REVISION SOCIAL MEDIA

# 

Centre number: 23332

Littleover Community School 23332 Candidate Number: 7121

## **Non-Exam Assessment**

## **Steven Beardsley**

Contents

***Background to project***

* Current systems 2
* Identification of an End User 2
* Prospective Users 2
* Evidence of analysis 3
* Observations 6
* Interview with end user 6
* Log/Diary of research 6

***Analysis***

* Modelling of the problem (initial DFD) 7
* Proposed solution details (hardware/software) 11
* Set of objectives 11
* Acceptable limitations 12

***Design***

* High level overview 13
* Flowcharts 15
* Pseudocode 16
* Database Design 23
* SQL 26
* Design of the user interface 28
* System security and integrity of data 34

***Testing***

* Testing 35
* Remedial Action 44

***Evaluation***

* The Project Report 45
* User feedback form 47
* Evaluation-interview 47

***Appendix***

* Appendix 48

**Background to project**

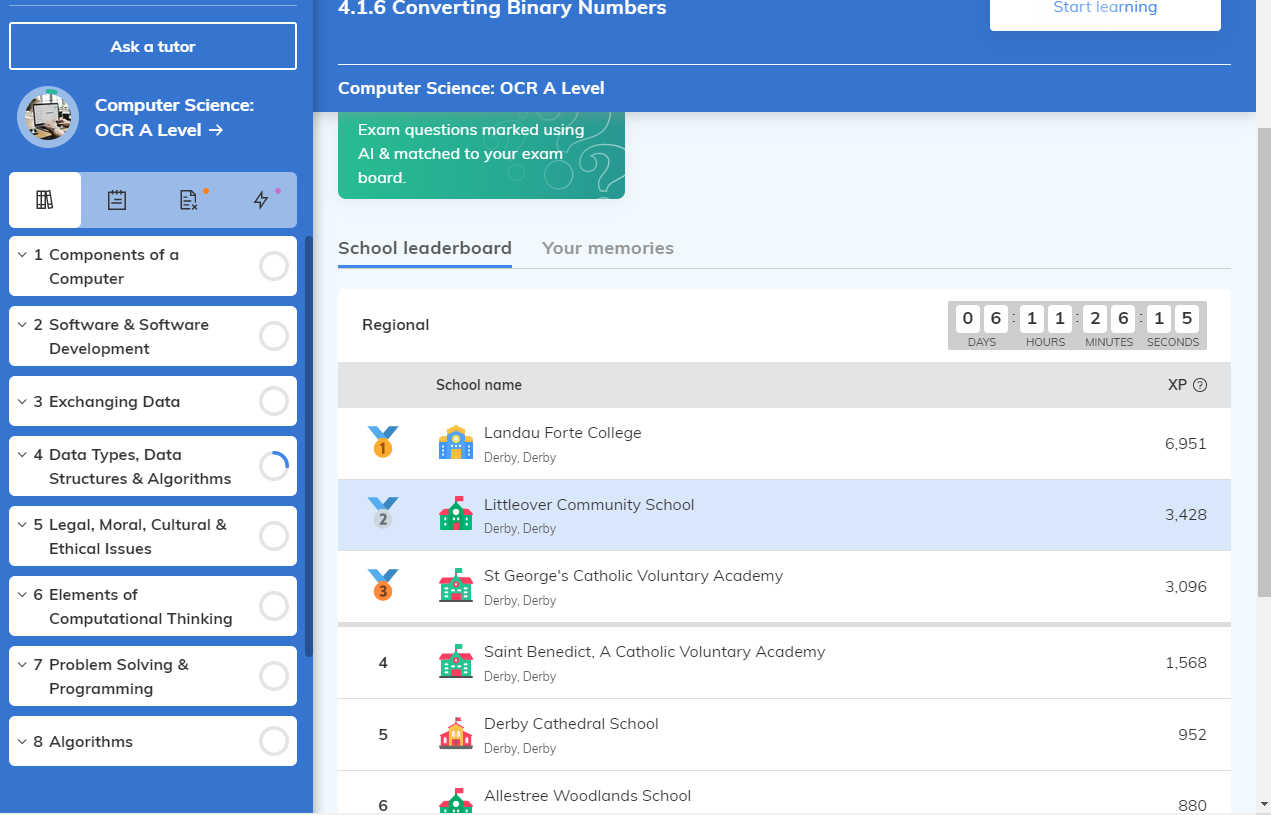
I am planning to make a revision software which operates as a social media for people to share their revision and compare with classmates and friends. I will use a GUI as the basis of this project to create an eye-catching software which is easy to use and understand. By creating this software, it means that the users will be able to optimise their revision time through looking at their statistics and revision habits and finding their best revision techniques.

Current systems

Right now, the most commonly used revision sharing interfaces are those inbuilt into testing websites, for example those on Seneca and Quizlet. These leader board systems only account for those specific websites and miss out on a range of other forms of revision including past papers, worksheets and making flashcards. My software will allow the student to upload and access all forms of their revision and look, in depth, into their revision habits and compare against other students’ accounts- learning and competing against others giving them motivation to revise whilst making revising more fun. I have included some screen shots of these current systems below.



Here is Quizlet’s software involving flashcards and revision. It has a clear title and the flashcards are the main focus, making it easy to traverse and view the set of flashcards. It also involves a few different features including a learn mode and a test mode as well as some other game modes.

Here is Seneca’s software. Here they have implemented a school leader board to compare and compete different local schools against each other in terms of revision times. This may be something I plan on implementing at a smaller scale.

Identification of an end-user

My end-user will be Jack McAteer. He is currently sitting year 12 exams and having to revise 2-4 hours each day. He currently has no database to showcase his revision habits and thus isn’t properly utilising his revision to get the best out of his time revising. He will use this system, once created, daily after revising to help prepare him for his exams.

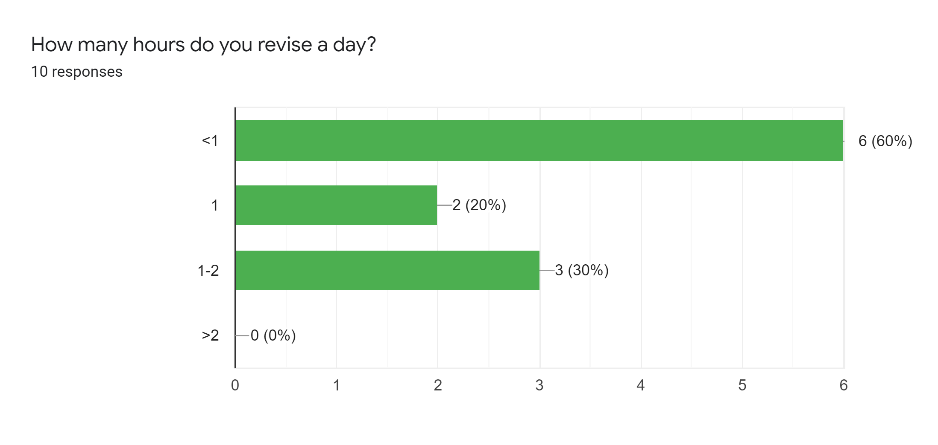
Prospective users

As well as my end user Jack McAteer, other students studying at any level will be able to access and log into my software to help propel their revision. Teachers will also have accounts to gain a valuable insight into their students timetables and revision habits to help integrate and enforce these techniques in their lessons with their students. It will also help them learn more about their students work ethics, revision websites they might enjoy and the subjects they are struggling with the most. My software could also be used by parents to track their kids’ revision and help keep them up to date.

Evidence of Analysis

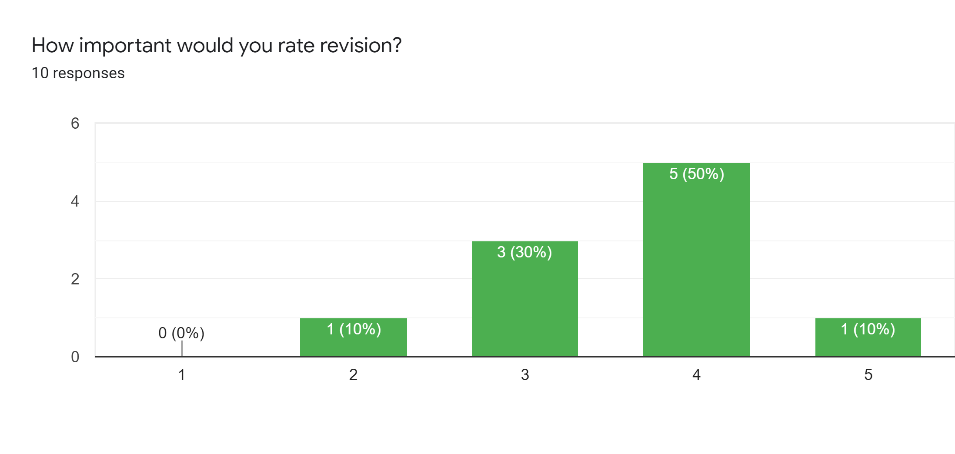
In this section I will begin to look at how my solution will function, what it should achieve and what I can manage to achieve. I will also conduct market research to gain inspiration and insight into what the users of my software will use and want out of it to make sure my solution is optimised and fits the needs of the users.

**Questionnaire**

I created a questionnaire consisting of 10 questions and got a total of 10 answers back to help analyse and outline features revolving around the software.

Time spent (hours)

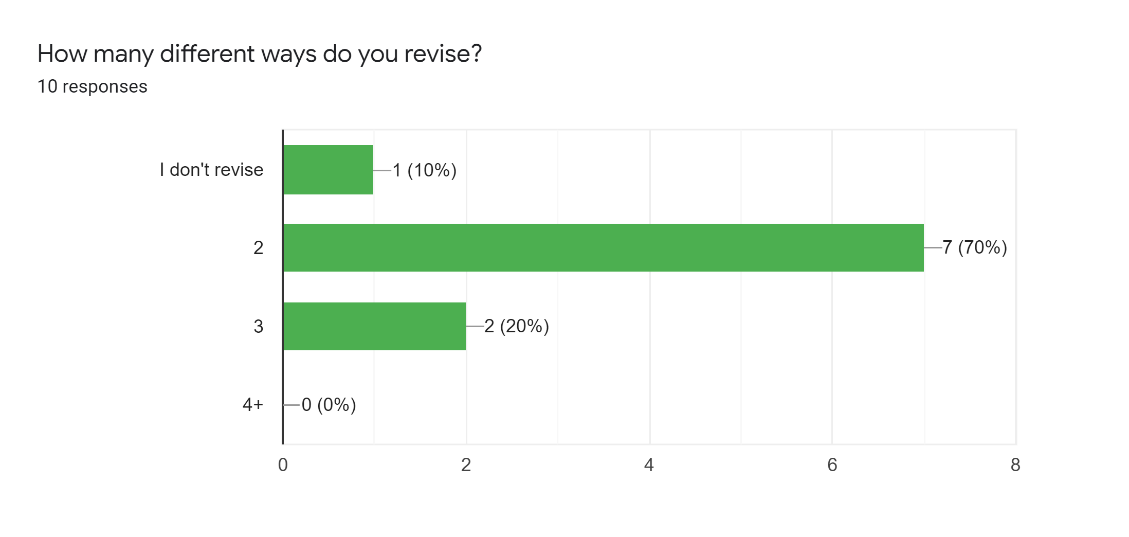
Number of responses

This question helps to illustrate the key aspect of revision in a student’s daily life. With most students revising a total of an hour after completing homework and class set tasks it shows that it’s an active thought in their mind and by creating something to minimise this time whilst still maintaining the effective of it will be very important.

Number of responses

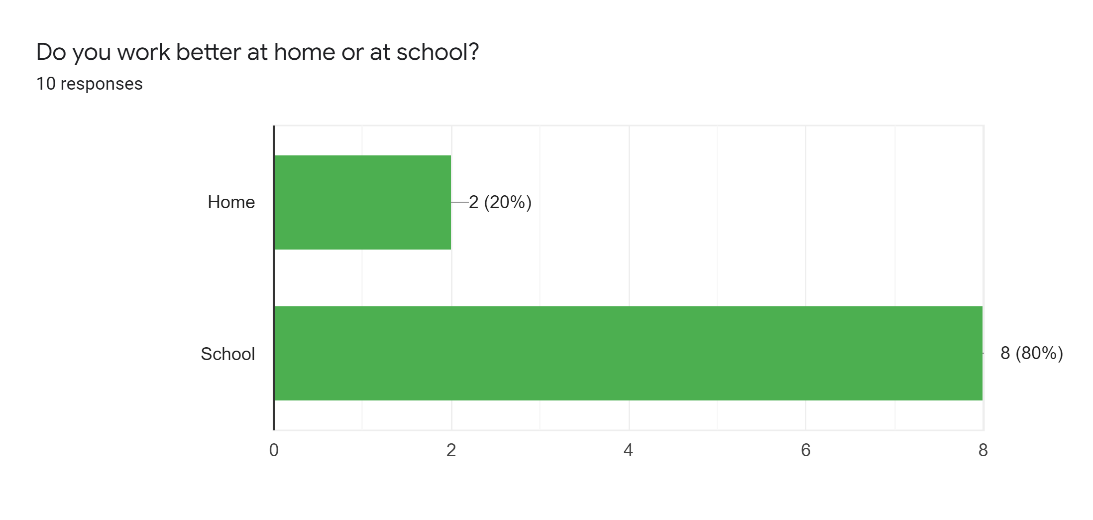
Importance rating

With 5 rated as revision being very important this chart shows how students think revision is important. However, only 1 person rated revision with an importance rating of 5, enforcing a common misconception that IQ is more important. I hope through the use of my software that I can work to fix this misconception by incentivising students to revise more frequently, proving the importance of revision.



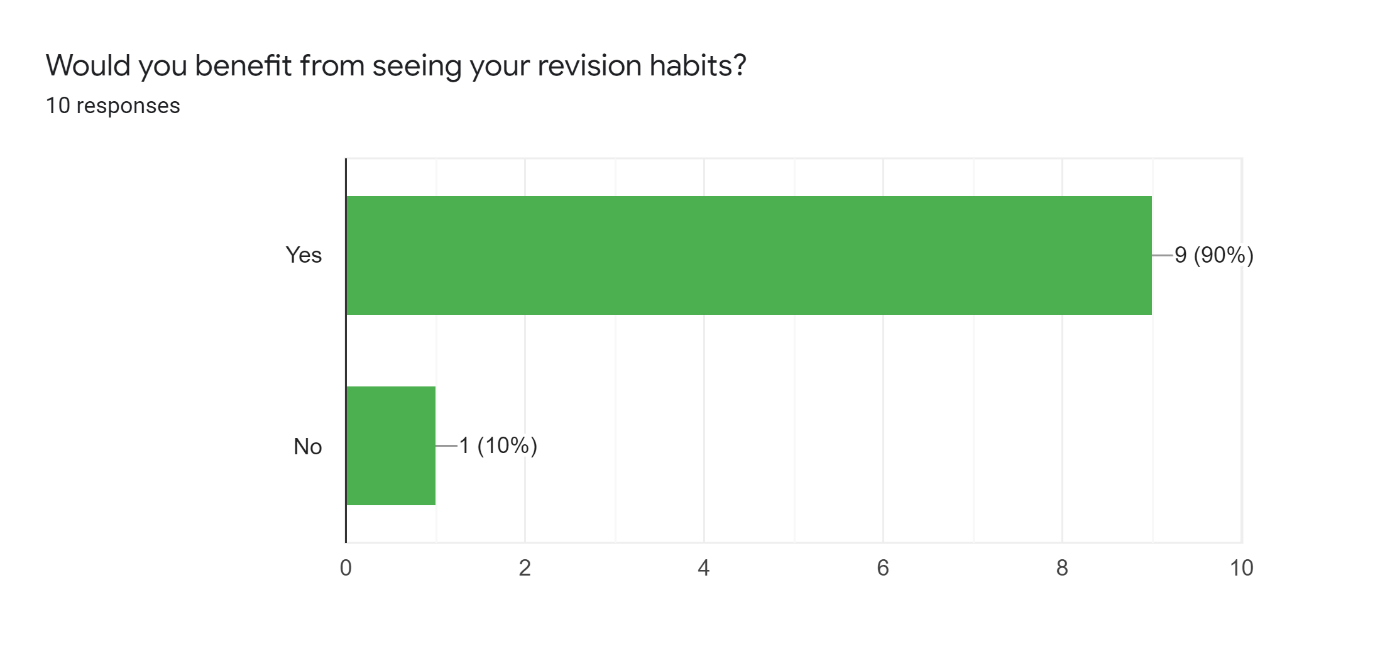
Ways of revising

Number of responses

With the majority of people revising only two ways this shows that they have the opportunity to try out different techniques and compare with classmates. The person who doesn’t revise would be best for my future software as they will be able to gain a clearer understanding of ways to make revision more fun and easier to help incentivise revision.

Number of responses

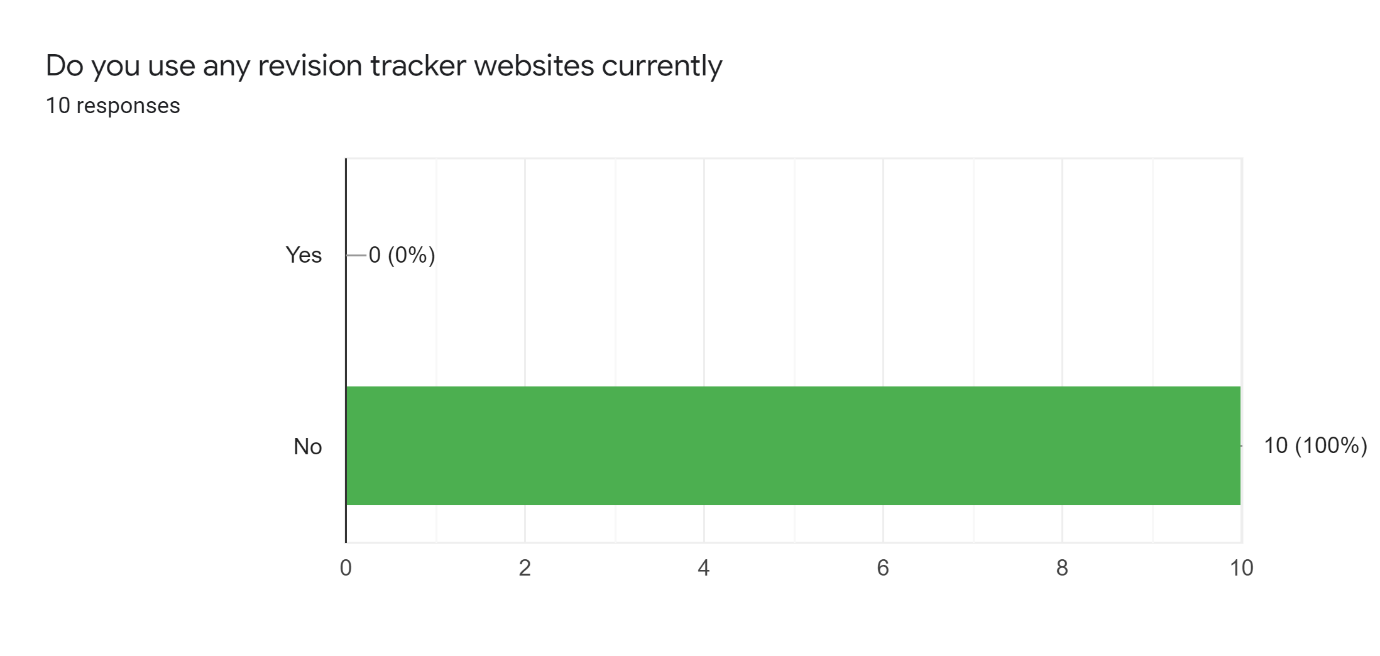
With most people working better at school I think this clearly shows a missed opportunity of making home revision more fun and enjoyable helping to drive people to revise at home. The reason this is so important is that school revision is often faced with obstacles such as opening times and other dependencies which will ultimately affect the student’s revision and could jeopardise important time on the build up to exams.



Amount of responses

Reponses

This is very reassuring and it further proves that students will want to minimise the time spent whilst maximising their gain. In an age where practically everything is recorded and monitored making full use of computers power it seems strange that something so important to almost everyone isn’t digitised in a way that is intuitive and easy.



Amount of responses

Reponses

This is a vast contradiction to the last question where almost all the participants said that they would’ve wanted a revision tracker website. This is a perfect indicator of a market gap and shows that there is a problem that I will address with my software.

Observations

As part of my analysis, I observed an A level student whilst revising for three nights in a week. Firstly, they used a minimal number of revision techniques within this time. The techniques they did use were often inefficient and passive forms of revision, often preferred because they are “easier” and other myths. Despite this they revised in 35-minute sessions accumulating to an hour and a half each day roughly. One thing I did notice was the lack of a timer when completing these sessions, the student often worked until they got bored or worn out. Although sounding like a good idea, continuously revising till exhaustion makes the revision feel longer and can often lead to the student being de-incentivised to revise and making less progress. By finishing before the student is completely bored it gives them a sense of achievement and keeps them fresher when coming into their next block of revision. This technique also minimizes procrastination, another common mistake I observed as the student repeatedly checked their phone and gradually became less and less focused on the revision. From my observations, timing seems to be a big issue that my software will need to address, perhaps showing the average revision block times within the class or giving the user rewards for breaks and blocks. Whilst making the software fun and engaging I have also got to be careful of distracting the user from the data and actually revising, this is because as seen in my observation’s students will avoid revision and procrastinate which will contradict the utility of my software.

Interview with the end-user

**I have interviewed my end user (transcript in appendix (1)). From this interview I learnt that my end user is willing to use a revision tracker software and he would find it useful for his revision. This is a good sign as he models the average student and this shows that there could be demand for my software. Furthermore, it showed me what an average student thinks about revision, highlighting the importance of avoiding procrastination and making the software as easy to use and simple as possible in doing so. Jack also picked up on the use of flashcards which may be something I plan on implementing. Additionally, Jack said “*I try and focus on my own revision*”, this especially shows that he would benefit from seeing other people’s revision habits, gaining inspiration and insight from his peers and hopefully making him a more effective revisor for his studies.**

Log/Diary of research collection

12th June- I began thinking about the concept of the solution and began thinking about how it would work and what features would be involved in creating the solution.

20th June – Settled on a final project idea and started to think more in depth about how my user will interact with the software and what they will want from it.

25th June – I identified an end user, Jack McAteer, and used him as a baseline to tailor my software towards whenever making a decision.

1st July – Put out a questionnaire as a form of market research in order to grasp the problems which my software was to solve and how popular, and feasible it would be. I found that there was definitely interest from students and that the software would need to be easy to use and access as not to take away from revision being the main activity the student does. It needed to be eye catching yet straight to the point and not unnecessarily distracting to help maintain focus for the user.

5th July – I looked at solutions available at the minute and analysed what features they had whilst investigating what they did well and what they were missing. I found there was very few software applications available to students that actively compared grades and showed it in such a motivating and eye-catching way which would incentivise the student to work harder and climb the leader board.

10th July – I interviewed my end user in order to attain an exact idea what he needed to gain from the solution and why he was using it. He told me he wanted to see the development over time and this made me think about showing progression over time and how the user has changed over a certain time frame.

13th July- I began observing an A level student in an attempt to gauge how much revision they undertake and what techniques they use. I was also looking at what could be done better, to get more out their revision for less effort, as well as what they were doing well.

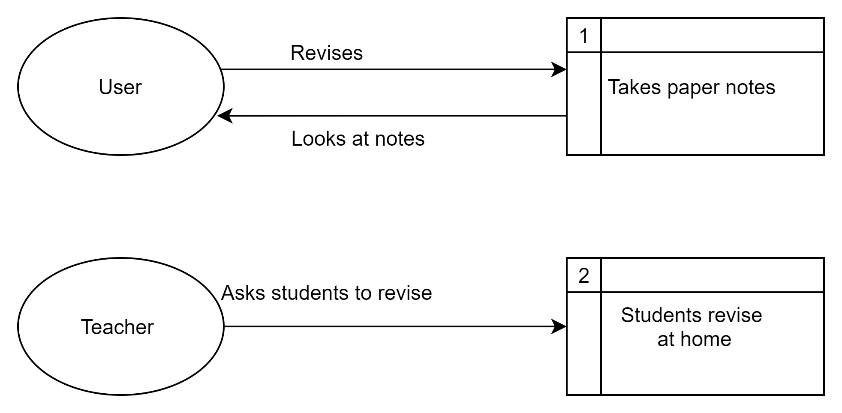
**Analysis**

This is my analysis section of my NEA document where I begin to look more closely at how I will develop the technical solution and how it will work.

Modelling of the problem (initial DFD)

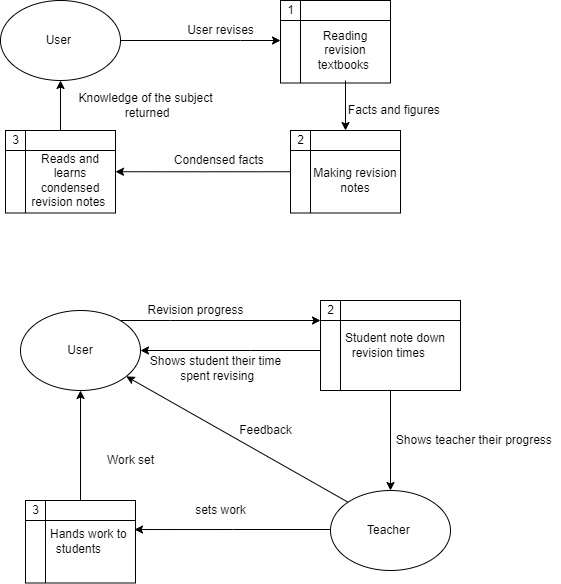
In this section of my analysis, I will begin to look at how students and teachers may interact with my software and begin looking into how data will flow between the components of the software. To help illustrate this I will make use of data flow diagrams (DFD’s). This tool graphically maps out the flow of information within my system whilst also creating a framework for the inputs and outputs within my solution. A DFD has two levels; level 0, also known as a context diagram, shows an abstracted view of the process through the use of one process and its relationship to external entities and level 1, where multiple processes are used as the diagram is decomposed further giving a better idea of how the data flows inside the system.

Current system- Level 0



The first part of the DFD shows the simplicity of how students are told to revise and how they do it.

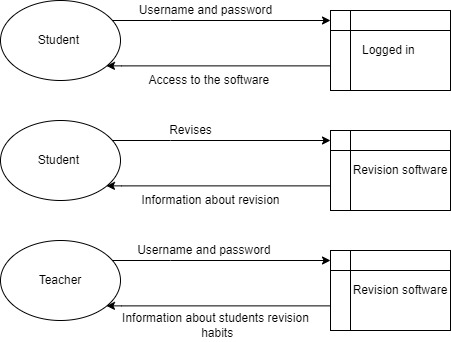
Current system- Level 1

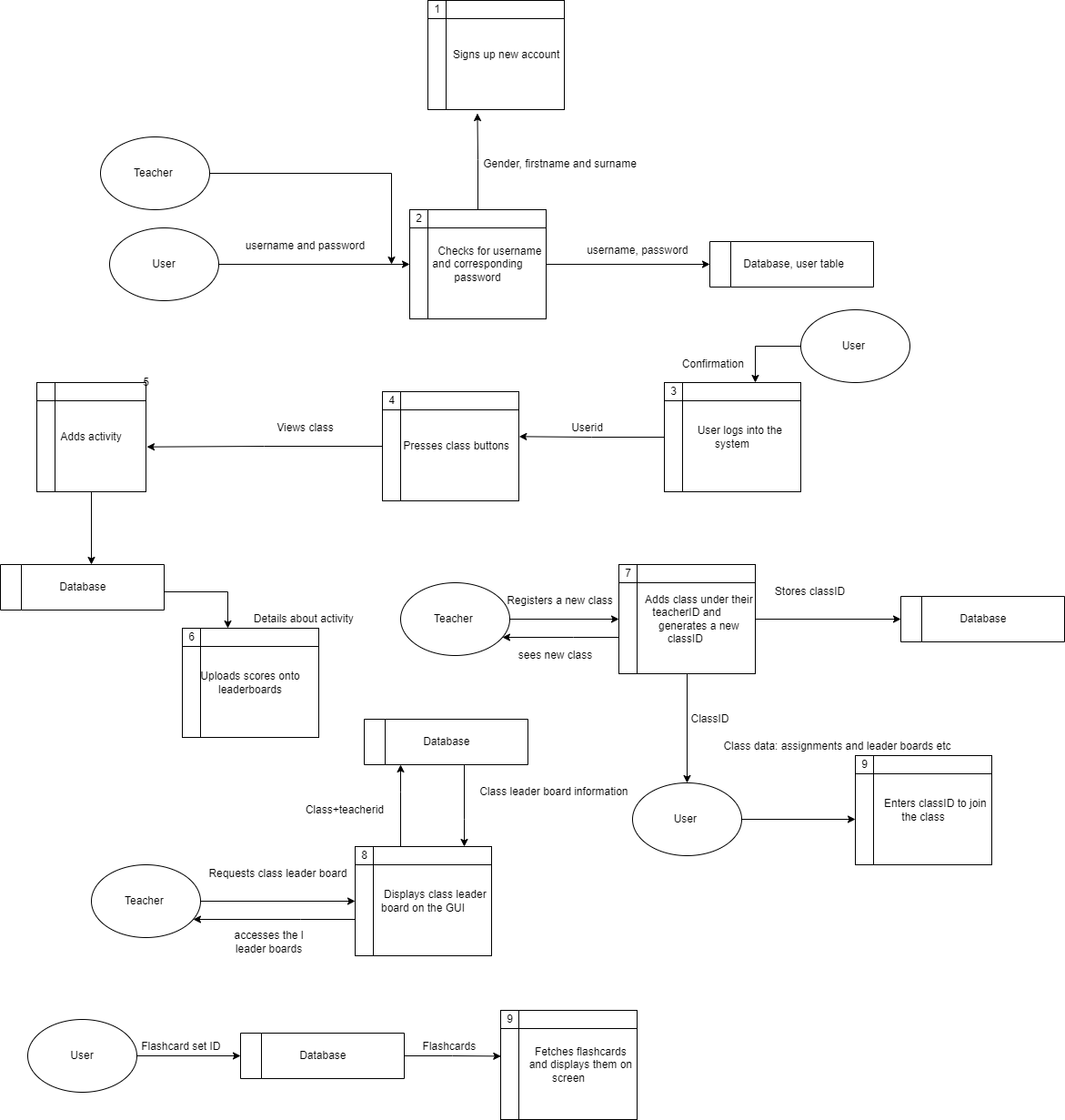


As you can see from the data flow diagram, the current system relies on the teacher being on top of revision and constantly checking up on students. This leaves a lot of space for human error which could cause students to go unnoticed and the teachers to lose track. This issue is especially prominent for teachers teaching many classes involving lots of students. This is something my program will help solve to help not only the students but also the teachers.

