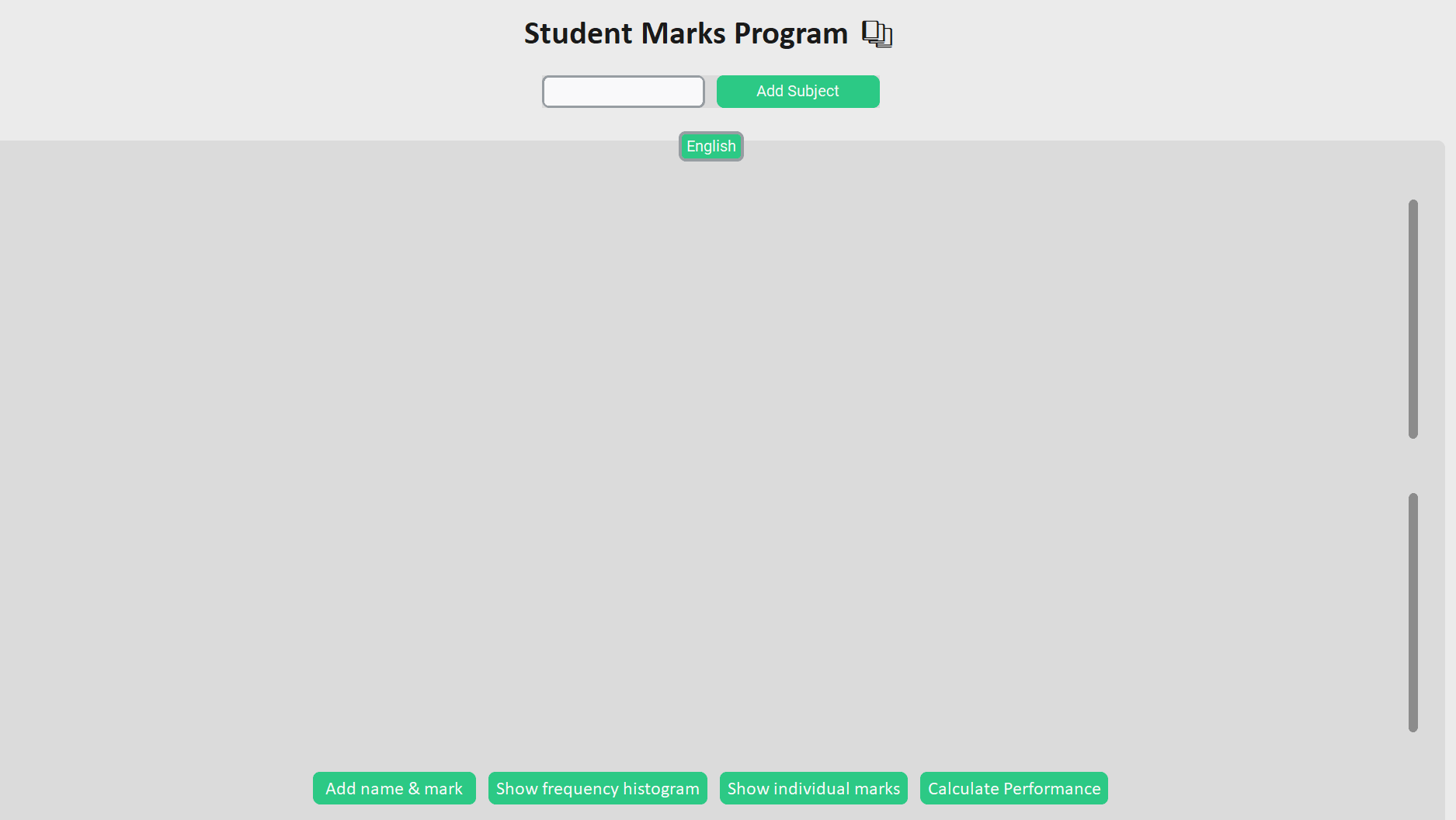
**Student Dashboard Management (SDM)**

**Name :** Warren Mao

**Github URL :** <https://github.com/Warrenmao1/Y12-AT3>

****

**Identifying and Defining**

**1.1 Problem Statement**

Many analytical systems used for judging student academics may overcomplicate data, using a range of indicators. This may create confusion or misunderstanding for those without a very strong background in statistics, generating inconvenience for those such as teachers plotting data or for audiences such as parents trying to understand it.

**1.2 Project Purpose and Boundaries**

I plan to create a program that allows teachers to easily monitor student performance through first inputting subjects, then student grades for that subject. The program would then automatically graph the data whilst providing a summary of each student 's performance. This would provide ease for teachers by automating graphing features and calculation features whilst summarising it in appealing graphics. However, it should be noted that predicting marks is beyond the boundaries of this project, with machine learning and incorporation being out of the scope due to concerns of its reliability.

**1.3 Stakeholder Requirements**

The stakeholders of this project are education institutions that require a modernised analytics system. These investors require the SDM to balance providing deep analytics whilst also summarizing this information in a clear communicative way. Additionally, the system should have a way of securely storing user data and information for significant organisations such as the department of education.

**1.4 Functional and Non - Functional Requirements**

* Functional
  + A user password system. It should allow the teacher to enter a username corresponding with a password which is either stored when signing up or checked if it matches one another for logging in.
  + The ability for students' names and marks inputted across multiple subjects of the user’s choosing.
  + A calculation algorithm to determine the mean and standard deviation of student grades. It should compare students' grades for their respective subject to the mean through the use of the Z score.
  + An external UI from the console which allows users to type and input marks and display a frequency histogram.
* Non functional
  + The sign up page should ensure the username is not the same as one existing in the database. Both the login and signup system should ensure a password and username are entered at all in the text boxes. Additionally hashing is to be incorporated for passwords
  + In each mark text box for each student, multiple scores should be allowed to be entered with an average being calculated from these scores.
  + The UI should show a very clear and readable curve, using colours to symbolize student performance.
  + Error handling is to be incorporated to check inputs that match a criteria such as marks being in the form of numbers rather than letter grades and provide a succinct error message of the reason for error.

