** | |
| -  -  -  -  - | username: String  isAdmin: boolean  file: File  companyList: CompanyList  debug: boolean |
| +  +  +  +  +  - | User(username: String, isAdmin: Boolean, filePath: String)  User()  getUsername(): String  getIsAdmin(): boolean  getCompanyList(): CompanyList  toggleDebug(toggle: boolean) |

UML Diagram 2 User class

|  |  |
| --- | --- |
| **CompanyList** | |
| -  -  +  + | head: Company  file: File  debug: boolean  companyListLoaded: boolean |
| +  +  +  +  +  +  +  + | CompanyList(file: File)  add(company: Company)  save()  toArray(): Company[]  length(): int  isEmpty(): boolean  exists(fileName: String): boolean  getHead(): Company |

UML Diagram 3 CompanyList class

|  |  |
| --- | --- |
| **Company** | |
| -  -  -  -  -  -  -  +  + | next: Company  name: String  description: String  country: String  tickerSymbol: String  revenues: ArrayList<Statistic>  costs: ArrayList<Statistic>  debug: boolean  companyLoaded: boolean |
| +  +  +  + | Company(filePath: String)  Company()  getNext(): Company  setNext(next: Company) |

UML Diagram 4 Company class

|  |  |
| --- | --- |
| **Statistic** | |
| --  - | name: String  filePath: String  data: ArrayList<Data> |
| +  +  +  +  +  + + | Statistic(name: String, filePath: String)  Statistic(name: String, data: ArrayList<Data>)  getName(): String  readData()  getData(): ArrayList<data>  toString(): String  extrapolateData(monthsToExtrapolate: int): ArrayList<Data> |

UML Diagram 5 Statistic class

|  |  |
| --- | --- |
| **Data** | |
| -  -  - | year: int  month: int  value: int |
| +  +  +  + + | Data(year: int, month: int, value: int)  getYear(): int  getMonth(): int  getValue(): int  toString(): String |

UML Diagram 6 Data class

# Hierarchal Chart

A representation of how the proposed program’s windows and interface could be linked together.

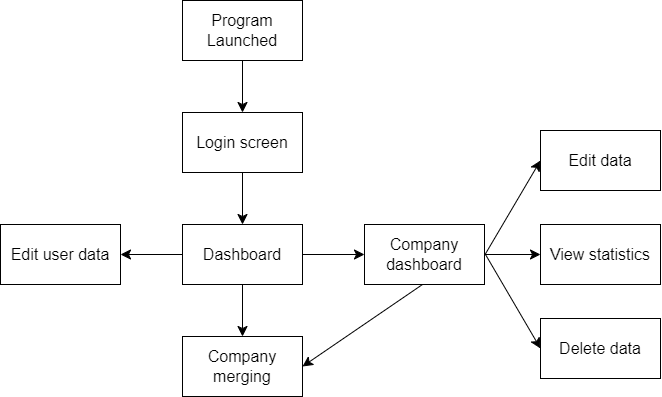


Figure 11 Hierarchal Chart

# Connection Chart

A representation of the class-relationship used in the program.

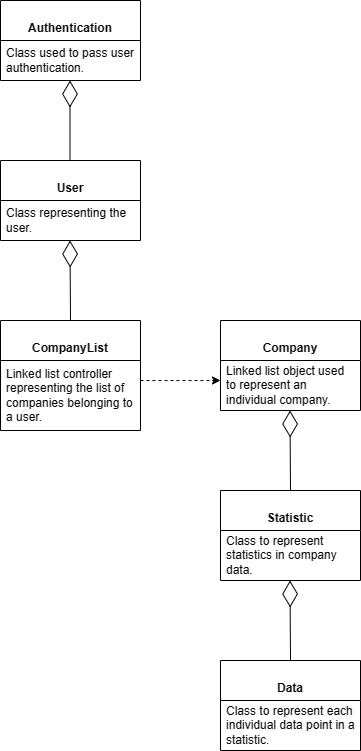
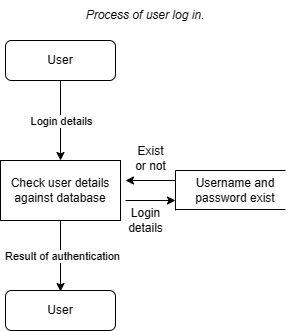


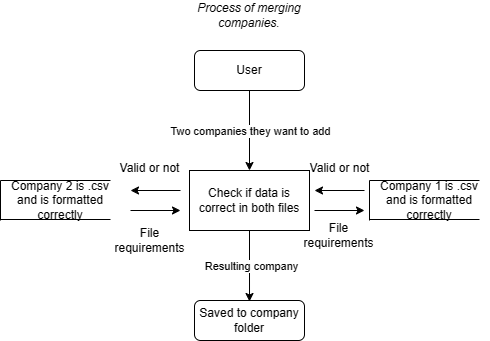
Figure 12 Connection chart.