Proposed system Level 0

The level 0 data flow diagram for my proposed system helps illustrate, at a basic level, how the users will interact with my proposed software in order to utilise it when working.



Proposed system level 1

The DFD illustrates how my proposed solution will act and transfer the user’s data. It shows that the database is constantly getting refreshed to keep the users’ data safe and secure. My program will be very minimalist in terms of operation as to not distract the user and this is why each function has a specific purpose to help the user. This helps save space on the device and increase its performance speed for a better experience.

Proposed solution details (hardware/software)

My solution will be a local revision software tool created via Python version 3. For my solution to function the user will need access to the internet in order for my solution to run. For the host computer to run the python it will need:

* Modern Operating System:
  + Windows 7 or 10
  + Mac OS X 10.11 or higher, 64-bit
  + Linux: RHEL 6/7, 64-bit (almost all libraries also work in Ubuntu)
* x86 64-bit CPU (Intel / AMD architecture)
* 4 GB RAM
* 5 GB free disk space

Set of objectives

I have created a set of objectives for my solution to meet. These are targets which help outline what my program will do the things I will aim to achieve whilst programming my solution.

1. Make a revision software which is accessible for students and teachers
2. The program must open within 5 seconds and not crash
3. The solution must be easy to navigate with buttons and clear labels
4. The solution must allow for students and teachers to create a unique username and password
5. The solution must implement a degree of security when storing sensitive information e.g. passwords
6. The password must be at least 7 characters long and involve a character
7. The program must be able to save students accounts and progress
8. The solution must present the data in a clear eligible way
9. The solution must be robust and allow for lots of accounts and classes
   1. Teachers will be allowed infinite
   2. Students will be allowed to join up to 3 classes
10. The solution must allow for cross class comparison for teachers
11. The solution must allow users to query and update the database

* For example, users must be able to update account statistics such as username, name and email.

1. The solution must involve a feature allowing students to look at universities
2. The solution must allow teachers to communicate with students within their classes
3. The solution must be robust and have enough error handling to account for user error

* For example, when users encounter an entry box the program must not crash and inform the user of the error whilst keeping running

Acceptable limitations

I have created a set of agreed limitations with my end user to make my solution custom to the user and to make sure they like it and can viably use it on a regular basis and still gain utility from it.

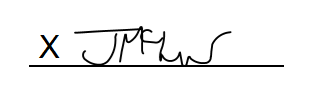
* The solution will not be web based
  + This is as a result of the lack of indication that users need a web-based revision tool, showing how it may become a redundant and over engineered solution which won’t be needed.
* The solution will work on a local database
* The solution will be run through the python module tkinter
* The solution will not be able to be ran on multiple devices simultaneously
* The user’s computer will need the source code downloaded for the program to run
* The user’s computer must be able to support python 3.7

End user confirmation

I have made these limitations as a result of my own programming knowledge as an A level student and the combined knowledge of what I know my end user wants out of my program.

*Signed by end user*

*I have seen and agree with the limitations, objectives and the analysis of the project so far.*



**Design**

In this design section of my coursework I will begin to look out how my program will work, look and its function to the user. I will consider the database and some important algorithms I intend on implementing into my solution as well as the colours and graphics of my coursework which will ultimately dictate how the software will look to the user.

High Level overview

In this section I will choose and describe chosen key algorithms which I plan on using within my solution. This will provide a framework to how I plan to design my solution, what it will do and how I will code it.

**User Registration Algorithm**

* I intend on adding a user registration software to my software for him to securely see his personal statistics and information. This algorithm will ensure that the passwords and usernames are unique to each user and are at a secure standard to ensure that the information is kept safe and protected. I will implement a hashing and salting algorithm to ensure that if the database containing the passwords was targeted the passwords remained encrypted. I will also check the length and character types of all passwords to ensure that the passwords are not easily guessed and are secure. These features help protect my software from brute force attacks and rainbow tables when trying to gain access to my software. I will also hide the password as its being typed in by the user to prevent shoulder surfing to further increase security.

**Email Algorithm**

* In my solution I intend for the teachers to be send emails to their students via any online mailing format to account for students using different websites e.g. gmail, Hotmail etc. This will allow teachers to remotely communicate with the students in safe way, the program will use its own email address acting as the teachers, this is to protect the teacher’s privacy and their email address and create a one-way communication channel from teacher to student.

**Web Scraping Algorithm**

* I intend on implementing a feature into my revision tool which will allow students to search for universities through my software and show relevant universities for their courses and the grades needed to study at these universities. This is a very useful feature as it will allow them to quickly scan through university courses at ease, without leaving the program and needed to roam through websites. The module I will use to help me with this is BeautifulSoup.

**Flashcard Test Algorithm**

* I intend on developing a section of my code that allows students to create and test themselves on their own flashcards. I plan the test to work on a multiple choice-based system where each question will have a right answer and a surrounding three random wrong answers. This will give the students a interactive engaging way for them to revise within my solution which will contribute to their revision time. I plan for the solution to count their score and show it to the user at the end of the test for the student to use as future reference and to see progress.

**Merge Sort Algorithm**

* I plan to implement a merge sort algorithm into my solution. I will use this algorithm when sorting through universities when they are displayed onto the UI. This will allow users to sort the universities presented to them in ascending and descending order. This creates a more efficient experience for the students, allowing them to focus on the universities which will apply to them and their grades. Furthermore, by using a merge sort algorithm it has a relatively fast time complexity of nlogn, making it ideal for searching through lots of universities at high speeds on my software. The merge sort uses a divide and conquer approach to sorting lists, where it breaks down the list into sub lists and sorting them whilst putting the sub lists back together to create a sorted main list.

**Class Ranking Algorithm**

* I wish to include a feature into my software where teachers can rank their classes amongst other classes. This will allow for direct comparisons between teachers and help them to build an idea where their class ranks amongst others, in terms of effort and revision hours. This could also further create a fun and competitive environment not just for the students but also for the teachers, helping to motivate classes. I will make the ranking system based on revision hours rather than grade as to include everyone regardless of their grades. This will mean the competition is fair and could be used across year groups. I plan on using a bubble sort to sort the classes onto the UI, this is because the number of classes will be very low per subject so the time it takes to carry out the program won’t be as important.

Pseudocode

In this section I will create a pseudocode example of how I intend on implementing each of my algorithms. This will act as a tool to show how the algorithm will function and each data structure work whilst helping me plan out a way to code the algorithms individually without having to worry about the complexities of working with programming syntax.

**User Registration Algorithm**

OUTPUT ‘Enter Username’

username <- USERINPUT

OUTPUT ‘Enter Password’

password <- USERINPUT

FUNCTION checkDB:

IF username AND password IS NOT IN database THEN:

OUTPUT ‘Wrong Details’

ENDIF

ELSE THEN:

IF LENGTH(password) GREATER THAN 7 THEN:

NumCount <- False

UpperCount <- False

FOR character IN password:

IF character IS INT THEN:

NumCount <- True

ENDIF

IF character IS UPPERCASE THEN:

UpperCount <- True

ENDIF

IF NumCount == True AND UpperCount ==True THEN:

HashPassword()

ENDIF

ENDIF

ELSE THEN:  
 OUTPUT ‘Password is too weak’

ENDELSE

PROCEDURE HashPassword:

HASH PASSWORD

CREATE salt

password <- password+ salt

STORE password IN database

I have written a pseudocode algorithm for what will be my Login algorithm. Here it is easier to see the iteration and the logic behind the idea, highlighting how the algorithm will run in a terminal.

**Email Algorithm**

OUTPUT ‘Enter header for email’

header <- USERINPUT

OUTPUT ‘Enter message to send via email’

message <- USERINPUT

OUTPUT ‘Select students to receive the email’  
FUNCTION SelectStudents;

OUTPUT ‘Enter student names who will receive your message’

student\_list <- USERINPUT

PROCEDURE send\_email:

port <- 587 // This is the default port for email submission and is compatible with stmp and gmail

smtp\_server <- ‘smtp.gmail.com’

email\_address <- ‘getsmartrevision@gmail.com’

password <- ‘coursework’

student\_list <- SelectStudents

server <- server.connect

html = f"""\

<html>

<head></head>

<body>

<p>{header }<br>

{message }

</p>

</body>

</html>

server.login(email\_address ,password)

for student in student\_list:

server.sendmail(email\_address, student, message)

send\_email

This is my pseudocode for my email system, which demonstrates how I plan to implement the algorithm into my final solution. I plan on using a module to aid me in making this algorithm work and make the program run smoothly. I will also use html code to format the email as that’s the code which is compatible with most email modules.

**Web Scraping Algorithm**

OUTPUT ‘Enter course name:’

course <- USERINPUT

url <- ‘https://www.theuniguide.co.uk/search/course?utf8=%E2%9C%93&c%5Bq%5D='

course\_words <- coursewords.split()

length <- len(coursewords)

if length>= 1 THEN

for x in range (length):  
 IF x== 0 THEN

my\_url <-

‘https://www.theuniguide.co.uk/search/course?utf8=%E2%9C%93&c%5Bq%5D='+words[0]

ENDIF

ELSE THEN:

my\_url <- my\_url+’+’+words[x]

ENDELSE

Soup <= soup(html,’html.parser’)

Universities <= soup.findall(‘universities’)

This is my pseudo code for my web\_scraping algorithm. This will allow me to search a website for a course and add it onto a URL to create a correct URL. The website I intend on using is theuniguide.co.uk, as this website is particularly ideal for web-scraping. From searching the course name, I will then search for key information, iterating through each university, for example: name, location and entry grades. This information I then plan on storing in a table which is easily accessible to the students using this tool.

**Flashcard Test Algorithm**

Flashcards <= FETCH flashcards FROM database WHERE setid == setid

SHUFFLE Flashcards

score<= 0

FUNCTION GetWrongQuestion(TempCards,Flashcards)

WQuestions <= []

FOR X IN RANGE (3):

Index = random.integer(0,len(TempCards))

WQ = TempCards(Index)

TempCards.remove(WQ)

WQuestions.append(WQ)

ENDFOR

RETURN WQuestions

PROCEDURE Question (Flashcards)

Question <= Flashcards [0]

TempCards < = Flashcards.remove(Question)

WQ <= GetWrongQuestion (TempCards,Flashcards)

OUTPUT (WQ,QUESTION)

Answer <= USERINPUT

IF Answer == Question THEN

RETURN 1

ENDIF

OUTPUT “Enter number of questions: ”

number <= USERINPUT

FOR X IN RANGE (number)

S < = Question(Flashcards)

Score <= score + S

ENDFOR

OUTPUT Score

This is my pseudocode for my Testing algorithm. As shown, it fetches each flashcard and shuffles them. This creates a random order of questions to prevent the students from cheating or expecting certain questions and keep the test fresh and new. Furthermore, it generates random wrong answers each time to try and catch the students out and make the test harder. It uses a function and a procedure in order to save code and iterate through each flashcard for each question.

**Merge Sort**

PROCEUDRE merge\_sort(List)

Length <- len(Liist)

If Length > 1 THEN #Base case

mid <= length //2

left <= List[:mid]

right <= List[mid:]

merge\_sort(left) #Recurison is used on both lists to sort and merge the lists

merge\_sort(right)

i <= 0 #these are counters used to traverse all 3 lists, i is for the left list

j <= 0 #j is for the right list

k <= 0 #k is for the main original list

WHILE i <len(left) AND j <len(right):  
 if left[i] < right[j] THEN #Compares two values to sort them

List[k] <= left[i]

i <= i + 1

k <= k + 1

ENDIF

ELSE THEN #If right list value is smaller the right value goes into main list

List[k] <= right[j]

J <= j+1

k <= k+1 #k is incremented as item is added so index increases

ENDELSE

ENDWHILE

#Checks for any values left out

WHILE I <len(left) #used when the right list has been completely sorted

List[k] <= left[i]

i <= i + 1

k <= k + 1

ENDWHILE

WHILE j<len(right) #Used when the left list is sorted

List[k] <= right[j]

J <= j+1

k <= k+1

ENDWHILE

merge\_sort(List)

This is my pseudocode for my merge sort algorithm. As labelled, it starts off with the divide stage and divides the main list into separate lists. I then use a main list pointer, k, and a left, I, and a right, j, pointer. These enable me to traverse the sub-lists when sorting them and creating a sorted list to put back into the main list.

**Class Ranking Algorithm**

PROCEDURE BubbleSort(List):

Length < = len(List)

FOR i in range (0,length-1):

FOR j in range (0,length-1-i):

IF List[j] > List [j+1] THEN

List[j],List[j+1] <= List[j+1], List[j]

ENDIF

ENDFOR

ENDFOR

Teachers <= (SELECT teacherid FROM database WHERE CLASSID == ‘1’)

Unique <= []

FOR teacher in teacherlist:

IF teacher not in Unique THEN

Unique.append(teacher)

ENDIF

ENDFOR

FOR teacher in Unique:

HourList <= (SELECT hours from database WHERE classid== ‘1’ AND teacherid == ‘teacher’)

Total <= 0

FOR hour in hourlist:

Total <= Total + hour

ENDFOR

Class <= (teacherid,totalhours)

ENDFOR

BubbleSort(Class)

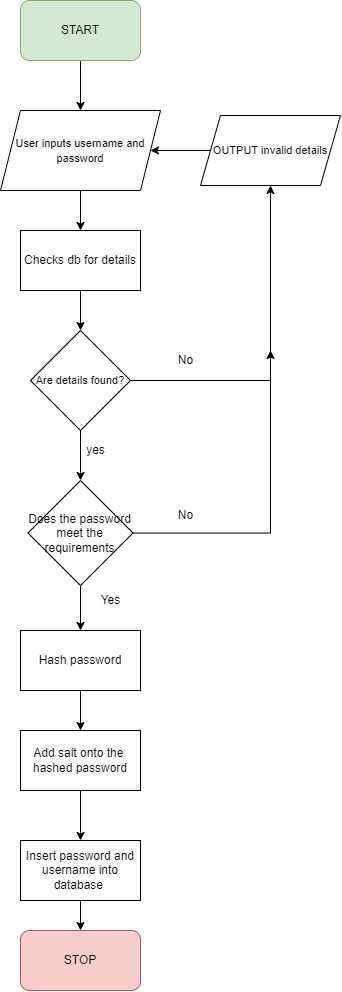
OUTPUT List

This is my class ranking algorithm code. I intend for this to allow teachers to compare class average and total revision times within the program with other classes studying the same subject. In this algorithm I intend to use a bubble sort. Despite this having a lower time efficiency than a merge sort, for the smaller inputs I expect using my program the disadvantage is negatable.

Flowcharts

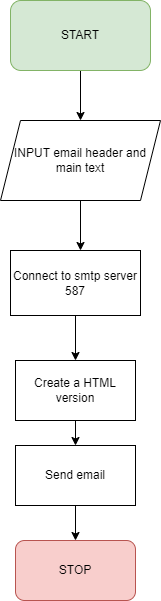
I have created flowcharts to help show my algorithms graphically and the way which I intend them to work by decomposing each process in the form of a flowchart.

**User Registration Algorithm**

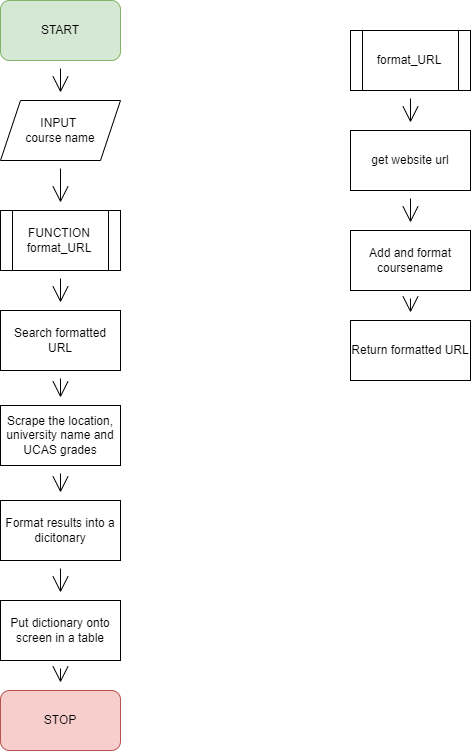
* Here is my flowchart for the login system algorithm, this shows the process each password must go through before being deemed secure and being put into my database. These parameters will help ensure the security of my software is high to protect the users from dictionary-based attacks.

**Email Algorithm**

* I have created a flowchart to show the email process within my software*.* This shows the modules I have incorporated to help create this process and help make the emails run smoother. It also illustrates the steps the algorithm must follow to allow the teacher to send emails to students, including selecting the desired students. This helps for better communication as teachers can help send relevant information to relevant students without disturbing the rest of the class. I have used a gmail server as this is a reliable, popular software which many of the students will already be familiar with.

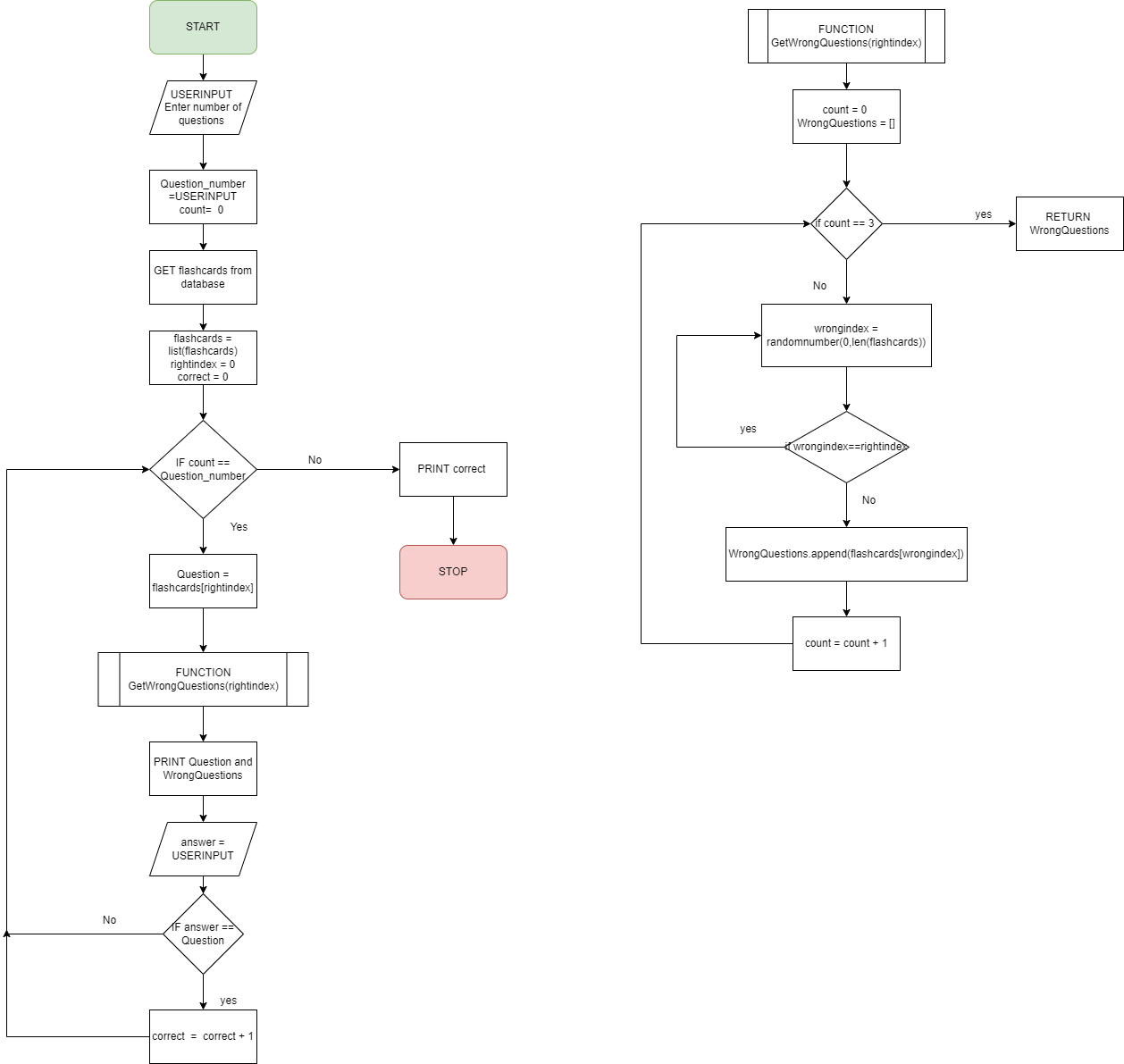


**Web Scraping Algorithm**

* I have created a flowchart to show the steps the algorithm must take to web scrape universities from a website. It shows the user inputting a course name and the program returning information about universities regarding their given course. By making an algorithm compatible with all the courses held in the website by adding the course name parameter it allows for much more freedom when searching for courses and a wider variety of courses available to display. I do this by using a format URL subroutine as showed which is returned and searched for. I will then go on to use the BeautifulSoup module to scrape the html code to retrieve the data.

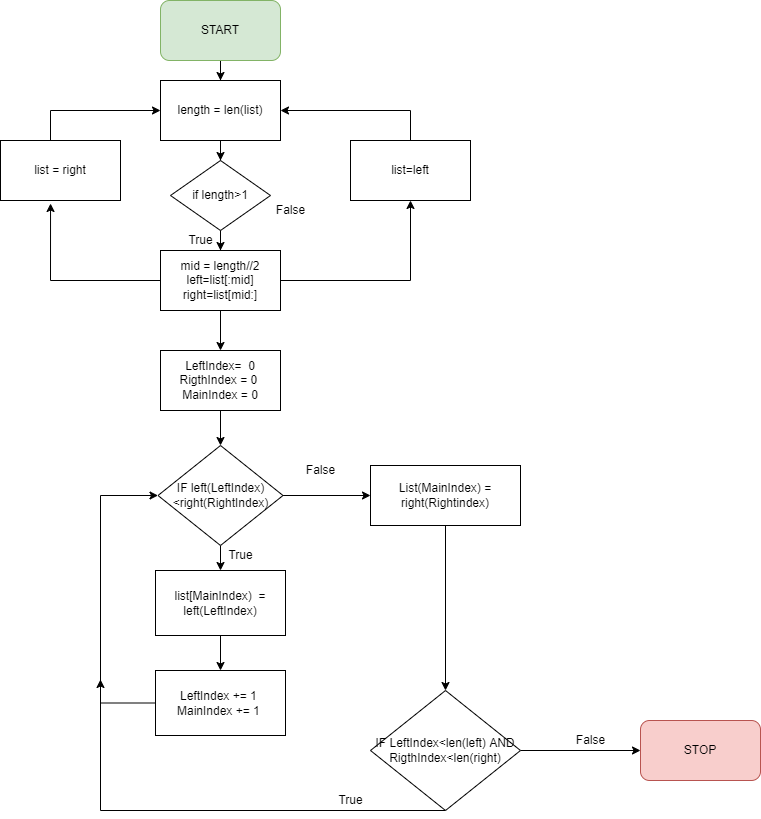
**Flashcard Test Algorithm**

* I have created a flowchart showing how I intend to create a test by using the user created flowcharts*.* It shows how I intend on creating the questions by fetching them from the database and how the user will input their choice and it will count whether the user is right or wrong, which it will use to display their score at the end of the test.

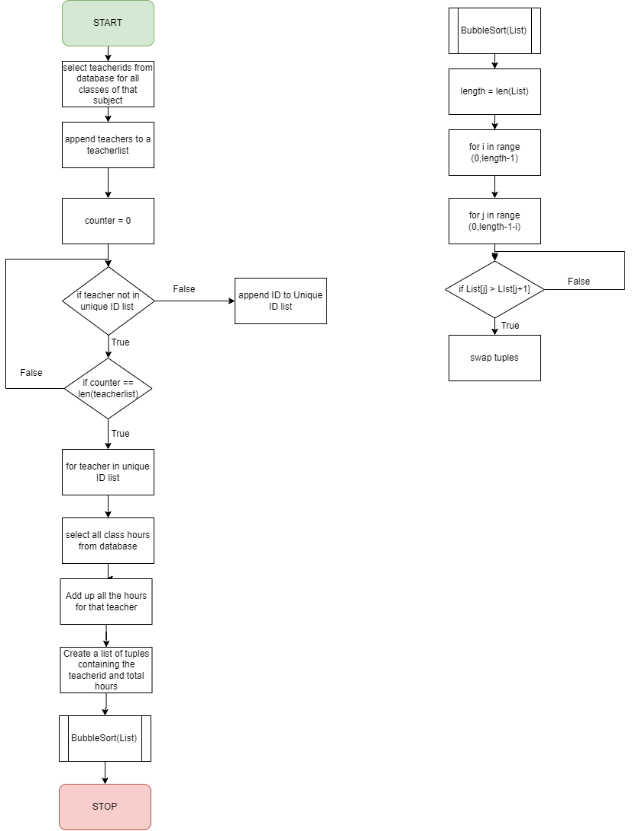


**Merge Sort**

- I have created a flow chart illustrating how my merge sort will function and work. It involves two sections, the division of the list into sub lists and then the merge of the sub lists to form an ordered final list. My flowchart clearly highlights how I have used recursion to save time and make my code more robust and less likely to have bugs.



**Class Ranking Algorithm**

* I have created a flowchart to illustrate the algorithm behind how my program will get and rank the classes in a subject*.* The flowchart shows how I will loop through the student class table in the database selecting all the teacher ID’s where the class ID’s correspond. This will then return me duplicate teachers so I create a new unique teachers list by iterating through the first list. Then by using these teachers IDs I am able to select the students’ total hours by using a parameterised SQL command. Additionally, I will sort the total hours by using a bubble sort algorithm, consisting of 2 for loops. This will return to me the results in an ordered way for which can be put straight onto the UI for the teacher to see.

Database Design

For my program to work I will be using 6 tables to construct a relational database. These tables and their entity descriptions are:

* account (**userid**,username,password,firstname,surname,gender,email)
* class (**classid**,subname)
* teacher (**teacherid**,title,surname,username,password)
* scjoin (***userid***,**classcode** ,*teacherid*,totalhours,averagehours)
* teacherclassjoin (**classcode**, ***teacherid****, classid* ,level)
* stats (***userid***,***classcode*** ,**DATE**,TIME,hours,yearday)
* flashcard (**setid**, heading,text)
* flashcardjoin(**setid**,***userid***,***classcode***,title)

\***Bold** columns indicate primary keys and *italic* columns are used to represent foreign keys.

\*All passwords will also be encrypted when stored into the database using a hashing algorithm.

# 

# Main Database Design

This is the total plan of my entire database I intend to implement. It features 8 normalised tables in a relational model. I intend to use this design as it stores the data in the most space efficient way. Through the use of join tables I have been able to relate students to classes and teachers to classes separately, allowing data to be stored in individual tables to help protect sensitive information and maintain data integrity.

Entity relationship modelling

I will begin to look into how I will use a relational database to store the backend information about my users to use for my program to run and what kind of relationship they will have.

**Student – Class Join Table Relationship**

|  |
| --- |
| account |
| **userid** |
| username |
| password |
| firstname |
| gender |
| email |

|  |
| --- |
| scjoin |
| ***userid*** |
| ***classcode*** |
| *teacherid* |
| totalhours |
| averagehours |

Here there is a one-to-many relationship between students and classes as one student will be in 3 classes. I have userid as a primary key to identify the student users with and have join that with the foreign classcode key to form a composite key to allow me to locate that user with that class.

**Class-Class Join Table Relationship**

|  |
| --- |
| class |
| **classid** |
| subname |

|  |
| --- |
| scjoin |
| ***userid*** |
| ***classcode*** |
| *teacherid* |
| totalhours |
| averagehours |

This shows the relationship between the classes students will be in and the student class joining table. To normalise my database and save storage I gave each class a classcode so I could reuse class names and make it easier to group subjects together despite when they are taught by different teachers. I have used a primary key composed of the two foreign keys userid and classcode for scjoin table which creates a one-to-many relationship with the class table, as there is one class per record in the scjoin.

|  |
| --- |
| scjoin |
| ***userid*** |
| ***classcode*** |
| *teacherid* |
| totalhours |
| averagehours |

**stats-scjoin Relationship**

|  |
| --- |
| stats |
| **userid** |
| **classcode** |
| **DATE** |
| TIME |
| hours |
| yearday |

The stats table will hold all the information about users’ revision times and uploads. This means because the same user with the same class can be in it more than once that I will use time stamps to differentiate between each entry. This will also be useful when sorting and displaying data. I will use these multiple entries and, by use of an algorithm, update the scjoin table with the users’ statistics, creating a many-to-one relationship.

**Flashcard- flashcardjoin Relationship**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  | flashcard |  |  | flashcardcardjoin |
|  | setid |  | | ***setid*** |
|  | heading | userid |
|  | info | classcode |
|  |  |  |  | title |
|  |  |  |  |  |
|  |  |  |  |  |

The flashcard table will not have a primary key. This is because each flashcard will not need to be individually queried and instead, I will only query the flashcard set using the setid. They will have a many to one relationship as each set will have one row in flashcardjoin where it’s title and the class it’s for is stored.

