**A Level computer Science**

Component 3

Physics Homework App

A black and blue logo

Description automatically generated

By: Konstantinos Papadopoulos

For: A. Issa

**Stoke on Trent Sixth Form College**

Table of Contents

Table of Contents

[Chapter One: Analysis of the problem 5](#_Toc194400186)

[1.1 Introduction 6](#_Toc194400187)

[1.2 Problem Identification 6](#_Toc194400188)

[1.3 Why the problem is suited to a Computational Method 7](#_Toc194400189)

[1.4 Stakeholders analysis 8](#_Toc194400190)

[1.4.1 Stakeholder Introduction 8](#_Toc194400191)

[1.4.2 Stakeholder Interviews 8](#_Toc194400192)

[1.4.3 Stakeholder Conclusion 11](#_Toc194400193)

[1.5 Research of solutions for similar problems 12](#_Toc194400194)

[1.5.1 S**imilar existing solutions** 12](#_Toc194400195)

[1.6 Hardware and software requirements 14](#_Toc194400196)

[1.6.1 User requirements 14](#_Toc194400197)

[1.6.2 Solution requirements 14](#_Toc194400198)

[1.7 The requirements of the solution 15](#_Toc194400199)

[1.8 Limitations of the solution 16](#_Toc194400200)

[1.9 Success Criteria 17](#_Toc194400201)

[1.9.1 Universal success criteria 17](#_Toc194400202)

[1.9.2 Teacher/Admin success criteria 17](#_Toc194400203)

[1.9.3 Student success criteria 18](#_Toc194400204)

[Chapter Two: Design 19](#_Toc194400205)

[2.1 Introduction 20](#_Toc194400206)

[2.2 Decomposition of the problem 20](#_Toc194400207)

[2.2.1 Decomposition Diagram 20](#_Toc194400208)

[2.2.2 Flow Charts / Data Flow Diagram 20](#_Toc194400209)

[2.2.3 Input Process Output Charts 22](#_Toc194400210)

[2.3 How All Solution Parts are Linked 24](#_Toc194400211)

[2.3.1 State Diagrams of the different sections 24](#_Toc194400212)

[2.3.2 How different functions /classes are connected 26](#_Toc194400213)

[2.4 Database Design 28](#_Toc194400214)

[2.4.1 Data dictionaries 28](#_Toc194400215)

[2.4.2 Entity Relationship Diagram 32](#_Toc194400216)

[2.4.3 SQL Pseudocode 32](#_Toc194400217)

[2.4.4 Test plan for database 35](#_Toc194400218)

[2.5 Design of other Parts of the Solution 36](#_Toc194400219)

[2.5.1 Login System 36](#_Toc194400220)

[2.5.2 Admin View 40](#_Toc194400221)

[2.5.3 Student View 52](#_Toc194400222)

[2.6 Stakeholders involvement 57](#_Toc194400223)

[2.6.1 Stakeholder feedback 57](#_Toc194400224)

[2.6.2 Stakeholder conclusion 58](#_Toc194400225)

[2.7 Testing plan to inform evaluation 58](#_Toc194400226)

[Chapter Three: Iterative Development and Testing 60](#_Toc194400227)

[3.1 Introduction 61](#_Toc194400228)

[3.2 Stage 1: Building the Database 61](#_Toc194400229)

[3.2.1 SQL and Python Code 61](#_Toc194400230)

[3.2.2 Testing 83](#_Toc194400231)

[3.2.3 Review 88](#_Toc194400232)

[3.3 Stage 2: Building The Log In 88](#_Toc194400233)

[3.3.1 Python code 88](#_Toc194400234)

[3.3.2 Testing 92](#_Toc194400235)

[3.3.3 Stakeholder Feedback 95](#_Toc194400236)

[3.3.4 Corrective actions 95](#_Toc194400237)

[3.4 Stage 3: Building The Validation 102](#_Toc194400238)

[3.4.1 Python code 102](#_Toc194400239)

[3.4.2 Testing 104](#_Toc194400240)

[3.4.3 Corrective actions and further testing 106](#_Toc194400241)

[3.4.4 Review 109](#_Toc194400242)

[3.5 Stage 4: Building The Admin View 109](#_Toc194400243)

[3.5.1 Python code 109](#_Toc194400244)

[3.5.2 Testing 131](#_Toc194400245)

[Comment on the result 132](#_Toc194400246)

[Show changes/correction and further testing 132](#_Toc194400247)

[3.5.3 Stakeholder feedback 132](#_Toc194400248)

[Review 132](#_Toc194400249)

[3.6 Stage 5: Building The Student View 132](#_Toc194400250)

[3.6.1 Python code 132](#_Toc194400251)

[3.7 Final Review, Improvements and Corrective Actions 133](#_Toc194400252)

[Chapter Four: Evaluation 136](#_Toc194400253)

[4.1 Introduction 137](#_Toc194400254)

[4.2 Testing to inform evaluation 137](#_Toc194400255)

[4.3 Evaluation 138](#_Toc194400256)

[Evaluating usability features 138](#_Toc194400257)

[Evaluating Robustness 138](#_Toc194400258)

[Limitations and Maintenance 138](#_Toc194400259)

# Chapter One: Analysis of the problem

## Introduction

I will be creating a Physics Homework App which will allow teachers to check homework and check students’ performance. It will also allow students to complete their homework. To complete this project, I will be collaborating with the Physics department in my school and some Physics students. They will be stakeholders for this project and they will give me ideas and recommendations of what features to have.

This chapter will outline the problem identification process and will document the process of planning the proposed solution.

## Problem Identification

While studying Physics at A-Level, we used multiple websites to do our homework. This was due to the fact that all of the different websites had their own flaws. The main flaw that I found is that no website that is available to us allows for both calculations and long answers to be entered as answers.

This is an example of a website allowing to answer a calculation question. As you can see in this example, you can only enter a value as an answer and then the website tells you if it is right or wrong. This is a big problem because, if you get the question wrong, you and your teacher will not be able to see where the problem in the calculation is and how to fix it.



As shown in this example, this particular website allows for a long answer to be entered as an answer. However, the limitation of this is that when you are marking your answer, it can only be marked as right or wrong. The solution to this problem is to allow the student to give their answer a mark, by checking their answer against the mark scheme, out of the number of marks that the teacher set. This will allow the student to see what they would score in the real exam.

## 1.3 Why the problem is suited to a Computational Method

Accessibility

A computational solution would improve accessibility to resources by allowing students to complete their work anytime and from anywhere in the world as long as they have an internet connection. This would allow for students that are absent to be able to still complete their work and stay up to date with the content that is taught. Additionally, a computational approach would allow students that have disabilities or bad motor skills to be able to complete their work. This is because, typing requires less physical effort which can strain and be uncomfortable to a person that is suffering from an injury or disability. This means that all students would be able to work and learn no matter what their situation is. As a result, a computational method is suitable to be used.

Usability

Using a computational method to solve my problem would allow for multiple users to access and complete their work simultaneously. Students can complete and mark their assignments while their teacher can view their overall progress. This would be a more efficient and faster approach than the teacher marking each completed assignment and then returning it to the student with feedback. Also, a computational method would allow for a simple user interface that is easy and quick to navigate. Other methods that students use can be very complicated and confusing (for example a file directory or a physical folder with printed worksheets), so a simple UI with the main features in a main window would streamline the user’s experience. Due to this, a computational method can be used to solve the problem.

Storage

When a student is assigned homework, they are given sheets of paper. These pieces of homework can be easily lost. Also, over time, the student would need a lot of physical storage space to store their past and current homework. A computational solution can provide a large amount of storage space that can store the numerous pieces of data. This can be achieved using databases hosted on an online server. This can allow for efficient data organisation, quick and easy deletion and creation of records and an easier analysis of data. Using data analysis, teachers can effectively look for trends in students’ performance and adapt their teaching style accordingly. This means that any important data will not be lost, due to the nature and backups of the database, and it will cut down on the use of paper which will benefit the environment. Hence, a computational method would be suitable and preferred for this solution.

Repetition

Setting and marking homework is a very long and repetitive process. An application would remove the need for the repetition of writing and printing out homework and giving it to students. Also, an application would remove the need for teachers to mark each homework for every student that they teach. As a result, the homework process would be less repetitive and more streamlined. Thus, a computational solution is suitable and preferable.

## 1.4 Stakeholders analysis

1.4.1 Stakeholder Introduction

The demographic for my project is for teachers and students in the Physics A-Level. For my stakeholders I picked one Physics teacher and three Physics students. The teacher stakeholder is Mina and my student stakeholders are Ali, Jahin and Violet. I picked a teacher and three students for my stakeholders because I wanted to have many different perspectives where I can take ideas from. I asked each group of stakeholders seven questions. I asked the teacher group the following questions:

1. What resources do you currently use to set physics homework?
2. What are some good features of the current resources that you use?
3. What are some bad features of the current resources that you use?
4. If a new homework website was made, what would you like to see included in it?
5. As a teacher, what are the biggest challenges when setting and marking homework?
6. What would make the website easier to navigate for you?
7. Anything you would like to add that wasn’t included in the questionnaire?

I asked the student group the following questions:

1. What resources do you currently use for your physics homework?
2. What are some good features of the current resources that you use?
3. What are some bad features of the current resources that you use?
4. If a new homework website was made, what would you like to see included in it?
5. What would make the website easier to navigate for you?
6. How can the website motivate you to do your homework?
7. Anything you would like to add that wasn’t included in the questionnaire?

1.4.2 Stakeholder Interviews

My teacher stakeholder interview was with Mina, an A-level Physics teacher. She answered my questions as follows:

What resources do you currently use to set physics homework?

“I use SENECA and Bromcom”

What are some good features of the current resources that you use?

“SENECA provides active learning which can be very beneficial for a student. Bromcom allows for the parents to easily check their kid’s progress, homework and behaviour at school.”

What are some bad features of the current resources that you use?

“Both websites have too many steps when setting a homework”

If a new homework website was made, what would you like to see included in it?

“I’d like to see a feature allowing the teacher to check a student’s progress”

As a teacher, what are the biggest challenges when setting and marking homework?

“Time. Setting a homework can be very time consuming because it involves a lot of steps”

What would make the website easier to navigate for you?

“Limiting the number of clicks required to reach important pages and actions. Also, a clean and uncluttered design”

Anything you would like to add that wasn’t included in the questionnaire?

“No”

My first student stakeholder interview was with Ali. He currently does Computer Science, Maths and Physics at A-Level. His responses were as follows:

What resources do you currently use for your physics homework?

“I currently use Isaac Physics and Carousel Learning for my homework”

What are some good features of the current resources that you use?

“I really like that there is a menu that allows you to review new and completed assignments. I also really like that there are hint options for when I am struggling with a question”

What are some bad features of the current resources that you use?

“One thing I dislike about Carousel Learning is that if I accidently go to the next question, I can’t go back to the previous question. I also really dislike that I can’t do theory questions on Isaac and calculations on Carousel which makes it really inconvenient to do my homework. Another thing I dislike about my current resources is that I don’t get notifications when new homework is set.”

If a new homework website was made, what would you like to see included in it?

“I would really like to see a feature that show how much progress I am making on my homework and how much progress I am making overall.”

What would make the website easier to navigate for you?

“My current resources have a bunch of useless stuff on the screen which makes it hard to navigate so I think a clean UI and design would make the website easier to navigate for me.”

How can the website motivate you to do your homework?

“A point reward system or a praise system when I get a question right and also a class leaderboard to see where I am compared to my classmates”

Anything you would like to add that wasn’t included in the questionnaire?

“No”

My next student stakeholder interview was with Jahin. He is studying Physics, Computer Science and Further Maths. His responses were as follows:

What resources do you currently use for your physics homework?

“I use past papers, Isaac Physics and Carousel Learning”

What are some good features of the current resources that you use?

“They let me work at my own pace, there’s no timers that rush me and stress me out”

What are some bad features of the current resources that you use?

“Some resources don’t have hints on how to solve questions I am stuck on”

If a new homework website was made, what would you like to see included in it?

“An option for hints for when I am stuck on a hard question”

What would make the website easier to navigate for you?

“A simple UI with no over-the-top styling”

How can the website motivate you to do your homework?

“Maybe a leaderboard comparing you against classmates”

Anything you would like to add that wasn’t included in the questionnaire?