# Data Flow

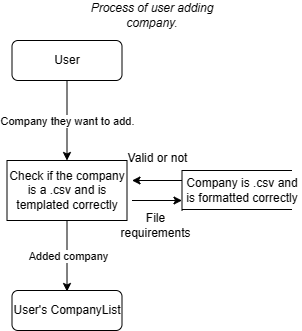
Various diagrams representing the flow of data in some of the programs’ most important processes.



Data Flow 1 Process of user log in.



Data Flow 2 Process of merging companies.



Data Flow 3 Process of user adding a company.

A diagram of a company data processing

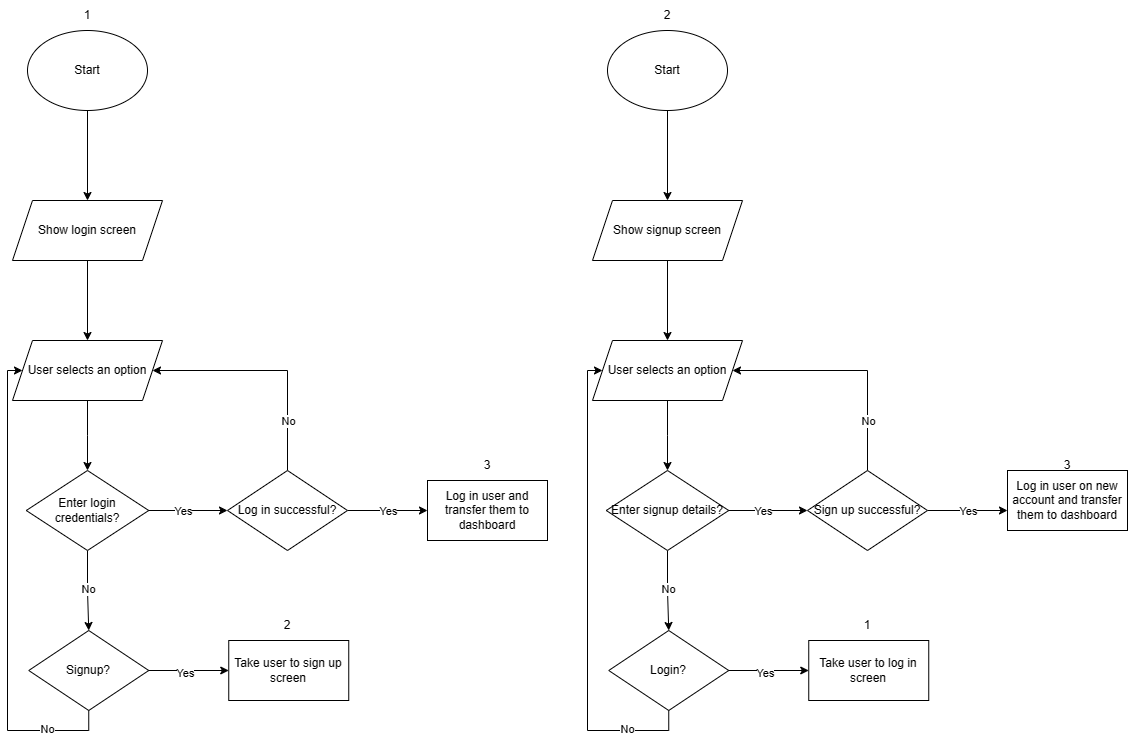
Description automatically generated

Data Flow 4 Process of creating company valuation.