SQL

Here is the sql code of my relational database which I will plan on using to store information in my final solution.

CREATE TABLE account (

userid INTEGER PRIMARY KEY,

username VARCHAR(50) NOT NULL,

password VARCHAR(50) NOT NULL,

firstname VARCHAR(50) NOT NULL

surname VARCHAR(50) NOT NULL,

gender VARCHAR(50),

email VARCHAR(50) )

CREATE TABLE class (

classid INTEGER PRIMARY KEY,

subname TEXT) )

CREATE TABLE teacher (

teacherid INTEGER PRIMARY KEY,

title TEXT,

surname TEXT,

username TEXT,

password TEXT,

username TEXT,

password TEXT) )

CREATE TABLE scjoin (

userid INTEGER,

classid INTEGER,

teacherid INTEGER,

totalhours INTEGER,

averagehours INTEGER,

FOREIGN KEY (teacherid) REFERENCES teacher(teacherid),

FOREIGN KEY (userid) REFERENCES account(userid),

FOREIGN KEY (classid) REFERENCES class(classid),

PRIMARY KEY (userid,classid) )

CREATE TABLE teacherclassjoin (

classid INTEGER,

teacherid INTEGER,

level TEXT,

FOREIGN KEY (teacherid) REFERENCES teacher(teacherid),

FOREIGN KEY (classid) REFERENCES class(classid),

PRIMARY KEY (teacherid,classid) )

CREATE TABLE stats (

userid INTEGER,

classid INTEGER,

DATE TIMESTAMP NOT NULL,

TIME TIMESTAMP NOT NULL,

hours INTEGER,

yearday INTEGER INTEGER NOT NULL,

FOREIGN KEY (userid) REFERENCES account(userid),

FOREIGN KEY (classid) REFERENCES class(classid),

PRIMARY KEY (userid,classid,DATE,TIME) )

CREATE TABLE flashcards(

setid INT,

heading TEXT,

info TEXT)

CREATE TABEL flashcardjoin(

setid INTEGER PRIMARY KEY,

userid INT,

classcode INT,

title VARCHAR(50),

FOREIGN KEY (userid) REFERENCES account (userid),

FOREIGN KEY (classcode) REFERENCES teacherclassjoin (classcode))

*Description of data structures*

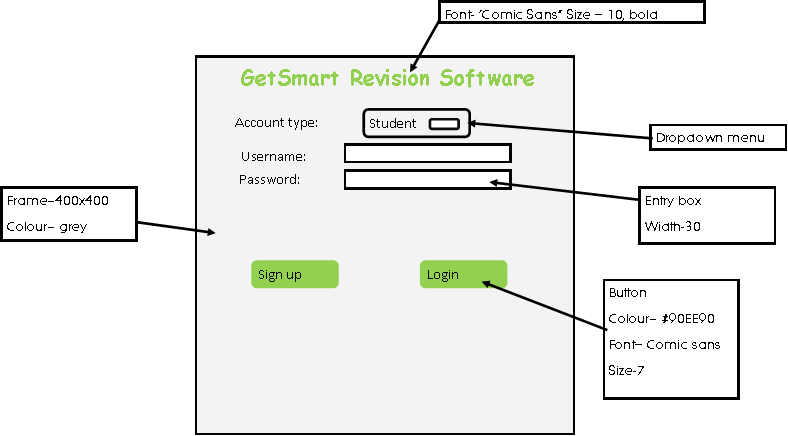
In this section I will look at the different types of data structures I will be using and how they will function inside the program. These will play an important part inside my program and by using the correct data types, it allows for much easier encapsulation, meaning it will be easier for me to work on the program incrementally and stop the program from ending up in any unwanted states.

|  |  |  |  |
| --- | --- | --- | --- |
| Data type name | Data structure | Intended function | Description |
| Universities | Dictionary | Stores a dictionary of universities which have been web scraped off the internet. | By using a dictionary to store universities it allows me to quickly reference certain characteristics, such as name or location. This will essentially be used instead of creating a database for each search which will only increase the time delay and space complexity of the university searching algorithm. |
| ClassArray | Array | Stores an array of indexed objects which are immutable. | If I implement a class array when fetching the classes of a certain teacher or student from a database it allows me to quickly traverse the indexed data and use it in any algorithms I might use. |
| Correct | Boolean | Stores whether the user has answered a flashcard correctly (either true or false) | By creating a Boolean data type, it creates a more efficient algorithm as the Correct variable can only be in two states (True/False). It also means that the variable cannot end up in any unwanted states and will always be one of the two states, helping maintain the program and allowing for easier error handling. |
| Flashcards | Stack | This will store a stack of the flashcards available to the user as the abstract data type stack formed from a list. | A stack is a LIFO structure, meaning the last card into the stack is the first card to be taken out, mimicking a stack of cards in real life, making it ideal for my solution. Furthermore, it allows for a variety of functions such as peek and pop to use the stack correctly. |
| Password | String | This will store the user’s password as a string of characters. | By creating the user’s password using a string it allows me to utilise inbuilt python string manipulation methods such as .upper(). It also is dynamic to allow for a variety of length passwords. |

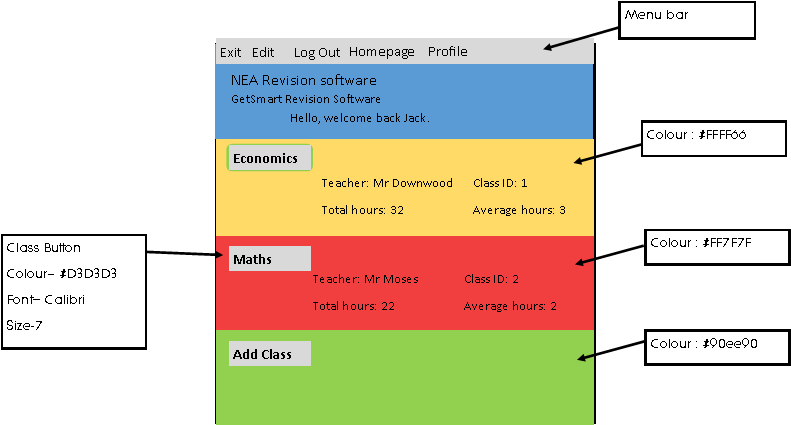
Design of the user interface

For my graphical interface I will be using the module tkinter. This is mainly because of I can use its functionality and wide variety of features to help create my program. I will go for a generally minimalist design as for a revision software the main focus should be on the statistics rather than fancy animations. I will try and stick to primary colours to create a colour scheme which attracts to both genders and students of all ages, increasing engagement in my software from students and teachers.

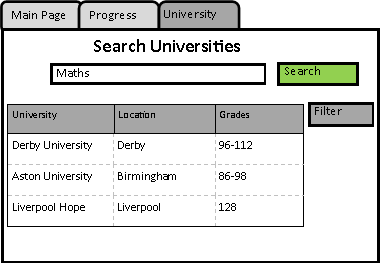
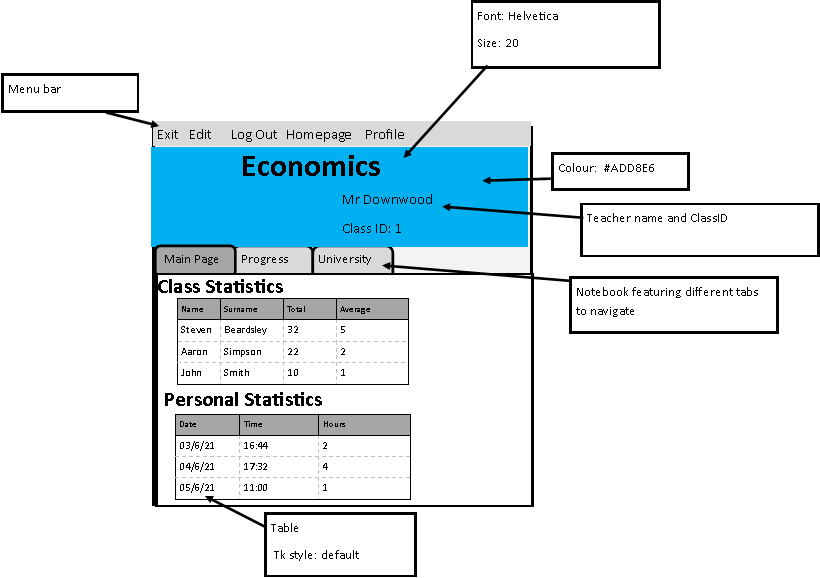
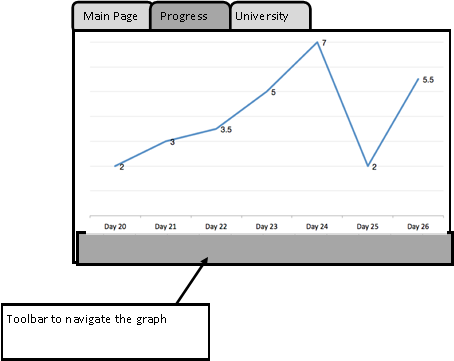
**Login Page**



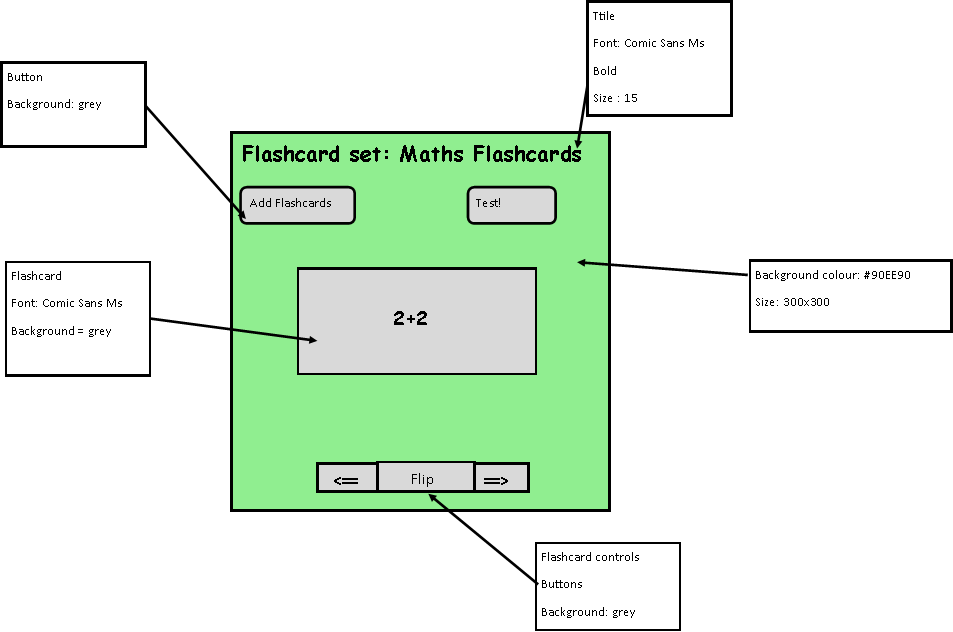
This is my design for my login screen which will greet the teachers and students logging into my software. It will clearly state the name of the software ‘GetSmart’ and uses green to indicate go and make the users want to use my software. I will use buttons to clearly show how to navigate my software and what to do.

**Student home screen**

Here is my mock student homepage. Each user will have capacity for three classes and these will be split up into the three clear sections, neatly differentiating the student’s classes whilst providing a creative and interesting homepage. Under each class section it will feature some brief details about that class at a glance and if no class is registered it will have a add class button for the user to easily add a class into their account. If the user does not have classes assigned it will show a clear ‘Add Class’ button which will help them create and sign themselves up for classes, using their class code.

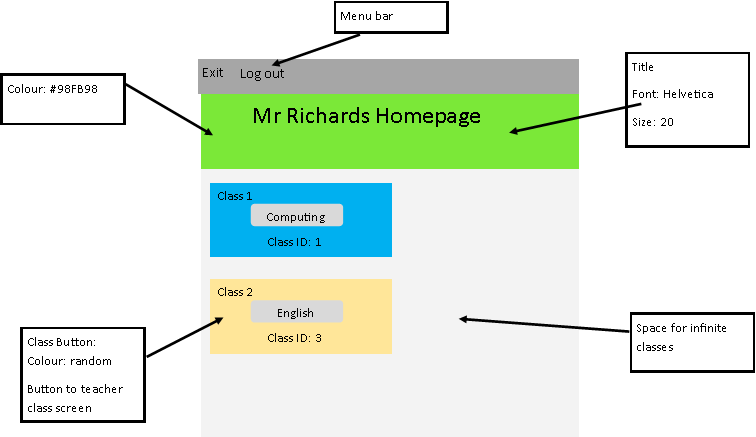
******Student class page**

This page will be very important as this is where the students will spend the most time whilst using the software, browsing and viewing their statistics. I intend to use a clear blue title bar to show the user what class they have selected whilst helping distinguish the classes they are looking at. Featuring two tables of Class and personal statistics, showing the top 3 performers in the class and all the students’ activities they have undertaken. This is important as it shows the activities are being recorded and are there to look back on. I also intend on using a tkinter notebook so the user can quickly swap between the windows and look at the other features.

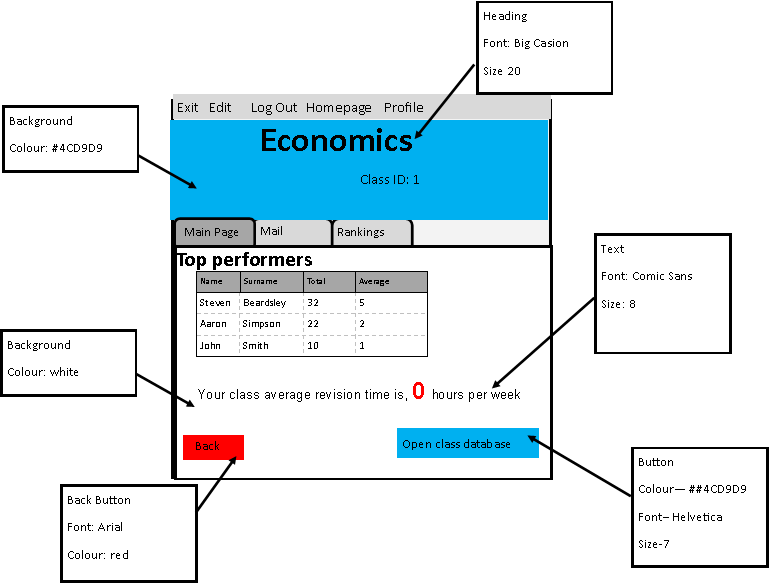
**Flashcard Page**

This page I want to stand out as unlike the other pages, this page had the students doing active revision, in the form of flashcards. The page consists of a bold green background to look lively and interesting whilst also representing the progress the students are making by revising. It will involve a clear display of the flashcard which are displayed one at a time for simplicity towards the student and can easily be traversed via the three easy to reach buttons at the bottom of the screen. Using a fun Comic Sans Ms font to make the page look more interesting and captivating towards the students to allow them to spend longer revising. There are also two clear buttons to add a flashcard and be tested on the flashcard set.

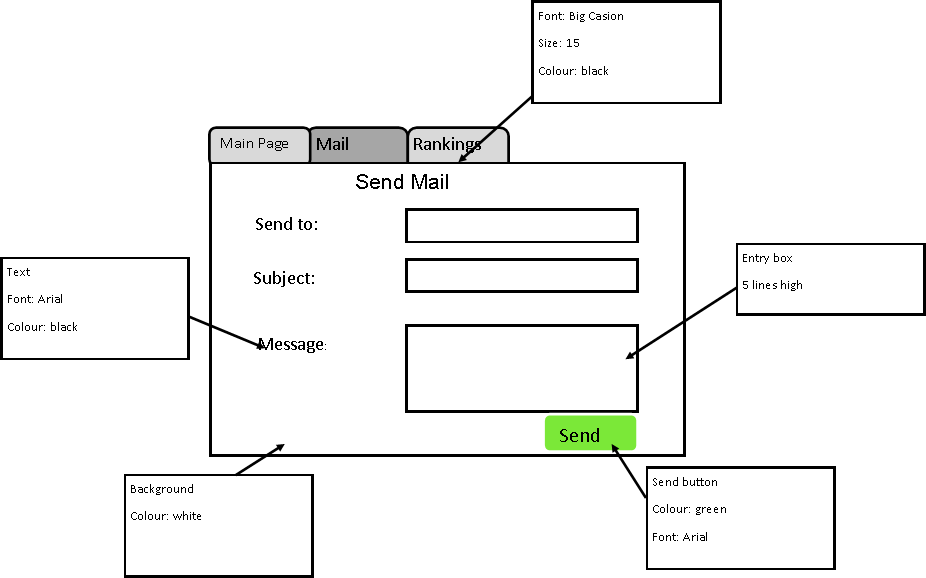
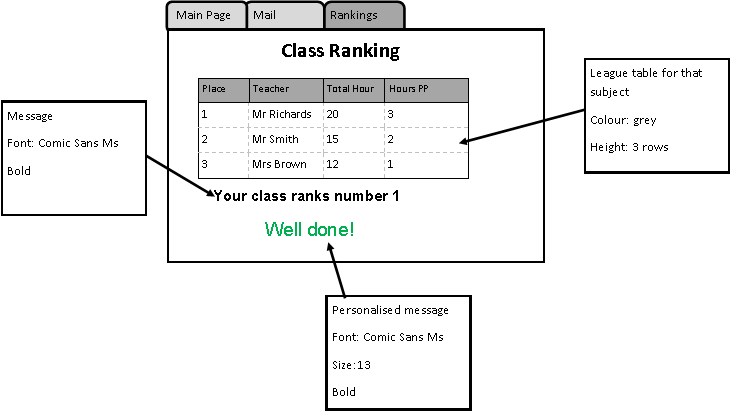
**Teacher Homepage**



The teacher homepage will be varied slightly to the student alternative. The title colour will be a bright green to help signify a teacher account and look professional. It also features the ability to have lots of classes, this is against a student’s cap on 3 classes. This is as teachers may teacher more than 3 classes or teach less than 3 depending on whether they work part time or full time or just the subject they teach. Each class is given its own little square with details at a glance and a button which will lead the teacher onto the class screen. I will keep with the simplistic design and bold headers as I think it keeps the software looking colourful yet maintaining professionalism and not drawing away from the original purpose of to help students revise.

**Teacher Class Page**

For the teacher class page, I intend to keep the look of the interface similar to the student’s class page to increase consistency and simplicity within the different areas of the software so it was easy to navigate as a teacher but also as a student. However, I toned down the colours and went with a white background to give a more professional feel for the teachers to use. It involves 3, easy to traverse windows, which include everything the teachers need to navigate the class and compare between classes. It also features a button to a class database which will show the whole class in one big table for a really in-depth look, especially useful for bigger classes. The aim of the window is to show which students are doing the best to give them recognition but also to show where the class stands in terms of revision time and help the teacher stay on top of the classes progress over time.

**Teacher Class Page-windows**

These are the designs for my intended windows on the class page on the teachers account which I plan on implementing. These windows will allow for quick and simple traversal of different aspects of the class allowing the teachers to see a wide variety of information about their students quickly and easily, optimising their time spent when using my software. Furthermore, by making an easy way to communicate with students without having to close down the software it incentivises communication and makes it simpler for teachers to keep in constant contact with students easily.

System security and integrity of data

In my program I will be storing sensitive information including students and teachers’ passwords. To help protect these passwords from attack I will hash them before storing them into the database. Hashing is a form of one-way encryption used as an alternative to storing the passwords on their own in the database. The hashing algorithm I will use will work by using a complex algorithm to create a unique string of characters. I will be then be able to hash the passwords the users enter into the entry box and check if it matches the hash in the database. By not storing the passwords inside the database if the hashes are hacked then the hacker won’t know what hash I used to hash the passwords and they will be tough to crack. I will also use a salt to encrypt the passwords which will further make the hashed passwords harder to crack. Furthermore, I will be adding password requirements for when users register their accounts. These passwords may involve the passwords being a certain word length and involving numbers. These requirements force the users to create a more unique password which is harder to crack or guess from hackers. This helps protect the system integrity from brute force attacks and rainbow tables.

Example of a hashing algorithm:

Password123 => GlH78&\*Gsdg

This hash will then be stored into the database instead of the password.

In the future, entered passwords will be hashed and compared with the original hash to see

if they create the same result (the password is correct).