“I would like to see that the app has a low bandwidth usage because the Wi-Fi in some places is really bad”

My final student interview was with Violet. She is studying Physics, Maths and Geography. Her interview went as follows:

What resources do you currently use for your physics homework?

“I use Physics and Maths tutor, questions from the textbooks and slides from the teachers”

What are some good features of the current resources that you use?

“They are easily accessible and have a wide range of topics”

What are some bad features of the current resources that you use?

“They have bad layouts which makes it hard to find the resources I need sometimes”

If a new homework website was made, what would you like to see included in it?

“I would like to see a main screen with all important features included in that screen”

What would make the website easier to navigate for you?

“A better layout with no clutter”

How can the website motivate you to do your homework?

“Some kind of praise system would be great”

Anything you would like to add that wasn’t included in the questionnaire?

“No”

1.4.3 Stakeholder Conclusion

From the interviews I conducted with my stakeholders, I can conclude the following:

What are some good features of the current resources that you use?

My stakeholders liked that the current resources let them work at their own pace. As a result, I will not be adding any timers to the assignments as this could stress out a student. My stakeholders also liked that the resources that they use are easily accessible. Due to this, I will make my solution into an executable file that will require minimal computer resources.

What are some bad features of the current resources that you use?

My stakeholders disliked that they have to use two different applications for their homework assignments. Due to this, I will allow for both calculation and written questions to be set and answered. My stakeholders also disliked that there are no hint options on their homework. This is a good feature to have, however, I will not be incorporating this feature into my solution. I chose to remove this feature because, if the student is stuck on a question, it would be more helpful to their education for them to do the research/revision by themselves instead of getting multiple hints that will get them to the answer. Also, if this feature is removed, the homework setting process would be much quicker and easier (this was the main problem that my teacher stakeholder identified).

If a new homework website was made, what would you like to see included in it?

My teacher stakeholder outlined that they would like to see a feature that would allow for them to check the students’ progress. My student stakeholders also identified this as a feature that they would like to see included. As a result, I have decided to include this feature to my solution as it was the main feature that my stakeholders wanted to see and I believe that checking your progress is vital to improving.

What would make the website easier to navigate for you?

All my stakeholders identified that they want a clean and uncluttered user interface that is easy to use. I will incorporate this into my solution by removing unnecessary buttons and having a main window with all of the important actions.

## 1.5 Research of solutions for similar problems

1.5.1 S**imilar existing solutions**

Seneca Learning

How does it work?

A screenshot of a video game

Description automatically generatedOne similar solution to my problem that I found is a website called Seneca Learning. This website has pre-made modules for many subjects and exam boards. The layout for homework does not allow students to do calculations. When completing a homework on Seneca Learning, the answer is given to you as an explanation and you are then given a question to answer (usually a complete the sentence question). This is website would be really good for students at the GCSE level but for A-Level students it would not be useful.

What components and features can be borrowed?

Seneca learning has many features that are very useful for a student to have. It has a class system that allows for teachers to create their own classes and then set homework for all the students in those classes. This feature will be borrowed because it will make it easy for teachers to set homework for their classes and it would allow teachers to group their students in their respective classes.

One feature that I find to be a bad feature, is the ability for a student to create their own account. I think this is bad because, when students are signing up, they can choose if they are a student, teacher or parent. Having access to this feature means that a person with malicious intentions can create multiple teacher accounts and cause a big problem in a school system. Due to this, for my solution, I will only allow teacher to create student accounts and then having the system send the student an email with their details.

Carousel Learning

How does it work?

Another similar solution I found to my problem is Carousel Learning. In this solution, teachers set homework (the teachers enter the questions and answers as there are no premade modules) and then students answer and mark their answers. When marking, the students can only mark their answers as correct or incorrect.

This solution is a website. Students access the website via a link that is sent to them by their teacher. This link takes students straight to the assignment (there is no student area where they can view progress/upcoming assignments). When the students are directed to the assignment, they log in using their first and last name (their accounts are created by their teacher and no password is required).

What components and features can be borrowed?

From this solution, I can only borrow the feature where the student marks their own questions. However, I will allow the students to assign themselves a mark instead of marking their answer as correct or incorrect. I am doing this as it is more beneficial for the student to see what mark they would have scored in the exam rather than seeing if their answer is correct.

I believe that the login system that the website currently uses is a big security issue as the users don’t have passwords. This would allow students to log in to other students’ accounts as they only need their name. As a result, I will have a main login window where the students/teachers need to enter their email address and password.



Up Learn

How does it work?

A similar solution to my problem is Up Learn.

What components and features can be borrowed?

## 1.6 Hardware and software requirements

1.6.1 User requirements

My solution will require a computer with the standard I/O devices (keyboard, mouse and monitor) so the user can interact with the application. My solution is a python based application. However, the user will not need to have python installed as the application will be compiled into an executable file. An internet connection will also be required so the application can access the database. This will be needed so the user can log in and see their assignments. The following requirements are the minimum requirements needed for the application to run. Most computers and laptops will already meet the requirements as python executable files are not very demanding when it comes to hardware and software requirements.

Hardware:  
- Processor: Any multicore processor. This will allow for faster computations and more efficient multitasking.

- RAM: At least 1GB as the application has a GUI and it will run slow with less RAM. (this amount is not required but it is recommended)

- Storage: At least 100MB to store the executable file and any temporary files.

- I/O devices: Keyboard, mouse and monitor. This will allow the user to use and interact with the application.

Software :

- Operating System: Windows 8 or above, macOS or a Linux based system. These are the current operating systems supported by python.

- Other software will not be needed as all the libraries used will be already bundled with the executable file.

1.6.2 Solution requirements

My solution will require an online PostgreSQL database. This is because it needs to store all the login and assignment information. A local database will not be suitable for my solution as there will be multiple teachers and students that need to access the database from different devices at the same time. An online database would ensure maximum data security as the data can be backed up and encrypted. Also, if an online database is used, the overall memory space needed for my solution would be reduced as the user wouldn’t need to store the database in their device. As a result, my solution will require a webserver to host the database.

## 1.7 The requirements of the solution

Using my stakeholder interviews and research into similar solutions, I decided that the following features would be included in my solution.

|  |  |  |
| --- | --- | --- |
| Feature | Description | Justification |
| Login System | Users should be able to login to the system with their credentials | This would allow for all users to access their personal area with their data. This would also allow for different people with different permissions (admin/student) to access the correct area. |
| Two-factor authentication system | Users should receive an email with a one time password and enter it correctly to log in. | This adds an extra level of security. A person with malicious intentions would need to also have access to the user’s email account. |
| Ability to change email and password of the user’s account | Users should be able to modify their account email and password | This would allow users to change their emails just in case they have lost access to their email account. Users should be able to change their password increases security as they can personalise their password. Also, giving users this ability means that they won’t have to contact admins to change their password and compromise security. |
| Ability to create assignments (Teachers only) | A teacher should be able to create an assignment and add questions to the assignment | Creating an assignment is a core part of a homework program. Teachers should be the only users that can set a homework because students with malicious intent could overload the system. |
| Ability to create and delete classes (Teachers only) | A teacher should be able to create and delete classes | A teacher should be able to group students in their respective classes. This would allow for assignment setting to be easier as the teacher won’t have to add every student to the assignment individually. |
| Ability to view active/past assignments and submissions (Teachers only) | A teacher should be able to view the current and past assignments. They should also be able to view the submissions to those assignments. | This would allow for teachers to monitor student progress and performance. |
| Ability to create user accounts (Teachers only) | A teacher/admin should be able to create accounts for students | Only teachers and admins should be able to do this. This is because, if a student had malicious intent, they could create many accounts and flood the databases and system with fake accounts |
| Ability to see active assignments (Students only) | A student should be able to see all their active assignments on the main screen | A student should be able to view their assignments on the main screen. This will allow the assignments to be accessible and easy to find at any time. |
| Ability to view past completed assignments (Students only) | A student should be able to see their past assignments and their scores on those assignments | Allowing a student to look at past assignments would allow the student to look at their past performance and progress. This would allow the student to see what topics they didn’t do well in and revise those topics. |
| Ability to complete assignments (Students only) | A student should be able to answer all the questions in an assignment and mark their answers | A student should be able to mark their own answers. This is because it would help them see what topics they perform bad in and it would allow them to see what they would have scored in the actual exam. |

## 1.8 Limitations of the solution

One of the main limitations of my solution is that it will only be available on a computer or laptop because these platforms offer better specifications than phones (such as faster processors and more storage space. This would lead to a smoother performance and overall better experience while using the application. Therefore, while the application runs better on laptops and computers, it will be inaccessible to users that do not own such a device and to users that prefer to work on their mobile devices. This could be improved if the solution was hosted on a website instead of an application.

Another limitation of the solution is that it is limited only to the Physics subject as it is designed so students can answer long answers and calculations. The current design could also be used for other subjects like Biology and Computer science but it could only be used for some types of questions. The solution could be altered to fit the rest of the A-Level subjects but it may not meet the full list of requirements for the needs of the Physics students, my target user group.

The last main limitation of my solution requires a constant, un-interrupted connection to the PostgreSQL database hosted on the webserver. Because of this, any outage in the host would mean that my solution would not function properly and most of the main features would be unusable. A local database could be used, but this would lead to several security issues and unauthorised access to the database. Therefore, an online database host is the only viable option for my solution.

## 1.9 Success Criteria

1.9.1 Universal success criteria

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Description | Justification | Evidence |
| Login System | A system that allows a user to enter their credentials and log in to the system | A login system would allow users to enter their respective area with their work and correct permissions (teacher/admin permissions or student permissions) | Screenshots and GIFs will be used as evidence to show the functionality of the login system |
| Two-Factor Authentication System | A 2FA system where the user receives a code in their email | A 2FA system adds a second layer of security. An attacker would need access to the user’s email account to access this 2FA code | Screenshots and GIFs will be used as evidence to show the functionality of the 2FA system |
| A user can change their email or password | A user should be able to modify their account email and password | A user can forget their password or lose access to their email account. Having this feature would allow for users to never lose access to their account | Screenshots and GIFs will be used as evidence to show the functionality of the change email/password system |

1.9.2 Teacher/Admin success criteria

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Description | Justification | Evidence |
| A teacher/admin can create an assignment | A teacher/admin should be able to create an assignment and set it to a class | Setting an assignment is the main feature of this application. Only teachers will be able to set assignments so students with malicious intents don’t overload the database | Screenshots and GIFs will be used as evidence to show that the assignment setting system functions correctly |
| A teacher/admin can create and delete classes | A teacher/admin should be able to group students together into their respective classes | This will make the assignment setting easier as students will be grouped in their respective classes | Screenshots and GIFs will be used as evidence to show the functionality of the class system |
| Ability to view past and current assignments | A teacher/admin should be able to view a list of all past and currently active assignments | This will allow the teacher to manage how much homework they set and what topics have already been assigned | Screenshots and GIFs will be used as evidence to show the functionality of the assignment viewing system |
| Ability to view student submissions to set assignments | A teacher/admin should be able to view a list of student submissions and achieved mark and date submitted | This will allow the teacher to track student progress and performance | Screenshots and GIFs will be used as evidence to show the functionality of the submission viewing system |
| Ability to create student accounts | A teacher/admin should be able to create accounts for students | Only teachers/admins should have the permission to create accounts. A student with malicious intent could create multiple accounts and overload the system if this permission was given to all users | Screenshots and GIFs will be used as evidence to show the functionality of the account creation system |

1.9.3 Student success criteria

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement | Description | Justification | Evidence |
| Ability to see all active assignments | A student should see all their active and uncompleted assignments on the main screen | The assignments of a student should be easily accessible | Screenshots and GIFs will be used as evidence to show the functionality of the assignment system |
| Ability to view past and completed assignments | A student should be able to view a list of their past, completed assignments and view the mark they achieved | A student should be able to view their performance and track their progress. This would allow the student to revisit topics they underperformed on | Screenshots and GIFs will be used as evidence to show the functionality of the past assignment list system |
| Ability to answer and mark assignments | A student should be able to answer, complete and mark an assignment | A student should be able to mark their own answers. This will allow them to see what they would have actually scored in the exam. | Screenshots and GIFs will be used as evidence to show the functionality of the assignment system |

# Chapter Two: Design

## 2.1 Introduction

In this section I will outline the design process of my solution. I will also show the decomposition of my solution into smaller and more manageable sub-sections.