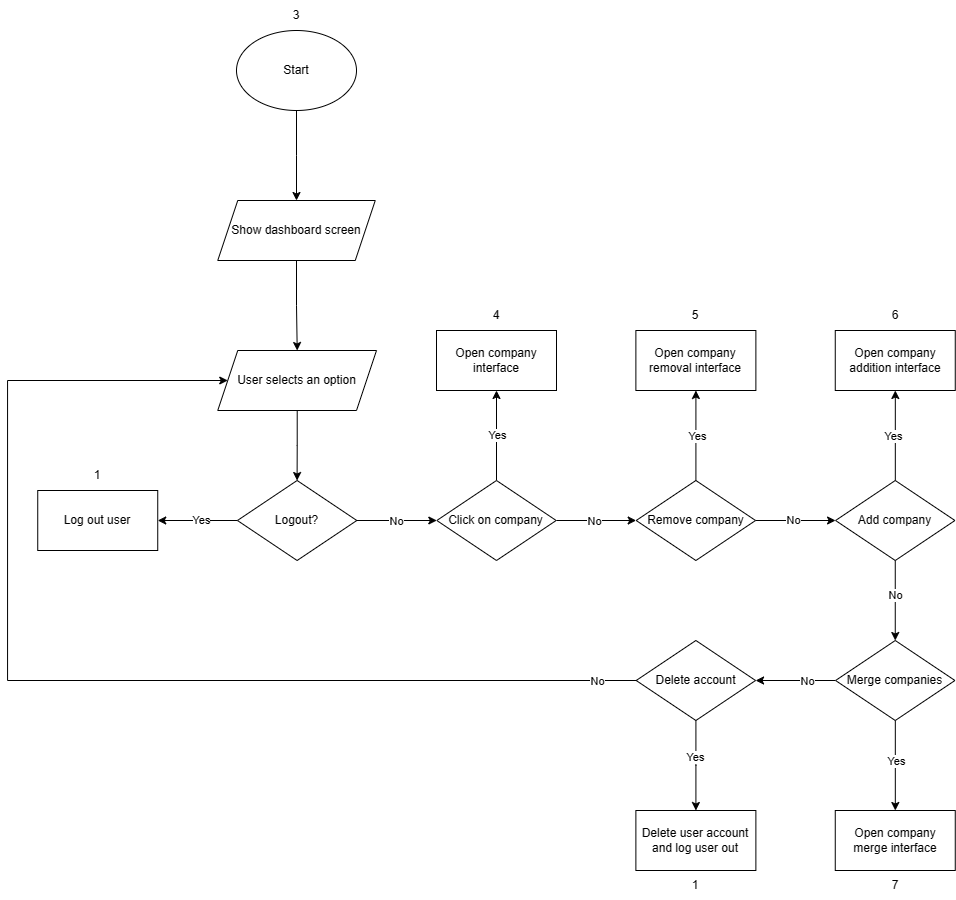
# Flowcharts

With flowcharts we can visualize the ‘flow’ or the way that the program runs and how a user would progress a program. This is useful to be able to fully plan out concepts and how they will specifically interact and call on each other.

To make the flow of the program easily understandable across all the separated flowcharts, numbers of the flow which each point on the flowchart corresponds to have been added. Wherever two numbers are the same it means that these places are connected in some way.