**Technical Solution**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330  331  332  333  334  335  336  337  338  339  340  341  342  343  344  345  346  347  348  349  350  351  352  353  354  355  356  357  358  359  360  361  362  363  364  365  366  367  368  369  370  371  372  373  374  375  376  377  378  379  380  381  382  383  384  385  386  387  388  389  390  391  392  393  394  395  396  397  398  399  400  401  402  403  404  405  406  407  408  409  410  411  412  413  414  415  416  417  418  419  420  421  422  423  424  425  426  427  428  429  430  431  432  433  434  435  436  437  438  439  440  441  442  443  444  445  446  447  448  449  450  451  452  453  454  455  456  457  458  459  460  461  462  463  464  465  466  467  468  469  470  471  472  473  474  475  476  477  478  479  480  481  482  483  484  485  486  487  488  489  490  491  492  493  494  495  496  497  498  499  500  501  502  503  504  505  506  507  508  509  510  511  512  513  514  515  516  517  518  519  520  521  522  523  524  525  526  527  528  529  530  531  532  533  534  535  536  537  538  539  540  541  542  543  544  545  546  547  548  549  550  551  552  553  554  555  556  557  558  559  560  561  562  563  564  565  566  567  568  569  570  571  572  573  574  575  576  577  578  579  580  581  582  583  584  585  586  587  588  589  590  591  592  593  594  595  596  597  598  599  600  601  602  603  604  605  606  607  608  609  610  611  612  613  614  615  616  617  618  619  620  621  622  623  624  625  626  627  628  629  630  631  632  633  634  635  636  637  638  639  640  641  642  643  644  645  646  647  648  649  650  651  652  653  654  655  656  657  658  659  660  661  662  663  664  665  666  667  668  669  670  671  672  673  674  675  676  677  678  679  680  681  682  683  684  685  686  687  688  689  690  691  692  693  694  695  696  697  698  699  700  701  702  703  704  705  706  707  708  709  710  711  712  713  714  715  716  717  718  719  720  721  722  723  724  725  726  727  728  729  730  731  732  733  734  735  736  737  738  739  740  741  742  743  744  745  746  747  748  749  750  751  752  753  754  755  756  757  758  759  760  761  762  763  764  765  766  767  768  769  770  771  772  773  774  775  776  777  778  779  780  781  782  783  784  785  786  787  788  789  790  791  792  793  794  795  796  797  798  799  800  801  802  803  804  805  806  807  808  809  810  811  812  813  814  815  816  817  818  819  820  821  822  823  824  825  826  827  828  829  830  831  832  833  834  835  836  837  838  839  840  841  842  843  844  845  846  847  848  849  850  851  852  853  854  855  856  857  858  859  860  861  862  863  864  865  866  867  868  869  870  871  872  873  874  875  876  877  878  879  880  881  882  883  884  885  886  887  888  889  890  891  892  893  894  895  896  897  898  899  900  901  902  903  904  905  906  907  908  909  910  911  912  913  914  915  916  917  918  919  920  921  922  923  924  925  926  927  928  929  930  931  932  933  934  935  936  937  938  939  940  941  942  943  944  945  946  947  948  949  950  951  952  953  954  955  956  957  958  959  960  961  962  963  964  965  966  967  968  969  970  971  972  973  974  975  976  977  978  979  980  981  982  983  984  985  986  987  988  989  990  991  992  993  994  995  996  997  998  999  1000  1001  1002  1003  1004  1005  1006  1007  1008  1009  1010  1011  1012  1013  1014  1015  1016  1017  1018  1019  1020  1021  1022  1023  1024  1025  1026  1027  1028  1029  1030  1031  1032  1033  1034  1035  1036  1037  1038  1039  1040  1041  1042  1043  1044  1045  1046  1047  1048  1049  1050  1051  1052  1053  1054  1055  1056  1057  1058  1059  1060  1061  1062  1063  1064  1065  1066  1067  1068  1069  1070  1071  1072  1073  1074  1075  1076  1077  1078  1079  1080  1081  1082  1083  1084  1085  1086  1087  1088  1089  1090  1091  1092  1093  1094  1095  1096  1097  1098  1099  1100  1101  1102  1103  1104  1105  1106  1107  1108  1109  1110  1111  1112  1113  1114  1115  1116  1117  1118  1119  1120  1121  1122  1123  1124  1125  1126  1127  1128  1129  1130  1131  1132  1133  1134  1135  1136  1137  1138  1139  1140  1141  1142  1143  1144  1145  1146  1147  1148  1149  1150  1151  1152  1153  1154  1155  1156  1157  1158  1159  1160  1161  1162  1163  1164  1165  1166  1167  1168  1169  1170  1171  1172  1173  1174  1175  1176  1177  1178  1179  1180  1181  1182  1183  1184  1185  1186  1187  1188  1189  1190  1191  1192  1193  1194  1195  1196  1197  1198  1199  1200  1201  1202  1203  1204  1205  1206  1207  1208  1209  1210  1211  1212  1213  1214  1215  1216  1217  1218  1219  1220  1221  1222  1223  1224  1225  1226  1227  1228  1229  1230  1231  1232  1233  1234  1235  1236  1237  1238  1239  1240  1241  1242  1243  1244  1245  1246  1247  1248  1249  1250  1251  1252  1253  1254  1255 | **from** **tkinter** **import** \*  **import** **datetime**  **import** **time**  **from** **tkinter** **import** messagebox  **from** **tkinter** **import** ttk  **import** **sqlite3**  **import** **csv**  **import** **random**  **import** **matplotlib**  **from** **matplotlib.figure** **import** Figure  **import** **numpy** **as** **np**  **import** **smtplib**, **ssl**  **import** **matplotlib.pyplot** **as** **plt**  **from** **matplotlib.figure** **import** Figure  **from** **matplotlib.backends.backend\_tkagg** **import** (FigureCanvasTkAgg, NavigationToolbar2Tk)  **from** **email.mime.multipart** **import** MIMEMultipart  **from** **email.mime.text** **import** MIMEText  **from** **bs4** **import** BeautifulSoup **as** soup  **import** **urllib.request**  **from** **urllib.request** **import** urlopen **as** uReq  **import** **os**  **import** **hashlib**  **import** **binascii**  root = Tk()  root.title("Steven Beardsley NEA")  root.geometry("400x400")  conn = sqlite3.connect(r"Steven\_Coursework.db")  conn.execute("PRAGMA foreign\_keys = 1")*#enables foreign keys*  c = conn.cursor()  c.execute("""CREATE TABLE IF NOT EXISTS account (userid INTEGER PRIMARY KEY,  username VARCHAR(50) NOT NULL,  password VARCHAR(50) NOT NULL ,  firstname VARCHAR(50) NOT NULL,  surname VARCHAR(50) NOT NULL,  gender VARCHAR(50),  email VARCHAR(50) )""")  c.execute("""CREATE TABLE IF NOT EXISTS class (classid INTEGER PRIMARY KEY,  subname TEXT)""")  c.execute("""CREATE TABLE IF NOT EXISTS teacher (teacherid INTEGER PRIMARY KEY,  title TEXT,  surname TEXT,  username TEXT,  password TEXT)""")  c.execute("""CREATE TABLE IF NOT EXISTS scjoin (userid INT,  classcode INTEGER,  totalhours INT,  averagehours INT,  FOREIGN KEY (classcode) REFERENCES teacherclassjoin(classcode),  FOREIGN KEY (userid) REFERENCES account(userid),  PRIMARY KEY (userid,classcode))""")  c.execute("""CREATE TABLE IF NOT EXISTS teacherclassjoin (classcode INTEGER PRIMARY KEY,  classid INTEGER,  teacherid INTEGER,  level TEXT,  FOREIGN KEY (teacherid) REFERENCES teacher(teacherid),  FOREIGN KEY (classid) REFERENCES class(classid))""")  c.execute("""CREATE TABLE IF NOT EXISTS stats (userid INT,  classcode INT,  DATE TIMESTAMP not null,  TIME TIMESTAMP not null,  hours INT,  yearday INT not null,  FOREIGN KEY (userid) REFERENCES account(userid),  FOREIGN KEY (classcode) REFERENCES teacherclassjoin(classcode),  PRIMARY KEY (userid,classcode,DATE,TIME))""")  c.execute("""CREATE TABLE IF NOT EXISTS flashcard (setid INT,  heading TEXT,  info TEXT) """)  c.execute("""CREATE TABLE IF NOT EXISTS flashcardjoin ( setid INTEGER PRIMARY KEY ,  userid INT ,  classcode INT,  title VARCHAR(50),  FOREIGN KEY (userid) REFERENCES account (userid),  FOREIGN KEY (classcode) REFERENCES teacherclassjoin (classcode)) """)  **class** **Create\_Login**:*#Login screen*  **def** \_\_init\_\_(self,master,c,conn):  self.master = master  self.c = c  self.conn = conn  self.TitleFrame = Frame(self.master)  self.TitleLabel = Label(self.TitleFrame, text = "GetSmart Revision Tracker",font =("Comic Sans",12,'bold'), foreground = "green")  self.TitleLabel.grid(row=0, column = 2, columnspan= 4,pady=5)  self.MainFrame = Frame(self.master)  self.AccLabel = Label(self.MainFrame, text= "Account type:")  self.AccLabel.grid(row=1,column=1)  self.accountclick = StringVar()  self.accountclick.set("Student")  self.AccountType = OptionMenu(self.MainFrame,self.accountclick,"Student","Teacher")  self.AccountType.grid(row=1,column=2,columnspan=2)  self.UsernameLabel=Label(self.MainFrame,text="Username",background="white")  self.UsernameLabel.grid(row = 2, column = 1)  self.UsernameEntry = Entry(self.MainFrame,width=30)  self.UsernameEntry.grid(row = 2, column = 2, columnspan = 2,pady = 5)  self.PasswordLabel = Label(self.MainFrame,text="Password",background = "white")  self.PasswordLabel.grid(row = 3, column = 0,columnspan = 2, pady= 5)  self.PasswordEntry = Entry(self.MainFrame,width = 30,show='\*')  self.PasswordEntry.grid(row = 3, column = 2, columnspan = 2)  self.EnterButton = Button(self.MainFrame,text = "Login", command = self.Login,bg = "#90EE90")  self.EnterButton.grid(row = 4, column = 2, pady= 15)  self.CreateAccButton = Button(self.MainFrame, text="Sign Up",command = self.SignUp,bg = "#90EE90")  self.CreateAccButton.grid(row = 4, column = 4,pady=20)  self.TitleFrame.pack()  self.MainFrame.pack()  **def** Invalid(self):  self.NW.destroy()  Create\_Login.ClearScreen(self)  Create\_Login(self.master,self.c,self.conn)    **def** WrongDetails(self,message):*#Wrong details popup showing its own error message*  self.NW = Toplevel()  self.NW.geometry("200x120")  self.NW.title('Revision tool')  self.NW.config(bg= "#E00719")  self.ErrorLabel = Label(self.NW,text= message,bg= "#E00719")  self.ErrorLabel.pack(pady=10,padx=10)  self.OkButton = Button(self.NW, text = 'OK', command = self.Invalid)  self.OkButton.pack()  **def** Login(self):  self.username,self.password = self.GetEntry()  *#Check the database*  self.message = "Wrong details"  **if** self.accountclick.get() == 'Student':  self.c.execute("SELECT username FROM account WHERE username == '"+self.username+"'")  **if** self.c.fetchall() == []:  self.WrongDetails('Username or password not found')  **else**: *##username is correct*  self.c.execute("SELECT password FROM account WHERE username == '"+self.username+"'")  StoredPassword = self.c.fetchone()[0]  Verify = EncryptPassword.CheckPassword(self,StoredPassword,self.password)  **if** Verify:  self.c.execute("SELECT userid FROM account WHERE username == '"+self.username+"' AND password == '"+StoredPassword+"'")  self.userid = self.c.fetchone()[0]  MainScreen(self.master,self.c,self.conn,self.userid)  **else**:  self.WrongDetails(self.message)  **if** self.accountclick.get() == 'Teacher':  self.c.execute("SELECT username FROM teacher WHERE username== '"+self.username+"'")  **if** self.c.fetchall() == []:  self.WrongDetails(self.message)  **else**:  self.c.execute("SELECT password FROM teacher WHERE username== '"+self.username+"' ")  StoredPassword = self.c.fetchone()[0]  Verify = EncryptPassword.CheckPassword(self,StoredPassword,self.password)  **if** Verify:  self.c.execute("SELECT teacherid FROM teacher WHERE username == '"+self.username+"' AND password == '"+StoredPassword+"'")  self.teacherid = self.c.fetchone()[0]  l=TeacherMainScreen(self.master,self.c,self.conn,self.teacherid)  **else**:  self.WrongDetails(self.message)  **def** CheckDetails(self,password,username):  *#Check for username and password in database*  Message1 = "Password must be **\n** atleast 7 characters **\n** and involve a number **\n** and a capital letter"  self.c.execute(f"SELECT username FROM account WHERE username== '{username}'")  check = self.c.fetchall()  **if** check != []:*#username is taken already*  self.WrongDetails("Username is taken")  **else**:  **if** len(password) < 7:*#password isnt long enough*  self.WrongDetails(Message1)  NumCount = 0  Upper = **False**  **for** character **in** password:  **if** character.isnumeric() == **True**:  NumCount += 1  **if** character.isupper() == **True**:  Upper = **True**  NumCount=5  Upper=**True**  **def** SignUp(self):  username,password = self.GetEntry()  self.CheckDetails(password,username)*#Checks passwords and username*  **if** self.accountclick.get() == "Student":  self.UserLoginFrame = Frame(self.master)  self.NameLabel = Label(self.UserLoginFrame, text = "First name:")  self.NameLabel.grid(row =5, column = 2,pady=15)  self.SurnameLabel = Label(self.UserLoginFrame,text = "Surname:")  self.SurnameLabel.grid(row=6, column = 2,pady=15)  self.NameEntry = Entry(self.UserLoginFrame, width = 30)  self.NameEntry.grid(row =5, column = 3, columnspan = 2)  self.SurnameEntry = Entry(self.UserLoginFrame,width = 30)  self.SurnameEntry.grid(row = 6, column = 3, columnspan = 2 )  self.GenderLabel =Label(self.UserLoginFrame,text = "Gender:")  self.GenderLabel.grid(row=7,column = 2,pady=15)  self.clicked = StringVar()  self.clicked.set("Male")  self.GenderEntry = OptionMenu(self.UserLoginFrame,self.clicked,"Male","Female","Other")  self.GenderEntry.grid(row=7,column=3,columnspan=2)  self.EmailLabel = Label(self.UserLoginFrame,text=' Address:')  self.EmailLabel.grid(row=8,column=2)  self.EmailEntry = Entry(self.UserLoginFrame, width=20)  self.EmailEntry.grid(row=8,column=3)  self.CreateButton = Button(self.UserLoginFrame, text= "Create", fg = "green",command = self.CreateAccount).grid(row = 8, column = 6,pady=15)  self.UserLoginFrame.pack()  **if** self.accountclick.get() == "Teacher":  self.TeacherCreateAccount()  **def** TeacherCreateAccount(self):  *# self.Frame1 = LabelFrame(self.master, text = "NEA Revision Software ",bg = "#ADD8E6",width = 400,font = 15)*  *#self.Frame1.pack(anchor = N, fill = "x", expand =False)*  self.TeacherFrame = LabelFrame(self.master,text="Teacher Details",bg="#ADD8E6",pady=25,padx=5)  self.TeacherFrame.pack()  **if** self.accountclick.get() == "Student":  **with** self.conn:*#inserts the values into the table*  self.c.executemany('INSERT INTO account (username,password,firstname,surname,gender,email) VALUES (?,?,?,?,?,?)',values)  self.conn.commit()  self.TitleLabel = Label(self.TeacherFrame,text = "Title:",bg="#ADD8E6")  self.TitleLabel.grid(row =0,column=0,pady=15,padx=10)  self.TitleClick = StringVar()  self.TitleClick.set("")  self.TitleMenu = OptionMenu(self.TeacherFrame,self.TitleClick,"Mr","Ms","Dr")  self.TitleMenu.grid(row=0,column=3)  self.SurnameLabel = Label(self.TeacherFrame, text="Surname:",bg="#ADD8E6")  self.SurnameLabel.grid(row=1,column=0,pady=10)  self.SurnameEntry = Entry(self.TeacherFrame)  self.SurnameEntry.grid(row=1,column=2,columnspan=2,pady=10)  self.EnterButton = Button(self.TeacherFrame, text= "Enter", bg='#90EE90',padx=20,command =self.TeacherSignIn)  self.EnterButton.grid(row=3,column=5,padx=20,pady=20)  **def** TeacherSignIn(self):  self.username,self.password = self.GetEntry()  self.surname= self.SurnameEntry.get()  self.title= self.TitleClick.get()  self.password = EncryptPassword.HashPassword(self,self.password)  values = [  (self.title,self.surname,self.username,self.password)  ]  **with** self.conn:*#inserts the values into the table*  self.c.executemany('INSERT INTO teacher (title,surname,username,password) VALUES (?,?,?,?)',values)  self.conn.commit()  self.c.execute("SELECT teacherid FROM teacher WHERE username = '"+self.username+"'")  self.teacherID = self.c.fetchone()  TeacherMainScreen(self.master,self.c,self.conn,self.teacherID[0])  **def** CreateAccount(self):  self.firstname = self.NameEntry.get()  self.username, self.password = self.GetEntry()  self.lastname = self.SurnameEntry.get()  self.gender = self.clicked.get()  self.Email = self.EmailEntry.get()  self.HashPassword = EncryptPassword.HashPassword(self,self.password)  values = [  (self.username,self.HashPassword,self.firstname,self.lastname,self.gender,self.Email)  ]  **if** self.accountclick.get() == "Student":  **with** self.conn:*#inserts the values into the table*  self.c.executemany('INSERT INTO account (username,password,firstname,surname,gender,email) VALUES (?,?,?,?,?,?)',values)  self.conn.commit()  *#GETS USER ID*  self.c.execute("SELECT userid FROM account WHERE username = '"+self.username+"'")  self.userid = self.c.fetchone()[0]  self.ValidateDetails(self.username,self.password)  **def** GetEntry(self):  username = self.UsernameEntry.get()  password = self.PasswordEntry.get()  **return** username, password  **def** ValidateDetails(self,username,password):  self.ClearScreen()  MainScreen(self.master,self.c,self.conn,self.userid)  **def** ClearScreen(self):*#Clears the screen, utility method*  list = self.master.pack\_slaves()  **for** l **in** list:  l.destroy()  grid = self.master.grid\_slaves()  **for** l **in** grid:  l.destroy()  **class** **TeacherMainScreen**(Create\_Login):  **def** \_\_init\_\_(self,master,c,conn,teacherID):  self.Dictionary = {}  super().\_\_init\_\_(master,c,conn)*#instantiates from the previous class*  self.teacherID=teacherID  self.ClearScreen()  self.PrintAccountDetails()  self.TitleFrame = Frame(self.master,bg="#98FB98")  self.TitleFrame.pack(expand= **True**, fill= "both",anchor =N,ipady=20)  self.ClassFrame = Frame(self.master, bg="white")  self.ClassFrame.pack(expand=**True**,fill="both",anchor=N,ipady=90)  self.CheckClass()  self.GetTeacherName()  self.Label1= Label(self.TitleFrame, text=f"{self.FullName} Homepage",font=('Helvetica',20),bg="#98FB98").pack()  self.Menu = Menu(master)  self.master.config(menu =self.Menu)*#configs menu to be used*  self.ExitMenu= Menu(self.Menu)  self.Menu.add\_cascade(label = "Exit", menu =self.ExitMenu)  self.ExitMenu.add\_command(label = "Exit", command = master.quit)  *###checks for classes*  self.ClassMenu = Menu(self.Menu)  self.LogMenu = Menu(self.Menu)  self.Menu.add\_cascade(label = "Class", menu=self.ClassMenu)  self.ClassMenu.add\_cascade(label = "Add Class",command =self.GenerateClass)  self.ClassMenu.add\_cascade(label= "Delete Class", command = self.DeleteClassScreen)  self.Menu.add\_cascade(label='Log Out',menu = self.LogMenu)  self.LogMenu.add\_cascade(label="Logout",command=self.LogOut)  self.ProfileMenu = Menu(self.Menu)  self.Menu.add\_cascade(label='Profile',menu=self.ProfileMenu)  self.ProfileMenu.add\_command(label='Profile',command= **lambda** : Profile(self.master,self.c,self.conn,self.teacherID,'teacher'))  self.HomepageMenu = Menu(self.Menu)  self.Menu.add\_cascade(label='Homepage',menu=self.HomepageMenu)  self.HomepageMenu.add\_command(label='Homepage',command = **lambda**: TeacherMainScreen(self.master,self.c,self.conn,self.teacherID))    *##Class Frame*  **def** DeleteClassScreen(self):  self.top = Toplevel()  self.top.geometry("300x150")  self.top.title('Revision tool')  self.top.config(bg= "#EA5252")  self.Title=Label(self.top,text="Delete Class",font =('Helvetica',12,'bold'),fg="white",bg="#EA5252")  self.Title.grid(row=0,column=0,ipadx=10,pady=5)  self.IDLabel = Label(self.top,text="Enter the classID:",font =('Helvetica',10),fg="black",bg="#EA5252")  self.IDLabel.grid(row=1,column=0,padx=10,pady=10)  self.IDEntry = Entry(self.top,width=10)  self.IDEntry.grid(row=1,column=1)  check=[]  self.DeleteButton = Button(self.top,text="Delete",command = self.DeleteClass)  self.DeleteButton.grid(row=2,column=0)  **def** DeleteClass(self):  classcode = self.IDEntry.get()  *##check they are assigned to the class*  self.c.execute(f"SELECT \* FROM teacherclassjoin WHERE classcode == '{classcode}' AND teacherid == '{self.teacherID}'")  classes = self.c.fetchall()  **if** classes == []:*#class does not exist*  self.Invalid = Label(self.top,text = "Invalid Class Entered",font =('Helvetica',10,"bold"),fg="black",bg="#EA5252")  self.Invalid.grid(row=5,column=0)  **else**:  **with** self.conn:  self.c.execute(f"DELETE FROM scjoin WHERE classcode == '{classcode}'")  self.c.execute(f"DELETE FROM stats WHERE classcode == '{classcode}'")  self.c.execute(f"DELETE FROM teacherclassjoin WHERE classcode =='{classcode}' AND teacherid == '{self.teacherID}'")  self.top.destroy()  Create\_Login.ClearScreen(self)  l =TeacherMainScreen(self.master,self.c,self.conn,self.teacherID)  **def** LogOut(self):  MainScreen.LogOut(self)  **def** GetTeacherName(self):  self.c.execute("SELECT surname FROM teacher WHERE teacherid == '"+self.teacherID+"'")  self.surname = self.c.fetchone()[0]  self.c.execute("SELECT title FROM teacher WHERE teacherid == '"+self.teacherID+"'")  self.title=self.c.fetchone()[0]  self.FullName = self.title + " "+ self.surname  **def** PrintAccountDetails(self):  self.teacherID=str(self.teacherID)  self.c.execute("SELECT \* FROM teacher WHERE teacherid =='"+self.teacherID+"'")  details = self.c.fetchall()  self.c.execute("SELECT \* FROM teacherclassjoin WHERE teacherid == '"+self.teacherID+"'")  AccountDetails = self.c.fetchall()  **def** CheckClass(self):  self.c.execute("SELECT \* FROM teacherclassjoin WHERE teacherid== '"+str(self.teacherID)+"'")  check= self.c.fetchall()  colourwheel = ["#28BFBF","#287BBF","#3D1EC7","#EBBC15","#EB1535","#DE8C98","#8CDEDC","#10E052","#B6F2C9"]  **if** len(check) ==0:*##No classes already*  self.NoClassesLabel = Label(self.ClassFrame, text="You have no current classes",font =('Helvetica',12),bg="white")  self.NoClassesLabel.pack()  **else**:  self.idlist = []  self.buttonlist=[]  **for** x **in** range (len(check)):*##for as many classes as there already are*  self.colour = colourwheel[random.randint(0,len(colourwheel)-1)]  Class = check[x-1]  classid= str(Class[1])  classcode = str(Class[0])  self.idlist.append(classcode)  level = Class[3]  self.c.execute("SELECT subname FROM class WHERE classid == '"+classid+"'")  subname = self.c.fetchone()  self.Frame = LabelFrame (self.ClassFrame, text = f"Class {x+1}",bg=self.colour)  self.Frame.grid(row=x,column=0,ipadx=50,pady=10)  self.buttonlist.append(Button(self.Frame, text = subname,bg="white",command = **lambda** x=x: self.GoToClassScreen(x)))  self.buttonlist[x].pack()  self.Description = Label(self.Frame, text = f"Class Code: {classcode} **\n** Level: {level}",bg=self.colour)  self.Description.pack()  **def** GoToClassScreen(self,buttonid):  button = (self.buttonlist[buttonid])  Create\_Login.ClearScreen(self)  classcode = self.idlist[buttonid]  Class\_Screen(self.master,self.c,self.conn,self.teacherID,self.colour,classcode)  **def** GenerateClass(self):  self.ClearScreen()  self.Frame1 = LabelFrame(self.master,bg="white")  self.Frame1.pack(fill=BOTH,ipady=300)  self.TitleLabel = Label(self.Frame1,text="Create a new class",font=('Helvetica',20),bg="white",fg="green")  self.TitleLabel.grid(row=0,column=0,padx=0,columnspan=12,sticky='NSEW')  self.SubjectLabel = Label(self.Frame1,text="Information:", font=('Helvetica',10),bg='white')  self.SubjectLabel.grid(row=2,column=6,sticky= 'NSEW')  self.SubjectMenu = Listbox(self.Frame1,bd=5)  self.GetClassList()  **for** classes **in** sorted(self.ClassList):*#Inserts classes into the list box alphabetically*  self.SubjectMenu.insert(END,classes)    self.SubjectMenu.grid(row=3,column=6,sticky='NSEW',rowspan=3)  self.CreateClassButton = Button(self.Frame1,text="Add class",command = self.AddClassGUI)  self.CreateClassButton.grid(row=3,column=7)  self.Subclicked = StringVar()  self.Subclicked.set("Academic level")  self.LevelMenu = OptionMenu(self.Frame1,self.Subclicked,"GCSE","A Level")  self.LevelMenu.grid(row=7,column=6,sticky='NSEW')  self.CreateButton = Button(self.Frame1, text= "Create new class", font= ('Helvetica',10), bg="#90EE90", activebackground = "#2d912c", command = self.AddClass)  self.CreateButton.grid(row=7,column=8,columnspan=3,sticky='W',pady=5,padx=50)  **def** InsertClass(self):  subject = self.ClassNameEntry.get()  self.ClassList.append(subject)  self.SubjectMenu.insert(END,subject)  self.ClassNameEntry.delete(0,END)  **def** AddClassGUI(self):  self.ClassNameEntry= Entry(self.Frame1)  self.ClassNameEntry.insert(0,"classname")  self.ClassNameEntry.grid(row=3,column=8,columnspan=2)  self.SubmitButton = Button(self.Frame1, text="Submit",command = self.InsertClass,bg="#63EC0D")  self.SubmitButton.grid(row=3,column=10)  **def** GetClassList(self):  self.c.execute("SELECT subname FROM class")*#Returns classlist*  self.ClassList = self.c.fetchall()  **def** AddClass(self):  self.SubjectName = self.SubjectMenu.get(ANCHOR)  **if** **not** self.SubjectName:  messagebox.showerror('Error','Class must have a name')  self.GenerateClass()  **if** self.SubjectName:  **if** type(self.SubjectName) == tuple:  self.SubjectName = self.SubjectName[0]  self.Level = self.Subclicked.get()  self.c.execute("SELECT classid FROM class WHERE subname=='"+self.SubjectName+"'" )  check = self.c.fetchall()  **if** len(check)== 0:*##Class does not exist already*  values = [  (**None**,self.SubjectName)  ]  self.c.executemany("INSERT INTO class VALUES (?,?)",(values))  self.conn.commit()  self.SubmitClass()  **def** SubmitClass(self):  *###Once the class is in the database*  *##Insert class details into classteacherjoin*  self.c.execute("SELECT classid FROM class WHERE subname == '"+self.SubjectName+"'")  self.ClassID= self.c.fetchone()[0]  values = [  (**None**,self.ClassID,self.teacherID,self.Level)  ]  **with** self.conn:  self.c.executemany("INSERT INTO teacherclassjoin VALUES (?,?,?,?)",values)  self.ClearScreen()  TeacherMainScreen(self.master,self.c,self.conn,self.teacherID)  **def** ClearScreen(self):  Create\_Login.ClearScreen(self)  **class** **Class\_Screen**(TeacherMainScreen):  **def** \_\_init\_\_(self,master,c,conn,teacherID,colour,classcode):*#Createes class sreen and involving windows*  super().