**1.5 Constraints**

The main constraints on this project is the time frame window and implementing a diverse range of features. A GANTT chart would be produced to allocate the sections of development and scale its feasibility. Furthermore as mentioned under the system design section, diagrams are to be created to understand the flow of the program.

**Research and Planning**

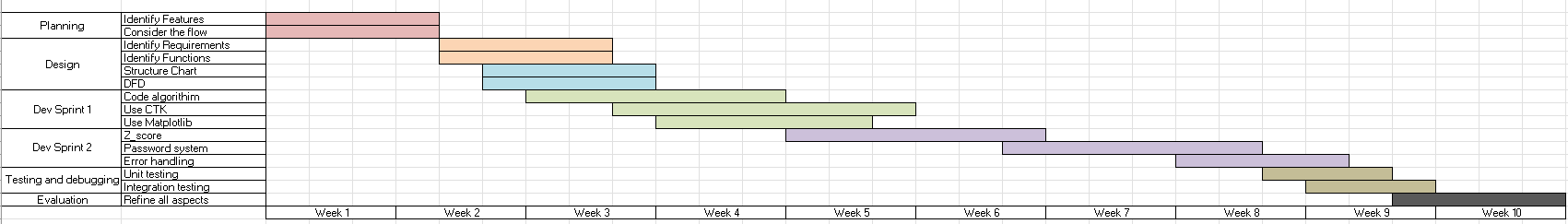
**2.1 Development Methodology**

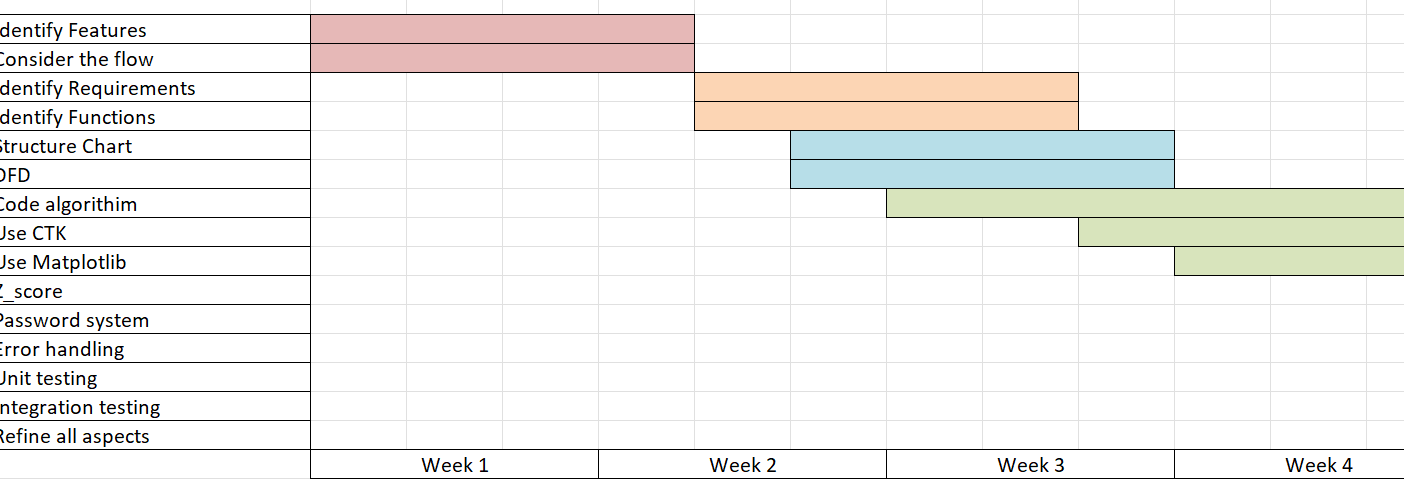
The Agile method has been chosen to approach this project due to its flexibility and ability to scale down larger projects. The Agile methodology is a project management approach that concentrates on iterative development to adapt the project. This is done by breaking down work loads into multiple development sprints rather than focusing on a large uniform workload to fully complete the project. By breaking it down into parts, it allows change to be more flexible and increases scalability of the whole project by targeting and focusing on individual parts.