Flowchart 2 Initial screen

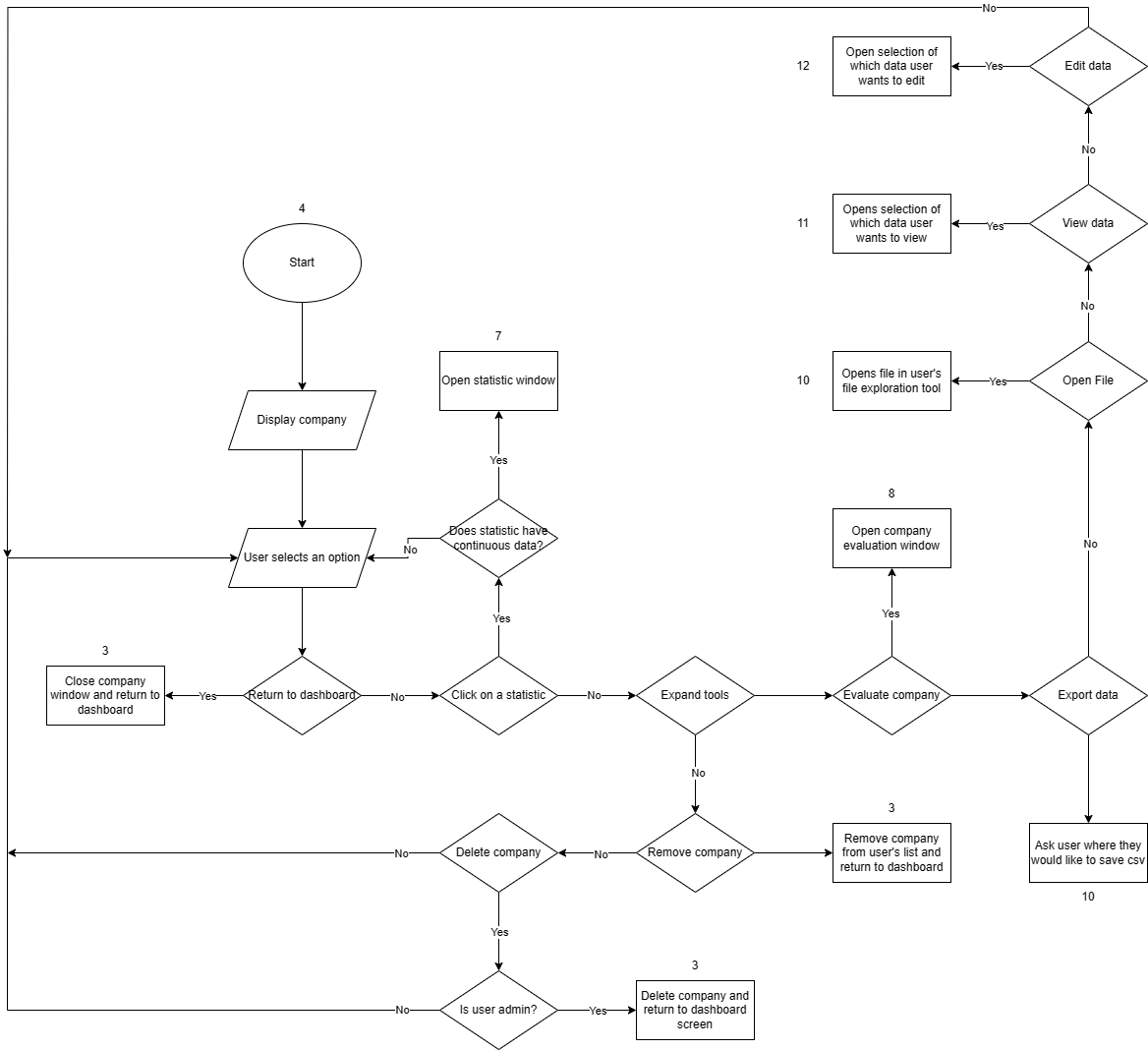


Flowchart 3 User dashboard

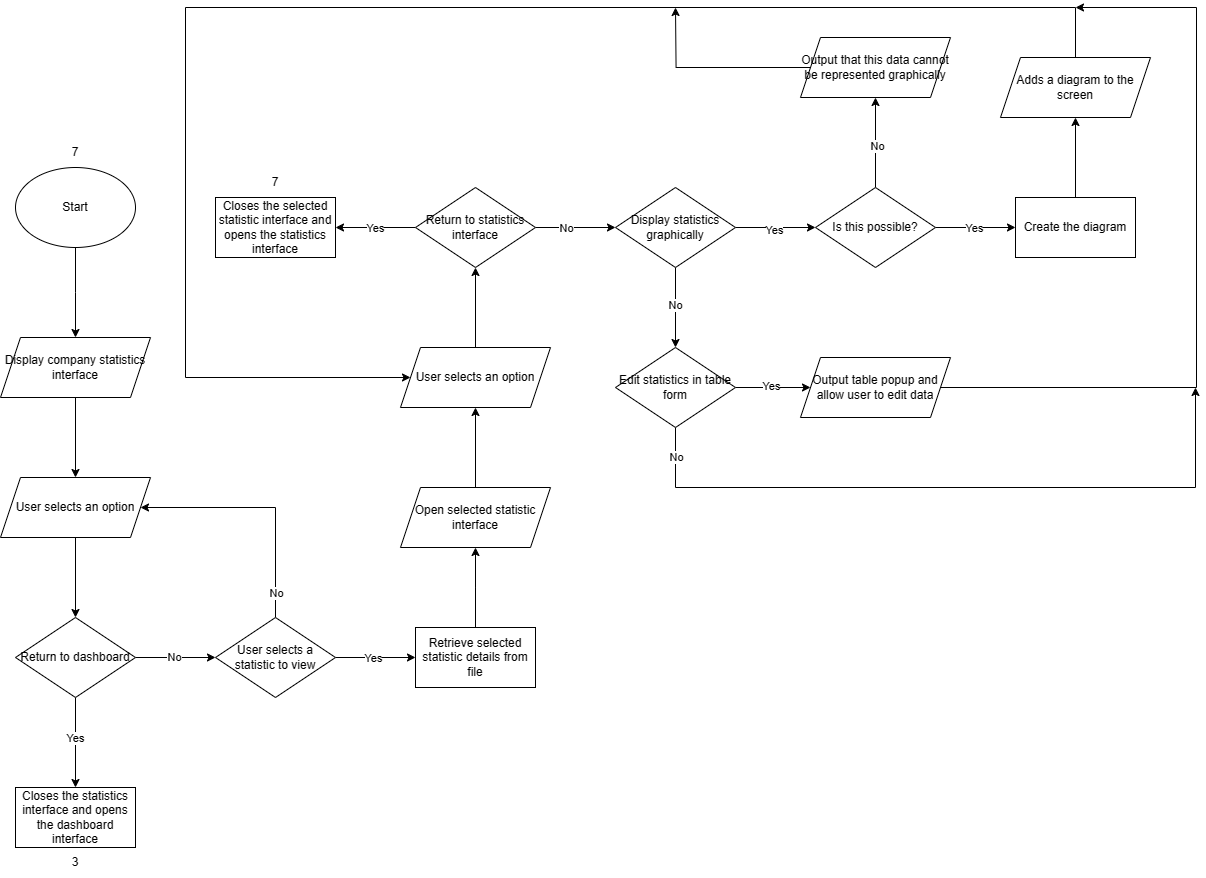
A screenshot of a diagram

Description automatically generated

Flowchart 4 Company addition and removal



Flowchart 5 Company dashboard flowchart.



Flowchart 6 Company statistics interface.

# Testing Plan

A plan of the various tests the program will have to undergo to make sure that it meets the success criteria.