\_\_init\_\_(master,c,conn,teacherID)  Create\_Login.ClearScreen(self)  self.colour = colour  self.classcode = classcode  colour = '#4CD9D9'  self.GetDetails()  *##Create the GUI*  Create\_Login.ClearScreen(self)  self.HeadingFrame = Frame(self.master,bg= colour)  self.HeadingFrame.pack(anchor=N,fill="x",expand=**True**,ipady=10)  self.Title = Label(self.HeadingFrame, text = self.Class[1],font =("Big Caslon",20),bg=colour)  self.Title.pack()  self.IDLabel = Label(self.HeadingFrame,text = f'Class ID: {self.classid}',bg=colour)  self.IDLabel.pack()  self.Notebook = ttk.Notebook(self.master)  self.MainFrame = Frame(self.Notebook, bg="white")  self.MainFrame.pack(fill= BOTH,side=BOTTOM,expand =**True**)  *#EmailFrame##################################*  self.EmailFrame = Frame(self.Notebook)  self.Header = Label(self.EmailFrame, text = 'Send Mail',font =("Big Caslon",15))  self.Header.grid(row=0,column=4)  self.SendToLabel = Label(self.EmailFrame, text='Send to:')  self.SendToLabel.grid(row=2,column=1,padx=10)  self.EmailFrame.pack(fill= BOTH,side=BOTTOM,expand =**True**)  self.GetClassList()  self.ClassListFrame =Frame(self.EmailFrame)  self.scrollbar = Scrollbar(self.ClassListFrame,orient = VERTICAL)  self.ClassList = Listbox(self.ClassListFrame,yscrollcommand=self.scrollbar.set,height=5,selectmode=MULTIPLE)  self.scrollbar.config(command=self.ClassList.yview)  **for** Class **in** self.classlist:  self.ClassList.insert(END, Class)  self.scrollbar.pack(side=RIGHT,fill=Y)  self.ClassList.pack()  self.ClassListFrame.grid(row=2,column=2,columnspan=3,pady=5)  self.SubjectLabel = Label(self.EmailFrame,text = 'Subject')  self.SubjectLabel.grid(row=3,column=1,pady=5)  self.Subject = Text(self.EmailFrame,width=20,height=1, font=('bold'))  self.Subject.grid(row=3,column=2,padx=5,columnspan=4)  self.MessageLabel = Label(self.EmailFrame, text='Message')  self.MessageLabel.grid(row=4,column=1,pady=5)  self.Message = Text(self.EmailFrame,width=30,height=5)  self.Message.grid(row=4,column=2,columnspan=4)  self.Send = Button(self.EmailFrame, text ='Send',bg='#90EE90',command =self.GetRecievers)  self.Send.grid(row=5,column=5)  *######################################*  self.Notebook.pack(ipady=200,expand=1,ipadx=200,fill=BOTH)  self.Notebook.add(self.MainFrame,text= "Main Page")  self.Notebook.add(self.EmailFrame,text= "Mail")  *###MainPage*  self.TPLabel = Label(self.MainFrame, text = "Top Performers",bg='white',font =('Helvetica',16))  self.TPLabel.grid(row=0,column=0,rowspan=2)  self.CreateTopPerformers()  self.TP.grid(row=2,column=0,padx=5,columnspan=5)  self.GetAvRev()  self.BackButton = Button(self.MainFrame, text = "Back", command = **lambda**: TeacherMainScreen(self.master,self.c,self.conn,self.teacherID),bg='#F43737')  self.BackButton.grid(row=8,column=0,ipadx=7,ipady=2)  self.ClassScreenButton = Button(self.MainFrame,text="Open Class Database",bg=colour,command = **lambda**: Class\_DBScreen(self.master,self.c,self.conn,self.teacherID,colour,self.classcode) )  self.ClassScreenButton.grid(row=8,column=3,pady=10,ipady=1)  *##Rank Page*  self.RankFrame = Frame(self.Notebook)  self.FillRankFrame()  self.RankFrame.pack(fill= BOTH,side=BOTTOM,expand =**True**)  self.Notebook.add(self.RankFrame,text="Rankings")  **def** FillRankFrame(self):  Header = Label(self.RankFrame, text="Class Ranking",font=('Comic Sans',17,'bold'))  Header.grid(row=0,column=0)  self.c.execute("SELECT teacherid FROM teacherclassjoin WHERE classid =='"+self.classid+"'")  teacherlist = self.c.fetchall()  **if** teacherlist == []:*#No students in class*  Label1 = Label(self.RankFrame, text = "No students are in this class!",font =("Arial",15),fg='red')  Label1.grid(row=4,column=0,padx=10)  **else**:  self.ClassDetails = []  **for** teacher **in** teacherlist:  *#Get teacher name*  self.c.execute("SELECT title, surname FROM teacher WHERE teacherid == '"+str(teacher[0])+"'")  title,surname = self.c.fetchone()  self.c.execute("SELECT classcode FROM teacherclassjoin WHERE classid == '"+self.classid+"' AND teacherid == '"+str(teacher[0])+"'")*#Gets all the Classes*  classlist = self.c.fetchall()  **for** classcode **in** classlist:  classcode = classcode[0]  self.c.execute("SELECT totalhours FROM scjoin WHERE classcode == '"+str(classcode)+"'")  HourList = self.c.fetchall()  TotalHours = 0  **for** hour **in** HourList:  TotalHours += hour[0]  *#Get hours pp*  size = len(HourList)  **if** TotalHours== 0 **or** size == 0 :  Class = (title+ ' '+ surname,0,0,classcode)  **else**:  PPhours = TotalHours// size  Class = (title+ ' '+ surname,TotalHours,PPhours,classcode)  self.ClassDetails.append(Class)  self.Sort()  self.ClassDetails.reverse()  **for** Class **in** self.ClassDetails:  **if** int(Class[3]) == int(self.classcode):  rank = self.ClassDetails.index(Class) + 1  style = ttk.Style()  style.theme\_use('default')  self.RankTableFrame = Frame (self.RankFrame)  self.RankTableFrame.grid(row=2,column=0,padx=5,pady=10)  self.TreeScroll = Scrollbar(self.RankTableFrame)  self.RankTable = ttk.Treeview(self.RankTableFrame,height=3,yscrollcommand=self.TreeScroll.set)  self.TreeScroll.pack(side =RIGHT, fill=Y)  self.RankTable['columns'] = ('Place','Teacher','Total Hours','Hours per person')  self.RankTable.column("#0",width=0,stretch=NO)  self.RankTable.column("Place",anchor=W,width=40)  self.RankTable.column("Teacher",anchor=W,width=80)  self.RankTable.column("Total Hours",anchor=W,width=70)  self.RankTable.column("Hours per person",anchor=CENTER,width=100)  *#Create Headings*  self.RankTable.heading("#0",text='',anchor=W)  self.RankTable.heading('Place',text='Place',anchor=W)  self.RankTable.heading('Teacher',text='Teacher',anchor=W)  self.RankTable.heading('Total Hours',text='Total Hours',anchor=W)  self.RankTable.heading('Hours per person',text = "Hours per person",anchor=W)  self.RankTable.pack()  self.TreeScroll.config(command=self.RankTable.yview)  count = 0  **for** Class **in** self.ClassDetails:  self.RankTable.insert(parent='',index='end',iid=count, values = (count+1,Class[0],Class[1],Class[2]),tags='blue')  count+= 1  self.c.execute("SELECT title,surname FROM teacher WHERE teacherid == '"+self.teacherID+"'")  name = self.c.fetchone()  name = name[0]+name[1]  **for** Class **in** self.ClassDetails:  **if** Class[0] == name:  rank = self.ClassDetails.index(Class) + 1  count += 1  self.font = ('Comic Sans',13,'bold')  message = Label(self.RankFrame, text=f"Your class ranks number {rank}",font = self.font)  message.grid(row=3,column=0)  **if** rank == 1:  label = Label(self.RankFrame,text="Well done!",fg='green',font = self.font)  **if** rank < 4 **and** rank != 1:  label = Label(self.RankFrame,text='On the podium',font=self.font,fg='orange')  **if** rank > 3:  label =Label (self.RankFrame, text='Keep trying',font=self.font,fg='red')  label.grid(row=4,column=0)    **def** Sort(self):  *##Bubble sort*  length= len(self.ClassDetails)  **for** i **in** range (0,length-1):*#iterates thrpugh each element in the list*  **for** j **in** range (0,length-1-i):*#Compares each element with the elements infront*  **if** self.ClassDetails[j][1] >self.ClassDetails[j+1][1]:  self.ClassDetails[j],self.ClassDetails[j+1] = self.ClassDetails[j+1],self.ClassDetails[j]*#Swaps if the number is greater*  **def** GetAvRev(self):*#Calculates the average revision time for a class*  self.c.execute(f"SELECT averagehours FROM scjoin WHERE classcode=='{self.classcode}'")  RevTimes = self.c.fetchall()  **if** RevTimes == []:  self.total = 0  **else**:  self.total = 0  **for** time **in** RevTimes:*#gets mean of revision time*  self.total += time[0]  self.total = self.total// len(RevTimes)  **if** self.total <3:  colour = "red"  **elif** self.total <3 **and** self.total< 5:*#Determines the colour depedant on the mean*  colour = "orange"  **elif** self.total> 4:  colour = "green"  self.RevLabel = Label(self.MainFrame, text='Your class average revision time is, ', bg='white')  self.RevLabel.grid(row=4,column=0)  self.TimeLabel = Label(self.MainFrame, text = f'{self.total}',bg='white',fg=self.colour,font=('Comic Sans MS',15,'bold'))  self.TimeLabel.grid(row=4,column=2)  **if** self.total == 1:  self.Label = Label (self.MainFrame, text = "hour per week",bg='white')  **elif** self.total > 1 **or** self.total == 0:  self.Label = Label (self.MainFrame, text = "hours per week",bg='white')  self.Label.grid(row=4,column=3)  **def** CreateTopPerformers(self):  self.TP = ttk.Treeview(self.MainFrame,height=3)  self.TP['columns'] = ('Name','Surname','Total', 'Average')  self.TP.column("#0",width=0,stretch=NO)  self.TP.column("Name",anchor=W,width=60)  self.TP.column("Surname",anchor=W,width=60)  self.TP.column("Total",anchor=CENTER,width=40)  self.TP.column("Average",anchor=W,width=40)  *#Create Headings*  self.TP.heading("#0",text='',anchor=W)  self.TP.heading('Name',text='Name',anchor=W)  self.TP.heading('Surname',text='Surname',anchor=W)  self.TP.heading('Total',text = "Total",anchor=CENTER)  self.TP.heading("Average",text = 'Average')  self.c.execute(f"SELECT \* FROM scjoin WHERE classcode== '"+self.classcode+"'")  self.classlist = self.c.fetchall()  self.classlist.sort(key=**lambda** tup: tup[3],reverse=**True**)  count = 0  **for** student **in** self.classlist:  self.c.execute(f"SELECT firstname, surname FROM account WHERE userid == '{student[0]}'")  firstname, surname = self.c.fetchone()  self.TP.insert(parent='',index='end',iid=count, values = (firstname,surname,student[2],student[3]),tags='blue')*#Inserts each student into the table*  count += 1  **def** GetRecievers(self):  *#get selected students*  Students = self.ClassList.curselection()  EmailList = []  **if** **not** Students:  messagebox.showerror('Error','No students are selected')*#Error handling if no students are selected*  **if** Students:  **for** Student **in** Students:  user = self.classlist[Student]  userid = user[0]  self.c.execute("SELECT email FROM account WHERE userid == '"+str(userid)+"'")  EmailList.append(self.c.fetchone()[0])  Subject = self.Subject.get('1.0',END)  Message = self.Message.get('1.0',END)  **for** email **in** EmailList:  Send\_Email(self.master,self.c,self.conn,email,Subject,Message)  **def** GetClassList(self):  self.c.execute("SELECT classcode FROM teacherclassjoin WHERE classid =='"+self.classid+"' AND teacherid =='"+self.teacherID+"'")  self.classcode =str( self.c.fetchone()[0])  self.c.execute("SELECT userid FROM scjoin WHERE classcode == '"+self.classcode+"'")  IDlist = self.c.fetchall()  self.classlist=[]  **for** ID **in** IDlist:  self.c.execute("SELECT firstname, surname FROM account WHERE userid == '"+str(ID[0])+"'")  name = self.c.fetchone() + ID  self.classlist.append(name)  **def** GetDetails(self):  self.c.execute("SELECT classid FROM teacherclassjoin WHERE teacherid == '"+self.teacherID+"' AND classcode == '"+self.classcode+"'")  self.classid = str(self.c.fetchone()[0])  self.c.execute("SELECT \* FROM class WHERE classid == '"+self.classid+"'")  self.Class = self.c.fetchone()    **class** **Send\_Email**():  **def** \_\_init\_\_(self,master,c,conn,emaillist,subject,message):  self.master = master  self.c = c  self.conn= conn  self.Emails = emaillist  self.subject= subject  self.text = message  self.port = 587 *# For starttls*  self.smtp\_server = "smtp.gmail.com"  self.context = ssl.create\_default\_context()  self.SenderEmail = 'getsmartrevision@gmail.com'  self.Password = 'coursework'  self.FormatMessage()  self.Send()  **def** FormatMessage(self):  msg = MIMEMultipart('alternative')  msg['Subject'] = self.subject  msg['From'] = self.SenderEmail  *#Creates a plain text and html version of the message*  text = f"{self.subject} \ {self.text}"  html = f"""**\**  <html>  <head></head>  <body>  <p>{self.subject}<br>  {self.text}  </p>  </body>  </html>  """  part1 = MIMEText(text, 'plain')  part2 = MIMEText(html, 'html')  msg.attach(part1)  msg.attach(part2)  self.message = msg.as\_string()  **def** Send(self):  **with** smtplib.SMTP(self.smtp\_server, self.port) **as** server:  **for** reciever **in** self.Emails:  server.starttls(context=self.context)  server.login(self.SenderEmail,self.Password)*#Logs into GetSmart gmail account*  server.sendmail(self.SenderEmail, reciever, self.message)*#Sends email*  **class** **Class\_DBScreen**(Class\_Screen):  **def** \_\_init\_\_(self,master,c,conn,teacherID,colour,classid):  super().\_\_init\_\_(master,c,conn,teacherID,colour,classid)  self.window = Toplevel()  self.colour = '#ADD8E6'  self.window.geometry("700x400")  self.window.title('Class Database')  self.window.config(bg= "#ADD8E6")  self.Title = Label(self.window, text="Class Database", bg= '#ADD8E6',font=("Helvetica",20,'bold'))  self.Title.grid(row=0,column=3)  self.DBFrame = Frame(self.window, bg=self.colour)  self.CreateTable()  **def** CreateTable(self):  style = ttk.Style()  style.theme\_use('default')  self.DB = ttk.Treeview(self.DBFrame,height=12)  self.DB['columns'] = ('ID','Name','Surname','Total', 'Average')  self.DB.column("#0",width=0,stretch=NO)  self.DB.column("ID",width=50,stretch=NO)  self.DB.column("Name",anchor=W,width=100)  self.DB.column("Surname",anchor=W,width=100)  self.DB.column("Total",anchor=CENTER,width=100)  self.DB.column("Average",anchor=W,width=100)  *#Create Headings*  self.DB.heading("#0",text='',anchor=W)  self.DB.heading("ID",text='ID',anchor= CENTER)  self.DB.heading('Name',text='Name',anchor=W)  self.DB.heading('Surname',text='Surname',anchor=W)  self.DB.heading('Total',text = "Total",anchor=CENTER)  self.DB.heading("Average",text = 'Average')  self.DBFrame.grid(row=2,column=0,rowspan=5,columnspan=5)  self.GetID()  self.DetailsFrame = Frame(self.window,bg=self.colour)  self.SearchLabel = Label(self.DetailsFrame, text='Search surname',bg=self.colour)  self.SearchLabel.grid(row=0,column=2)  self.SearchEntry = Entry(self.DetailsFrame,width=20)  self.SearchEntry.grid(row=0,column=3)  self.SearchButton = Button(self.DetailsFrame, text="Search",bg='#ADD8E6',command =**lambda**: self.SearchDB(self.SearchEntry.get()))  self.SearchButton.grid(row=1,column=3)  self.DeleteButton = Button(self.DetailsFrame,text="Delete Student",command = self.DeleteStudent)  self.DeleteButton.grid(row=2,column=3,pady=10)  self.SaveButton = Button(self.DetailsFrame, text='Save table',command= self.SaveTable)  self.SaveButton.grid(row=3,column=3,pady=10)  self.ResetButton = Button(self.DetailsFrame, text= 'Reset',command = self.CreateTable)  self.ResetButton.grid(row=10,column=3)  self.DetailsFrame.grid(row=2,column=5)  **def** DeleteStudent (self):  *#Get selected student*  student = self.DB.item(self.DB.focus())  userid = str(student['values'][0])  *##Delete student from databases*  **with** self.conn:  self.c.execute("DELETE FROM stats WHERE userid == '"+userid+"' AND classcode == '"+self.classid+"'")  self.c.execute("DELETE FROM scjoin WHERE userid == '"+userid+"' AND classcode == '"+self.classid+"'")  self.CreateTable()  **def** FormatName(self):  TeacherMainScreen.GetTeacherName(self)*#Get Teacher Name*  self.c.execute("SELECT subname FROM class WHERE classid == '"+self.classid+"'")  subname = self.c.fetchone()[0]  self.FileName = (f'{self.FullName}\_{subname}.csv')  **def** SaveTable(self):  self.FormatName()  **with** open(self.FileName, "w", newline='') **as** myfile:  csvwriter = csv.writer(myfile, delimiter=',')  **for** row\_id **in** self.DB.get\_children():  row = self.DB.item(row\_id)['values']  csvwriter.writerow(row)  **def** DeleteTreeview(self):  **for** item **in** self.DB.get\_children():  self.DB.delete(item)  **def** SearchDB(self,surname):  self.c.execute("SELECT userid FROM account WHERE surname = '"+surname+"'")*#Searches for student by surname in database*  result = self.c.fetchall()  **if** **not** result:*#Error handling if student is not in class*  self.NoResult = Label(self.DetailsFrame, text= 'No student found')  self.NoResult.grid(row=7,column=3)  self.EnterValues(result)  **def** GetID(self):  self.c.execute("SELECT userid FROM scjoin WHERE classcode == '"+self.classcode+"'")  self.classlist= self.c.fetchall()  self.EnterValues(self.classlist)  **def** EnterValues(self,classlist):  count = 0  self.DeleteTreeview()  **for** student **in** classlist:  student = str(student[0])  self.c.execute("SELECT firstname,surname FROM account WHERE userid == '"+str(student)+"'")  firstname,surname = self.c.fetchone()  self.c.execute(f"SELECT totalhours,averagehours from scjoin WHERE classcode == '{self.classcode}' AND userid == '{student}'")  totalhours, averagehours = self.c.fetchone()  self.DB.tag\_configure('oddrow',background='white')  self.DB.tag\_configure('evenrow',background='#ADD8E6')  **if** count % 2 == 0:  self.DB.insert(parent ='',index='end',iid=count,text='',values= (student,firstname,surname,totalhours,averagehours),tags=('evenrow',))  **else**:  self.DB.insert(parent='',index='end',iid=count,text='', values = (student,firstname,surname,totalhours,averagehours),tags=('oddrow',))  count += 1  self.DB.grid(row=0,column=0,columnspan=5,rowspan=5)  **class** **MainScreen**():*#Homepage*  **def** \_\_init\_\_(self,master,c,conn,userid):  self.master = master  self.userid = str(userid)  self.c = c  self.conn = conn  *#Creates the frames*  Create\_Login.ClearScreen(self)  self.Frame1 = LabelFrame(self.master, text = "NEA Revision Software ",bg = "#ADD8E6",width = 400,font = 15)  self.Frame1.pack(anchor = N, fill = "x", expand =**False**)  self.TitleLabel = Label(self.Frame1, text = "GetSmart revision tracker software",bg = "#ADD8E6")  self.TitleLabel.grid(row =0, column = 0,columnspan=2 )  self.c.execute("SELECT firstname FROM account WHERE userid== '"+self.userid+"'")  name = self.c.fetchone()[0]  self.WelcomeLabel = Label(self.Frame1,text = f"Hello, welcome back {name}",bg = "#ADD8E6")  self.WelcomeLabel.grid(row = 1, column = 1,sticky = W+E)  self.Frame2 = LabelFrame(master,bg="white")  self.Frame2.pack(anchor=W,fill="both",expand=**True**)  self.Class1Frame = Frame(self.Frame2,bg ='#FFFF66',height =100)  self.Class1Frame.pack(anchor=N,fill="both",expand =**True**)  self.Class2Frame = Frame(self.Frame2, bg='#FF7F7F')  self.Class2Frame.pack(anchor=N,fill="both",expand =**True**)  self.Class3Frame = Frame(self.Frame2, bg='#90ee90')  self.Class3Frame.pack(anchor=N,fill="both",expand =**True**)  **for** x **in** range (3):  classlist = [self.Class1Frame,self.Class2Frame,self.Class3Frame]  self.Frame = classlist[x-1]  self.CheckClass()  self.Menu = Menu(master)  self.master.config(menu =self.Menu)*#configs menu to be used*  self.ExitMenu= Menu(self.Menu)  self.Menu.add\_cascade(label = "Exit", menu =self.ExitMenu)  self.ExitMenu.add\_command(label = "Exit", command = master.quit)  self.EditMenu = Menu(self.Menu)  self.Menu.add\_cascade(label = "Edit", menu=self.EditMenu)  self.EditMenu.add\_cascade(label = "Upload activity", command = self.upload\_activity)  self.EditMenu.add\_cascade(label= 'Remove Class',command = self.RemoveClassGUI)  self.LogOutMenu = Menu(self.Menu)  self.Menu.add\_cascade(label = "Log Out",menu=self.LogOutMenu)  self.LogOutMenu.add\_cascade(label = "Log Out",command = self.LogOut)  self.HomepageMenu = Menu(self.Menu)  self.Menu.add\_cascade(label='Homepage',menu=self.HomepageMenu)  self.HomepageMenu.add\_command(label='Homepage',command = **lambda**: MainScreen(self.master,self.c,self.conn,self.userid))  self.ProfileMenu = Menu(self.Menu)  self.Menu.add\_cascade(label='Profile',menu=self.ProfileMenu)  self.ProfileMenu.add\_command(label='Profile',command= **lambda** : Profile(self.master,self.c,self.conn,self.userid,'student'))  **def** RemoveClassGUI(self):  self.top = Toplevel()  self.top.geometry("150x150")  self.top.title('Revision tool')  self.top.config(bg= "#ADD8E6")  self.title = Label(self.top, text = "Delete Class",font =("Helletica",10,"bold"),bg="#ADD8E6")  self.title.grid(row=0,column=0,columnspan=2)  self.ClassIDLabel=Label(self.top,text='Class ID:',bg="#ADD8E6")  self.ClassIDLabel.grid(row=1,column=0)  self.ClassIDEntry = Entry(self.top,width =8,)  self.ClassIDEntry.grid(row=1,column=2)  self.ClassIDEntry.insert(0,"000")  self.EnterButton = Button(self.top,text="Enter", bg ="#ADD8E6",command =self.RemoveClass)  self.EnterButton.grid(row=4,column=1,pady=10)  **def** RemoveClass(self):  classcode = self.ClassIDEntry.get()  *##check they are assigned to the class*  self.c.execute(f"SELECT \* FROM scjoin WHERE classcode == '{classcode}' AND userid == '{self.userid}'")  classes = self.c.fetchall()  **if** classes == []:*#class does not exist*  self.Invalid = Label(self.top,text = "Invalid Class Entered",font =('Helvetica',10,"bold"),fg="black",bg="#EA5252")  self.Invalid.grid(row=5,column=0)  **else**:  **with** self.conn:  self.c.execute(f"DELETE FROM scjoin WHERE classcode =='{classcode}' AND userid == '{self.userid}'")  self.c.execute(f"DELETE FROM stats WHERE userid == '{self.userid}' AND classcode == '{classcode}'")  self.top.destroy()  Create\_Login.ClearScreen(self)  MainScreen(self.master,self.c,self.conn,self.userid)  **def** CheckClass(self):*#Checks for any classes the user has*  self.c.execute("SELECT classcode FROM scjoin WHERE userid == '"+self.userid+"' ")  colours = ['#FFFF66','#FF7F7F','#90ee90']  classlist = self.c.fetchall()  self.ButtonList= []  **for** x **in** range (3):*#loops througb the 3 classes popping each class which is done*  FrameList = [self.Class1Frame,self.Class2Frame,self.Class3Frame]  self.Frame = FrameList[x]  **if** classlist == []:*#No classes*  self.AddClassButton = Button(self.Frame, text = "Add Class", command = self.AddClass)  self.AddClassButton.grid(row=0,column=0)  **else**:  classcode = str(classlist[0][0])  classid = self.GetClassID(classcode)  self.c.execute("SELECT subname FROM class WHERE classid == '"+str(classid)+"'")  subname = self.c.fetchone()  self.ButtonList.append(Button (self.Frame, text = subname,command = **lambda** x=x: self.GoToUserClassScreen(x),padx=5))  self.ButtonList[x].grid(row=0,column=0,pady=5,padx=5)  self.c.execute(f"SELECT totalhours, averagehours FROM scjoin WHERE classcode == '{classcode}' AND userid == '{self.userid}'")  info = self.c.fetchone()  self.TeacherInfo(classcode)  details = Label(self.Frame,text=f"Teacher: {self.teachername} Class ID: {classcode} **\n** Total Hours: {info[0]} Average Hours: {info[1]}",bg=colours[x],font = ('Hellevetica',10))  details.grid(row=2,column=2,pady=5)  classlist.remove(classlist[0])  **def** TeacherInfo(self,classcode):  self.c.execute("SELECT teacherid FROM teacherclassjoin WHERE classcode == '"+classcode+"'")  self.teacherid = str(self.c.fetchone()[0])  self.c.execute("SELECT title,surname FROM teacher WHERE teacherid == '"+self.teacherid+"'")  teachername = self.c.fetchone()  self.teachername = teachername[0]+' '+ teachername[1]  **def** GetClassID(self,classcode):  self.c.execute("SELECT classid FROM teacherclassjoin WHERE classcode == '"+classcode+"'")  **return** str (self.c.fetchone()[0])  **def** GoToUserClassScreen(self,buttonid):  self.c.execute("SELECT classcode FROM scjoin WHERE userid == '"+self.userid+"' ")  classlist = self.c.fetchall()  Create\_Login.ClearScreen(self)  button = self.ButtonList[buttonid]  classcode = classlist[buttonid]  User\_ClassScreen(self.master,self.c,self.conn,self.userid,classcode)  **def** UpdateClass(self):*##Adds class to the scjoin table*  self.classcode = self.ClassIDEntry.get()  *#Get classID*  self.c.execute("SELECT classid FROM teacherclassjoin WHERE classcode == '"+self.classcode+"'")  classid = self.c.fetchone()  **if** **not** classid:  messagebox.showerror("Error","Invalid class ID")  self.AddClass()  **if** classid:  classid = str(classid[0])  self.c.execute("SELECT subname FROM class WHERE classid == '"+classid+"'")  SubName = self.c.fetchall()  **if** SubName == []: *#class does not exist*  self.ErrorMessage()  **else**:*##class exists*  *#Check if the users already in the class*  self.c.execute(f"SELECT \* FROM scjoin WHERE userid == '{self.userid}' AND classcode == '{self.classcode}'")  check = self.c.fetchone()  **if** **not** check:  SubName = SubName[0]  self.c.execute ("SELECT teacherid FROM teacherclassjoin WHERE classid == '"+classid+"'")  teacherid = self.c.fetchone()[0]  values = [self.userid,self.classcode,0,0]  self.conn.commit()  **with** self.conn:  self.c.execute("INSERT INTO scjoin VALUES (?,?,?,?)",values)*#Inserts a new class into scjoin*  self.top.destroy()  MainScreen(self.master,self.c,self.conn,self.userid)  **if** check:  messagebox.showerror('Error','Class is already registered to this account')  **def** AddClass(self):*##Add class window*  self.top = Toplevel()  self.top.geometry("150x150")  self.top.title('Revision tool')  self.top.config(bg= "#ADD8E6")  self.title = Label(self.top, text = "Create Class",font =("Helletica",10,"bold"),bg="#ADD8E6")  self.title.grid(row=0,column=0,columnspan=2)  self.ClassIDLabel=Label(self.top,text='Class ID:',bg="#ADD8E6")  self.ClassIDLabel.grid(row=1,column=0)  self.ClassIDEntry = Entry(self.top,width =8)  self.ClassIDEntry.grid(row=1,column=2)  self.ClassIDEntry.insert(0,"000")  self.EnterButton = Button(self.top,text="Enter", bg ="#ADD8E6",command =self.UpdateClass)  self.EnterButton.grid(row=4,column=1,pady=10)  **def** ErrorMessage(self):  self.NW = Toplevel()  self.NW.geometry("150x100")  self.NW.title('Revision tool')  self.NW.config(bg= "#E00719")  self.ErrorLabel = Label(self.NW,text= "Invalid class code",font=('Helvetica',8,'bold'),bg="#E00719",fg="white")  self.ErrorLabel.pack(pady=10,padx=10)  self.top.destroy()  self.OkButton = Button(self.NW, text = 'OK', command= **lambda**:self.NW.destroy())  self.OkButton.pack()  **def** upload\_activity(self):  Upload\_Activity(self.master,self.c,self.conn,self.userid)  **def** LogOut(self):  Create\_Login.ClearScreen(self)  Create\_Login(self.master,self.c,self.conn)  **class** **Upload\_Activity**():  **def** \_\_init\_\_(self,master,c,conn,userid):  self.master = master  self.c = c  self.conn = conn  self.userid=userid  Create\_Login.ClearScreen(self)  self.TitleLabel = Label(self.master,text = "Upload Activity",font=('Comic Sans',15))  self.TitleLabel.grid(row=0,column=0,columnspan=2)*#*  self.GetClasses()  **if** self.classlist == []:  self.NoClass = Label(self.master, text='You have no current classes',font= ('Hellvetica',12),fg = 'red')  self.NoClass.grid(row=2,column=1,pady=10)  **else**:  self.classlabel = Label(self.master,text = "Class:",font=("Comic Sans",12))  self.classlabel.grid(row=1,column=0)  self.GetClasses()  self.clicked = StringVar()  self.clicked.set(self.classlist[0])  self.ClassMenu = OptionMenu(self.master,self.clicked, \*self.classlist)  self.ClassMenu.grid(row=1,column=1,pady=5)  self.TimeSpent = Label(self.master,text= "Time Spent (Hours)",font=('Comic Sans',12))  self.TimeSpent.grid(row=2,column=0)  self.hoursEntry = Spinbox(self.master,from\_=0,to=10,font=("Comic Sans",12))  self.hoursEntry.grid(row=2,column=1,pady=5)  self.ManualEntry = Button(self.master,text="Create Timer", command = self.Timer,bg="#90EE90")  self.ManualEntry.grid(row=3,column=0)  self.Submit = Button(self.master,text="Submit",bg="#90EE90",command = **lambda**: self.SubmitActivity(self.hoursEntry.get()))  self.Submit.grid(row=3,column=1,ipadx=5,pady=5)  **def** Timer(self):  self.hour=StringVar()  self.minute=StringVar()  self.second=StringVar()  self.hour.set("00")  self.minute.set("00")  self.second.set("00")  self.TimeFrame = Frame (self.master)  self.TimeFrame.grid(row=4,column=1,pady=20)  HourEntry= Entry(self.TimeFrame, width=3, font=("Helvetica",18,""),textvariable=self.hour)  HourEntry.grid(row=0,column=0)  MinuteEntry = Entry(self.TimeFrame, width=3, font=("Helvetica",18,""),textvariable=self.minute)  MinuteEntry.grid(row=0,column=1)  SecondEntry = Entry(self.TimeFrame, width=3, font=("Helvetica",18,""),textvariable=self.second)  SecondEntry.grid(row=0,column=2)  TimerButton = Button(self.TimeFrame, text="Start", command = self.RunTimer, bg ='#90EE90')  TimerButton.grid(row=3,column=4)  **def** RunTimer(self):  TotalTime = int(self.hour.get())\*3600 + int(self.minute.get())\*60 + int(self.second.get())  self.Time = TotalTime  **while** TotalTime > -1:  min,sec = divmod(TotalTime,60)*#Divides and returns the remainder*  hours = 0  **if** min > 60:  hours, min = divmod(min,60)  self.hour.set("{0:2d}".format(hours))  self.minute.set("{0:2d}".format(min))  self.second.set("{0:2d}".format(sec))  self.master.update()  time.sleep(1)  TotalTime -= 1  **if** TotalTime == 0:  messagebox.showinfo("GetSmart Revision","Activity complete ")  Upload= Button (self.TimeFrame, text = 'Upload', command = self.UploadTimer,bg='light blue')  Upload.grid(row=4,column=4)    **def** UploadTimer (self):  self.Time = self.Time/ 3600 *#converts seconds to hours*  self.Time = round(self.Time\*2) / 2  self.SubmitActivity(self.Time)  **def** Updateuserstats(self):  self.GetClassCode(self.subject)  self.c.execute("SELECT \* FROM scjoin WHERE userid=='"+self.userid+"' AND classcode == '"+str(self.classcode)+"' ")  self.UserStats = self.c.fetchone()  **if** self.UserStats == **None**:*#No activites for the classes*  values = [  self.userid,self.classcode,0,0  ]  **with** self.conn:  self.c.execute("INSERT INTO scjoin VALUES (?,?,?,?) WHERE userid == '"+str(self.userid)+"' AND classid == '"+str(self.classid)+"'",values)  hours = 0  totalhours = 0  *#Get total hours*  self.c.execute("SELECT totalhours FROM scjoin WHERE userid == '"+self.userid+"' AND classcode =='"+str(self.classcode)+"' ")  total\_hours = self.c.fetchone()[0]  *#Add new hours*  total\_hours = int(self.hours) *#+ int(total\_hours)*  **with** self.conn:  self.c.execute("UPDATE scjoin SET totalhours = '"+str(total\_hours)+"' WHERE userid == '"+self.userid+"' AND classcode =='"+str(self.classcode)+"'")  week\_ago = int(self.year\_day) -7  self.c.execute("SELECT hours FROM stats WHERE yearday>= '"+str(week\_ago)+"'")  hourlist = self.c.fetchall()*#past weeks revision hours*  total = 0  **for** hours **in** hourlist:  total += hours[0]  average\_hours = total//7  **with** self.conn:  self.c.execute("UPDATE scjoin SET averagehours = '"+str(average\_hours)+"' WHERE userid == '"+self.userid+"' AND classcode =='"+str(self.classcode)+"'")  **def** GetClasses(self):  self.c.execute("SELECT classcode FROM scjoin WHERE userid == '"+str(self.userid)+"'")  self.CodeList = self.c.fetchall()  self.classlist = []  **for** codes **in** self.CodeList:  codes = str(codes[0])  self.c.execute("SELECT classid FROM teacherclassjoin WHERE classcode == '"+codes+"'")  classid = str(self.c.fetchone()[0])  self.c.execute("SELECT subname FROM class WHERE classid == '"+classid+"'")  self.classlist.append(self.c.fetchone()[0])  **def** GetClassCode(self,subname):  self.c.execute("SELECT classid FROM class WHERE subname == '"+subname+"'")  classid = str(self.c.fetchone()[0])  self.c.execute("SELECT classcode FROM teacherclassjoin WHERE classid == '"+classid+"'")  classcodes = self.c.fetchall()  **for** code **in** classcodes:  self.c.execute("SELECT classcode FROM scjoin WHERE classcode == '"+str(code[0])+"' AND userid == '"+self.userid+"'")  check = self.c.fetchone()  **if** check:  self.classcode = check[0]  **def** SubmitActivity(self,hours):  subject = self.clicked.get()  self.subject = subject  self.GetClassCode(subject)  self.hours = hours  self.c.execute("SELECT classid FROM class WHERE subname == '"+subject+"'")  self.classid = self.c.fetchone()[0]  *##Insert into stats*  full\_date = datetime.datetime.now()  *#date = full\_date.strftime("%x")*  date = datetime.date.today()  current\_time = full\_date.strftime("%X")  self.year\_day = full\_date.strftime("%j")  *##Check for activity already on that day*  self.c.execute(f"SELECT \* FROM stats WHERE date == '{date}' AND userid == '{self.userid}' AND classcode == '{self.classcode}'")  result = self.c.fetchone()  **if** result == **None**:  values =[self.userid,self.classcode,date,current\_time,self.hours,self.year\_day]  **with** self.conn:  self.c.execute("INSERT INTO stats VALUES (?,?,?,?,?,?)",values)  **else**:*##Update that days stats*  **with** self.conn:  *#get hours*  hours = result[4]  self.hours = hours + int(self.hours)  self.c.execute(f"UPDATE stats SET hours = '{self.hours}' WHERE userid == '{self.userid}' AND classcode== '{self.classcode}' AND DATE == '{date}'")  self.Updateuserstats()  Create\_Login.ClearScreen(self)  l = User\_ClassScreen(self.master,self.c,self.conn,self.userid,str(self.classcode))  **class** **User\_ClassScreen**(MainScreen):  **def** \_\_init\_\_(self,master,c,conn,userid,classcode):  super().