## 2.2 Decomposition of the problem

Since a project with many features can be very complex, my solution will be broken down into more manageable sub-sections.

For the design of my solution, I have split my program into three smaller sub-sections. The login system, the admin view and the student view. The login system, will allow users to enter their credentials and log in to their respective area. This enables users to access their personal data and assignments. The second section is the admin view. This section provides the ability to create student accounts, create assignments and classes, and view student submissions and assignments. The final section, the student view, will allow students to complete and mark their assignments, and view their past submissions. Both the admin view and student view will have access to the settings area.

2.2.1 Decomposition Diagram

A screenshot of a computer screen

AI-generated content may be incorrect.The following diagram outlines the proposed decomposition design of the solution.

(Figure 1 in evidence)

2.2.2 Flow Charts / Data Flow Diagram

The following diagrams outline where and how the data flows through the different parts of the program.

A screenshot of a computer screen

AI-generated content may be incorrect.

A black background with many different colored rectangular objects

AI-generated content may be incorrect.Data flow diagram for the class system (Figure 3 in evidence)

Data flow diagram for the assignment system (Figure 4 in evidence)

A screenshot of a computer screen

AI-generated content may be incorrect.

Data flow diagram for the account system (Figure 5 in evidence)

2.2.3 Input Process Output Charts

Login system:

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Email and password | Hashes the password, then checks against the database to see if the credentials are valid/invalid | Error message or two-factor authentication window |
| One-time password | Verifies the one-time password and returns True or False | Error message or creation of amin/student window |

Registering student account:

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Student information (name, email, password) | Checks if both passwords match, validates the name, email and password, and checks if the email is already in the database. Then writes data to student accounts table (if all tests pass) | Error or success message |

Creating assignments:

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Assignment information (Title, class assigned to and due date) | Validates data entered. If all data is valid, it creates a table with a randomly generated name and inserts the assignment information to the assignments table | Error message or creation of the add questions window |
| Question information (Question, answer, mark) | Validates data. If valid add question information to the table for that assignment. | Completion message or next question window (depending on if the user clicks complete or next) |

Creating classes:

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Class information (class name and students) | Validates if data is empty. If validation succeeds, creates a table for the class with the selected students and adds the class information (class name and id) to the classes table | Error or success message |

Deleting classes:

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Class name | Validates if a class name has been picked. If so, class table is deleted and record is removed from the classes table | Error or success message |

Settings system:

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Email | Checks if emails entered match, validates the email and checks if it already exists. If all tests pass, email is updated in the database | Error or success message |
| Password | Checks if passwords entered match, validates the password If all tests pass, the password is hashed and updated in the database | Error or success message |

Answering assignments:

|  |  |  |
| --- | --- | --- |
| Input | Process | Output |
| Answer | Checks if answer is empty. If not, a Json file is created and stores the answers | Error message or next question (or marking window) |
| Mark | Checks if it is an integer. If it is, checks if it is the last question. If it is, the Json file is deleted, marks are totalled up and added to the submissions table. | Error message or next question marking window or marking over message |

## 2.3 How All Solution Parts are Linked

2.3.1 State Diagrams of the different sections

Login system:

A black and white rectangular object with text

AI-generated content may be incorrect.

(Figure 6 in evidence)

Admin view:

A screenshot of a computer screen

AI-generated content may be incorrect.(Figure 7 in evidence)

Student view:

A screenshot of a computer screen

AI-generated content may be incorrect.

(Figure 8 in evidence)

2.3.2 How different functions /classes are connected

Login system:

A black and yellow rectangle with black text

AI-generated content may be incorrect.

(Figure 24 in evidence)

Admin system:

A screenshot of a computer screen

AI-generated content may be incorrect.(Figure 23 in evidence)

Student system:

A screenshot of a computer screen

AI-generated content may be incorrect.(Figure 25 in evidence)

## 2.4 Database Design

2.4.1 Data dictionaries

Table name: admin\_acc

This table will store all the admin and teacher accounts. More rows can’t be added to this table unless the database is accessed directly in the database website.

It will store the following information:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data Type | Length | Rules | Comments | Example |
| id | Integer | One digit and above | N/A | Primary key. Generated by the database | 13 |
| email | Varchar | N/A | Must be in the form: example@domain.com | Stores the user’s email | “JohnSmith@hotmail.com” |
| password | Varchar | N/A | N/A | Stores the user’s hashed password | 1d460455f292dc60ed53ad  bea5365bfa27262c25e724  a6eecec799f203a69d7a |
| name | Varchar | More than 5 characters | Must have a space in the middle | Stores the user’s name | “John Smith” |

Table name: main\_acc

This table stores all the student accounts. This table can be changed by any admin or teacher when creating an account.

It stores the following information:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data Type | Length | Rules | Comments | Example |
| Id | Integer | One digit and above | N/A | Primary key. Generated by the database. | 32 |
| name | Varchar | More than 5 characters | Must have a space in the middle | Stores the user’s name | “Rachel Smith” |
| password | Varchar | N/A | N/A | Stores the user’s hashed password | 1d460455f292dc60ed5  3adbea5365bfa27262c  25e724a6eecec799f20  3a69d7a |
| email | Varchar | N/A | Must be in the form: [example@domain.com](mailto:example@domain.com) | Stores the user’s email | “RachelSmith@gmail.com” |
| teacher\_id | Integer | One digit or above | N/A | Foreign key. Links with the id column in the admin\_acc table | 21 |

Table name: stud\_classes

This table stores all the class names and which teacher own them. This table is updated when a class is created or deleted.

It stores this information:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data Type | Length | Rules | Comments | Example |
| Id | Integer | One digit and above | N/A | Primary key. Generated by the database. | 54 |
| class\_names | Varchar | More than one character | N/A | Stores the name of the class | “Physics Class 7E” |
| teacher\_id | Integer | One digit or above | N/A | Foreign key. Links with the id column in the admin\_acc table | 12 |

Table name: assignments

This table stores the information of every assignment. This table is needed to allow for many-to-many table relationships to happen. It also makes looking for an assignment easier.

It follows this structure:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data Type | Length | Rules | Comments | Example |
| assign\_id | Integer | One digit and above | N/A | Primary key. Generated automatically by the database | 31 |
| title | Varchar | More than 2 characters | N/A | Stores the title of the assignment | “The Solar System” |
| class\_id | Integer | One digit and above | N/A | Foreign key. Links to the stud\_classes table. Stores the class which the assignment is assigned to . | 54 |
| due\_date | Datetime | N/A | Must be in the format yyyy-mm-dd | Stores the due date of the assignment. Chosen by the teacher | 2024-09-14 |
| teacher\_id | Varchar | One digit and above | N/A | Foreign key. Links to the admin\_acc table. Stores which teacher set the assignment | 13 |
| title\_id | Varchar | N/A | Must be in the format a12345678 | Created automatically when creating an assignment. Id is unique to each assignment | "a99217410" |

Table name: submissions

This table is used to store the student’s homework submissions.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data Type | Length | Rules | Comments | Example |
| subm\_id | Integer | One digit and above | N/A | Primary key. Generated automatically when the assignment is submitted | 14 |
| assignment\_id | Integer | One digit and above | N/A | Foreign key. Links to the assignments table. | 54 |
| student\_id | Integer | One digit and above | N/A | Foreign key. Links to the main\_acc table. Stores which student submitted the assignment | 21 |
| submission\_date | Datetime | N/A | Must be in the format yyyy-mm-dd | Stores when the assignment was submitted. | 2025-12-13 |
| mark | Integer | One digit and above | N/A | Stores the mark that the student achieved | 13 |

When a teacher creates an assignment, a table is created into the database that stores all the questions and answers. The structure of that table is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data Type | Length | Rules | Comments | Example |
| questionnum | Integer | One digit and above | N/A | Primary key. Generated in a sequential order. | 3 |
| assignment\_id | Integer | One digit and above | N/A | Foreign key. Links with the assignments table. | 4 |
| question | Varchar | More than 5 characters | N/A | Stores the question | “What is the mass of the Sun?” |
| answer | Varchar | More than one character | N/A | Stores the answer | “17 kilograms” |
| marks | Integer | One digit and above | N/A | Stores the marks the question is worth | 2 |
| question\_type | Varchar | N/A | The teacher can select one of two options when creating the assignment (Standard answer/Calculation) | Stores the type of question. The UI when answering the question changes depending on the type. | “Standard answer” |

When a class is created, a table is made that stores all the students that are in that class. It follows this structure:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Name | Data Type | Length | Rules | Comments | Example |
| student\_id | Integer | One digit and above | N/A | Primary key. Generated by the database when the class is created | 23 |
| student\_name | Varchar | More than 5 characters | Must have at least one space | Stores the student’s name | “Laurence Stone” |

A computer screen shot of a diagram

AI-generated content may be incorrect.2.4.2 Entity Relationship Diagram

(Figure 2 in evidence)

2.4.3 SQL Pseudocode

The following Pseudocode defines creation of the tables I need in the database.

CREATE TABLE IF NOT EXISTS admin\_acc(

Id INT AUTO INCREMENT NOT NULL,

Email varchar(255) NOT NULL,

Password varchar(255) NOT NULL,

Name varchar(100) NOT NULL,

PRIMARY KEY(Id)

);

CREATE TABLE IF NOT EXISTS main\_acc(

Id INT AUTO INCREMENT NOT NULL,

Name varchar(100) NOT NULL,

Password varchar(255) NOT NULL,

Email varchar(255) NOT NULL,

Teacher\_id INT NOT NULL,

PRIMARY KEY(Id),

FOREIGN KEY(Teacher\_id) REFERENCES admin\_acc(Id)

);

CREATE TABLE IF NOT EXISTS stud\_classes(

Id INT AUTO INCREMENT NOT NULL,

Class\_names varchar(255) NOT NULL,

Teacher\_id INT NOT NULL,

PRIMARY KEY(Id),

FOREIGN KEY(Teacher\_id) REFERENCES admin\_acc(Id)

);

CREATE TABLE IF NOT EXISTS assignments(

Assign\_id INT AUTO INCREMENT NOT NULL,

Title varchar(255) NOT NULL,

Class\_id INT NOT NULL,

Due\_date datetime NOT NULL,

Teacher\_id INT NOT NULL,

Title\_id varchar(9) NOT NULL,

PRIMARY KEY(Assign\_id),

FOREIGN KEY(Class\_id) REFERENCES stud\_classes(Id)

FOREIGN KEY(Teacher\_id) REFERENCES admin\_acc(Id)

);

CREATE TABLE IF NOT EXISTS submissions(

Subm\_id INT AUTO INCREMENT NOT NULL,

Assignment\_id INT NOT NULL,

Student\_id INT NOT NULL,

Submission\_date datetime NOT NULL,

Mark INT NOT NULL,

PRIMARY KEY(Subm\_id),

FOREIGN KEY(Assignment\_id) REFERENCES assignments(Assign\_id),

FOREIGN KEY(Student\_id) REFERENCES main\_acc(Id)

);

The following table is created by the program when an assignment is created. It is assigned a randomly generated name following the form ‘a12345678’. This name is stored in the assignments table under the column Title\_id. Due to this, the program can automatically detect if the same randomly generated string is already in use which makes the use of a “IF NOT EXISTS” statement unnecessary. The name used in the SQL query is an example of what the name of the table would look like.

CREATE TABLE a12456381(

Questionnum INT AUTO INCREMENT NOT NULL,

Assignment\_id INT NOT NULL,

Question varchar(500) NOT NULL,

Answer varchar(500) NOT NULL,

Marks INT NOT NULL,

Question\_type varchar(30) NOT NULL,

PRIMARY KEY(Questionnum),

FOREIGN KEY(Assignment\_id) REFERENCES assignments(Id)

);

The following table is created when a teacher creates a class. It stores which students are in that class. This table is used to avoid a many-to-many relationship between main\_acc and assignments. Each table name will be different and set by the teacher. ‘Physics\_Y2’ will be used as an example.