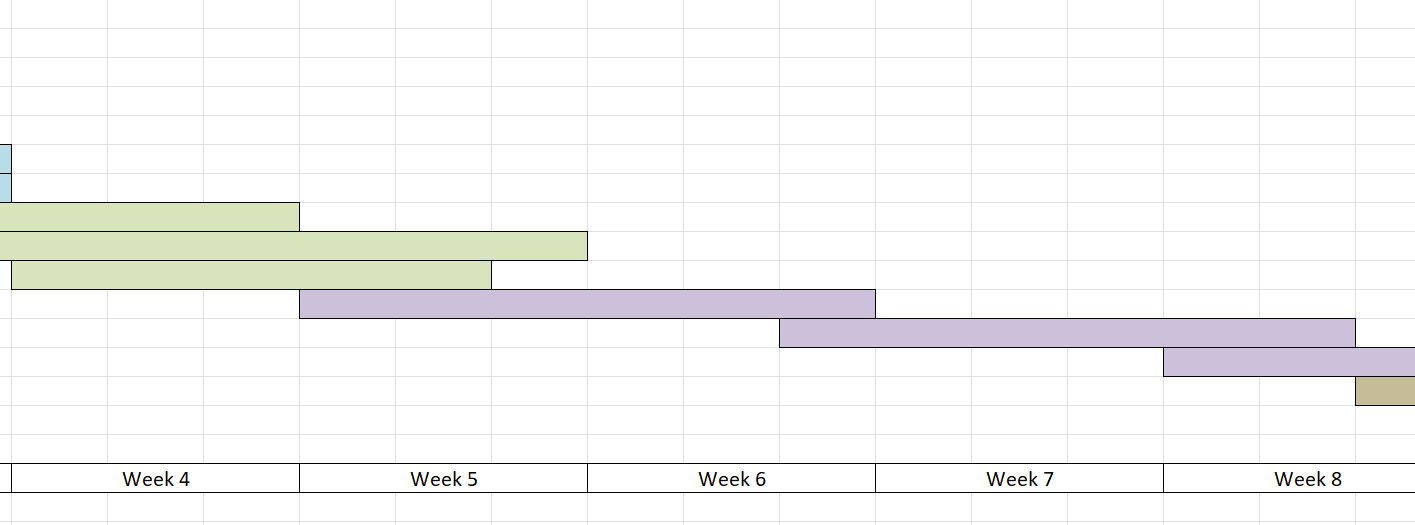
**2.2 Tools and Technology**

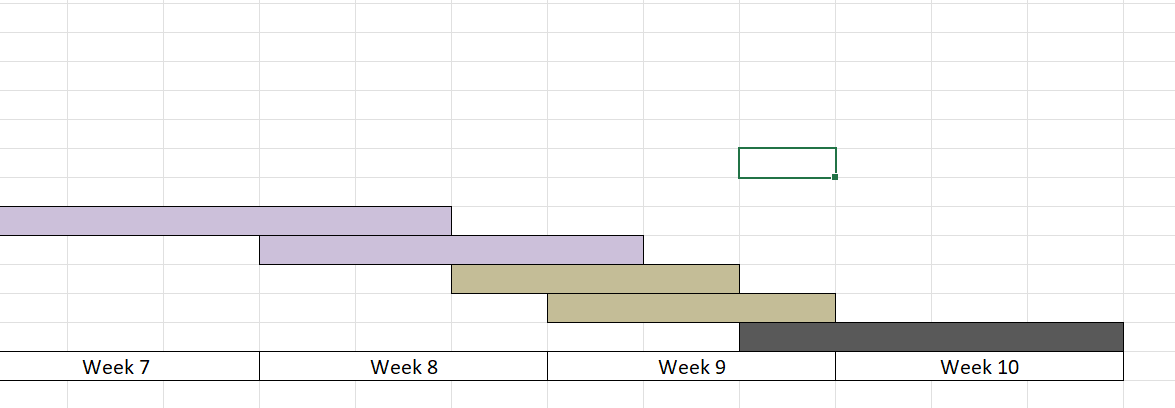
Python is the only coding language that is to be used in this subject. This is to increase ease of development as a result of extreme familiarity. Visual Studio is the IDE that should also be used due to its ability to be stored locally and also be connected to Github. Github is to be used to store files and documentation such as “read me”, creating a central location for all files to be stored. Regarding libraries, Custom Tkinter is the main one to be incorporated in order to provide an external UI alongside Matplotlib to graph results. Numpy is to be used for data calculations. Finally, Json and Hashlib are to be integrated for storing user data securely.