\_\_init\_\_(master,c,conn,userid)  self.classcode = str(classcode[0])  colour = '#4CD9D9'  self.GetDetails()  Create\_Login.ClearScreen(self)  self.Notebook =ttk.Notebook(self.master)  self.HeadingFrame = LabelFrame(self.master,bg = colour)  self.HeadingFrame.pack(ipadx=100,side=TOP,ipady=0)  self.HeadingTitle = Label(self.HeadingFrame, text = self.SubName, bg= colour,font =('Helvetica',16,'bold') )  self.HeadingTitle.grid(row=0,column=0,ipadx=100,columnspan=4)  self.TeacherSubHeader = Label(self.HeadingFrame, text= self.TeacherName,bg= colour, font =('ComicSans'))  self.TeacherSubHeader.grid(row=1,column=3)  self.ClassHeader = Label(self.HeadingFrame, text=f"Class Code: {self.classid}", bg=colour, font=('ComicSans'))  self.ClassHeader.grid(row=2,column=3)  self.MainFrame = Frame(self.Notebook, bg= 'white')  self.MainFrame.pack(fill= BOTH,side=BOTTOM,expand =**True**)  self.ClassStats= Label(self.MainFrame,text = "Class Statistics",font =('Helvetica',14,'bold'),bg='white')  self.ClassStats.grid(row=0,column=0)  self.LeaderBoardFrame = Frame(self.MainFrame)  self.LeaderBoardFrame.grid(row=1,column=0)  *##Add scrolllbar*  self.TreeScroll = Scrollbar(self.LeaderBoardFrame)  self.TreeScroll.pack(side =RIGHT, fill=Y)  self.LB = ttk.Treeview(self.LeaderBoardFrame,height=3,yscrollcommand =self.TreeScroll.set)  self.LB['columns'] = ('Name','Surname','Total', 'Average')  self.LB.column("#0",width=0,stretch=NO)  self.LB.column("Name",anchor=W,width=60)  self.LB.column("Surname",anchor=W,width=60)  self.LB.column("Total",anchor=CENTER,width=60)  self.LB.column("Average",anchor=W,width=60)  *#Create Headings*  self.LB.heading("#0",text='',anchor=W)  self.LB.heading('Name',text='Name',anchor=W)  self.LB.heading('Surname',text='Surname',anchor=W)  self.LB.heading('Total',text = "Total",anchor=CENTER)  self.LB.heading("Average",text = 'Average')  self.LB.pack()  self.TreeScroll.config(command=self.LB.yview)  self.CreateLB()  self.Pstats =Label(self.MainFrame,text = "Personal Statistics",font =('Helvetica',14,'bold'),bg='white')  self.Pstats.grid(row=2,column=0)  self.PLBFrame = Frame(self.MainFrame)  self.PLBFrame.grid(row=3,column=0,ipadx=10)  self.PLBTreeScroll = Scrollbar(self.PLBFrame)  self.PLBTreeScroll.pack(side =RIGHT, fill=Y)  *#Add style*  style = ttk.Style()  *#Pick a theme*  style.theme\_use("clam")  self.PLB = ttk.Treeview(self.PLBFrame,height=3,yscrollcommand =self.PLBTreeScroll.set)  self.PLB['columns'] = ('Date','Time','Hours')  self.PLB.column("#0",width=0,stretch=NO)  self.PLB.column("Date",anchor=W,width=60)  self.PLB.column("Time",anchor=W,width=60)  self.PLB.column("Hours",anchor=CENTER,width=60)  *#Create Headings*  self.PLB.heading("#0",text='',anchor=W)  self.PLB.heading('Date',text='Date',anchor=W)  self.PLB.heading('Time',text='Time',anchor=W)  self.PLB.heading('Hours',text = "Hours",anchor=CENTER)  *#self.PLB.grid(row=3,column=0)*  self.PLB.pack(ipadx=20)  self.PLBTreeScroll.config(command=self.PLB.yview)  self.GetPLB()  self.Notebook.pack(ipady=100,expand=1,ipadx=100)  self.Notebook.add(self.MainFrame,text= "Main Page")  CreateTable(self.master,self.c,self.conn,self.userid,self.classcode,self.Notebook)  UniversityPage(self.master,self.c,self.conn,self.userid,self.classid,self.Notebook)  FlashCards(self.master,self.c,self.conn,self.userid,self.classid,self.Notebook).GetCards()  **def** CreateLB(self):  self.GetClassList()  **def** GetDetails(self):  self.userid=str(self.userid)  self.c.execute("SELECT classid FROM teacherclassjoin WHERE classcode == '"+self.classcode+"'")  self.classid = str(self.c.fetchone()[0])  self.c.execute("SELECT subname FROM class WHERE classid== '"+self.classid+"'")  self.SubName = self.c.fetchall()  self.c.execute("SELECT teacherid FROM teacherclassjoin WHERE classcode== '"+self.classcode+"' ")  teacherid = str(self.c.fetchone()[0])  self.c.execute("SELECT title,surname FROM teacher WHERE teacherid == '"+teacherid+"' ")  title,surname = self.c.fetchone()  self.TeacherName = title + " "+ surname  **def** GetClassList(self):  self.c.execute("SELECT \* FROM scjoin WHERE classcode == '"+self.classcode+"'")  classlist = self.c.fetchall()  count = 0  **for** students **in** classlist:  self.c.execute("SELECT firstname, surname FROM account WHERE userid == '"+str(students[0])+"'")  names = self.c.fetchall()[0]  firstname, surname = names[0], names[1]  details = [firstname, surname, students[2],students [3]]  self.LB.insert(parent='',index='end',iid=count, values = (details[0],details[1],details[2],details[3]))  count += 1  **def** GetPLB(self):  self.c.execute("SELECT \* FROM stats WHERE userid == '"+self.userid+"' AND classcode == '"+self.classcode+"'")  records = self.c.fetchall()  records.reverse()*#sorts list in order it was added to the database*  count =0  **for** x **in** range (len(records)):  self.PLB.insert(parent='',index='end',iid=count, values = (records[x][2],records[x][3],records[x][4]),tags='blue')  count += 1  **class** **FlashCards**:  **def** \_\_init\_\_(self,master,c,conn,userid,classcode,frame):  self.master = master  self.c = c  self.conn = conn  self.userid,self.classcode = userid, classcode  self.Notebook = frame  self.Frame1 = Frame(self.Notebook,bg='white')  self.Frame1.pack(expand=1)  self.Notebook.add(self.Frame1,text="Flashcards")  self.S1 = Stack()  **def** GetCards(self):  self.c.execute("SELECT setid FROM flashcardjoin WHERE userid == '"+self.userid+"' AND classcode == '"+self.classcode+"'")  sets = self.c.fetchall()  **if** sets:*#If the user has any flashcard sets already created*  Title = Label(self.Frame1, text = "Revision Flashcards",font = ("Arial",13),bg='white')  Title.grid(row=0,column=3)  self.idlist = []  buttonlist = []  SetFrame = Frame(self.Frame1,bg='white')  SetFrame.grid(row=2,column=0)  AddSetButton = Button(SetFrame, text= "Add set!",fg= 'green',command = self.AddSet)  AddSetButton.pack()  RemoveSetButton = Button(SetFrame, text = "Delete set", fg='red', command = self.DeleteSetWindow)  RemoveSetButton.pack(pady=5,padx=5)  **for** x **in** range (len(sets)):  self.idlist.append(sets[x])  self.c.execute("SELECT title FROM flashcardjoin WHERE setid == '"+str(sets[x][0])+"'")  title = self.c.fetchone()  buttonlist.append(Button(SetFrame,text=f"{title[0]} **\n** Set ID:{sets[x][0]}",bg='#72f542',command = **lambda** x=x: self.GoToSet(x)))  buttonlist[x].pack(ipadx=5,ipady=5,pady=5,padx=5)  **if** **not** sets:*#If the user has no created flashcard sets*  Label1 = Label(self.Frame1, text = "You have no current set of flashcards",fg = 'red', font = ("Arial",12),bg='white')  Label1.grid(row=1,column=0,padx=30,pady=30)  AddSetButton = Button(self.Frame1, text= "Add set!",fg= 'green',command = self.AddSet)  AddSetButton.grid(row=2,column=0,pady=5,ipadx=5)    **def** DeleteSetWindow(self):  self.window = Toplevel()  self.window.geometry("200x200")  Title = Label (self.window,text = "Delete Flashcards", font = ("Comic Sans Ms",13,"bold"),fg='red')  Title.grid(row=0,column=0,columnspan=2,pady=10)  Label1 = Label(self.window,text="Enter set ID:",font = ("Arial"))  Label1.grid(row=2,column=0)  Entry1 = Entry(self.window,width=15)  Entry1.grid(row=2,column=1,pady=5,padx=10)  Entry1.insert(0,"000")  Delete = Button(self.window,text = "Confirm", fg='red',command = **lambda**: self.DeleteSet(Entry1.get()))  Delete.grid(row=3,column=1,pady=5)    **def** DeleteSet(self,setid):  **with** self.conn:*#Deletes set from database*  self.c.execute(f"DELETE FROM flashcardjoin WHERE setid == '{setid}'")  self.c.execute(f"DELETE FROM flashcard WHERE setid == '{setid}' ")  self.window.destroy()  User\_ClassScreen(self.master,self.c,self.conn,self.userid,self.classcode)  **def** GoToSet(self,index):  self.setid = str(self.idlist[index][0])  *#Create\_Login.ClearScreen(self)*  self.Set()  **def** AddSet(self):  window = Toplevel()  colour = '#42ddf5'  window.geometry("300x300")  window.title('Revision tool')  window.config(bg=colour)  Title = Label(window,text= "Add Revision set",bg=colour,font = ('Arial',13,'bold'))  Title.grid(row=0,column=3)  NameLabel= Label(window,text = "Set name",bg=colour)  NameLabel.grid(row=1,column=1,pady=5)  NameEntry = Entry (window,width=20)  NameEntry.grid(row=1,column=2,columnspan=2,padx=3)  NextButton = Button(window,text='Next',bg='#90EE90',command = **lambda**: self.CreateSet(NameEntry.get(),window))  NextButton.grid(row=3,column=3,pady=5)    **def** CreateSet(self,setname,window):  values = [(**None**,self.userid,self.classcode,setname)]  *# self.c.executemany('INSERT INTO account (username,password,firstname,surname,gender,email) VALUES (?,?,?,?,?,?)',values)*  **with** self.conn:  self.c.executemany('INSERT INTO flashcardjoin (setid,userid,classcode,title) VALUES (?,?,?,?)',values)  Create\_Login.ClearScreen(self)  window.destroy()  User\_ClassScreen(self.master,self.c,self.conn,self.userid,self.classcode)    **def** Set(self):  self.c.execute("SELECT title FROM flashcardjoin WHERE setid == '"+self.setid[0]+"'")  title = self.c.fetchone()[0]  colour = '#90EE90'  window = Toplevel()  self.c.execute('SELECT heading, info FROM flashcard WHERE setid == "'+self.setid+'"')  cards = self.c.fetchall()  **for** card **in** cards:  self.S1.Push(card)*#Pushes card onto stack*  window.geometry("300x300")  window.title('Revision tool')  window.config(bg=colour)  TitleLabel = Label(window, text = f"Flashcard set: {title}",font = ("Comic Sans Ms", 15,'bold'),bg=colour)  TitleLabel.grid(row=0,column=0,columnspan=3)  self.c.execute("SELECT heading, info FROM flashcard WHERE setid == '"+self.setid+"'")  self.cards = self.c.fetchall()  AddButton = Button(window, text = 'Add Flashcards',bg ='white',command = self.AddCard)  AddButton.grid(row=1,column=0,pady=5,ipadx=5,ipady=2,padx=5)  **if** len(self.cards) > 3:*#If the user has enough vards to test*  TestButton = Button(window, text = "Test!",bg = 'white',command = **lambda**: Test(window,self.c,self.conn,self.setid,self.userid,self.classcode,self.master))  TestButton.grid(row=1,column=2,pady=5,ipadx=5,ipady=2,padx=5)  self.CardFrame = Frame(window,bg= 'white')  self.CardFrame.grid(row=3,column=0,columnspan=4,rowspan=5,pady=20)  self.Toolbar = Frame(window)  self.Toolbar.grid(row=10,column=0,columnspan=4,pady=5)  self.FillCardFrame(**False**,0)  **def** FillCardFrame(self,Info,index):  **for** widgets **in** self.CardFrame.grid\_slaves():  widgets.destroy()  self.c.execute("SELECT heading, info FROM flashcard WHERE setid =='"+self.setid+"'")  cards = self.c.fetchall()  **if** self.S1.IsEmpty():*#Checks if the stack is empty and returns a Boolean val*  Label1 = Label (self.CardFrame, text = "Add Cards now!",font = ("Comic Sans Ms",13,"bold"),bg='#90EE90')  Label1.grid(row=0,column=0)  **else**:  **if** index == len(cards):  index = 0  **if** index == -1:  index = len(cards)-1  card = cards[index]  **if** Info == **False**:  Text = Label(self.CardFrame, text = card[0],font = ("Comic Sans Ms",12))*#Determines what side of the card to show*  **else**:  Text = Label(self.CardFrame, text = card[1],font = ("Comic Sans Ms",12))  Text.grid(row=0,column=0,ipadx=10,ipady=10,columnspan=3)  Back = Button(self.Toolbar, text = '<==',command = **lambda**: self.FillCardFrame(**False**,index-1))  Back.grid(row=1,column=0)  Flip = Button(self.Toolbar,text="Flip",command = **lambda**: self.FillCardFrame(**not** Info,index))*##recursion to flip the card*  Flip.grid(row=1,column=1)  Forward = Button(self.Toolbar,text = '==>',command = **lambda**: self.FillCardFrame(**False**,index+1))  Forward.grid(row=1,column=3)  **def** AddCard(self):  root = Toplevel()  root.geometry("400x200")  root.title("Flashcard Maker")  Label1 = Label(root,text="Keyword")  Label1.grid(row=0,column=0,pady=5,padx=5)  self.WordEntry = Entry(root,width = 20)  self.WordEntry.grid(row=1,column=1)  Label2 = Label(root,text="Description")  Label2.grid(row=2,column=0,pady=5)  self.Info = Text(root,width=30,height=5)  self.Info.grid(row=3,column=1)  Add = Button(root,text='Add',bg = '#72f542',command = **lambda**: self.SaveCard(self.WordEntry.get(),self.Info.get('1.0',END)))  Add.grid(row=4,column=1)    **def** SaveCard(self,word,info):  values = [(self.setid,word,info)]  self.S1.Push((word,info))  **with** self.conn:  self.c.executemany('INSERT INTO flashcard (setid,heading,info) VALUES (?,?,?)',values)  self.WordEntry.delete(0,END)  self.Info.delete('1.0','end')  **class** **Stack** (FlashCards):  **def** \_\_init\_\_(self,stack=**None**):  **if** stack ==**None**:  self.stack = []  **else**:  self.stack = stack    **def** Shuffle (self):  **for** i **in** range (len(self.stack)-1,0,-1):  rand = random.randint(0,i)  self.stack[i], self.stack[rand] = self.stack[rand],self.stack[i]  **return** self.stack  **def** Pop(self):*#Remove and return the card from the top of the stack*  self.card = self.stack[-1]  self.stack.pop(-1)  **return** self.card    **def** Push(self,card):*#Push card to the top of the stack*  **if** card **not** **in** self.stack:  self.stack.append(card)  **def** Peek(self):*#Return the card on the top of the stack*  **return** self.stack[-1]  **def** IsEmpty(self):  **if** len(self.stack) == 0:  **return** **True**  **else**:  **return** **False**  **def** PrintStack(self):  **for** card **in** self.stack:  print (card)  **class** **Test**:  **def** \_\_init\_\_(self,frame,c,conn,setid,userid,classcode,master):  *#super().\_\_init\_\_(master,c,conn,userid,classcode,frame)*  self.frame = frame  self.c =c  self.conn = conn  self.setid = setid  self.userid = userid  self.classcode = classcode  self.master = master  self.Testr(self.frame)  **def** Testr(self,window):  window.destroy()  window = Toplevel()  self.MainWindow = window  window.geometry("500x400")  window.title("Flashcard test")  self.Frame = Frame(window)  self.Frame.pack(expand=1,ipady=100)  Title = Label(self.Frame, text= "Flaschard Test!",font = ('Comic Sans Ms',15,'bold'))  Title.grid(row=0,column=0,columnspan=2)  Label1 = Label(self.Frame, text="Number of questions: ")  Label1.grid(row=2,pady=20,column=1)  self.SortClick = StringVar()  self.SortClick.set(10)  Number = OptionMenu(self.Frame,self.SortClick,5,10,15,20)  Number.grid(row=2,column=3)  self.Qnumber = 0  Go = Button(self.Frame,text="Start!",fg="green",command = self.TestWindow)  Go.grid(row=3,column=3,pady=10,ipady=5,ipadx=10)  **def** TestWindow(self):  self.number = self.SortClick.get()  **for** widget **in** self.Frame.winfo\_children():  widget.destroy()  self.GetQuestions()  **def** GetQuestions(self):  self.c.execute("SELECT heading,info FROM flashcard WHERE setid == '"+self.setid+"'")  self.CardsList = self.c.fetchall()  self.Cards= Stack(self.CardsList)  print ()  print (self.Cards)  self.Cards.PrintStack()  self.Cards = self.Cards.Shuffle()  *#self.Cards.PrintStack()*  QuestionNumber = 1  self.totalright = 0  self.Stats = Frame(self.Frame)  self.Stats.grid(row=1,column=4)  self.Correct = "Correct: 0"  self.QuestionTitle = "Question 0"  self.Prompt = Label (self.Frame, text=self.Cards[0][0],font = ("Comic Sans Ms",15,"bold"),fg='blue')  self.Prompt.grid(row=1,column=2)  self.React = Label(self.Frame,font =("Comic Sans Ms",12),fg='green')  self.React.grid(row=4,column=2,columnspan=2)  self.NextQuestion(QuestionNumber)  **def** NextQuestion(self,QuestionNumber):  self.Title = Label (self.Frame, text = self.QuestionTitle, font = ("Arial",15,"bold"))  self.Title.grid(row=0,column=0,columnspan=2)  self.CorrectQ = Label(self.Stats, text= self.Correct,font = ("Comic Sans Ms",10))  self.CorrectQ.grid(row=0,column=0)  self.QNum = QuestionNumber  *##Quiz Loop*  **if** self.QNum == int(self.number)+1:  **for** widget **in** self.Frame.winfo\_children():  widget.destroy()  self.EndScreen(self.totalright)  **else**:  self.temp = []  self.card = self.Cards[0]  *#Print the question*  self.QuestionFrame = Frame(self.Frame,borderwidth=10,highlightbackground= 'blue',relief = RIDGE)  self.QuestionFrame.grid(row=3,column=1,columnspan=3,pady=10,ipady=10,ipadx=20)  **for** card **in** self.Cards:  self.temp.append(card)  self.temp.remove(self.card)  self.GetWrongAnswers()  self.GenerateQuestions()  **def** UpdateStats(self,right):  self.totalright += right  self.Correct = f"Correct {self.totalright}"  self.CorrectQ.config(text=self.Correct)  self.Cards.pop(0)  self.Cards.insert(-1,self.card)*#Moves the card from the back to the front*  *#Update the screen*  self.QuestionTitle = f"Question {self.QNum}"  self.Title.config(text=self.QuestionTitle)  self.Prompt.config(text=self.Cards[0][0])  **if** right == 1:  text = "✔ CORRECT"  fg1 = 'green'  **else** :  text = "✘Incorrect Answer"  fg1 = 'red'  self.React.config(text = text,fg=fg1)  self.QuestionFrame.destroy()  self.NextQuestion(self.QNum+1)  **def** GetWrongAnswers(self):  self.wrong\_answers = []  **for** x **in** range (3):  index = random.randint(0,len(self.temp)-1)  self.wrong\_answers.append(self.temp[index][1])  self.temp.pop(index)  **def** GenerateQuestions(self):  question = self.card[0]  right\_answer =self.card[1]  coords = [(0,0),(1,0),(0,1),(1,1)]  index = random.randint(0,3)  rightcoords = coords[index]  coords.pop(index)  r = IntVar()  r.get  *#r.set()*  R1 = Radiobutton(self.QuestionFrame,text=right\_answer, value = 1, variable = r,command = **lambda**: self.clicked(r.get()))  R1.grid(row= rightcoords[0],column = rightcoords[1])  c = 2  **for** x **in** range (3):  R1 = Radiobutton(self.QuestionFrame, text=self.wrong\_answers[x], value = c, variable = r , command = **lambda**: self.clicked(r.get()))  R1.grid(row=coords[x][0], column= coords[x][1])  c += 1  **def** clicked(self,value):  **if** value == 1:*#right answer*  right = 1  **else**:*#wrong answer*  right = 0  self.UpdateStats(right)  **def** EndScreen(self,totalright):  Title = Label(self.Frame, text = "Test Complete",font = ("Comic Sans Ms",15))  Title.grid(row=0,column=0,pady=5)  Label1 = Label(self.Frame, text = f"You finished with {totalright} answers out of {self.number}",font = ('bold'),fg='blue')  Label1.grid(row=2,column=0)  percentage = (totalright / int(self.number)) \* 100  percentage = round(percentage,1)  **if** percentage < 50:  colour = 'red'  **if** percentage >= 50 **and** percentage < 80:  colour = 'orange'  **if** percentage > 79 :  colour = 'green'  Label2 = Label(self.Frame, text = f"{percentage}%",font = ("Comic Sans Ms ",25,'bold'),fg=colour)  Label2.grid(row=3,column=0)  Return = Button (self.Frame, text = "Return",command = **lambda**: self.MainWindow.destroy())  Return.grid(row=4,column=0)  **class** **CreateTable**():  **def** \_\_init\_\_(self,master,c,conn,userid,classcode,notebook):  self.master = master  self.c = c  self.conn = conn  self.userid = userid  self.classcode = classcode  self.Notebook = notebook  *##CREATE TAB*  self.Frame1 = Frame(self.Notebook,bg="grey")  self.Frame1.pack(expand=1)  self.Notebook.add(self.Frame1,text = 'Progress')  self.CreateGraph()  **def** CreateGraph(self):  self.c.execute("SELECT \* FROM stats WHERE userid == '"+self.userid+"' AND classcode== '"+self.classcode+"'")  dates = self.c.fetchall()  dates\_list = []  time\_list = []  fig = plt.Figure(figsize = (5, 4),  dpi = 100)  canvas = FigureCanvasTkAgg(fig,master = self.Frame1)  **for** value **in** dates:  date = value[2]  time = value[4]  dates\_list.append(date)  time\_list.append(time)  fig.add\_subplot(111).plot(dates\_list,time\_list)  canvas.draw()  *# placing the canvas on the Tkinter window*  toolbar = NavigationToolbar2Tk(canvas,self.Frame1)  toolbar.update()  canvas.get\_tk\_widget().pack()  **class** **UniversityPage**(User\_ClassScreen):  **def** \_\_init\_\_(self,master,c,conn,userid,classid,notebook):  self.master = master  self.c = c  self.conn = conn  self.userid = userid  self.classid = classid  self.Notebook = notebook  self.UniFrame = Frame(self.Notebook)  self.UniFrame.pack(fill= BOTH,side=BOTTOM,expand =**True**)  self.Title= Label(self.UniFrame, text = 'Search University Courses',font = ('Helvetica',15))  self.Title.grid(row=0,column=3)  self.CourseEntry = Entry(self.UniFrame, width=20)  self.CourseEntry.grid(row=1,column=2,pady=5,columnspan=2)  self.CourseEntry.insert(0,"Course Name")  self.Search = Button(self.UniFrame, text = 'Search', command = **lambda**: GetUni(self.master,self.c,self.conn,self.CourseEntry.get(),self.UniFrame),bg = '#90EE90')  self.Search.grid(row=1,column=5)  self.Notebook.add(self.UniFrame,text = 'University')  **class** **GetUni**(UniversityPage):  **def** \_\_init\_\_(self,master,c,conn,coursename,UniFrame):  self.master = master  self.c = c  self.conn = conn  self.coursename = coursename  self.UniFrame = UniFrame  self.FormatURL()  self.page\_soup = self.OpenWebsite(self.URL)  self.GetValues()  self.PrintValues()  self.FilterButton = Button(self.UniFrame, text = 'Filter', command = self.Filter)  self.FilterButton.grid(row=2,column=5)  **def** Filter(self):  self.colour = "#8DF8D7"  self.window = Toplevel()  self.font = ('Hellvetica',10)  self.window.geometry("300x300")  self.window.title('Revision tool')  self.window.config(bg=self.colour)  self.Title = Label(self.window,text='Filter Universities', bg = self.colour,font=('Apple Color Emoji',20))  self.Title.grid(row=0,column=0,columnspan=4,ipadx=30)  self.SortByLabel = Label(self.window,text="Sort by", font = self.font,bg=self.colour)  self.SortByLabel.grid(row=1,column=0,pady=10)  self.SortClick = StringVar()  self.SortClick.set('')  self.SortMenu = OptionMenu(self.window,self.SortClick,"Ascending","Descending")  self.SortMenu.grid(row=1,column=3)  self.SortButton = Button(self.window,text="Sort",command = **lambda** : self.MergeSort(self.SortClick.get()))  self.SortButton.grid(row=5,column=0)  **def** OpenWebsite(self,url):  self.opener = AppURLopener()  self.uClient = self.opener.open(url)  self.page\_html = self.uClient.read()*#saves it as a variable*  self.uClient.close()*#closes it*  self.page\_soup = soup(self.page\_html,"html.parser")  **return** self.page\_soup  **def** PrintValues(self):*#Puts universities onto the page*  self.TableFrame = Frame(self.UniFrame)  self.scrollbar = Scrollbar(self.TableFrame,orient = VERTICAL)  self.scrollbar.pack(side=RIGHT,fill=Y)  self.UniTable = ttk.Treeview(self.TableFrame,height=6,yscrollcommand = self.scrollbar.set)  self.UniTable['columns'] = ('University','Location','Grades')  self.UniTable.column("#0",width=0,stretch=NO)  self.UniTable.column("University",anchor=W,width=120)  self.UniTable.column("Location",anchor=W,width=100)  self.UniTable.column("Grades",anchor=CENTER,width=80)  *#Create Headings*  self.UniTable.heading("#0",text='',anchor=W)  self.UniTable.heading('University',text='University',anchor=W)  self.UniTable.heading('Location',text='Location',anchor=W)  self.UniTable.heading('Grades',text = "Grades",anchor=CENTER)  self.scrollbar.config(command=self.UniTable.yview)  count = 0  **for** uni **in** self.UniList:*#Insert values into table*  self.UniTable.insert(parent='',index='end',iid=count, values = (uni['name'],uni['location'],uni['grade']),tags='blue')  count += 1  self.UniTable.pack()  self.TableFrame.grid(row=3,column=0,columnspan=5,rowspan=2)  **def** GetValues(self):*#Gets Universities*  location = self.page\_soup.findAll('span',{'class':'institution\_location'})  ucas = self.page\_soup.findAll('div',{'class':'points'})  unilist = []  ucaslist = []  **for** grade **in** ucas:  ucaslist.append(grade.text)  UniList = []  dictionary = {}  **for** x **in** range (len(location)):  both = location[x].text.split('|')*#gets the location and the uni name*  uniname = both[0]  unilocation = both[1]  grade = ucas[x].text.split()[2]  dictionary = {'name': uniname,  'location':unilocation,  'grade':grade}  UniList.append(dictionary)*#Creates a list of dictionaries containing each university*  self.UniList = UniList  **def** MergeSort(self,sorter):  **def** Sort(List):  **if** len(List)> 1:*#base case*  mid = len(List)//2  left = List[:mid]  right = List[mid:]  Sort(left)  Sort(right)  *#Merge*  i = 0*#Left counter*  j = 0*#Right counter*  k = 0*#Main list counter*  **while** i<len(left) **and** j<len(right):  **if** left [i]['grade'].split('-')[0] =='':*#Fixes the webscraped numbers*  left [i]['grade'].split('-')[0] = left[i]['grade'] = '0''0'  **if** right[j]['grade'].split('-')[0] =='':  right[j]['grade'].split('-')[0] = right[j]['grade'] = '0'  **if** int(left[i]['grade'].split('-')[0]) < int(right[j]['grade'].split('-')[0]):  List[k]= left[i]  i += 1  k += 1  **else**:  List[k] = right[j]  j += 1  k += 1  **while** i<len(left):  List[k] = left[i]  i += 1  k += 1  **while** j<len(right):  List[k] = right[j]  j+= 1  k +=1  Sort(self.UniList)  **if** sorter == 'Descending':  self.UniList.reverse()  count =0  self.UniTable.delete(\*self.UniTable.get\_children())  **for** uni **in** self.UniList:*#Insert values into table*  self.UniTable.insert(parent='',index='end',iid=count, values = (uni['name'],uni['location'],uni['grade']),tags='blue')  count += 1  self.UniTable.pack()  self.TableFrame.grid(row=3,column=0,columnspan=5,rowspan=2)  **def** FormatURL(self):  url = 'https://www.theuniguide.co.uk/search/course?utf8=%E2%9C%93&c%5Bq%5D='  words = self.coursename.split()  **if** len(words) >= 1:  **for** x **in** range (len(words)):  **if** x == 0:  my\_url = f'https://www.theuniguide.co.uk/search/course?utf8=%E2%9C%93&c%5Bq%5D={words[0]}'  **else**:  my\_url = my\_url+f'{"+"}{words[x]}'  self.URL = my\_url  **class** **AppURLopener**(urllib.request.FancyURLopener):  version = "Mozilla/5.0"    **class** **EncryptPassword**(Create\_Login):  **def** \_\_init\_\_(self,root,c,conn,password):  super().\_\_init\_\_(root,c,conn)  **def** HashPassword(self,password):  salt = hashlib.sha256(os.urandom(60)).hexdigest().encode('ascii')*#Creates a salt*  password\_hash = hashlib.pbkdf2\_hmac('sha512',password.encode('utf-8'),  salt, 100000)*#Uses a hashing algorithm to create the hash*  password\_hash = binascii.hexlify(password\_hash)  **return** (salt+password\_hash).decode('ascii')  **def** CheckPassword(self,stored\_password, user\_password):  salt = stored\_password[:64]*#Gets the salt*  stored\_password = stored\_password[64:]  password\_hash = hashlib.pbkdf2\_hmac('sha512',user\_password.encode('utf-8'),  salt.encode('ascii'),  100000)  password\_hash = binascii.hexlify(password\_hash).decode('ascii')  **return** password\_hash == stored\_password  **class** **Profile**:  **def** \_\_init\_\_(self,master,c,conn,ID,AccType):  self.master = master  self.c = c  self.conn = conn  self.ID = ID  self.AccType = AccType  self.Frame1 = Frame(self.master)  self.font = ('Hellvetica', 10)  Create\_Login.ClearScreen(self)  self.Title = Label (self.Frame1, text= 'Your Details',font= ('Hellvetica',20,'bold'),fg="green")  self.Title.grid(row=0,column=2,columnspan=2)  self.GetDetails()  self.UpdateButton = Button(self.Frame1,text='Update',bg='#90EE90',command = self.Update)  self.UpdateButton.grid(row=6,column=4,ipadx=10,ipady=3,pady=10)  **if** AccType == 'teacher':  self.TeacherProf()  **if** AccType == 'student':  self.StudentProf()  self.Frame1.pack(fill=BOTH)  **def** GetDetails(self):  **if** self.AccType == 'teacher':  self.c.execute(f"SELECT \* FROM teacher WHERE teacherid == {self.ID} ")  **if** self.AccType == 'student':  self.c.execute(f"SELECT \* FROM account WHERE userid == '{self.ID}' ")  self.details = self.c.fetchone()  **def** TeacherProf(self):  self.Title = Label(self.Frame1, text= 'Title' , font = self.font)  self.Title.grid(row=1,column=0)  self.TitleClick = StringVar()  self.TitleClick.set(self.details[1])  self.TitleMenu = OptionMenu(self.Frame1,self.TitleClick,"Mr","Ms","Dr")  self.TitleMenu.grid(row=1,column=3)  self.NameLabel = Label(self.Frame1,text='Name', font = self.font)  self.NameLabel.grid(row=2,column=0)  self.NameEntry = Entry(self.Frame1,width=20)  self.NameEntry.grid(row=2,column=3,pady=5)  self.NameEntry.insert(0,self.details[2])  **def** StudentProf(self):  self.UsernameLabel = Label(self.Frame1,text='Username', font = self.font)  self.UsernameLabel.grid(row=1,column=0)  self.UsernameEntry = Entry(self.Frame1,width=20)  self.UsernameEntry.insert(0,self.details[1])  self.UsernameEntry.grid(row=1,column=3)  self.FirstnameLabel = Label(self.Frame1,text='Firstname', font = self.font)  self.FirstnameLabel.grid(row=2,column=0)  self.FirstEntry = Entry(self.Frame1,width=20)  self.FirstEntry.insert(0,self.details[3])  self.FirstEntry.grid(row=2,column=3)  self.LastLabel = Label(self.Frame1,text='Lastname',font= self.font)  self.LastLabel.grid(row=3,column=0)  self.LastEntry = Entry(self.Frame1,width=20)  self.LastEntry.insert(0,self.details[4])  self.LastEntry.grid(row=3,column=3)  self.GenderLabel =Label(self.Frame1,text = "Gender:")  self.GenderLabel.grid(row=4,column = 0)  self.clicked = StringVar()  self.clicked.set(self.details[5])  self.GenderEntry = OptionMenu(self.Frame1,self.clicked,"Male","Female","Other")  self.GenderEntry.grid(row=4,column=3)  self.EmailLabel = Label(self.Frame1,text='Email Address')  self.EmailLabel.grid(row=5,column=0)  self.EmailEntry = Entry(self.Frame1,width=30)  self.EmailEntry.insert(0,self.details[6])  self.EmailEntry.grid(row=5,column=3,columnspan=2,padx=20)  **def** Update(self):  **if** self.AccType == 'student':  statement = (f"UPDATE account SET username = '{self.UsernameEntry.get()}',firstname = '{self.FirstEntry.get()}',surname= '{self.LastEntry.get()}',gender='{self.clicked.get()}',email='{self.EmailEntry.get()}' WHERE userid == '{self.ID}' ")  **if** self.AccType == 'teacher':  statement = (f"UPDATE teacher SET title='{self.TitleClick.get()}', surname='{self.NameEntry.get()}' WHERE teacherid == '{self.ID}' ")  **with** self.conn:  self.c.execute(statement)  label = Label(self.Frame1,text="Update Successful",fg='green')  label.grid(row=7,column=4)  c=Create\_Login(root,c, conn)  root.mainloop()  conn.close() |