|  |  |  |  |
| --- | --- | --- | --- |
| **Success criteria tested** | **Description of test** | **Test method** | **Expected outcome** |
| 1 | Test to see if you can add a company to the user file. | Open the file before adding it and check the contents. Then, run the program and open the dashboard and press the “Add Company” button and attempt to add a Company. Then, close the program and open the file and check the contents of the user file again. | The user file should have the new company added on the end of the first line of companies. |
| Test to see if you can delete companies from a user file. | Open the file before adding it and check the contents. Then, run the program and open the dashboard and press the “Remove Company” button and attempt to add a Company. Then, close the program and open the file and check the contents of the user file again. | The user file should now no longer contain the file name of the company that was removed. |
| 2 | Check if all company data is stored to file. | Open the file of a company (either a new one created through the program or added to it) and check the data there. Then open the program and perform something that would change data in any way of the company. | The contents of the company file should be in some way altered to what they were before. |
| 3 | Check to see if user inputs are validated. | Input abnormal, normal, and extreme data into various input forms throughout the program. | Program should output that the data entered is invalid and allow the user to input again. |
| Input blank spaces into any input. | Program should output a corresponding error message or popup. |
| Inputting a float into an input which requires an integer. | Program should output a popup which asks the user if they would like to keep the first part of the float (integer part, without the decimal) or if they would like to re-input a value that is an integer. |
| Inputting a string into an input which requires an integer. | Program should output a corresponding error message or popup. |
| Check that username inputted during sign up does not already exist in the database. | Input a username that already exists in the database in the signing up screen. | Program should output a corresponding error message or popup informing the user that the username they tried to input already exists in the database. |
| Check if user year inputs during company prediction are valid. | Input a year that is unfeasibly predictable (if data is from 2010 try to predict for the year 2100). | Program should output a message informing the user that the prediction they are about to make is not realistic, allow them to either proceed and get the unrealistic value or input again. |
| Input a year before the year that company data was recorded when trying to predict the value of the company. | Output a message informing the user that the program is not able to attempt to predict data before it reached the point at which it is inputted (chronologically). |
| Check that company file inputs are valid. | Input a file of a random file extension into the directory of the company files. | Simply skip over the item in the list as it is not considered a company file. |
| Do not put all the company details according to the company file template when loading a new company (from file) into the program. | Program should not load the company and output an error message to the user that the company selected is formatted wrongly and give them documentation on proper company templating. |
| 4 | Test to see how easy-to-use the interface of the program is. | Give the program to someone who is unfamiliar with it ask them to navigate or perform a certain task. | User performs the task without the need of documentation (a simple one) in a reasonable amount of time. |
| Survey people and ask them how they would rate various aspects of user friendliness and ease of user after using the program for a while. This is a more subjective approach but will get the opinions of people who could be users. | People should hopefully give the program a high rating in this aspect. It should be rated as easy to use. |
| Ask someone unfamiliar with the program to perform something complicated. | User should easily be able to access the documentation of the program (tooltips etc.) and easily figure out how to perform the more complicated task that they have been given. |
| 5 | Check to see if documentation about the program is easily accessed. | Ask someone unfamiliar with the program to perform something complicated. | User should easily be able to access the documentation of the program (tooltips etc.) and easily figure out how to perform the more complicated task that they have been given. |
| Check if documentation is accessible. | Open the program and hover over various things. | Tooltips should pop up where necessary. |
| Open the program and click on the information/notes that are provided about certain features. | Popups or information panels should appear when these buttons are pressed explaining how to perform actions relevant to the context in which these buttons were pressed (what they are next to). |
| 6 | Check if company data is retrievable by file. | Go to the company directory and open a company file. | Company file opened should be readable by a user. The user should also have access to read and edit the file as well as copy data from it. |
| Check if company data is retrievable by the program | Open the program dashboard. | Companies should be loaded, and their names should appear in the dashboard (the companies belonging to the user). |
| Open the company panel from the dashboard. | Program should have all the data that is included in the file being able to be read. |
| 7 | Test to see if applicable data of a company can be viewed by the user. | Open the company panel from the dashboard. | Basic company details should be outputted there (name, description etc.). |
| Open the company panel from the dashboard and then attempt to view some data as raw numbers (statistical data). | The program should output in the interface an interactable table of all the values that the user wanted to view. |
| Test to see if applicable statistics data of a company can be graphed by the user. | Try to graph discrete data that is in the company panel. | Data should be graphed in a suitable manner and various correlations and other details about the relationship of the data should be outputted. |
| Try to graph continuous data that is in the company panel. | Data should be graphed in a suitable manner and correlations should be shown. |
| Try to add a prediction time for revenue data. | Graphs should be generated with an imaginary extension line if the data is valid. The extension line should make sense unless the program warns of too little data. |
| Try to graph multiple streams of revenue data on the same window. | Data should be graphed in a suitable manner and correlations should be shown. Data should be side by side and there should be able to be comparisons performed between the data in some way. |
| 8 | Check if user can easily modify company data. | Try to edit discrete company data in table form. | Table should be opened, and the user should simply be able to click on values and try to edit them and it should work, and these values should be saved. |
| Edit a company csv file when the program is not running. | Changes should be saved – this is not really happening as part of the program so there should be no issue here. |
| Check if user is easily able to find the place where they can modify data. | Get a list of company files (actual files). | Files should be clickable and openable from the interface that appeared. |
| Ask a user unfamiliar with the program to attempt to open it and try to change some company data. | They find any form (whether file or interface) to edit the data without needing a lot of documentation (or simply easily being able to find the documentation to do so). |
| 9 | Check if user can use program to estimate company value. | Import a company for which the prediction calculations have been done beforehand and is known and input it into the program and attempt to estimate the company value for that specific year that has been pre-calculated. | Output the company value that is expected in that year and ask the user if they want a more in-depth view. The company value outputted should be near or the same as the value that was calculated beforehand. |
| 10 | See if the value that the program projects for the company can be viewed in a graph form over a variety of a time | Attempt to estimate how the value of a company will change until a specific year, the company should be one where the expected value until this year has been precalculated and verified. | Graph of company statistics as well as an imaginary extension of the value shown, and a confidence rating also given. The imaginary extension should cross or be near to values that were pre-calculated. |
| 11 | Users should be able to calculate value of two-companies when a proposed merger goes into action. | The value that a specific selection of companies when they are merged should be calculated beforehand. Then, these companies should be merged into the merging screen and merged. | The value of the combined estimation or value should be like that calculated before running the program. |

1. Please see Appendix B, 2nd Interview With the Client. [↑](#footnote-ref-2)