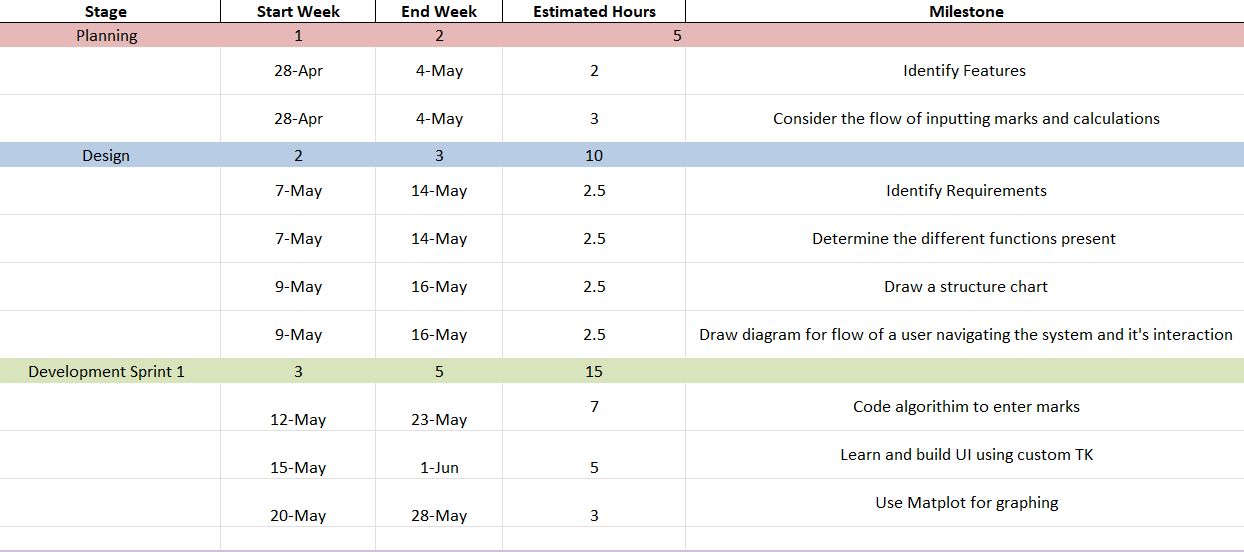
**2.3 Gantt Chart and Planning Table**

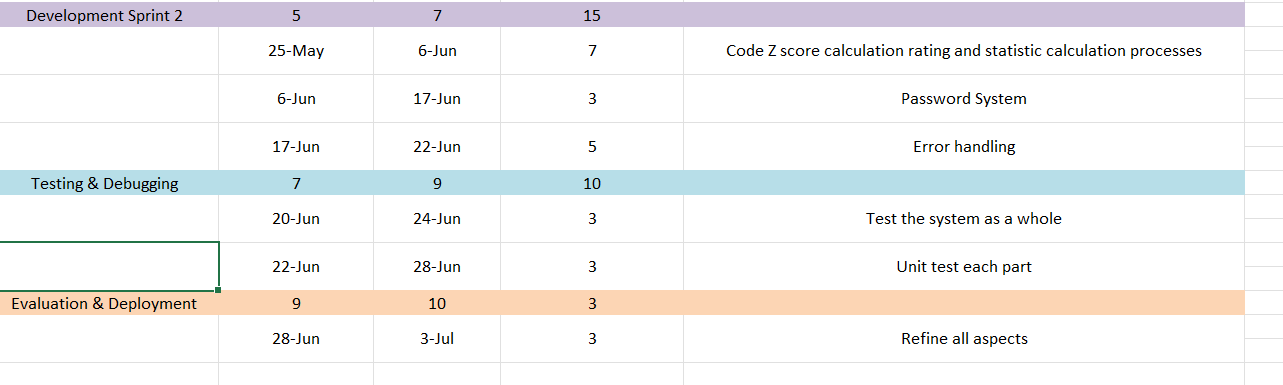
****







*Note : Same Gaant chart, with the second to third screenshots just being the original one split into multiple screenshots* 

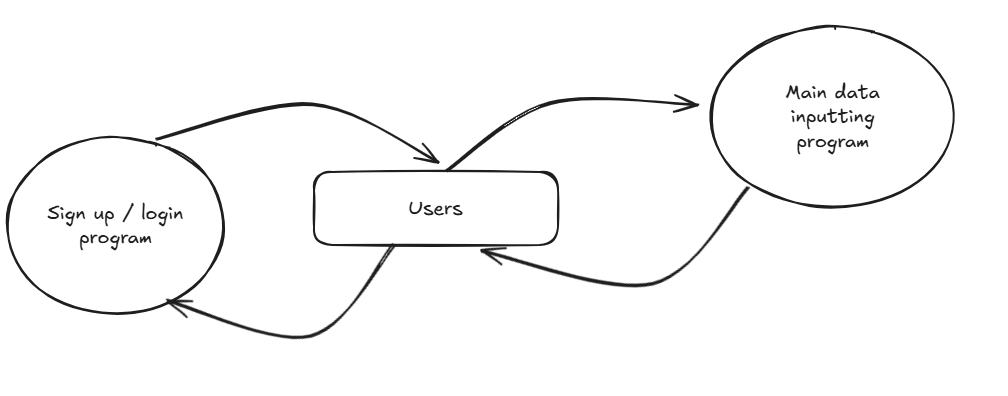
****

**2.4 Communication Plan**

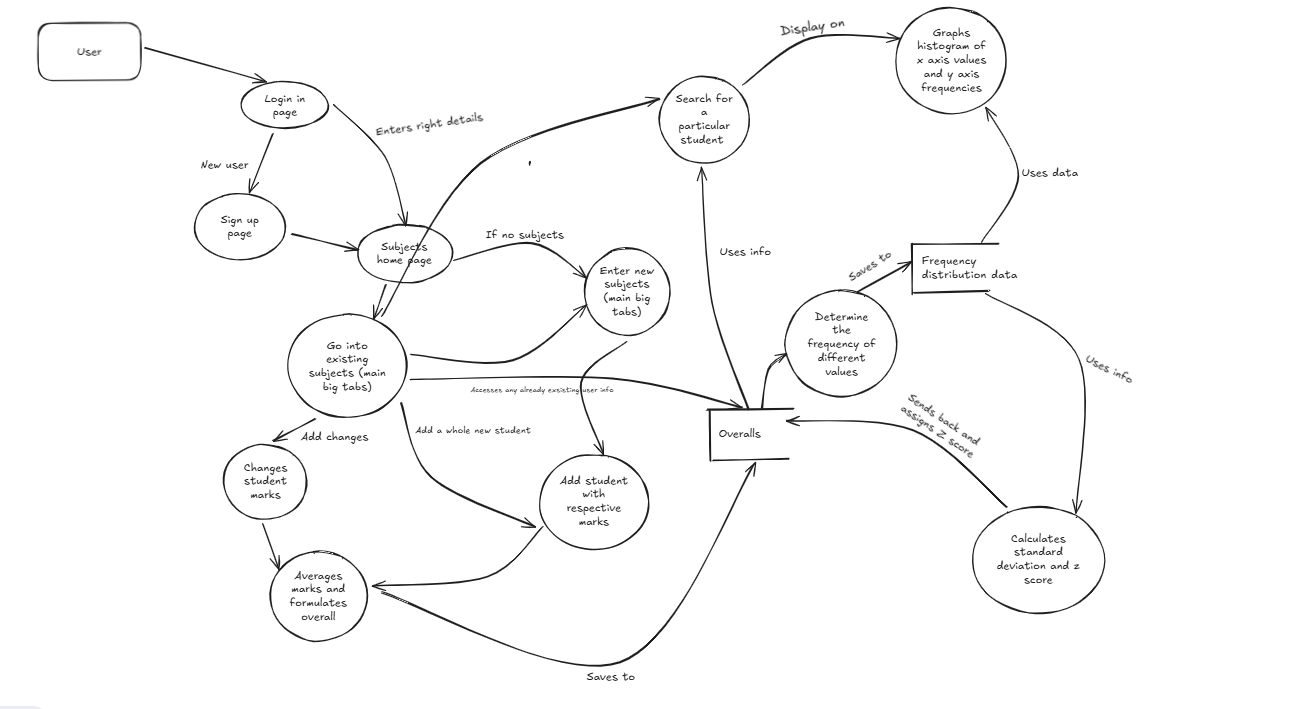
Feedback for the program would be fathered through user testing. User testing is a form of testing by having multiple users “trial” the program. By using this form of testing, feedback can also be gathered from their experiences and further enhance the program. This would also ensure the program is able to fulfil all the functional and nonfunctional requirements to a high standard.