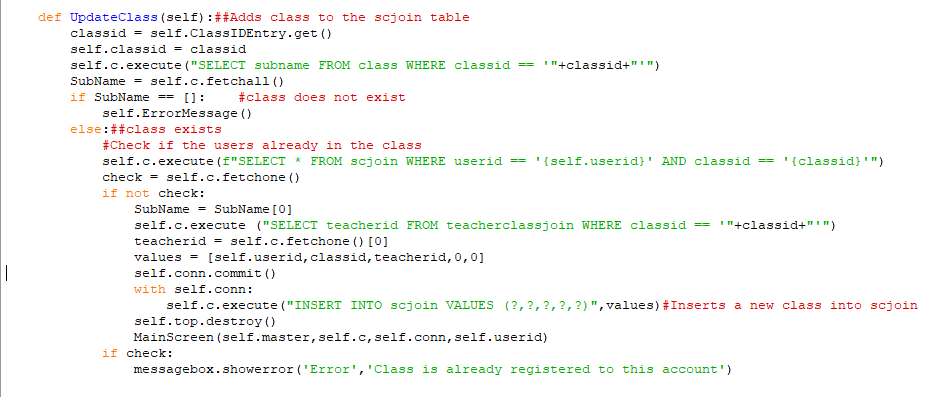
**Testing**

In this section I will test my working code. I will test it in a variety of ways to check its robustness and how easy it is to use for a user as a piece of software.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test number | Test | Test type | Objective | Test data | Expected outcome | Result |
| 1 | Program loads within 5 seconds | UI testing | 1 | N/A | Program loads within 5 seconds | SUCCESS-program loaded in under 5 seconds |
| 2 | Sign up a user | Acceptance | 1 | Username:’steven123’  Password:’6Stev7888’ | User is added into the database and the class page appears | SUCCESS- user was entered into the database as username and password was accepted |
| 3 | Enter invalid password to signup user, below 7 characters | Error | 4,6 | Username:’steven123’  Password’Short’ | Error message is shown and Login screen refreshes | SUCCESS- user was shown an error message and wasn’t allowed to sign up |
| 4 | Enter invalid password to signup user, no capital letters or number | Error | 4,6 | Username:’steven123’  Password:’nocapital’ | Error message is shown and Login screen refreshes | SUCCESS- user was shown an error message and wasn’t allowed to sign up |
| 5 | Enter long username to sign user up | Boundary | 4 | Username:’11111…..’  Password’16STeveQH’ | User is added into the database and the class page appears | SUCCESS- user was entered into the database as username and password was accepted |
| 6 | The username must be unique | Erroneous | 4 | USERNAME:’steven123’ | Program displays error message and doesn’t allow the username | SUCCESS-program displayed message showing the username was already taken |
| 7 | User can add a class | Acceptance | 9 | Classid: ‘1’, a predefined classid | Program adds class to users’ home screen | SUCCESS- class shows up on users homescreen for them to access |
| 8 | User inputs invalid classid | Acceptance | 14 | Classid:’27893’ | Program displays error message showing it is an invalid classid | SUCCESS-Error window was displayed and class is not added to the database |
| 9 | User can add 3 separate classes | Boundary | 9 | 3 Classids:1,2,3 | All classes are added and accessible for the user to click | SUCCESS-All classes are showed on the users homepage |
| 10 | User cannot add the same class twice | Erroneous | 14 | Classids ‘1’, ‘1’ | The software displays an error message and class is not added | ERROR- database unique constraint failed as error handling wasn’t coded |
| 10A | User cannot add the same class twice | Erroneous | 14 | Classids ‘1’, ‘1’ | The software displays an error message and class is not added | SUCCESS- The software displays an error to the user and the program keeps running |
| 11 | User can delete a class from their account | Acceptance | 9 | Classid:’1’ | The class is deleted and the class no longer appears on the homescreen | SUCCESS- user entered classid and class was removed from database and UI |
| 12 | User can click a class and view class screen | Acceptance | 3 | N/A | The GUI changes to show the class screen | SUCCESS-The correct class screen is showed to the user |
| 13 | User can go back to homepage through class page | Acceptance | 3 | N/A | The screen is reset back to the home screen | SUCCESS-Home screen button successfully returns user back to their home page |
| 14 | User can view their profile statistics | Acceptance | 8 | N/A | The profile page is shown | SUCCESS- the profile page is shown showing the users information |
| 15 | Users can edit their information | Acceptance | 11 | ‘matthew’ | The update is successful and a message appears | SUCCESS- the database is updated and a message appears |
| 16 | Users can delete their email addresses | Erroneous | 11 | N/A | The update is successful and their email is removed | SUCCESS- their emails are removed from the database and a message appears |
| 17 | A user can logout | Acceptance | 1 | N/A | The user clicks logout and it returns to the login page | SUCCESS- the program returns to the login page when logout option is pressed |
| 18 | A user can add an activity to a class | Acceptance | 7 | ‘5’ | The activity screen shows up and the activity is added | SUCCESS- activity is added into database |
| 19 | A user cannot submit a >10-hour long activity | Erroneous | 7,14 | ‘11’ | The counter on the entry box won’t allow numbers greater than 10 | SUCCESS- A user cannot input more than 10 hours, an unrealistic amount, in one activity |
| 20 | The activity is added and updates the class table | Acceptance | 7 | N/A | The table on the class screen should display the activity statistics | SUCCESS- Both tables are updated and show the activity |
| 21 | The user enters a lot of activities | Boundary | 7 | ‘10’ | The program should still run with the large data values | SUCCESS- The program keeps running despite the larger numbers |
| 22 | The user tries to enter a 0-hour activity | Boundary | 7,14 | ‘0’ | The activity is added like any other amount of time | SUCCESS-Activity is added |
| 23 | The activity is shown on the graph window | Acceptance | 8 | N/A | The total activity statistics are shown on the graph | SUCCESS- Program plots a graph showing the users total hours over a period of time |
| 24 | The toolbar works for the graph | Acceptance | 8 | N/A | The toolbar should be able to zoom, move and save the graph | SUCCESS- the toolbar works and has complete functionality |
| 25 | The student is able to search for a university of their choice | Acceptance | 12 | ‘Maths’ | A table is created which shows information regarding the course at each university | SUCCESS- A table is displayed showing information |
| 26 | A course is spelt wrong when searching for university courses | Erroneous | 12,14 | ‘enlish’ | Nothing is displayed and the program still runs | SUCCESS-The program keeps running and nothing is displayed |
| 27 | The user enters an obscure course name | Boundary | 12,14 | ‘Ancient history’ | The course checks the website and displays any courses it finds | SUCCESS-The course is displayed |
| 28 | The user can sort the table into ascending order | Acceptance | 8 | N/A | The table is sorted via a merge sort into ascending order | SUCCESS - The universities are displayed in ascending order |
| 29 | The user can sort the list into descending order | Acceptance | 8 | N/A | The table is sorted via a merge sort into descending order | SUCCESS- The table is sorted into descending order and displayed |
| 30 | A teacher is able to make an account | Acceptance | 1 | ‘username123’  ‘Passw0rd’  ‘Mr’  ‘Downwood’ | The teacher is able to sign up and an account is created | SUCCESS- Teacher is signed up and an account is created |
| 31 | A teacher is able to login to their account | Acceptance | 4 | ‘username’  ‘Passw0rd’ | The teacher is logged into their account | SUCCESS- teacher is logged in |
| 32 | A teacher can create a class | Acceptance | 9 | ‘English’  ‘A level’ | The teacher’s class is created and available for students to join | SUCCES- The class is created and is displayed |
| 33 | A teacher can delete a class | Acceptance | 9 | ‘5’ \*Class ID | The teacher’s class is deleted | SUCCESS- The class deleted |
| 34 | A teacher can access the class page | Acceptance | 3 | N/A | The class page is shown to the user | SUCCESS-The class screen is displayed |
| 35 | Teacher can send emails to students in their classes | Acceptance | 13 | ‘Homework set’  ‘Please complete homework’ | The email is sent and the program doesn’t crash | SUCCESS-The email is sent and formatted correctly to the intended students |
| 36 | The teacher is able to send emails to multiple students | Boundary | 13 | ‘Homework set’  ‘Please complete the homework set’ | The email is sent to multiple students | SUCCESS – The email is sent to multiple students |
| 37 | The teacher selects no students | Boundary | 13 | N/A | The program shows an error that no student is clicked and doesn’t crash | SUCCESS- An error message is shown and the program keeps running |
| 38 | The teachers’ class is able to be ranked | Acceptance | 10 | N/A | The teacher’s class is shown on the class leader board and it shows a rank | SUCCESS- The teacher’s class is shown and a message appears |
| 39 | The message changes depending on the classes rank | Acceptance | 10 | N/A | The software displays different messages dependant on the classes rank | SUCCESS- The ranking page shows different messages dependant on the rank of the class |
| 40 | A teacher can access the class database | Acceptance | 11 | N/A | The Software displays a table including all the students from their class | SUCCESS- The class database is shown. |
| 41 | The class database can include multiple students | Boundary | 8 | N/A | The software displays each student inside the database | SUCCESS- Each student is shown |
| 42 | The search works when multiple students have the same surname | Boundary | 14 | ‘Beardsley’ | Each occurrence of the surname shows up | SUCCESS- Each student with the surname appears |
| 43 | The search works where the surname is not matched with any student | Boundary | 8 | ‘Giles’ | No names appear in the table | SUCCESS- No students appear in table and the program doesn’t crash |
| 44 | The table can be reset after searching for a name | Acceptance | 8 | N/A | The table is reset | SUCCESS- The table is reset and all students are shown |
| 45 | The teacher can save the table into an excel document | Acceptance | 8 | N/A | The database is saved onto an excel file | SUCCESS- The table is saved into an excel document |
| 46 | The student can create a flashcard set | Acceptance | 8 | ‘English Flashcards’ | The Flashcard set is created and appears onto the window | SUCCESS- a button appears with the name and setID on and the set is created |
| 47 | The student is able to delete a set of flashcards | Acceptance | 8 | ‘1’-set id | A window appears and the user deletes the set of cards by entering the set id | SUCCESS- The flashcard set is deleted |
| 48 | A user can add two sets with the same name | Acceptance | 8 | ‘English Flashcards’  ‘English Flashcards’ | Both sets are created but with different set IDs | SUCCESS- Both sets are created and appear on screen |
| 49 | A user can add a card to the set | Acceptance | 8 | ‘noun’, ‘An object or thing’ | The card is added and shows up on screen | SUCCESS- The card is created and shows up on screen |
| 50 | Multiple cards can be added | Boundary | 8 | ‘verb’, ‘a doing word’  ‘adjective’, ‘a descriptive word’ | All the new cards are added | SUCCESS- All the cards are added and able to be tested on |
| 51 | The student can traverse the cards in the window | Acceptance | 8 | N/A | The student can view and flip all the cards to allow them to revise from them | SUCCESS- The user can traverse the cards and cycle through them |
| 52 | The student cannot be tested on a set of cards with a length of 3 or below | Erroneous | 14 | N/A | The test button does not appear for until 4 cards are added | SUCCESS- The button only appears when 4 cards are added |
| 53 | A user can start a test with a given number of questions despite the length of the set of revision cards | Acceptance | 14 | ‘10’ | The test cycles through the cards until all have been tested then cycles through again until the question number has been reached | SUCCESS- The program loops through the cards asking questions |
| 54 | The program keeps track of how many questions the user has got right | Acceptance | 14 | N/A | The counter goes up every time a user gets a question correct | SUCCESS- The program only increments the counter when a question is answered correctly |
| 55 | At the end of a test the program shows an end screen | Acceptance | 8 | N/A | A screen is displayed showing the test stats and percentage | SUCCESS- A final screen is shown with the percentage of right questions answered |
| 56 | The percentage changes colour dependant on the result | Acceptance | 8 | N/A | The text changes colour dependant on the result | SUCCESS- The text changes colour |

Remedial Action

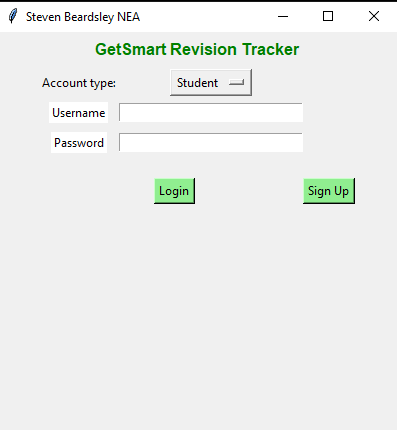
**Test 10A:**

Test number 10 failed and would crash when a user attempted to join a class they were already apart of. I have fixed this problem by adding a form of error checking by checking if an instance of that user in that class already exists inside the database.

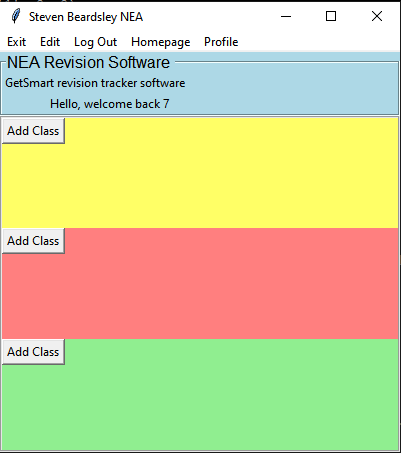
Error checking to see if the instance is in the database

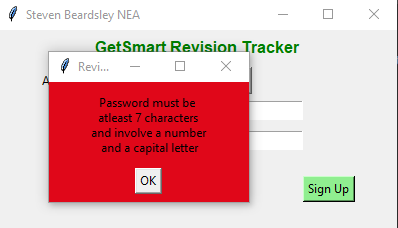
I have retested test 10. The test passed.

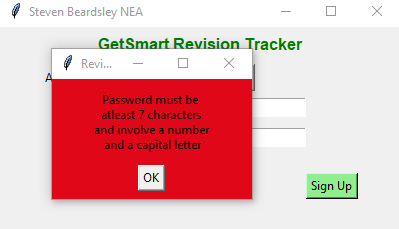
**Testing screen shots**

1)

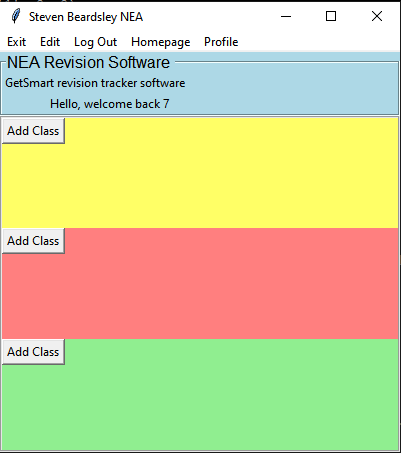
2)

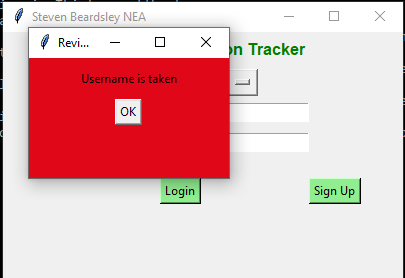


3)

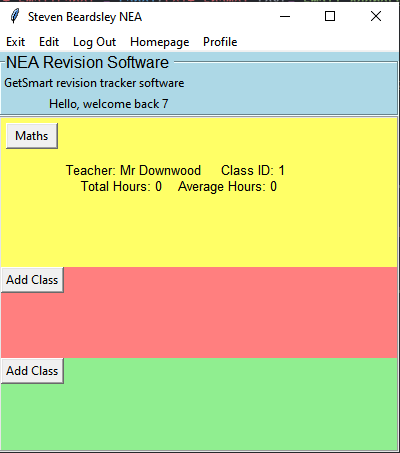
4)

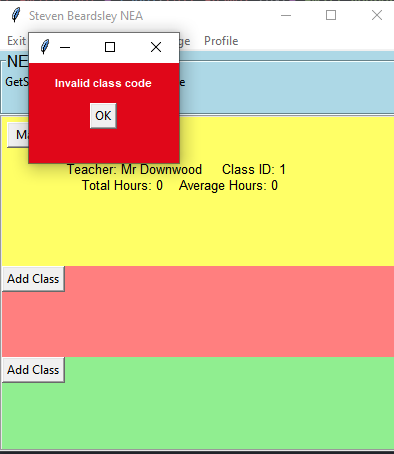
5)



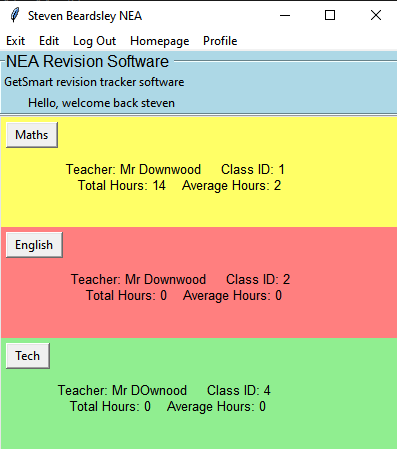
6)

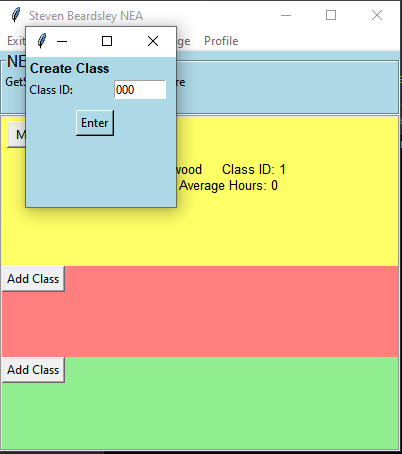
7)

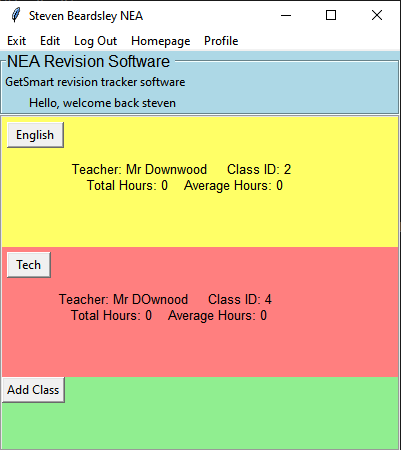


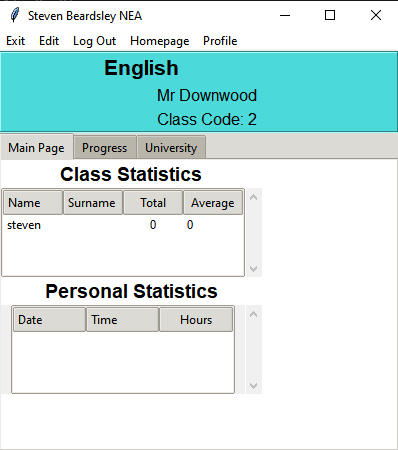


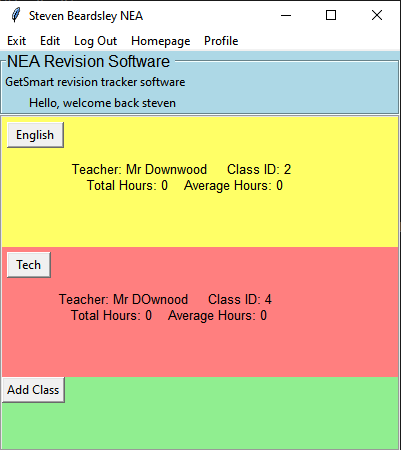
8)

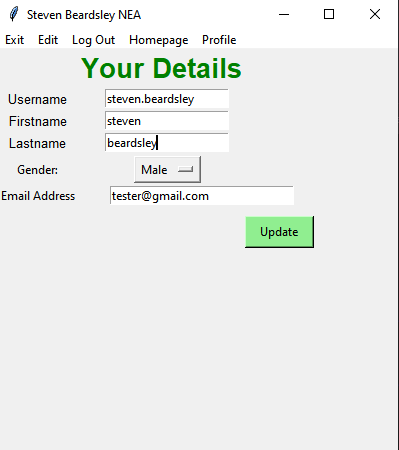
9)

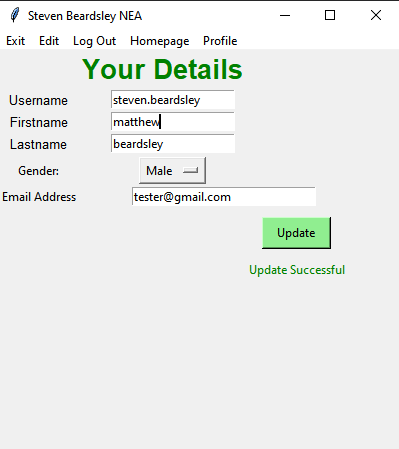
10)

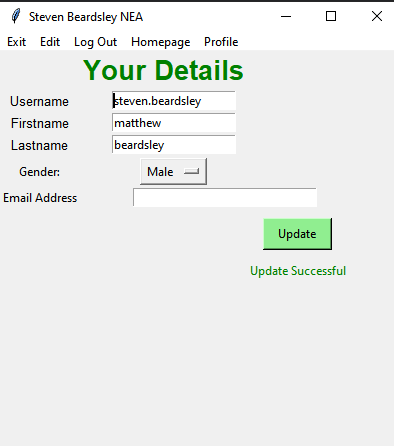
11)

12)

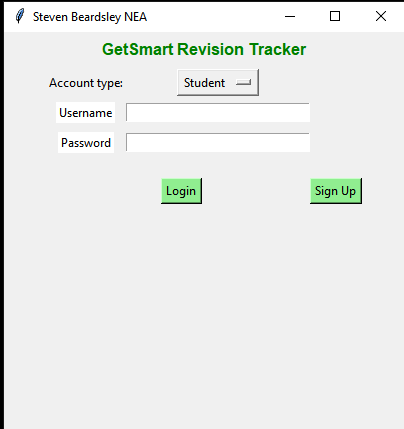
13)

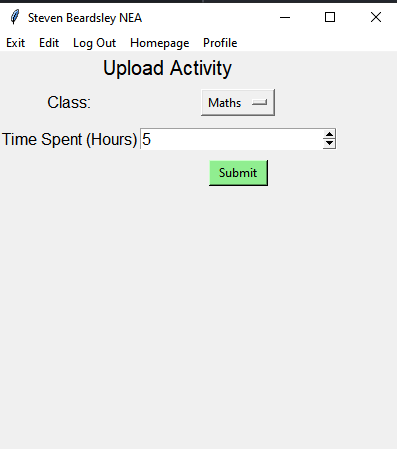
14)

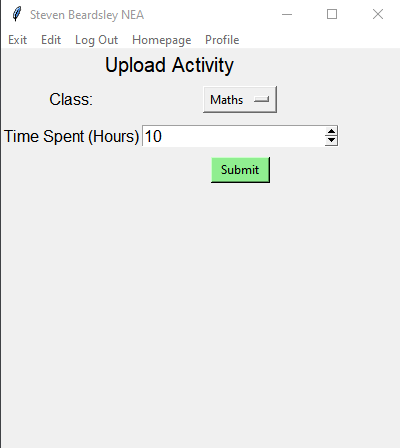
15)

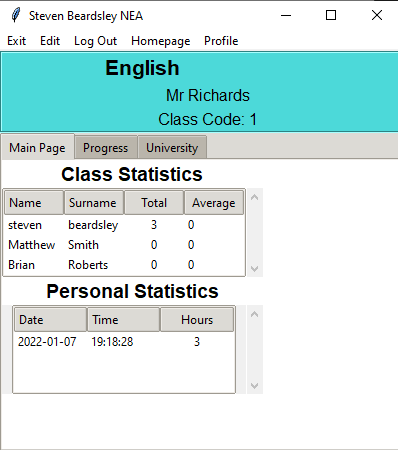
16)

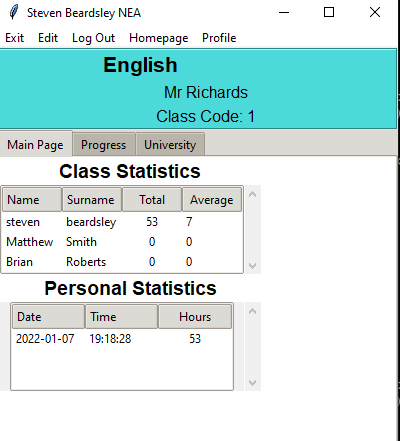
17)

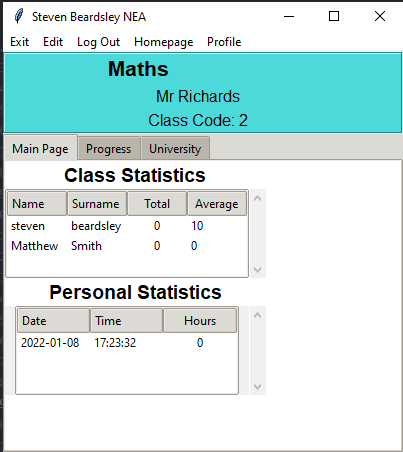


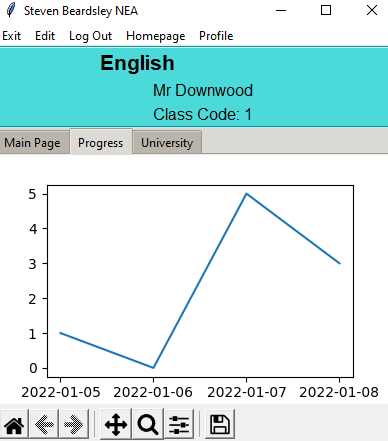
18)

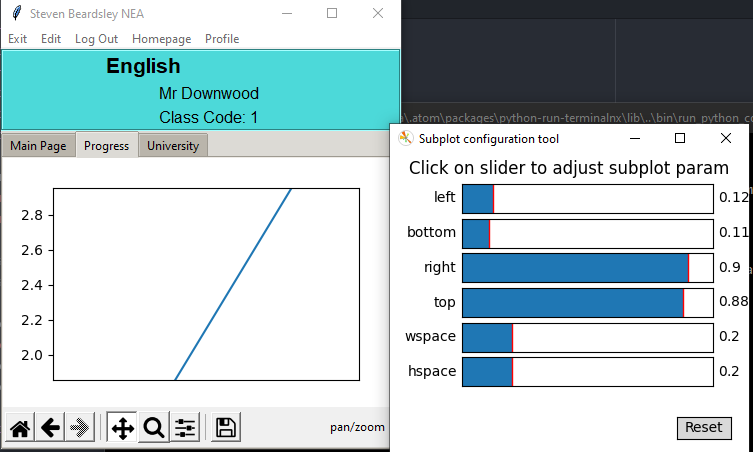
19)

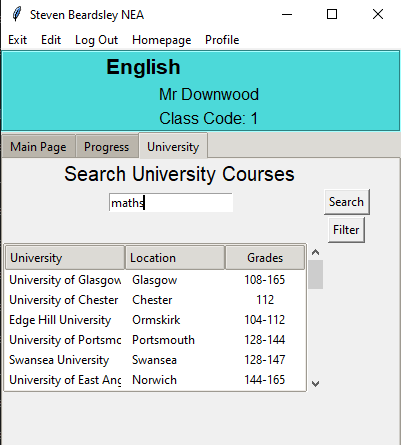
20)

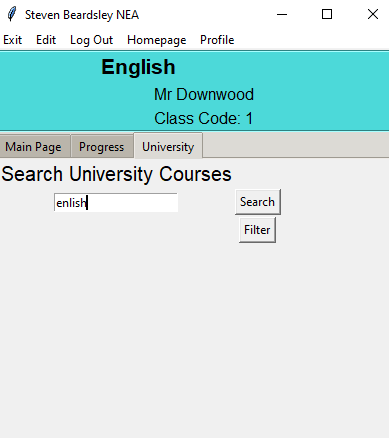
21)

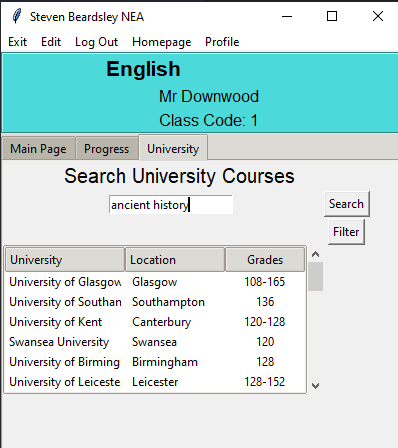
22)