CREATE TABLE IF NOT EXISTS Physics\_Y2(

Student\_id INT NOT NULL,

Student\_name NOT NULL,

PRIMARY KEY(Student\_id),

FOREIGN KEY(Student\_id\_) REFERENCES main\_acc(Id)

);

2.4.4 Test plan for database

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test data | Justification |
| 1.0 | Program can connect to the database | Correct database URL | The program needs to be able to make a connection to the database so it can read and write data. |
| 1.1 | There is an error pop-up when connection is not achieved | Incorrect database URL | It is important for an error to be displayed if there is no connection to the database. This is because, if the database host is experiencing issues or there isn’t an internet connection, the program can stop attempting to make a connection. |
| 1.2 | Data can be read from the database | Read a student’s name | It is important for the program to be able to read data accurately from the database. |
| 1.3 | Data can be inserted into a table | Creating a new student | It is important for the program to be able to write to the database. This will allow for new users to be created |
| 1.4 | Data can be deleted from a table | Remove a student | It is important for the program to delete records from tables. This will allow for the deletion of out-of-date data. |
| 1.5 | Tables can be created | Create a class | It is important for the program to be able to create new tables in the database. This will allow for new classes and assignments to be created |
| 1.6 | Tables can be deleted | Delete a class | It is important for the program to be able to delete tables from the database. This will allow old classes to be deleted from the records. |

## 2.5 Design of other Parts of the Solution

2.5.1 Login System

2.5.1.1 User Interface

A screenshot of a computer screen

AI-generated content may be incorrect.

(Figure 9 in evidence)

For this project, I am using the Tkinter library to create the user interface. I chose Tkinter due to the fact that it is a built-in library so it won’t use any storage space on the user’s computer. Also, the Tkinter library works across every operating system. This would lead to an easier development process and improved accessibility to students with different devices. Therefore, Tkinter will provide a user-friendly interface with minimal compatibility issues.

I designed the following diagrams of how I would like the user interface to look like.



(Figure 10 and Figure 11 in evidence)

I decided on a light grey background with black text (this will stay the same throughout the project) because both colours contrast each other. Also, this background colour, doesn’t strain the eyes as much as a pure white background would. This means that the user interface will have a good readability. Therefore, people that have trouble reading could use the application thus improving the accessibility of this application.

The exit button, once pressed, closes the login window. The eye symbol next to the password text box toggles the visibility of the password. When entering the password, it is automatically shown as asterisks. This is done to prevent shouldering and increase the security of the system. The log in button, once clicked, gets the information entered in the text boxes, hashes the password and checks the credentials against the database. If the credentials are correct, the two-factor authentication window is created and overlayed over the login window. If the credentials entered do not match a record in the database, an error message window is created and overlayed over the login window.

On the 2FA window, the resend code button resends the email with the one-time password to the user. The verify button checks if the OTP entered is correct. If the OTP entered is incorrect, an error message window is overlayed over the 2FA window. If the OTP is correct, the login and 2FA windows are destroyed and the admin/student view is opened.

2.5.1.2 Validation Rules

|  |  |  |
| --- | --- | --- |
| Data | Validation rule | Justification |
| Email | Must be in the format “example@domain.com” | This is the accepted and recognised email format |
| Password | Must be between 8 and 20 | A password that is 8 characters and above ensures an added level of complexity and security. Restricting the maximum characters to 20, limits the storage space used in the database. |
| Password | Must have at least 1 uppercase letter, 1 lowercase letter, 1 number and 1 of the following symbols !@\_& | This adds a level of security that would deter brute force attacks. This means no unauthorised person can enter the system |

2.5.1.3 Key Variables/Data Structures/Classes

|  |  |  |
| --- | --- | --- |
| Item | Data type | Description |
| Email | Varchar | This stores the user’s email. This is the main identifier when logging in. |
| Password | Varchar | This stores the user’s password. It is hashed before it is checked against the database. |
| OTP | Varchar | This stores the entered one-time password. It is checked and verified against the generated OTP. Without a correct OTP, the user will not be able to log in |

2.5.1.4 Algorithms and Pseudocode

Password validation pseudocode:

Function validatePassword(password):

If password matches the regular expression pattern '[A-Za-z0-9!@\_&]{8,20}':

If password does not contain any uppercase letter:

Return False

Else If password does not contain any lowercase letter:

Return False

Else If password does not contain any digit:

Return False

Else If password does not contain any special character from '!@\_&':

Return False

Else:

Return True

Else:

Return False

Endif

End function

Email validation pseudocode:

Function validEmail(email):

Define regex as the regular expression pattern for a valid email:

'([A-Za-z0-9]+[.-\_])\*[A-Za-z0-9]+@[A-Za-z0-9]+(\.[A-Z|a-z]{2,})+'

If email matches the regex pattern:

Return True

Else:

Return False

Endif

End function

2.5.1.5 Test Plan

The following is the test plan that I will adhere to. This test plan will ensure that all combinations of data entry are tested.

Login window:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.1 | Empty fields produce an error message | Email: NULL  Password: NULL | Invalid data |
| 1.2 | Empty password field produces an error message | Email: testemail@gmail.com  Password: NULL | Invalid data |
| 1.3 | Empty email field produces an error message | Email: NULL  Password: Password1! | Invalid data |
| 1.4 | Wrong credentials entered produce an error message | Email: notreal@gmail.com  Password: FakePass1! | Invalid data |
| 1.5 | Correct credentials successfully log in | Email: kostispapd@gmail.com  Password: Password1! | Will show if correct credentials log the user in to the correct view (admin/student) |

2FA window:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.1 | Empty field produces an error message | NULL | Invalid data |
| 1.2 | Invalid code produces an error message | Any code that isn’t the correct code | Invalid data |
| 1.3 | Valid code logs user in | Correct code (received when correct login credentials entered) | Will show if the logic works correctly and if the correct window is created |

2.5.2 Admin View

2.5.2.1 User Interface

I decided to structure the main window of the admin view as follows.



(Figure 12 in evidence)

At the top of the window, there is a tool bar. Clicking on the Account section will open a drop-down menu where the user will be able to create a student account, change their email address and their password. Clicking on the classes section will open a drop-down menu with the option to create or delete a class. When the user clicks sign out, the admin window will be destroyed and the login window will open. Clicking on exit will destroy the admin window and stop the program.

When clicking on the create assignment button the following window (the one on the left) will be created. When clicking the class drop-down menu, the teacher will be able to see all of the active classes that they have created. From there, they will be able to pick which class the assignment gets assigned to. When the date button is clicked, a separate window with a calendar will be displayed. When the teacher picks a date on the calendar and clicks select, the calendar window will be destroyed and the date picked will be displayed in the display box. The cancel button will destroy the window and return you to the main screen. The next button will save the assignment information to the database, destroy the create assignment window and create the window on the right.

On the add question window, the type drop-down menu will display the options “Calculation” and “Standard Answer”. This will affect the layout in the student view. When add question is clicked, the question information is saved to the database, the current window gets destroyed and an identical window is created. When the user is finished with adding questions, the finish button will return them to the main admin window.



(Figure 14 and Figure 15 in evidence)

When the view assignments button in the main admin window is pressed, the following window will be created.



(Figure 16 in evidence)

This window will be using the notebook function of the Tkinter library. I am using this function because it will allow for the user to switch quickly between the assignments and submissions. Both windows will have a list with the respective data. The button labelled “Id” will sort the lists by id in ascending order. The button labelled "Title” will sort the list by title in ascending order. I added this function because it would allow the user to find the assignment/submission they were looking for more efficiently.



(Figure 17 in evidence)

The settings menu will allow the user to change their password or email address. On the change password window, the eye symbol will toggle the visibility of the password. The cancel button will destroy the window and return the user back to the main admin window. The change password button, when clicked, will get both of the passwords entered, compare them and validate them. If they are not identical or invalid, an error message window will be overlayed. If the passwords match and are valid, they will be hashed and updated in the database. A success message pop-up will be created and then the window will be closed and the user will be returned to the admin window.

On the change email window, the cancel button will destroy the window and return the user to the admin view. The change email button will compare the emails entered and validate them, and if they do not match or are invalid, an error pop-up will be displayed. If the emails match and are valid, the database will be updated, a success message pop-up will be displayed and the change email window will be destroyed. This will return the user to the admin view.

A screenshot of a login box

AI-generated content may be incorrect.(Figure 18 in evidence)

The above diagram is the proposed design for the Register account window. This feature will be accessible from the settings drop-down menu. The eye symbols will toggle the visibility of the passwords that are being entered. The exit button will destroy the window and return the user to the admin window. Once the register account button is clicked, all data will be validated and the passwords will be compared to each other. Also, the email will be checked against the database to see if it is already in use. If any of the validation fails, the passwords don’t match or the email is already in use, an error pop-up will be displayed. If all tests pass, an success message will be displayed, the window will be destroyed and the user will be returned to the admin window.



(Figure 19 in evidence)

In the class drop-down menu, the user will have the option to create or delete a class. When the create class option is selected, the create class window will be created. Here, the user will have the option to add a class name and add students. When the add students button is pressed, a separate window is created and overlayed above the class window. In this window, a list of all students will be displayed. The user will be able to click on each student that they want to add to the class and then click select. Once select is clicked, the users that are selected will be stored in a list and the window will be destroyed, returning the user to the create class window. When the cancel button is clicked, the window is destroyed and the user is returned to the create class window. Once the user presses create class, the program checks if the class name or the students list is empty. If at least one is empty, an error message is displayed. If both class name and students are not empty, the database is updated and a new table is created with all of the students. Then, a success message is displayed and the user is returned to the admin window.

A screenshot of a computer

AI-generated content may be incorrect.(Figure 20 in evidence)

When the delete class option is selected from the class drop-down menu, the above window will be created. A sorted list of all classes belonging to that user will be displayed on the left. The user will be able to click on a class and then press delete. This will delete the class table and remove the record from the classes table. A success message will be displayed and the user will be returned to the admin view. If the user clicks delete without selecting a class, an error message will be displayed. When the user clicks cancel, the window will be destroyed and the user will be returned to the admin window.

2.5.2.2 Validation Rules

|  |  |  |
| --- | --- | --- |
| Data | Validation rule | Justification |
| Full name | More than 5 characters | This ensures that a name that is entered is real and valid |
| Email | Must be in the format “example@domain.com” | This is the recognised email format |
| Password | Must be between 8 and 20 | A password that is 8 characters and above will ensure an added level of complexity and security. Restricting the maximum characters to 20, limits the storage space used in the database. |
| Password | Must have at least 1 uppercase letter, 1 lowercase letter, 1 number and 1 of the following symbols !@\_& | This adds a level of security that would deter brute force attacks. This means no unauthorised person can enter the system |
| Class Name | Not Null | A class should have an identifier so the user can easily distinguish between classes |
| Students | Not Null | A class should have at least one student |

2.5.2.3 Key Variables/Data Structures/Classes

|  |  |  |
| --- | --- | --- |
| Item | Data type/Structure | Description |
| Full Name | Varchar | Stores the student’s name when their account is being created |
| Email | Varchar | Stores the user’s email. This is used when creating an account or changing email address |
| Password | Varchar | Stores the user’s password. Used when creating an account or changing password. The password is hashed before is it entered into the database |
| Class Name | Varchar | Stores the entered class name. Used when creating a class |
| Selected Students | List | Stores all the students that are being selected to be entered into a class. Used when creating a class |
| Selected Class | List | Stores the class name that is being deleted. Used when deleting classes. |

2.5.2.4 Algorithms and Pseudocode

Assignment system

createAssign:

The purpose of this procedure is to create the assignment information form. When the data is added, it calls the createAssignment procedure.

createAssignment:

The purpose of this procedure is to receive the assignment information data and validate it. If the data is valid, this procedure adds it to the database. Then, it runs the createQs procedure.

createQs:

The purpose of this procedure is to create the question entry form. The data that is entered is passed to the addQuestions procedure.

addQuestions:

The purpose of this procedure is to receive the question data and then validate it. If the given data is valid, it is added to the database. The createQs procedure is then called again.

Assignment viewing system

submissionViewCreate:

This creates the assignment viewing system window. This procedure calls the assignmentObjects and submissionObjects procedures.

assignmentObjects:

This procedure holds all the UI objects for the View Assignment tab. This procedure call the getAssignInfo procedure.

getAssignInfo:

This procedure gets all the assignments and all the information about those assignments from the database. It then passes that information to the assignmentObjects procedure.

submissionObjects:

This procedure holds all the UI objects for the View Submissions tab. This calls the getSubmissions procedure.

getSubmissions:

This procedure gets all the submission information from the database. It then passes that information to the submissionObjects procedure.