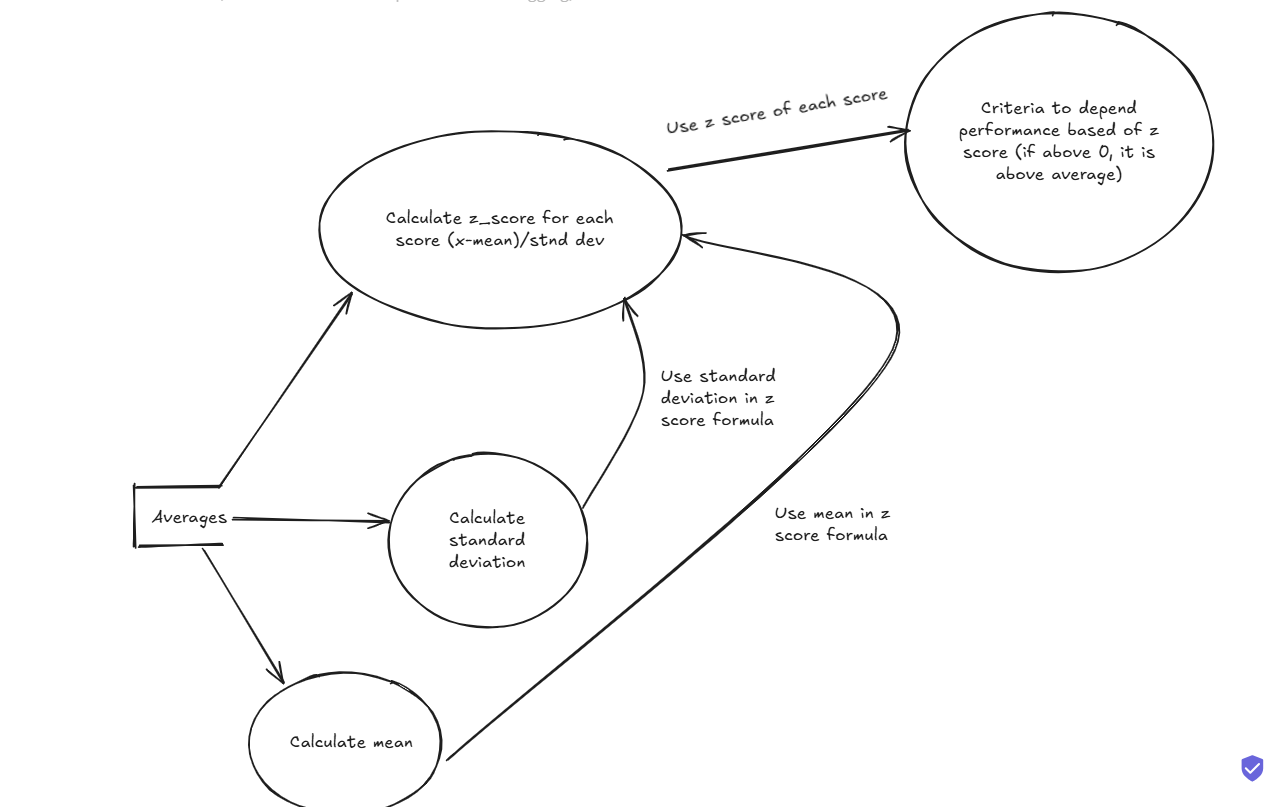
**System Design**

**3.1 Context Diagram**

****

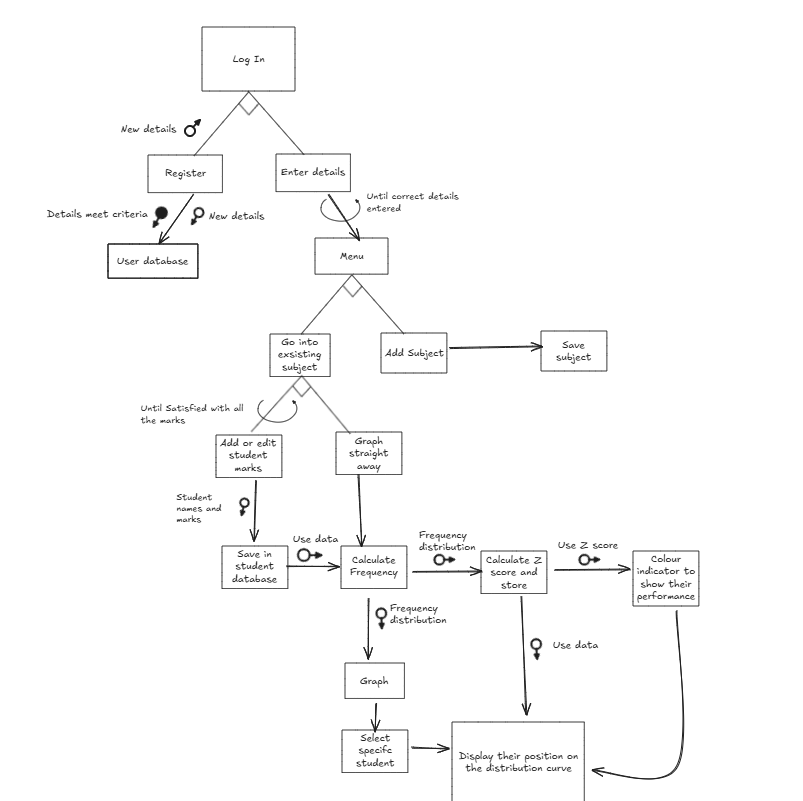
**3.2 Data Flow Diagram (level 1 and 2 respectively)**