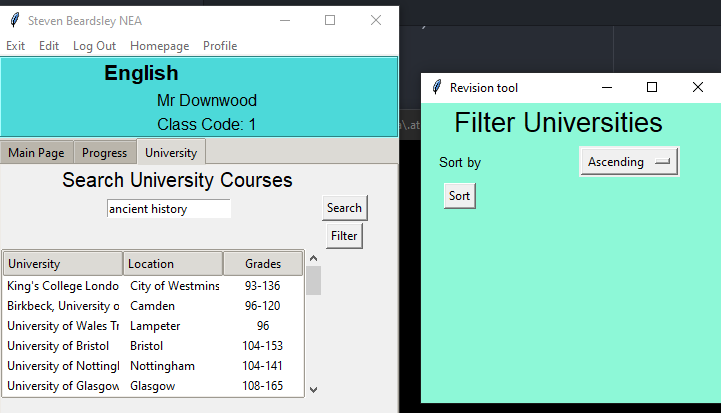
23)

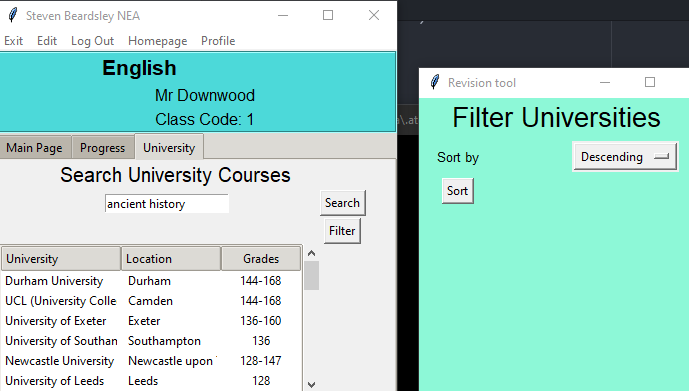
24)

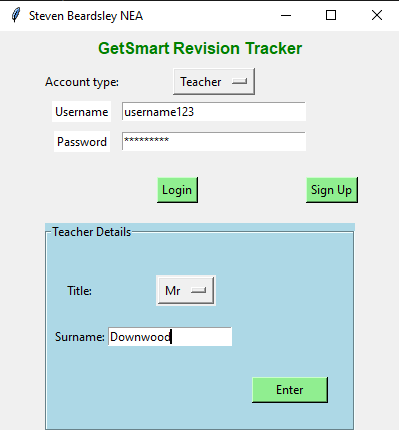
25)

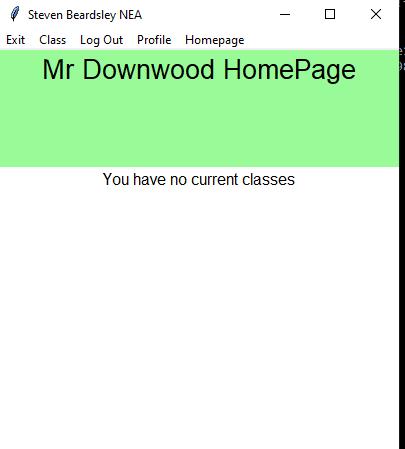
3.26)

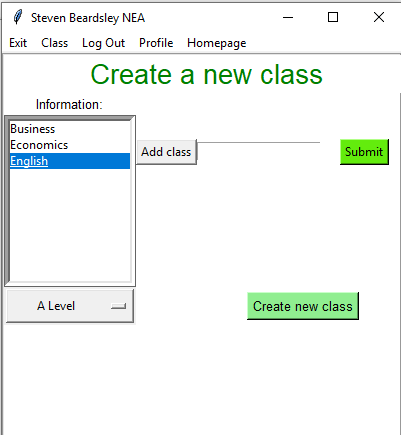
27)

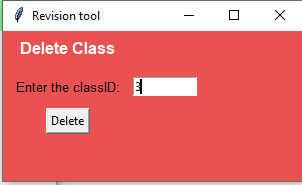
28)

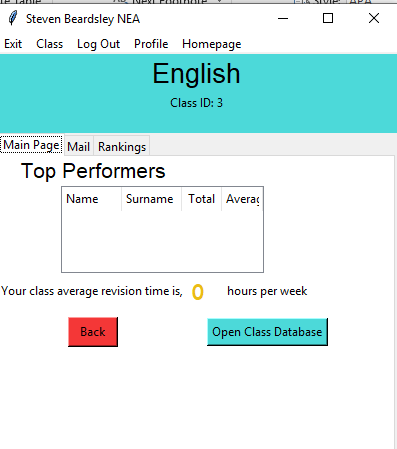
29)

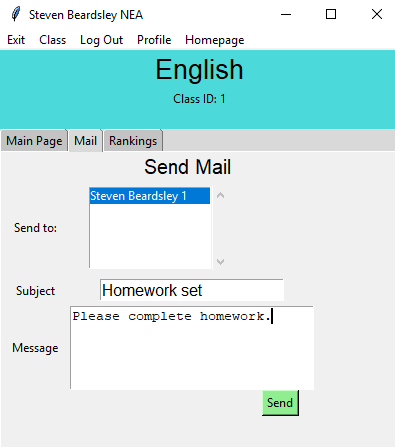
30)

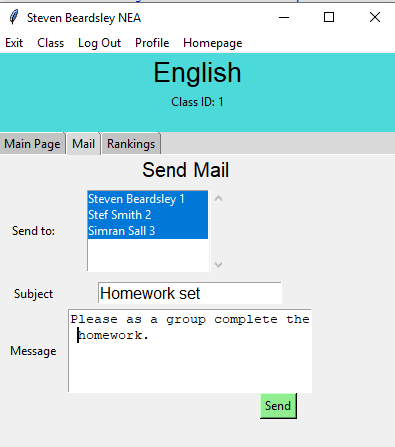
31)

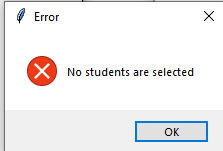
32)

33)

34)

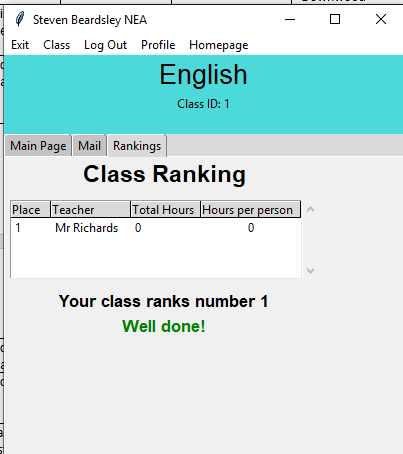
35)

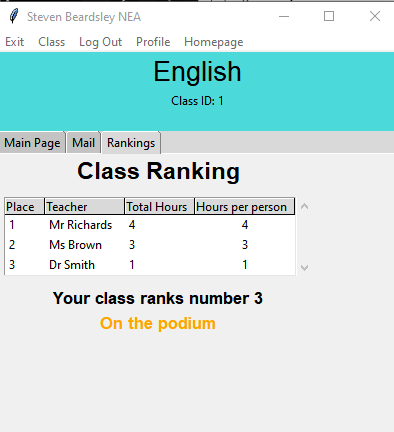
36)



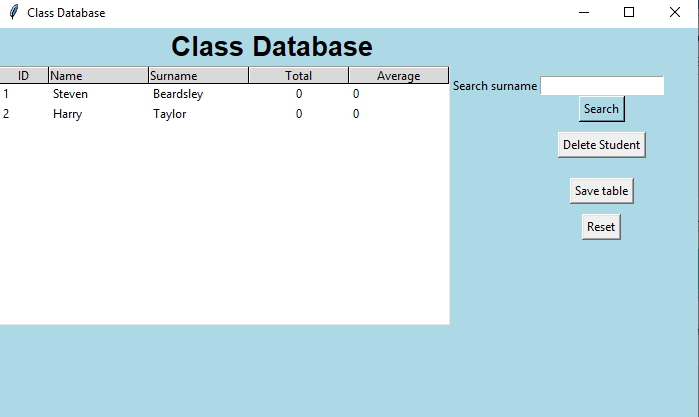
37)

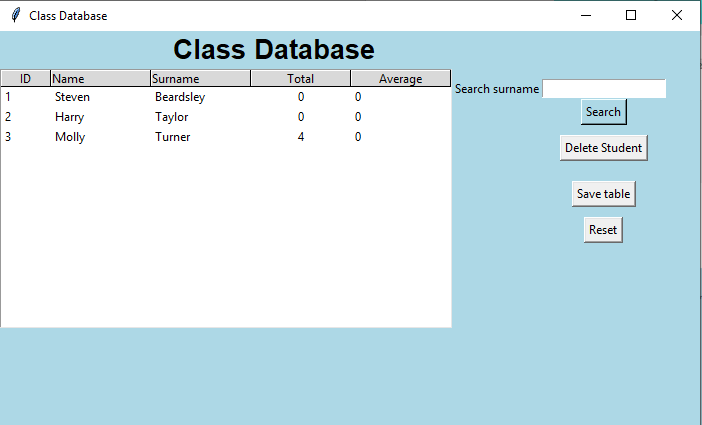
38)



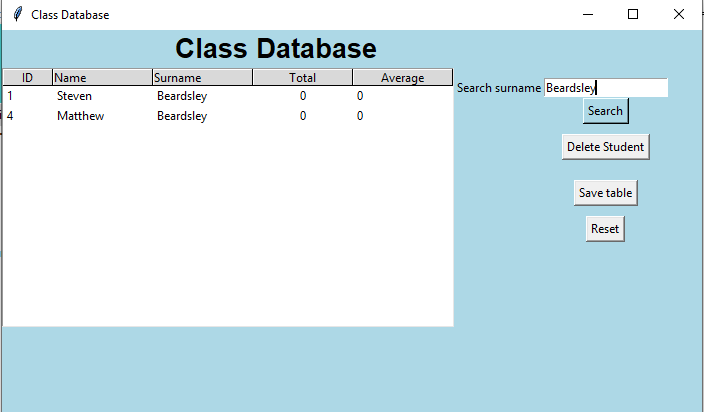
39)

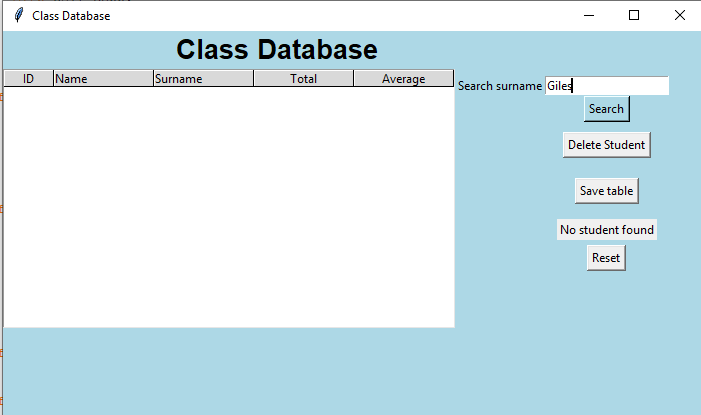
40)

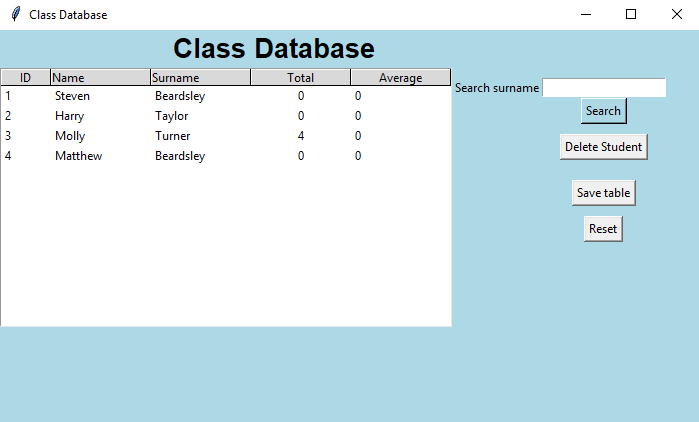


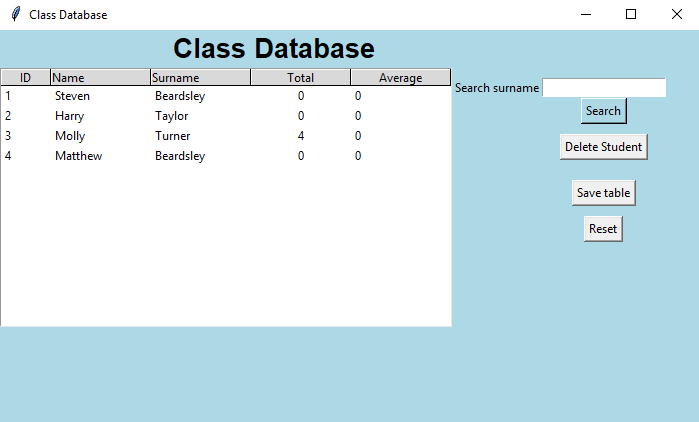
41)

42)

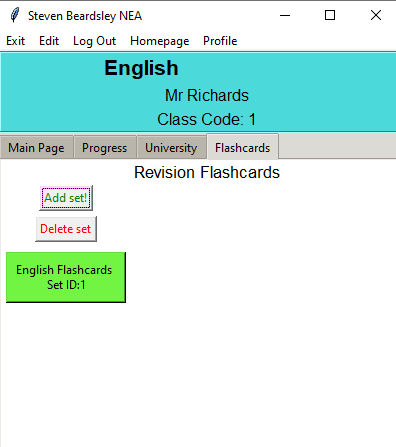


43)

44)

45)

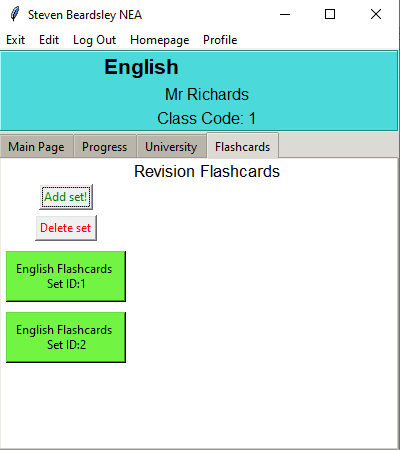
46)

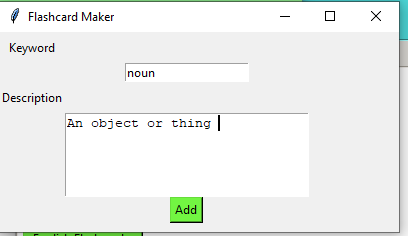


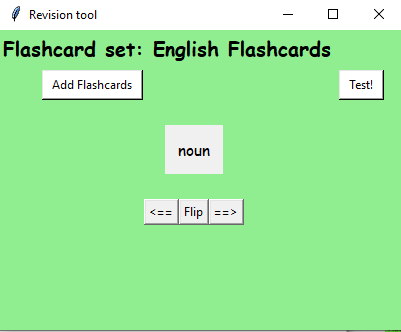
47)

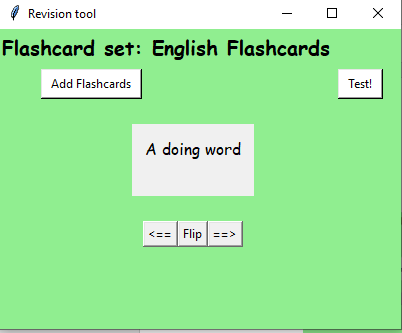


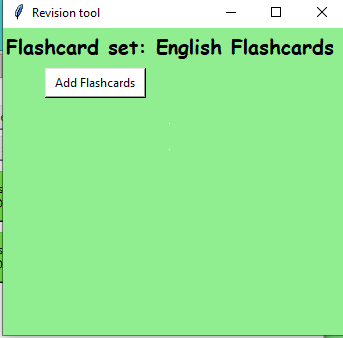
48)



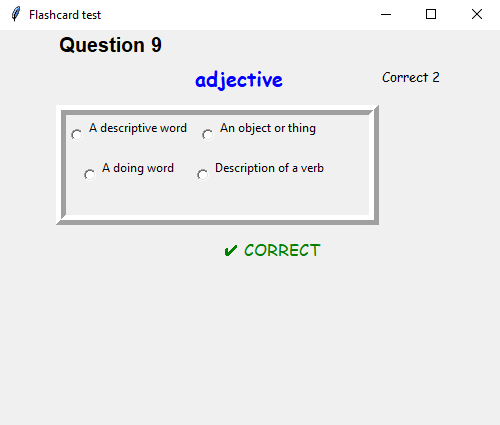
49)

50)

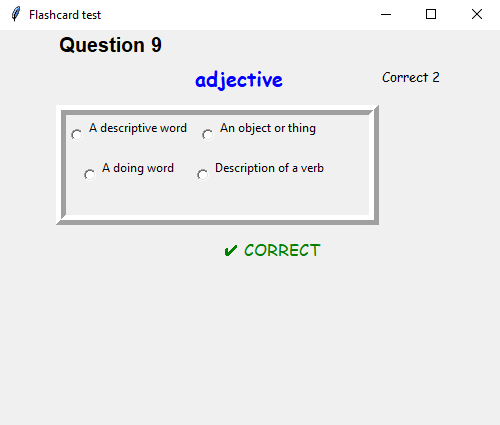
51)

52)

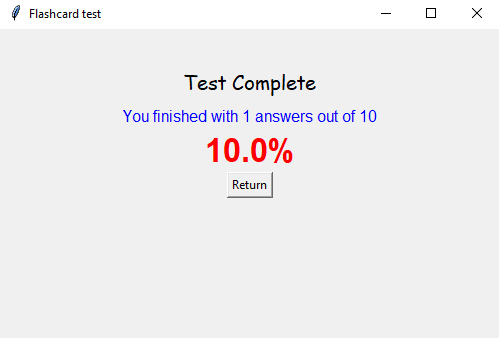
53)

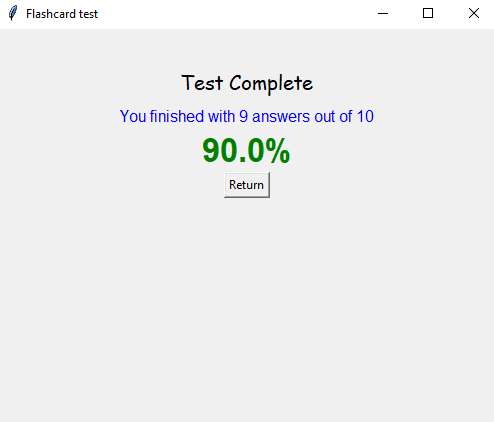


54)



55)



56)

**Evaluation**

In this section I will reflect through my program, comparing my final, completed solution to my objectives whilst getting more feedback from my end user and his thoughts about how successful I have been in my project.

The Project Report

**Self-assessment**

1. Make a revision software which is accessible for students and teachers

I have succeeded with this objective by creating an easy to use and useful program

1. The program must run within 5 seconds and not crash

I have completed this objective, shown by my testing that my program doesn’t crash.

1. The solution must be easy to navigate with buttons and clear label

My solution features lots of buttons and label clearly helping to navigate the user in order to make full use of all the features. I have succeeded at this objective.

1. The solution must allow for students and teachers to create a unique username and password

My solution makes sure each username is unique and passwords match via error checking. I have succeeded.

1. The solution must implement a degree of security when storing sensitive information e.g. passwords

I have tried to achieve this objective by implementing a hashing algorithm when storing user passwords. In evaluation, I could’ve secured more of the users’ data or used an even more complex algorithm.

1. The password must be at least 7 characters long and involve a character

I have perfectly implemented this as users cannot create a password which doesn’t match these requirements.

1. The program must be able to save students accounts and progress

I have implemented this using a stats table. This stores the total hours they have revised for and for what subject. Looking back, I could’ve included a potential description or title for each revision block but I didn’t feel it was necessary.

1. The solution must present the data in a clear eligible way

I have completed this objective with the use of tkinter tree view and by using the module matplotlib. Both of these prevent the users’ statistics in a clear engaging way which make the revision trends easier to spot.

1. The solution must be robust and allow for lots of accounts and classes

I have created a solution where the user can only have 3 classes registered whilst a teacher can have infinite classes. This is because most students will only have a maximum of three classes and it wasn’t needed to include an infinite class for students whereas teachers may teacher any number of classes depending if they work part time or full time.

1. The solution must allow for cross class comparison for teachers

I have created a feature for the teachers to compare their total class revision time against other teachers teaching that subject. This could’ve been improved by allowing teachers to compare against all classes and compare different kind of statistics, average and total hours. This would be a relatively simple thing to implement and if I had more time I would’ve liked to incorporate this into my solution.

1. The solution must allow users to query and update the database

This is achieved as my solution allows users to search the database to help select students and classes. It also allows students to check their profiles and update and change their information, ensuring their account stays current and relevant to their lives.

1. The solution must involve a feature allowing students to look at universities

For student, I have created a feature where the user can search course names and it will return a table featuring the entry grades at each university for that course and the location of that university. To improve this, I could’ve included more information about the course (for example modules and distance away from the user’s information). However, I managed to complete this objective to an acceptable standard.

1. The solution must allow teachers to communicate with students within their classes

In my solution I have allowed teachers to contact students via a GetSmart revision email account creating a one-way form of communication. The feature also allows for students to send the same email to multiple students in their class. One way I could improve this feature is by allowing students to respond. However, this would involve teachers including their own personal emails which they may be reluctant to do in the wake of cyber security and safeguarding.

1. The solution must be robust and have enough error handling to account for user error

As shown through my testing, my product handles all user errors and respond with an appropriate response, allowing the program to keep running and not crash. I have met this objective fully.

User feedback form

In this section I gave my end user a tick sheet involving the objectives from my SMART objective. *See appendix 3.0.* This gave me a chance to see what my end user, Jack, thought of the program first hand and how satisfied he was that they met the objectives.

As shown by the form my end user ticked all the boxes showing he was very happy with the program and felt the program met all the objectives. This is good because it was what the program was designed to do and indicates that the program was a success.

Evaluation- Interview

As a tool to help evaluate my solution I have interviewed my end user, asking questions about what he though about the software and how it may be improved. This also gave me a more in-depth evaluation than the user feedback form. \*\**see appendix 3.0*

This interview was positive and gave me more ideas for things I could add into my coursework in the future. I am happy that the end-user I designed my software for liked the software and would use it on a near daily basis.

One useful point my end user made was the implementation of a ‘quality’ scale to track ‘quality not just quantity’. I also think this would’ve been a useful feature and one where if I had more time, I would’ve certainly liked to look into the algorithm behind it.

*Final Conclusion*

Overall, I am happy with the standard and completeness of my solution. It provides a clear and easy framework for students to use whilst revising, helping to keep them motivated and competitive against their classmates, and alternatively enables teachers to track their student’s revision in an attempt for them to work off of the data collected to enhance the students overall learning. I have also included different features including flashcards and a university search bar as a way for students to be engaged with my software whilst still keeping the core focus of academics at heart. My solution is robust and my end user was able to, and continue to use it on a daily basis.

**Appendix**

1)Interview transcript with my end user:

1. **Hello Jack I’m just going to ask questions related to my coursework. Can you tell me how important revision is and what it means to your exam results?**

*Revision for me is very important relative to my exam grades. It helps build a foundation and gives me confidence going into the exam. I try to do it frequently so I have consistent results across all subjects.*

1. **How do you avoid procrastinating during your revision sessions?**

*I often procrastinate, however, I set myself time limits so I achieve my deadlines in time to a good standard. I most commonly procrastinate by spending time on my phone which I need to avoid.*

1. **Are there any revision techniques you haven’t tried and would like to which you’ve heard about?**

*I rarely use flashcards actively as they are often hard and awkward to make as proposed to a poster. I often end up losing or miss ordering them which inhibits my ability and motivation to revise making it harder to inspire myself to revise. A feature that would fix this and make flashcards quick and accessible would be great!*

1. **What do you know about other people’s revision habits?**

*Not much honestly, I tend not to ask and try and focus on my own revision.*

1. **What features would you like to see in a revision tracker software which would benefit you?**

*Time spent revising, the quality of my revision and my memory of the subject revised afterwards.*

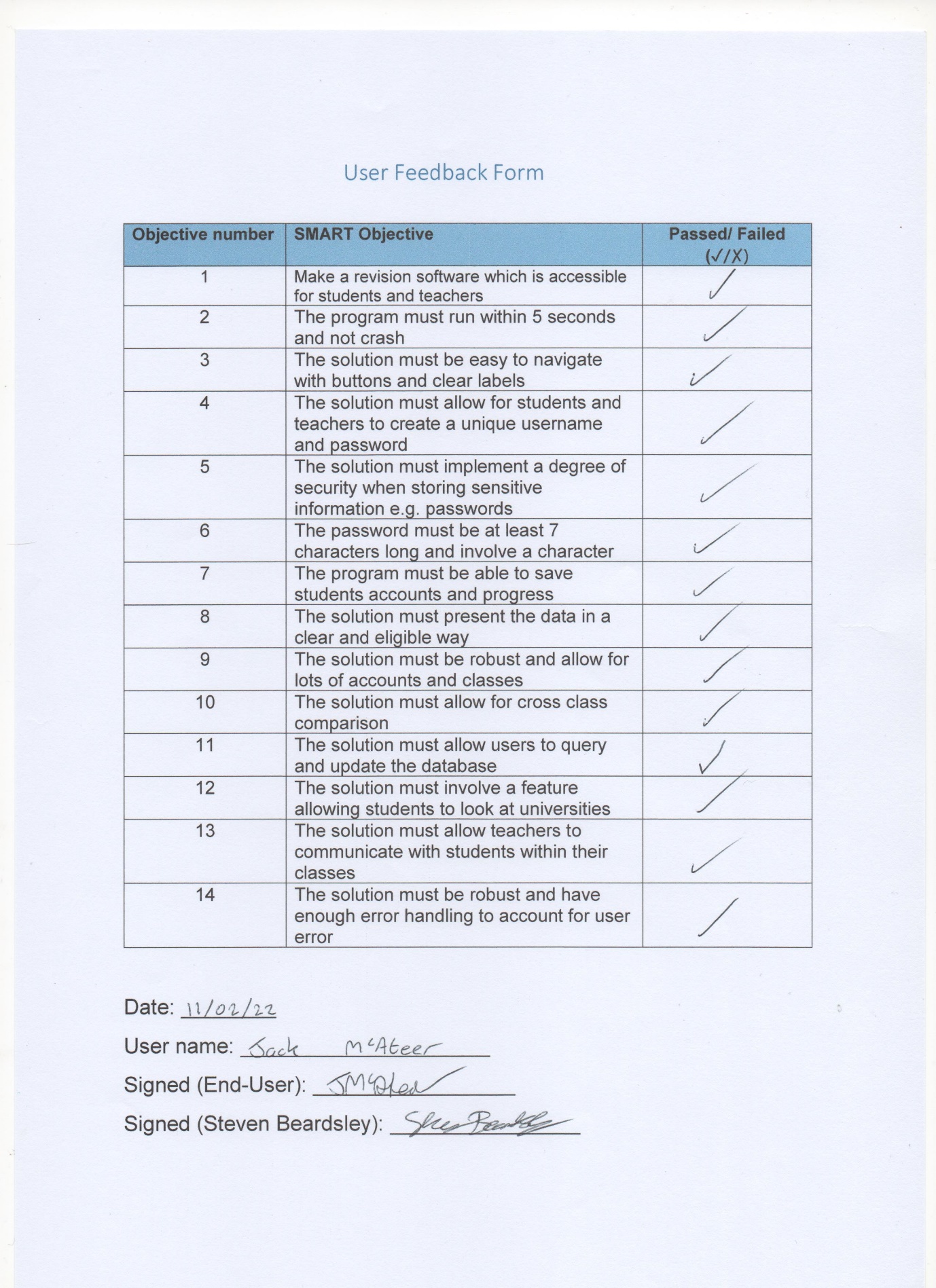
1. **Would being able to see all your revision and statistics be useful and motivating for you or maybe put you off revising?**

*It will be very useful seeing my development over time and where my weaknesses lie so I can focus on them later.*

1. **Do you find working in a team effective and are you often comparing your revision against someone else’s?**

*Yes, I find working in a team very effective as it helps drive me to do better. I rarely compare my revision to others.*

3.0) User feedback form



4.1) Evaluation Interview

1. **Hello Jack, what did you think of the software as a whole?**

*It was of a good quality and it met all of the requirements it said it needed to. Some aspects needed clarity, but they were minor errors and the software said it was an error clearly*

1. **What features of the software were your favourite?**

*On the teacher’s side I liked how they could compare class statistics and it would give data over a period of time, it was very functional and would be good in actual practice.*

1. **Can you imagine using this on a near daily basis as a way to track your revision?**

*Yes, as it was easy enough to use, and it was able to tell me how much I had revised for, and will allow for me, in the long term, to track my growth.*

1. **How did you think the software looked, was it easy to navigate as a student who’d never seen the program before?**

*It was fairly clear, however there were some functions which needed to be clearer, however the concept was successful and it generally was easy to navigate through the software.*

1. **Do you think the program met all the objectives we agreed upon?**

*Yes, I think it met all of the objectives, was up to a high enough standard and I think that some of them were completed to a higher degree than others, but all of them to a good enough level.*

1. **Did you think the university tool was helpful?**

*It was useful for sixth form students, who are looking for a course or specific university in mind, however, if you are unsure of what you want to do, it wouldn’t have been very useful.*

1. **Are there any other features you would want added in the future or you feel the program is missing?**

*Within the data of “hours completed”, maybe a rating system of how the quality of revision went would be beneficial, as then you could track quality aswell as quantity.*