Settings system

changePassUI:

This procedure will create the form window for the change password UI. This procedure calls the checkPassword procedure when “Change password is clicked”

checkPassword:

The purpose of this procedure is to validate the password entered. If it is valid, the changePass procedure is called.

changePass:

The purpose of this procedure is to update the database with the new user password.

changeEmailUI:

The purpose of this procedure is to create the user interface for the change email part of the settings. Calls the checkEmail procedure when “Change email” is pressed.

checkEmail:

The purpose of this procedure is to validate the email entered. If the email is valid, the changeEmail procedure is called.

changeEmail:

The purpose of this procedure is to update the database with the user’s new email address.

Account register system

createAccount:

The purpose of this procedure is to create the window for the register account system. Calls getVal when “Register Account” is selected.

getVal:

The purpose of this procedure is to get all of the values that were entered in the register account form. Then it passes those values to the checkVal procedure.

checkVal:

The purpose of this procedure is to validate all of the data that it receives as parameters. If all the data is valid, the registerAcc and sendEmailCreate procedures are called.

registerAcc:

The purpose of this procedure is to update the database with the new user account.

sendEmailCreate:

The purpose of this procedure is to send an email to the new user with their account details.

Class system

createClassUI:

The purpose of this procedure is to create the window and form for the class creation. Once “Add students” is clicked, it calls the stuListUI procedure. When “Create class” is clicked, the createClass procedure is called.

stuListUI:

This procedure creates the user interface for the selection of students. When the window is created, the getStudents procedure is called.

getStudents:

The purpose of this procedure is to get a list of all the students in the database and return them as a list.

createClass:

The purpose of this procedure is to update the database with the new class information. First, it creates a table with the class name as its name and then adds all of the students and their IDs in that table. Then, it updates the classes table with the class information (name and teacher ID).

deleteClassUI:

The purpose of this procedure is to create the window with a list of all available classes under a teacher. To get that list of classes, the getClass procedure is called. When a class is selected and “Delete” is clicked, the deleteClass procedure is called.

getClass:

The purpose of this procedure is to get all the class names under a teacher from the class table. It then returns them to deleteClassUI as a list.

deleteClass:

The purpose of this procedure is to delete a class from the database. First, it drops the table from the database and then removes the record from the classes table.

2.5.2.5 Test Plan

Assignment system:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.0 | Empty fields produce an error message | Assignment title: NULL  Class: NULL  Date: NULL | NULL fields are invalid data |
| 1.1 | Empty assignment title produces an error message | Assignment title: NULL  Class: Test class  Date: 01/03/2025 | NULL fields are invalid data |
| 1.2 | Empty class field produces an error message | Assignment title: “Test assignment”  Class: NULL  Date: 01/03/2025 | NULL fields are invalid data |
| 1.3 | Empty due date field produces an error message | Assignment title: “Test assignment”  Class: Test class  Date: NULL | NULL fields are invalid data |
| 1.4 | Empty question field produces an error message | Question: NULL  Marks: 3  Type: Calculation  Answer: “Test answer” | NULL fields are invalid data |
| 1.5 | Empty marks field produces an error message | Question: “This is a question”  Marks: NULL  Type: Calculation  Answer: “This is an answer” | NULL fields are invalid data |
| 1.6 | Empty type field produces an error message | Question: “This is a question”  Marks: 17  Type: NULL  Answer: “Answer” | NULL fields are invalid data |
| 1.7 | Empty answer field produces an error message | Question: “Question”  Marks: 32  Type: Standard answer  Answer: NULL | NULL fields are invalid data |
| 1.8 | Mark that isn’t an integer produces an error message | Question: “Test question”  Marks: “mark”  Type: Calculation  Answer: “Answer” | A mark should be an integer. Every other data type is invalid data |
| 1.9 | Valid data produces a success message | Question: “Test question”  Marks: 32  Type: Calculation  Answer: “Answer” | This will show that the system works correctly |

Assignment viewing system:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.0 | Empty assignment list displays a no assignments message | NULL | The user should know if there are no assignments |
| 1.1 | Empty submission list displays a no submissions message | NULL | The user should know if there are no submissions |
| 1.2 | An assignment list displays the assignments | A list of assignments | Shows if the program works correctly |
| 1.3 | A submission list displays the submissions | A list of submissions | Shows if the program works correctly |
| 1.4 | Sort by ID button sorts by the lDs | N/A | Shows if the program works correctly |
| 1.5 | Sort by title button sorts by the titles | N/A | Shows if the program works correctly |

Settings system:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.0 | Empty fields produce an error message | New password: NULL  Re-type password: NULL | NULL fields are invalid data |
| 1.1 | Empty new password field produces an error message | New password: NULL  Re-type password: “Password1!” | A NULL field is invalid data |
| 1.2 | Empty new password field produces an error message | New password: “Password1!”  Re-type password: NULL | A NULL field is invalid data |
| 1.3 | Passwords that do not match produce an error message | New password: “Password1!”  Re-type password: “PasswordOne1!” | Both passwords should match so when the user types in their new password they don’t make a typing mistake |
| 1.4 | Invalid passwords produce an error message | New password: “Password”  Re-type password: “Password” | A password should follow correct validation rules. If it is invalid, the user should know if it is. (This will be tested more in depth in chapter 3) |
| 1.5 | Valid passwords and passwords that match produce a success message | New password: “Password1!”  Re-type password: “Password1!” | Will show that the system works correctly |
| 1.6 | Empty fields produce an error message | New email: NULL  Re-type email: NULL | NULL fields are invalid data |
| 1.7 | Empty new email field produces an error message | New email: NULL  Re-type email: “testemail@gmail.com” | NULL fields are invalid data |
| 1.8 | Empty re-type email field produces an error message | New email: “testemail@gmail.com”  Re-type email: NULL | NULL fields are invalid data |
| 1.9 | Emails that do not match produce an error message | New email: “testemail@gmail.com”  Re-type email: “notemail@gmail.com” | Both emails should match so the user doesn’t make a typing mistake |
| 1.10 | Emails that are invalid produce an error message | New email: “email@.com”  Re-type email: “email@.com” | An invalid email wouldn’t exist so the user will not be able to receive 2FA codes |
| 1.11 | Valid and matching emails produce a success message | New email: “testemail@gmail.com”  Re-type email: “testemail@gmail.com” | Will show that the system works correctly |

Account register system:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.0 | Empty fields produce an error message | Full name: NULL  Email address: NULL  Password: NULL  Re-type password: NULL | Invalid data |
| 1.1 | Empty Name field produces an error message | Full name: NULL  Email address: “testemail@gmail.com”  Password: “Password1!”  Re-type password: “Password1!” | Invalid data |
| 1.2 | Empty email address field produces an error message | Full name: “Name Test”  Email address: NULL  Password: “Password1!”  Re-type password: “Password1!” | Invalid data |
| 1.3 | Empty password field produces an error message | Full name: “Name Test”  Email address: “testemail@gmail.com”  Password: NULL  Re-type password: “Password1!” | Invalid data |
| 1.4 | Empty re-type password field produces an error message | Full name: “Name Test”  Email address: “testemail@gmail.com”  Password: “Password1!”  Re-type password: NULL | Invalid data |
| 1.5 | Invalid email produces an error message | Full name: “Name Test”  Email address: “testemail@gmail”  Password: “Password1!”  Re-type password: “Password1!” | Invalid data |
| 1.6 | Passwords that don’t match produce an error message | Full name: “Name Test”  Email address: “testemail@gmail.com”  Password: “Password1!”  Re-type password: “Password!” | Invalid data |
| 1.7 | Invalid passwords produce an error message | Full name: “Name Test”  Email address: “testemail@gmail.com”  Password: “password”  Re-type password: “password” | Invalid data |
| 1.8 | Valid data produces success message | Full name: “Name Test”  Email address: “testemail@gmail.com”  Password: “Password1!”  Re-type password: “Password1!” | Will show that the system works correctly |

Class system:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.0 | Empty fields produce an error message | Class name: NULL  Students: NULL | Invalid data |
| 1.1 | Empty class name field produces an error message | Class name: NULL  Students: List of students | Invalid data |
| 1.2 | Empty student list produces an error message | Class name: “Test class”  Students: NULL | Invalid data |
| 1.3 | Valid student list and class name produces a success message | Class name: “Test class”  Students: List of students | Will show that the system works correctly |

2.5.3 Student View

2.5.3.1 User Interface

I decided to structure the main window of the student view as follows:



(Figure 13 in evidence)

The student will have access to a toolbar at the top of a window. From there, the user will be able to access their submissions and settings. When the student logs in, they will see a list of all of their active assignments. Each assignment will appear as a button with the assignment name and due date. When the user clicks on an assignment, the assignment view will open. When the settings button is clicked, a drop-down menu will appear. It will give the user the option to change password, change email or sign out. Clicking on the progress button will open a window with all of the user’s past submissions to assignments.



(Figure 21 in evidence)

The above is the proposed design for the assignment system UI. The diagram on the left, outlines the UI for answering a question. Clicking the “Previous” button will take the user to the previous question. If the user is on question 1 of the assignment, the “Previous” button will not be displayed. The “Next” button will save the user’s answer and take them to the next question. If the user is on the last question of the assignment, the “Next” button will be replaced by a “Submit” button. When the user clicks “Submit”, they will be taken to the marking window. Here, they will be shown the correct answer to the question and their answer. The user will be able to write how many marks their answer has scored in the text box. The “Next” button will take them to mark the next question. When the user is marking the last question, the “Next” button will be replaced by a “Submit” button. Clicking this will save the user’s submission to the database, show their total mark and return them to the main student window. The assignment will then be removed from the assignment list.

The change email and password UI will be the same across the admin and student view (design can be found in section 2.5.2.1).

When the user clicks on the progress button, the following window will be overlayed over the student window.

A screenshot of a computer

AI-generated content may be incorrect.(Figure 22 in evidence)

This window will display a list of all of the user’s submissions. The ID button will sort the list by Submission ID in ascending order. The title button will sort the submissions by title in ascending order.

2.5.3.2 Validation Rules

|  |  |  |
| --- | --- | --- |
| Data | Validation Rule | Justification |
| Answer | Can’t be null | An answer must be provided when answering a question |
| Mark | Can’t be null | A mark should be provided when marking an answer |
| Mark | Must be an integer | A mark should be an integer as it is the traditional and accepted marking system |

2.5.3.3 Key Variables/Data Structures/Classes

|  |  |  |
| --- | --- | --- |
| Item | Data type/Structure | Description |
| Answers | List | Stores the student’s answers |
| Marks | List | Stores the student marks for each question |
| Questions | Class | This class will handle the UI and logic for the answering questions system |
| Marking | Class | This class will create the UI for the marking system and will handle the logic for the system |
| Assignments | Class | This class will get all the student’s assignments and display them in a list |

2.5.3.4 Algorithms and Pseudocode

Questions System

Questions

This class will hold all the methods that create the UI windows that display the questions. Also, it will hold the methods that get all the question information.

createWindow

This procedure will hold all the UI elements for the questions system. This procedure calls the getQuest function.

getQuest

This function gets the question from the database and returns it to the createWindow procedure.

Marking

This class will hold all the methods for the marking system.

createWindow

This procedure will hold all the UI for the marking system. This procedure will call the getAnsw function. Once submit is pressed, the submitMark procedure will be called.

getAnsw

This function will get the answers from the database and return them to the createWindow procedure.

submitMark

This procedure will save the students final mark to the database and return the student to the main student window.

Assignments system

createStudent

This procedure creates the main student window. When it is called, once the student logs in, the Assignments class is initialised.

Assignments

This class will hold all of the methods that display the list of assignments on the student window.

showAssign

This method in the Assignments class calls the function getAssignments and displays a list of all assignments on the student window

getAssignments

This function gets all of the assignments that the student has not completed and returns them to showAssign.

openAssign

This procedure is called when an assignments is selected. This procedure creates an instance of the Questions class.

Submissions system

studentProgress

This procedure creates the UI for the Submissions list

getStudentProgress

This function is called when the studentProgress is called. This function gets all of the student’s submissions from the database and returns them.