****

****

*Note : DFD level 2 focuses on the “calculates standard deviation and z score” process in the bottom right corner of DFD level 1*

**3.3 Structure Chart**

****

**3.4 IPO Chart**

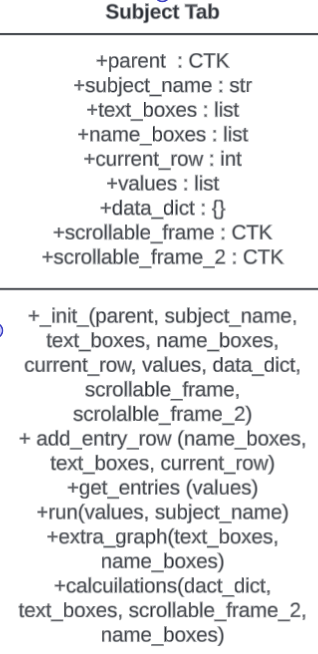
| **Input** | **Process** | **Output** |
| --- | --- | --- |
| Account credentials when signing up (username and password) | Saves them to a JSON file and hashes | Allows user to enter a program |
| Login details in order to login | Check them with the JSON file to see if the username matches the hashed password. | If the credentials match, it allows them to enter. Else, error message appears |
| Names of students | Saves them to a dictionary alongside there mark | Name appears in textbox and on diagram with individual marks |
| Marks of students | Saves them to two lists, one which has the raw values and one which averages them out the raw values by splitting them. | Marks appear on the diagram, with it being on y axis for the frequency histogram and individual marks column graph. |
| Marks of students | Saves them to a list, which splits it at commas and spaces. The numbers are then checked if they match a criteria and averaged. | Updates the list which is used for calculating performance |
| Averaged marks of students | Z score is calculated for each mark by first finding the standard deviation and mean. This is then plugged into the formula :  The value of the z score is then used to judge student performance. | Shows student performance on scrollable frames in Custom Tkinter. |
| Inputted subjects | Creates an instance of the subject using the class. Additionally, it is saved to a dictionary with subjects | Subject is added to the subjects tab at the top. |

**3.5 Data Dictionary**

| **Name** | **Type** | **Desc** |
| --- | --- | --- |
| app | Widget | The main Customs Tkinter application variable |
| self.parent | Attribute | Acts as the parent window, with a new one created for each subject (parameter in class). It stores the two main scrollable frames for input and displaying z scores. |
| self.subject\_name | Attribute | Attribute for subject names. It uses the name of the subject (parameter in class) entered in subject\_name. |
| self.text\_boxes | List | Adds marks for each student unique to each subject |
| self.name\_boxes | List | Adds names for each student unique to each subject |
| self.values | List | Stores average marks if the user adds multiple marks to each text\_boxes. |
| self.data\_dict | Dictionary | Adds names of students and their respective Z scores |
| self.scrollable\_frame | Widget | Adds a scrollable frame in each tab for the entry. It’s master is the parent window for each tab |
| self.scrollable\_frame\_2 | Widget | Adds a scrollable frame in each tab to display z\_scores. It’s master is the parent window for each tab |
| row\_frame | Widget | New frame created every time a new entry is made. |
| avg | Int | Calculates average of the marks using the multiple entries |
| data | Array | Converts the marks list of averages to an array in order to be used for calculations |
| mean | Int | Finds mean of all the marks in data using numpy |
| names | List | List of all names to be assigned to z scores |
| marks | List | List of avg |
| z\_scores | List | List of all z\_scores. It is rounded to three decimal places |
| parts | List | List of the parts in each text box if multiple marks were typed in. All commas are first converted into spaces. Each string is split into its own list of parts |
| nums | Int | Converts string in parts to int |
| std\_dev | Int | Finds the standard deviation for the set of data scores using numpy |

**3.6 Class Diagram**

Since there is only one class, the following is the diagram for one of the classes :



**Producing and Implementing**

**4.1 Development Process**

In order to ensure the quality of the program and that it covers all functions and requirements, diagrams were first drawn in order to comprehend the scale of the program. For example, data flow diagrams were crucial in envisioning the cohesion of how a user interacted with the program, also providing insight into background processes. Additionally, fundamental features were first programmed to provide a “base” for other features to stem off from. For example, the login/sign up system was only integrated after the main program had been finalised, ensuring a foundation used to be built upon and thus aiding the development process. Furthermore, Agile is the primary design development approach to be used due to its ability to break down the project into smaller steps. The project has many features that have their own functions but also integrate with other features. In order to make this more scalable and approachable, the project was broken to development sprints with the process of each function being identified before attempting to understand how it would integrate amongst other functions.