2.5.3.5 Test Plan

Questions system:

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1.0 | Empty answer produces an error message | Answer: NULL | A NULL field is invalid data |
| 1.1 | Valid answer goes to the next question | Answer: “This is an answer” | This will test if the system works correctly |
| 1.2 | Empty mark field produces an error message | Mark: NULL | Invalid data |
| 1.3 | A string entered in the mark field produces an error message | Mark: “12e” | Invalid data type |
| 1.4 | A float entered in the mark field produces an error message | Mark: 12.3 | Invalid data type |
| 1.4.a |  | Mark: 3.0 | Invalid data type |
| 1.5 | A mark that is above the maximum mark produces an error message | Mark: depends on the question | Invalid data |
| 1.6 | Boundary data testing | Mark: Maximum that is allowed (e.g. if the question is 4 marks, then 4 marks are rewarded) | Good to test for the higher boundary as the logic could be wrong and decline the valid data |
| 1.6.a |  | Mark: One mark above maximum mark | Invalid data as it’s above the boundary |
| 1.7 | A valid integer entered in the mark field goes to the next question | Mark: Depends on the question | Valid data |
| 1.7.a | A valid integer entered submits the marks (Only on last question) | Mark: Depends on the question | Valid data |

## 2.6 Stakeholders involvement

2.6.1 Stakeholder feedback

After I finalised my initial design for the solution, I sent all of the UI diagrams and flowcharts to my stakeholders to get feedback as they are the targeted users for my solution. Their responses were as follows:

Mina (Teacher):

“I really liked that I can access every important action from the main page so I will be able to navigate the app easily. I also liked that the setting of assignments is simple and straightforward so it will save me time”

Ali (Student):

“One feature I like is that I can go to the previous question as I cant do that currently. I like that the form design is simple so it won’t be hard to navigate. I don’t like that the hints were removed so when I’m struggling, I wont be able to get help”

Jahin (Student):

“I like that the program will only need internet connection to get stuff from the database which will keep the bandwidth usage low, which is one thing I really wanted to see. I also like that you focused on self learning so there are no timers which will allow me to do research on a question I am stuck on. The UI is clean and simple which is something I wanted from this program.”

Violet (Student):

“I like the layout of the UI and I like that there is a main window that holds every important function. An uncluttered UI was really nice to see.”

2.6.2 Stakeholder conclusion

Overall, the feedback from the stakeholder was very positive. I will be incorporating the same design and layout in to my program which will be outlined in the next chapter.

## 2.7 Testing plan to inform evaluation

The following test plan is linked to the success criteria of my solution and it is designed to ensure that the criteria is met.

|  |  |  |  |
| --- | --- | --- | --- |
| Test No. | Test | Test Data | Justification |
| 1 | A connection with the database can be established | Wi-Fi connection | If there is not a connection between the program and the database, the main function of the program will not work. |
| 2 | The user can log in to their respective area | Correct username, password and OTP | This is a main part of my success criteria, which is a secure login and 2FA system |
| 3 | User cannot login with invalid credentials | Invalid username and password | This will test my validation and error handling |
| 4 | User can change their email and password | Valid email or password | This test will ensure that the customisation criteria is met. |
| 5 | Teacher can create an assignment and add questions | Valid parameters | This will test that the assignment system works correctly |
| 6 | Classes can be created and deleted | Valid classes | Will test the class system works properly |
| 7 | Past and active assignments can be viewed (Teacher only) | N/A | A teacher should be able to see every assignment that has ever been created. This will test that function |
| 8 | Submissions can be viewed (Students and teachers) | N/A | Part of my success criteria is for students and teachers to be able to view progress. This will test that the system works |
| 9 | Accounts can be created | Valid data | Creating student accounts is a vital part of the system. This will test that the system works |
| 10 | Active assignments can be viewed and opened (Students only) | N/A | A student should be able to view their active assignments. This will test if a student can do that. |
| 11 | Assignments can be answered and marked | Valid answers and marks | This will ensure that the student answers and marks their assignments correctly. |
| 12 | Any invalid data produces an error message | Invalid data | Will test the error handling of the program |

# Chapter Three: Iterative Development and Testing

## 3.1 Introduction

I have split my solution into five main parts. The database, the login, the validation, the admin view and the student view. In this section, I will document the development and testing of the different parts of my solution. All sections of my project will be tested thoroughly and altered accordingly.

For my solution, I will be using a mix of event-driven programming and object-oriented programming. I have chosen to do this as the event-driven programming will allow user interaction with my solution and the OOP will allow for reusable code.

## 3.2 Stage 1: Building the Database

All of the SQL code is kept in the same file. This will make the program more organised and easy to maintain.

3.2.1 SQL and Python Code

Creating Tables

To create the main tables needed for my database, I ran the following code in a separate file.





This file will create all of the database tables to fit the layout of my design. I chose to keep this code separate from the main file as the users will not have access to this code. This will only be run when the database needs to be initialised.

Imports

The main SQL file will hold the rest of the SQL for the application. It imports the following libraries and files:



Psycopg2 – This library is a PostgreSQL adapter for python. I chose to use this library as it is a secure adapter. This is because, it allows for parametised SQL queries which prevent SQL injections.

Dotenv and os – I am keeping my database connection key in a .env file. I am doing this because it will allow me to get the key from anywhere in the program instead of writing it everywhere I need. Another benefit of this is that I can retrieve the key from anywhere in the program without needing to duplicate it in the code. Due to the fact that I am using a .env file, I need a .env file handler. I chose the dotenv library as it is very lightweight, so it won’t slow down the application, and because it is secure.

Tkinter messagebox – This library allows me to create error/information windows on the user’s screen. I chose to use Tkinter as it is a lightweight and built-in library. This will reduce storage space and make the application faster compared to using other UI libraries.

processWindows – This line imports the file processWindows.py. The processWindows file will hold a lot of the main functionality of the application. The SQL file needs to import this file due to the fact that it will be calling different functions/procedures from there.

Login System

The checkLogIn function from the SQL file can be seen below. This function checks the entered login credentials against both of the databases. When the function is called, a connection with the database is established. If a connection cannot be established, an error message is displayed. Then, the password is salted and hashed and the credentials are checked against the student account database. If the credentials are found in that database, the function returns the user’s permission level, name, email address and password. If the credentials are not found in the database, the admin database is checked. If the credentials are found, the function returns the user’s permission level, name, ID, password and email address. If the credentials are not in the second database, the function returns “None”.

The function is structured with a try, except statement. This means that if any part of the code fails, an error message will be shown immediately and no changes will be made to the code/database.



Before the password is checked against the database, it is salted and hashed. This adds an extra layer of security. The function that hashes the password is as follows:



In this function, the hashlib library is imported. This library allows me to hash the password. Before the password is hashed, a salt is added to the end of the password. A unique salt could be used for every user but his would require a lot of extra space in the database. While a static salt is not the most secure encryption method, it adds a basic level of security and it will prevent dictionary attacks.. After the password is salted, it is hashed using the SHA-256 method from the hashlib library. I chose to use this as it is a secure encryption method that will prevent preimage attacks. Once the password is hashed, a hexadecimal value is assigned to each hashed character. This adds the last level of security and it will make sure that every password is in a consistent length and format.

When an admin or teacher logs in, their user ID is fetched from the database. The code below outlines how the ID is fetched. This code will be reused throughout the program for different functions and procedures. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



Registration System

The following procedure outlines the main code that is used for creating an account. When this procedure is called, it establishes a connection with the database. Then it hashes the password entered and makes the email into lowercase letters. I chose to convert all emails to lowercase as it will ensure consistency and will avoid any case sensitivity issues. Then, the main\_acc table is updated with the new student account. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code checks whether the email entered when creating an account is already in use. This is a very important function as it will prevent duplicate data and won’t allow multiple users to have the same email address. If the email address is already in use, False is returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



Settings System

The following code snippet shows the code for changing the email in the database. This code connects to the database. Then, an if statement is used to check whether the email that needs to be changed is a student or admin email. After this check, the email is updated in the respective database table. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code outlines the process of updating a user’s password in the database. When this procedure is called, a connection is established with the database. Then, the password entered by the user is hashed. An if statement decides if the table that is being updated is the admin or student table. Once the selection is made, the respective table is updated with the new password. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



Class system

The following code outlines the class creation process. When the procedure is called, a connection is established with the database. Then, a table name is created. This table name is a mix between the given class name and the teacher ID, which will allow the program to distinguish between classes that have the same name. Then, a table with the created name is added to the database (this will store all of the students). The class name is added to the stud\_classes table and assigned an ID (automatic process done by the database). Once the database has been updated, a for loop adds all of the student information (name and ID) to the created table. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



This code gets a list of all of the students that are under a certain teacher. This process is used when a teacher is selecting which students to add to a class. When this function is called, a connection is established with the database. Then, all of the students that are under the specified teacher are fetched from the database. A list of all the student names is then created and returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code gets a list of all classes under a certain teacher. This function is used when a teacher is selecting which classes to delete. A connection is established with the database when this function is called. Then, all of the class names that are linked to the specified teacher are fetched from the database. A list of the fetched class names is then created and returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code outlines the class deletion process. A connection is established with the database. Then, for each class item in the classes list, the class with the same name is removed from the database. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code gets the ID of a specified class from the database. This is used when creating an assignment. First, a connection is established with the database. Then the ID is fetched from the table and then returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



Assignment system

The following code snippet shows the assignment creation process. When this procedure is called, an if statement is used to validate whether there are any empty parameters. Once validation is passed, a connection is established with the database and the class ID is fetched from the database. If an ID is not found, an error message is displayed. If the selected class has an ID, the assignment information (table name, title, class, due date and teacher) is inserted into the assignments table. Then, a new table is created in the database (this table will store the question information). After this step is complete, the assignment ID is fetched and passed on to the next stage in the assignment process. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code fetches the assignment ID from the assignments table. When the function is called, a connection with the database is established. Then, the assignment ID for the provided assignment is fetched and returned. If the assignment is not found in the database, an error message is displayed. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code outlines the process of adding a question to the database. An if statement is used to validate whether there are any empty parameters. If validation is passed successfully, a connection is established with the database. The assignment name is then fetched. Then, the question information (question, answer mark and question type) is inserted into the respective assignment table. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code fetches the assignment name from the assignments table. When this function is called, it establishes a connection with the database. Then the title ID is fetched from the database and returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following function gets the specified question from the database. This function is used to get the question the student is answering and display it on the window. When the function is called, a connection with the database is established. The question is then fetched and returned to the main program. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code checks what type a question is (calculation or standard answer). Different types of questions will have different UI layouts. When this function is called, it establishes a connection with the database. Then, the question type is fetched from the assignment table and returned to the main program. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



When an assignment is created, it is assigned a randomly generated number in the format “a12345678”. The following code checks whether the generated number is already in use. When this function is called, a connection is established with the database. The SQL query checks the assignments table and returns the value 1 if the assignment number already exists. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



When a student is answering the last question, the “Next question” button should change to a “Submit assignment” button. The following function fetches the question number of the last question in an assignment. A connection with the database is established when this function is called. Then, the number of the last question is fetched and returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following function gets the answer of the specified question. When the function is called, a connection with the database is established. Once a connection is successfully established, the answer and mark is fetched from the database. Then it is returned back to the main program. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following code saves the student’s submission to the database. Once this procedure is called, a connection is established with the database. Once a connection is successful, the assignment ID, student ID, date and mark is added to the submissions table. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following function gets the student and teacher ID from the database. This is used for when all of the student’s assignments get fetched. When this function is called, a connection with the database is established. Then, the student ID and teacher ID from the main account table. When both IDs get fetched successfully, they are returned back to the student view. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following procedure saves the student’s submission to the database. When a student finishes marking their assignment, this procedure is called and a connection with the database is established. Then, the assignment ID, student ID, submission date and student mark is inserted into the submissions table. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following function gets the students name. When this function is called, once marking is finished and saved, a connection with the database is established. Then the student’s name is fetched from the main account table and returned back. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



Submissions system

The following function gets all of the submissions under a certain teacher from the submissions table. When this function is called, a connection with the database is established. Once a connection is successfully connected, the assignments and main account tables are connected with the submissions table. This will allow for the program to fetch data from multiple tables at once. Then, the assignment title, submission id, student name, submission mark and submission date are fetched and returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following function gets all of the assignment information from the database. Once this function is called, a connection with the database is established. Then, the student classes table is connected with the assignments table. This will allow the program to fetch data from multiple tables at once. The assignment ID, assignment title, due date and class name is fetched from the respective tables. Then the data is returned back. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error.



The following function gets all of a respective student’s submissions. When this function is called, a connection is established. Then, the assignments table is joined with the submissions table. This will allow the program to fetch data from both tables at once. The assignment table, submission ID, mark and submission date is fetched and returned. A try except statement has been used to show an error message if the connection can’t be established or if there is an unexpected error. 

3.2.2 Testing

The following table will outline all of the database testing according to the testing plan in section 2.4.4.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test Description | Outcome | Evidence | Comment |
| 1.1 | Program can connect to the database. |  | A screenshot of a computer  AI-generated content may be incorrect.(GIF 18 in evidence) | Expected result |
| 1.2 | Database tables get created successfully | All tables created successfully. This is the desired result. | A screenshot of a computer  AI-generated content may be incorrect.(GIF 15 in evidence) | Expected result |
| 1.2.a | N/A | N/A | A screenshot of a computer  AI-generated content may be incorrect.(Figure 26 in evidence) | Extra evidence |
| 1.3.a | Error popup when connection isn’t established  (False database URL) | An error message successfully pops up when a connection with the database is not established | A screenshot of a computer program  AI-generated content may be incorrect.(GIF 16 in evidence) | Expected result |
| 1.3.b | Error popup when connection isn’t established  (No internet connection) | An error message successfully pops up when a connection with the database is not established | (GIF 17 in evidence) | Expected result |
| 1.4 | Data can be read from the database | Data from the database read successfully | A screenshot of a computer program  AI-generated content may be incorrect.  (GIF 19 in evidence) | Expected result |
| 1.5 | Data can be inserted into the database | Data can be inserted into the database | A screen shot of a computer  AI-generated content may be incorrect.  (GIF 20 in evidence) | Expected result |
| 1.5.a | N/A | N/A | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 21 in evidence) | Extra evidence |
| 1.6 | Data can be deleted from the database | Data is deleted from the database successfully | (GIF 22 in evidence) | Expected result |
| 1.6.a | N/A | N/A | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 23 in evidence) | Extra evidence |
| 1.7 | Tables can be created | Table created successfully | A computer screen with text and images  AI-generated content may be incorrect.(GIF 24 in evidence) | Expected result |
| 1.7.a | N/A | N/A | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 25 in evidence) | Extra evidence |
| 1.8 | Tables can be deleted from the database | Tables deleted successfully from the database | (GIF 26 in evidence) | Expected result |
| 1.8.a | N/A | N/A | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 27 in evidence) | Extra evidence |

3.2.3 Review

All database tests were performed according to the test plan is section 2.4.4 and all tests were successful. This means that all of the database functions will work correctly and no corrective actions need to be taken.

## 3.3 Stage 2: Building The Log In

The following section will outline the development process of the login system.

3.3.1 Python code

Imports

The login system file will need the following imports.



Tkinter and Tkinter messagebox – My application will need a GUI. The Tkinter library allows for a GUI to be created. I chose to use Tkinter as it is a lightweight and built in library. The messagebox method will allow the program to create error and information messages on the user’s screen.

SQLfunctions – SQLfuntions is the file where every SQL function is stored (development of this file was outlined in section 3.2).This file needs to be imported as the login system will need to check the database to check the credentials entered by the user.

PIL/Pillow – Pillow is an imaging library for python. This library will allow the application’s logo to be displayed on the login window.

Student/Admin view – These files hold the code for the student and admin view (development outlined in the following sections of this chapter). The login file will need to call the respective file to create the different view windows.

UI creation

The following procedure will create the login window and form where the user will be able to enter their credentials. When the procedure is called, a window instance is created. Then the position of the middle of the screen is calculated and the window is displayed in that position. The elements (labels, text boxes, logo frame etc) are defined and displayed on the window. The enter() subroutine is ran when the enter key is pressed.





The following procedure toggles the visibility of the password text box. When the procedure is called, when clicking the password view button, the current visibility of the text box is fetched. Then, the text box is assigned the opposite visibility state.



Credential check

The following procedure checks the credentials entered and creates the respective user view. When this procedure is called, the entered username and password are fetched from the login form. Then, the credentials are passed onto the checkLogIn function in the SQL file (development outlined in section 3.2.1) and the result is stored in the check variable. If the check variable is empty, an error message is created. If there is a value assigned to the check variable, the respective user view is created.



3.3.2 Testing

The following table will outline all of the login system testing according to the testing plan in section 2.5.1.5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test Description | Outcome | Evidence | Comment |
| 1.1 | Test that the window opens | The window opens successfully | A computer screen shot of a computer code  AI-generated content may be incorrect.  (GIF 1 in evidence) | Expected result |
| 1.2 | Show password button toggles the password visibility | Button changes the visibility of the password | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 2 in evidence) | Expected result |
| 1.3 | Exit button closes the window | Exit button closes the window successfully | A screen shot of a computer program  AI-generated content may be incorrect.  (GIF 3 in evidence) | Expected result |
| 1.4 | Empty fields produce an error message | Error message created when both fields are empty | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 4 in evidence) | Expected result |
| 1.5 | Empty password field produces an error message | Error message is created successfully when the password field is empty | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 5 in evidence) | Expected result |
| 1.6 | Empty username field produces an error message | Error message is created successfully when the username field is empty | A screenshot of a computer  AI-generated content may be incorrect.(GIF 6 in evidence) | Expected result |
| 1.7 | Wrong login information entered produces an error message | Wrong login information entered produces an error message successfully | A screenshot of a computer  AI-generated content may be incorrect.(GIF 7 in evidence) | Expected result |
| 1.8 | Correct student information entered creates student view | Student view created successfully when correct student information is entered | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 13 in evidence) | Expected result |
| 1.9 | Correct admin information entered creates the admin/teacher view | The admin view is created successfully when the correct information is entered | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 14 in evidence) | Expected result |

All login system tests were passed successfully and according to the test plan in section 2.5.1.5. No corrective actions will need to be taken.

3.3.3 Stakeholder Feedback

After I completed the login system, I showed it to my stakeholders and they were also pleased with the result. However, they pointed out that the two-factor authentication system is not a feature. This goes against my success criteria.

3.3.4 Corrective actions

For my solution to adhere to my success criteria, the 2FA system will have to be incorporated into my program.

**3.3.4.1 Python code**

Imports

The following imports have been added to the login system file.



Pyotp – This library will allow my program to generate an one-time password and verify it. I have chosen to use this library because it’s a lightweight and easy to use library.

SendEmailOTP (processWindows) – This procedure that is imported sends an email to the user logging in with their one time password. The processWindows file holds a lot of code that will be re-used throughout the application. The code for the imported procedure is outlined below.



This procedure uses the email.message library. I chose to use this library as it allows for easy and simple construction of email messages. Also, the email.message library has good integration with the smtplib library which allows Python to send email messages via SMTP servers. This procedure receives check (all of the user’s information) and the otp as parameters. It then gets the user’s email from the check parameter and crafts the email message using the EmailMessage() procedure (imported from email.message library). It then creates a connection with the SMTP server using the provided port and by logging in to the sender email. Then, it sends an email to the user with their one time password.

Two-factor authentication UI

The following procedure will create the user interface and form for the 2FA system. When the procedure is called, a window instance is created. The position of the middle of the screen is calculated and the window is placed and displayed there. Then, the labels, text boxes and buttons are defined and displayed on the window. The enter subroutine is ran when the enter key is pressed. Once the enter key is pressed, the logIn subroutine is ran. In this subroutine, the one-time password is verified and the respective user view is created.





The 2FA window is called when the entered credentials are correct. The logIn procedure has been altered accordingly to account for the new system. The changes are outlined below.



The following functions create an otp and verify it. The generateKey function creates a random base32 key for the otp creation. This key is then returned to the generateOTP function. Firstly, the function creates a time-based one-time password using the provided key. Then, an otp is generated and returned using the now() method of the TOTP object. The verify function verifies the entered one-time password and returns True or False (depending whether the otp entered is correct or not.



**3.3.4.2 Testing**

The following table will outline all of the 2FA system testing according to the testing plan in section 2.5.1.5.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test No. | Test Description | Outcome | Evidence | Comment |
| 1.1 | 2FA window opens when correct information is entered | 2FA window gets created successfully when the correct login information is entered | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 8 in evidence) | Expected result |
| 1.2 | Resend code button sends the same code | When the resend button is clicked, the same code is sent to the user | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 9 in evidence) | Expected result |
| 1.3 | Invalid 2FA code produces an error message | An invalid 2FA code successfully produces an error message | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 10 in evidence) | Expected result |
| 1.4 | Valid 2FA code logs the user in to the student view | When a student enters a valid 2FA code, they are logged in to the student view | (GIF 11 in evidence) | Expected result |
| 1.5 | Valid 2FA code opens admin view when entered by an admin/teacher | The admin view opens when an admin/teacher enters a valid 2FA code | A screenshot of a computer  AI-generated content may be incorrect.  (GIF 12 in evidence) | Expected result |

All of the tests for the two-factor authentication passed successfully. Therefore, no more corrective actions need to be taken. The addition of the new 2FA system means that the program now follows the success criteria.

**3.3.4.3 Stakeholder feedback**

After the testing was concluded, I asked my stakeholders for feedback on the 2FA system. Overall, the feedback was positive and the stakeholders were happy with the system.

## 3.4 Stage 3: Building The Validation

The most common validation in my solution will be input validation, which will validate user inputs like passwords and emails. The development and testing of the validation will be explained in this section.

3.4.1 Python code

All of the main validation will be held in a separate file that will be imported to the different modules. The validation file is outlined below.

Imports

The following imports will be used in the validation file.



Re – This is a regex library. This library will be used during email and password validation and will be used to check whether the entered string matches the regular expression pattern.

Kickbox – This is a client library for the kickbox API. This API will be used to check whether the entered email exists.

Email validation



The code above validates the email entered. It does this by using a regex pattern. First, the regex pattern is defined and compiled. The email entered must have at least one alphanumeric value followed by an optional dot, hyphen or underscore. It must the n=have an @ symbol followed by at least one alphanumeric value. This is then followed by a dot and the domain. The fullmatch() function then checks whether the supplied email follows that string. If the email matches, then True is returned. If not, False is returned.

Email verification



The code above verifies whether the email enter actually exists. It does this by using the kickbox API. When this function is called, a kickbox client is created with the provided API key. Then, the email entered is verified by sending an API request with the verify() method. The result of the verification is returned as a dictionary. The result is then checked to see if the email was deliverable or risky (this means that the email can be delivered but the address has a higher chance of being problematic) and True is returned if that is the case. False is retuned if the email cannot be delivered.

Password validation



The function above validates the password entered by the user. When this function is called, the password is checked against the regular expression. If it is a full match, then the password undergoes additional checks to see if it includes lowercase characters, uppercase characters, numbers and any of the listed special symbols. If all of the tests are passed, True is returned. If any of the tests fail or the password isn’t a full match with the regex pattern, False is returned.

3.4.2 Testing

I will be testing my validation using the unittest library as it allows for automated testing. The unittest code is outlined below.

3.4.2.1 Python code

Imports

The following imports will be used to unit test the validation.



Unittest – This is the library that will allow for simple and automated testing of code.

isValid – This import is the file where all of the validation is stored. This is the file that will be tested so the import is necessary.

Testing code



The code above outlines the unit testing process. The testPassword method of the Validate\_isValid class is used to test whether the password validation code works correctly. The testEmail method of the Validate\_isValid class is used to test whether the email validation code works correctly. The testVeifyEmail method of the Validate\_isValid class is used to test whether the email verification code works correctly. The testing for the validation is outlined below.

3.4.2.2 Unit testing

Email validation and verification testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No. | Description | Test data | Expected Outcome | Actual Outcome | Comment |
| 1.1 | Attempt to validate and verify a normal email | b32908@sfc.potteries.ac.uk | “True True”  “True True” | A computer screen with white text  Description automatically generated | Expected result |
| 1.2 | Attempt to validate and verify an invalid email | b32908@uk | “False False” “False False” | A screen shot of a computer  Description automatically generated | Expected result |
| 1.3 | Attempt to validate and verify a valid but not real email | b32908@uk.com | “True True”  “False False” | A screenshot of a computer error  Description automatically generated | Not the expected result. Corrections will have to be made |

The verification in test 1.3 failed. The expected outcome should be false as the email address doesn’t exist but the email verification returns True.