**4.2 Key Features Developed**

The core features of this program are distinguished through the functions and classes of this program

| **Feature** | **Description and Justification** |
| --- | --- |
| Subject Tab Class | This class provides a blueprint for all the functions of the program per subject tab, using the values entered in each tab for each subject's graphs and z scores. By having a class, it ensures values placed into each subject tab are unique to that tab and thus maintaining organisation. |
| Class initialization | This defines all the attributes of the class and runs every time a new tab is created. It has the parameters of self (the copy of the tab), parent(houses all the widgets for each subject) and subject\_name (name of the subject entered outside the class). It is essential to the class by defining all the attributes. |
| Get\_entries | This function extracts all values entered in name\_boxes and text\_boxes and saves it to lists. Additionally, it splits the entries if multiple were in the text box. This is crucial in bolstering the usability of the program, eliminating the need for users to formulate averages manually. |
| Add\_entry\_row | This function is run every time button\_add\_row is pressed in order to create a new row to add a name and mark. This is done by creating a new frame for each row and placing widgets in this row. |
| Run | This function creates the graph using matplotlib. It first creates bins to group the data which allows a polygon to be graphed by connecting a red line to the centre of each bin. This function is essential to plotting the graph. |
| Calculations | Calculates the z\_scores of the scores. This is done by first extracting data from the name\_boxes and text\_boxes and putting them into lists. The list containing the extracted marks from text\_boxes will then be converted into an array, before the mean for all scores was found using the array of marks. The standard deviation for each score was calculated before the z score for each score was found. Each z score was assigned to the person of that mark in the data dict. This function is necessary to determine the performance of each score. |
| Add\_subject | This function adds inputted subjects to the tab\_view on the top of the ctk window. It first checks if there is any input in the subject\_entry text box and whether the subject already exists in the subject\_tabs dictionary. By doing so, it allows the input of subjects to be handled correctly. |
| Master\_program | This is the function that stores the entire main program, with the function running after the login/signup criteria have been satisfied. This enables the smooth transition from the sign up/login system to the main program. |
| Login | The function is first run before signup, allowing users to input details to login, making sure the inputted username and password match saved data in the Json file. This is crucial in meeting the requirements of producing secure software architecture, complying with industry standards by also using hashing in the hash function. |
| Sign\_up | Similarly to login, it allows users to create an account, ensuring it meets the criteria of having a unique username. This is crucial in meeting the requirements of functionality in establishing the basis of secure software architecture. |
| Extra\_graph | This plots an extra graph, using names list as the x axis and marks list as the y axis. This is essential in providing variety for the user for different ways to graphically show student marks, increasing understanding. |

**4.3 Interface**

| **Screen Shots** | **Desc** |
| --- | --- |
|  | This is the login page of the system which is the first window that appears when the program is run. It allows users that have already signed up to input their username and password. If a user has yet to sign up, they can press the sign up button to go to redirect them to sign up. |
|  | This is the main page once a user has logged in. They can type in the name of a subject and press add subject to add it to their tabs. For example, if a user was to write “English", “English” would appear at the top of the tabview. |
|  | The user can add the name and mark of a student by pressing “Add Name and Mark”. Additionally, marks are unique for each subject. Furthermore, by pressing the “X”, it deletes the frame and therefore the entry of that row. |
|  | Once marks are entered, users can press “add frequency histogram” to see a graph of a histogram, with frequency graphed on the y axis and the types of scores graphed on the x axis. Additionally, a polygon is added for interpolation. |
|  | Users can also show a bar graph of the student’s individual marks by pressing “Show Individual Marks” compared against each other. |
|  | An overall summary of each student is also given when “Show performance” is pressed. It shows each student’s z\_score and also their performance compared to the average. The text in brackets show the range of standard deviations that they lie from the mean. |

**Testing and Evaluating**

**5.1 Testing Methods Used**

Testing is a process in Agile development, which ensures that the system functions properly, meeting criteria derived in previous stages of development. In order to effectively debug the program, there are multiple methods of testing as defined below :

* Unit Testing : This is done by breaking the main program into individual functions and modules. Each individual component is then tested to identify and address separate bugs.
* Integration Testing : This is opposite to unit testing, where individual components are combined. This provides understanding of how different features interact with one another
* Acceptance Testing : Runs the program with all the components to see how it operates in a real world scenario.
* User Testing : Allows users to run the program and also observe possible errors that may occur in real world usage.

**5.2 Testing Logs**

| **Name** | **Type** | **Desc** | **Results** |
| --- | --- | --- | --- |
| Opening Json file and inputting usernames and passwords | Unit testing | Unit testing was used to first determine the fundamentals of opening json files before integrating with the whole program. | Fail. It was determined that each user had to have a key with their name and a value of their password. |
| Opening Json file and extracting usernames and passwords | Unit testing | Unit testing was applied to see if the program was able to extract information from the json file | Pass |
| Creating the sign up page to input usernames | Integration Testing | The sign up function as a whole integrated the input features | Pass |
| Creating the login page | Integration testing | Integration testing determined if the login page was able to work alongside the sign up function, using newly inputted usernames and passwords. | Pass |
| Determining if passwords or usernames exists | Unit testing | Unit testing was done to see if a for loop was successful in reading a translated json file. | Pass |
| Integrating sign up/login with main program | Integration Testing | Integration testing was applied to see if containing the main program in a function would work | Pass |
| Classes for each subject | Acceptance and integration testing | Acceptance testing was done by running the class as a whole by giving an example instance of a subject. Integration testing was done to observe how | Pass |
| Histograms for subjects | Unit testing | Unit testing was done by creating a separate file to run and create a distribution. | Fail. The values plotted on the x axis had to be filtered from scores with 0 frequency. |
| Calculating z scores | Unit testing | The process of gathering data from lists and calculating the z scores. Unit testing was done to ensure that this function worked before sorting it out in the main program | Fail. the entry.get() was needed in order to convert the widget objects to the values in the widget boxes. Without entry.get(), it saved values entered in the textbox as widgets rather than the values put in the text box. |
| Error handling for textboxes with no content | Integration testing | The ability for the program to place an error message while disrupting the rows was done through integration testing and was tested alongside the function of adding rows. | Fail. Grid and pack cannot be mixed in the same parent widget. |
| Determining nature of values in widgets | Integration Testing | In order to add further error handling to ensure all marks are values, testing was done to determine the original nature (string or int) of marks when extracted from input. | Pass |
| Ability to average marks | User testing and unit testing | The ability for a string to be broken down was tested separately from the program. This was given to users to have feedback on it’s operation (e.g if when users inputted marks, it correctly averages their marks) | Pass |

**5.3 Evaluation Against Requirements**

The solution produced has covered all the components required in the criteria, meeting the goals established at the beginning of the Agile process. For example, functional requirements have all been met such as having the foundations of a sign up login system and allowing users to input their own subjects. Additionally, some non - functional requirements have been met, including an easy to navigate UI and a hashing process to increase security. By meeting these requirements, it ensures that users have access to all the basic features in the program.

**5.4 Improvements and Future Work**

Although the program has met all functional requirements, some of the non - functional requirements were met to their full potential. The ability for the program to display where a student is on the polygon could have been integrated, making the program more visually appealing and providing features that may help users better analyse student performance compared to the cohort. Furthermore, the ability for subjects to be saved for each student after the runtime session concluded was planned to be added yet wasn’t due to time constraints.

**Feedback and Reflection**

**6.1 Client Feedback**

Client feedback was collected at the end of development and throughout several stages. The following were common points of feedback :

| **Name** | **Feedback** |
| --- | --- |
| A | * Enable the ability for users to write multiple marks in one textbox for each student, representing percentages they achieved in the assessment tasks. Average these marks to find the overall percentage, eliminating the need for teachers to manually figure it out. |
| B | * Allow the program to save subjects for each user even after the runtime session is over. |
| C | * The interface should be further modernised, specifically coloured emojis for the z\_score system |
| D | * It should give a deeper summary of each student, with maybe a tab for each student in a new window |

**6.2 Personal Evaluation**

Many valuable skills were learnt and developed through this project. For example, this project allowed me to experiment with external UI’s rather than being confined to the terminal, understanding how spacing features such as packing works. Furthermore, the basics of user security were developed through the username password system, understanding how to save information in a json file. Additionally, software design skills were further refined through working on diagrams.

**Appendix**

**Log**

| **#** | **Date** | **Desc** |
| --- | --- | --- |
| 1 | May 19th | Created the first tk window with the title as well and two tabs. |
| 2 | May 22nd | Through the design stage, a scrollable frame was identified to be the most suitable option as the main parent frame, allowing users to scroll.    Additionally, buttons were added to the main frame. |
| 3 | May 23rd | Text boxes for names and the mark values were added to the program within the master scrollable frame. Lists were also made for names and marks to append the user input. |
| 4 | May 23rd | Changes were made to the rows, with each row now receiving their own frame in order to create a delete feature later on. |
| 5 | May 25th | Added delete row feature. |
| 6 | May 28th | get\_entries() was made to extract the marks in the list and convert them to be integers. |
| 7 | May 28th | run() was made to create the matplotlib histogram using the values list created earlier. |
| 8 | May 29th | Changes were made to exclude bins with zero frequencies to prevent them from being plotted |
| 9 | May 29th | Frequency polygon was added, with the bin centers first being identified before a line graph was integrated. |
| 10 | May 30th | An option to create a column graph was added. |
| 10 | May 30th | An error was encountered regarding the format of name\_boxes and text\_boxes in the new graph. Changes were made |
| 11 | June 1st | Calculation system is started to be worked on |
| 12 | June 3rd | Changes were made to the z scores list, by using the list function instead. However, when put into a dictionary with names as keys and z scores as values, the program doesn’t print anything. |
| 13 | June 9th | On the new data tab, a row frame similar to the one in the entry tab was added. This was to display the z scores as a label on each row, with each row being its own frame. |
| 14 | June 11th | In order to classify those with higher and lower z scores, an if else statement was used depending if they were below or above 0. |
| 15 | June 18th | Further detail was added to the z score classification system, making it easier to differentiate marks. |
| 16 | June 18th | The two tabs were removed as it added confusion to where different features were located. Instead, all features are located on the default parent tab. |
| 17 | June 21st | The feature to add multiple subjects and have designated marks for each subject was created. This was done by incorporating object oriented programming. Everytime a new subject is created, it uses the class as a blueprint. |
| 18 | June 25th | A login feature was added. It was first unit tested before being integrated with the main program. The main change was the order being shifted, with a lot of the main master program initialization put at the start and a large function was made to encapsulate the main program. |
| **19** | June 30th | Error handling was first added to the system. The first part of error handling was in the username and password system by ensuring the textboxes had to be filled (for an error could occur if someone signed up with no username or password which was save as  “ “ : “ “ in the json file. |
| **20** | July 2nd | Error handling was added to the master program, primarily through the use of if not, which checks if any criteria (e.g if there are marks entered or not) is false, which therefore is flagged. This was added to ensure all names and marks were filled out and that the marks were numbers. |
| **21** | July 3rd | The performance classification was slightly changed to cover a greater range of standard deviations to further differentiate students. |
| **22** | July 3rd | Hashing was added to ensure further security. This was first done by importing hashlib. The hash password takes in the password as plain text and returns as a hashed version. When a user tries to login, the password they entered is hashed and checked if it matches the hashed password in the json file. |

**GitHub Link**

<https://github.com/Warrenmao1/Y12-AT3>