3.4.3 Corrective actions and further testing

Upon further inspection of my code and the Kickbox API documentation, I discovered that the problem had to do with the API key. When I set up the API key, I configured it incorrectly and it simulated every test that I tried. This meant that all the results I was given were fake. To fix this, I created a new API key with the correct configuration. This change only affects the subroutine verifyEmail(). The affected and changed code can be seen below.



The tests for the altered code are outlined below:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No. | Description | Test data | Expected Outcome | Actual Outcome | Comment |
| 2.1 | Attempt to validate and verify a normal email | b32908@sfc.potteries.ac.uk | “True True”  “True True” | A screenshot of a computer  Description automatically generated | Expected result |
| 2.2 | Attempt to validate and verify an invalid email | b32908@uk | “False False” “False False” | A computer screen with red text  Description automatically generated | Expected result |
| 2.3 | Attempt to validate and verify a valid but not real email | b32908@uk.com | “True True”  “False False” | A black screen with white text  Description automatically generated | Expected result |
| 2.4 | Attempts to validate and verify erroneous email | b | “False False” “False False” | A screenshot of a computer error  AI-generated content may be incorrect. | Expected result |

The new tests were all passed successfully. No more corrective actions will have to be taken.

I then tested the password validation. Each password should have 8-20 characters, upper and lowercase characters, symbols and numbers. The tests went as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test No. | Description | Test data | Expected Outcome | Actual Outcome | Comment |
| 1.1 | Attempt to validate a valid password | passwA!or33345 | “True True” | A screen shot of a computer  Description automatically generated | Expected result |
| 1.2 | Attempt to validate a password with no uppercase letters | password1! | “False False” |  | Expected result |
| 1.3 | Attempt to validate a password with exactly 8 characters | Pass1!rd | “True True” | A black screen with white text  Description automatically generated | Expected result |
| 1.4 | Attempts to validate a password with less than 8 characters | Pass1!r | “False False” | A screen shot of a computer  Description automatically generated | Expected result |
| 1.5 | Attempts to validate a password with exactly 20 characters | PasswordLengthTest1! | “True True” | A black screen with white text  Description automatically generated | Expected result |
| 1.6 | Attempts to validate a password with more than 20 characters | PasswordLengthTest1!! | “False False” | A computer screen shot of a test  Description automatically generated | Expected result |
| 1.7 | Attempt to validate a password with no lowercase letters | PASSWORD1! | “False False” | A screen shot of a computer  Description automatically generated | Expected result |
| 1.8 | Attempt to validate a password with no symbols | Password1 | “False False” |  | Expected result |
| 1.9 | Attempt to validate a password with no numbers | Password! | “False False” | A black screen with white text  Description automatically generated | Expected result |

I am happy with the password validation testing. All tests passed successfully and no code had to be altered.

3.4.4 Review

Overall, only one error was encountered when testing the email address verification. The error was fixed and further testing proved that it was fixed successfully. No other corrective actions were taken. This means that the validation is now complete which follows my success criteria.

## 3.5 Stage 4: Building The Admin View

The following section will outline the development and testing of the admin view.

3.5.1 Python code

Imports

The following libraries and files will need to be imported to the admin view file.



Tkinter – This library will be used for the user interface. I am using this library as it is lightweight and built in to python. This means that it won’t take up storage space.

logInMenu – This file is the login system file. It needs to be imported as the login menu will need to be created when the user signs out.

processWindows – As outlined in section 3.3.4.1, the processWindows file stores different functions that will be needed throughout the program. The functions that are imported into the admin view system will be used for different procedures (will be outlined in the following sections)

User interface



The user interface will be created according to the plan in section 2.5.2. The code above creates the main admin view window. First, when the user logs in, the window instance is created and placed in the middle of the user’s screen. Then, a menu bar instance is created (development outlined below) and displayed on the screen. Then, the labels and buttons are defined and placed on the window.



The class above creates a menu bar and displays it at the top of the admin view window. The constructor of this class calls the constructor of the parent frame class (from the menu class in the Tkinter library). Then, it stores the passed parameters in different variables and then calls the method to create the menu bar.

The toolBarMenu method of the class creates a tool bar instance and puts it on the top of the window. Then, It creates different menus and adds options and commands to them. When all the commands and menus have been defined, they are added to the tool bar instance.

The exit method exits the program. It is called when the exit command on the tool bar is pressed.

The signOut method destroys the admin view and creates a login window. It is called when the sign out command on the tool bar is selected.

3.5.1.1 Menu Bar Commands

Create Student account





The code above creates the student registration window in accordance to the design in section 2.5.2.1. Once this procedure is called, a Top level window instance is created. Then, the middle of the user’s screen is calculated and the window is placed there. Then, all of the form information (labels, text boxes, buttons, etc) are defined and placed on the window. When the student account information is added and the register account button is clicked, the following getVal procedure is called.



When the getVal procedure (shown above) is called, all of the information entered in the register window text boxes is fetched and stored in different variables. Then, the procedure checks if any of the fields are empty. If any fields are empty, an error message is created on the user’s screen. If no fields are empty, a for loop to determine where the position of the space in the student’s name is ran. Once the space is located, an if statement determines whether the entered name is too short. If that is the case, an error message is created on the user’s screen. If the name is an appropriate length, the first letter of the first and last name of the student is capitalised. Once that is complete, the program checks whether the entered passwords match. If they don’t an error message is created. If both passwords match each other, the checkVal procedure is called.



The checkVal procedure (shown above) validates the entered account information and register’s the student’s account. Once this procedure is called, the entered email is validated using the email validation function (developed in section 3.4.1). If the email is invalid, an error message is created on the screen. If the email is valid, the email is checked against the database to check whether it is already in use (development outlined in section 3.2.1). If the email is already in use, an error message is created on the user’s screen. If the email is not in use, the email is verified (development in section 3.4.1). If the email is not verified successfully, an error message is created. If the email is verified successfully, the password entered is validated (developed in section 3.4.1). If the password is not valid, an error message is created on the screen. If all tests pass, the account is created and added to the database (developed in section 3.2.1). Then, an email with the account information is sent to the student and a success message is displayed on the screen.



The procedure shown above sends an account creation email to the student. A try except statement is used as validation to avoid unexpected errors. Once this procedure is called, the email.message library is imported. This will allow the program to send emails to users. After the library is imported, the email server, port, sender email login information, receiver email, email subject and body are defined. The email is then created and a secure connection to the email server is established. Once a connection is established, the program logs in to the email and sends the email to the student. If any errors occur, an error message is displayed.

Settings system



The procedure above creates the UI for the password changing system. Once the procedure is called, a window instance is created and displayed in the middle of the screen. Then, all of the form information (labels, text boxes and buttons) are defined and placed on the window. When the new password is entered and the change password button is pressed, the checkPassword procedure is called.



The checkPassword procedure (shown above) validates the entered password and changes it in the database. Once this procedure is called, the validatePassword function is imported from the validation file. Then, the passwords entered are checked to see if they match. If the passwords do not match, an error message is displayed. If the passwords match, the password is then validated. If the password is invalid, an error message is displayed on the user’s screen. If the password is valid, the password is changed in the database (developed in section 3.2.1) and a success message is displayed.



The code above creates the UI for the email changing system. When this procedure is called, a new window instance is created. Then all the form information (labels, text boxes, buttons etc) are defined and placed on the window. When the new email is entered and the change email button is pressed, the checkEmail procedure is called.



The checkEmail procedure (shown above) verifies and validates the email entered. When the procedure is called, the validEmail and verifyEmail functions are imported from the validation file. Then, the emails entered are checked to see if they match. If they don’t match, an error message is displayed. If both emails match, the email is validated and then verified. If any of the checks fail, an error message is displayed on the screen. If all tests pass, the email is changed in the database (development in section 3.2.1) and a success message is displayed.

Class System



The procedure shown above creates the UI for the class creation system in accordance to the design in section 2.5.2.1. When the procedure is called, a new window instance is created. The middle of the user’s screen is calculated and the window instance is placed there. Then, all of the form information (labels, text boxes, buttons etc) are defined and placed on the window. When the add students button is clicked, a new window with a list of students is created (development outlined below) and the selected students are stored in the students list. When the create class button is pressed, the create\_class() subroutine is called. When this subroutine is called, the length of the students list and class name are checked if they are empty. If any of the fields are empty, an error message is displayed on the screen. If both fields are not empty, the class is created in the database using the createClass() subroutine in the SQL file (developed in section 3.2.1) and the window is destroyed.



The procedure above creates the student list UI according to the design in section 2.5.2.1.. This procedure is called when the user presses the add students button on the create class window. When this procedure is called, a window instance is created and it is placed in the middle of the user’s screen. All of the students are fetched from the database (development in section 3.2.1) and stored in a list variable. Then, a list box is defined and placed on the window. The student list is then sorted into alphabetical order and placed in the list box. An empty list of selected is defined, this will be used to store all of the students that the teacher picked. When the wanted students are selected and the select button is pressed, all of the selected names are added to the student list and returned back to the create class form.



The procedure above creates the delete class UI in accordance to the design in section 2.5.2.1. When the procedure is called, a window instance is created and placed in the middle of the user’s screen. All of the classes under a certain teacher are fetched from the database (development outlined in section 3.2.1) and stored in a list variable. Then, a list box is created and placed in the window. The fetched classes are sorted alphabetically and placed in the list box. An empty list of selected classes is defined. When the classes that the user wants to delete are selected and the delete button is clicked, the delete() subroutine is called. This subroutine adds the selected classes to the selectedClass list and deletes the classes from the database (developed in section 3.2.1). If the classes are deleted successfully, a success message is displayed. If any errors occur, an error message will be displayed.

2.5.1.2 Main menu commands

When the main admin view is created according to the design in section 2.5.2.1, there should be two buttons on the main window. The development for both of those systems is outlined below.

Create assignment system





The procedure shown above creates the create assignment UI in accordance with section 2.5.2.1. When the procedure is called (by pressing the create assignment button on the main admin view), a window instance is created and all of the form information is defined and placed on the window. Then, all of the classes that a teacher manages are fetched from the database (development in section 3.2.1) and stored in a list. The fetched classes are then placed in a option menu. This will allow the teacher to chooses to which class the assignment is assigned to. When the choose date button is clicked, the calendar() subroutine is called. This subroutine creates a new window with a calendar where the teacher can choose a due date. When the due date is selected, it is displayed in a read-only text box. When the next button is selected, the assignment is added to the database (developed in section 3.2.1) and the nextAssign() procedure is called.



When the procedure above is called, it destroys the supplied window. Then it begins the create question process or goes to the next create question window (depending on where in the assignment process the user is).



The procedure above creates the add question window and form in accordance to the design in section 2.5.2.1. When the procedure is called, a new window instance is created and placed in the middle of the user’s screen. Then, the form information is defined and placed in the window instance. An option menu is created and placed so the user can choose what type of question they are creating. When the add question button is pressed, the question is inserted into the assignment in the database (development outlined in section 3.2.1) and the nextAssign() subroutine is called. This process repeats until the user clicks the finish button which will destroy the window.

View assignment system



The procedure above creates the submission view window using the notebook method from the Tkinter library. All of the objects for the different tabs are placed in frames in different procedures (outlined below).



The procedure above defines and places the view assignment tab objects in accordance to the design in section 2.5.2.1. When the procedure is called, a window title is defined and placed on the top of the window. Then, all of the assignment information is fetched from the database (development outlined in section 3.2.1) and stored in the assignment variable. If the assignment variable is empty, a “No assignments” label is placed on the window. If the variable is not empty, a tree view table is created and its columns are defined. Then, the assignment information is placed in the respective column. Sort by buttons are also created and placed underneath the table. These buttons will sort the table in ascending order of a certain column.

3.5.2 Testing

Comment on the result

Show changes/correction and further testing

3.5.3 Stakeholder feedback

Review

## 3.6 Stage 5: Building The Student View

The following section will outline the development and testing of the admin view.

3.6.1 Python code

## 3.7 Final Review, Improvements and Corrective Actions

I also tested if the UI of the application displayed correctly on different operating systems. I tested my UI on Windows 11 and a Linux based system.

|  |  |  |
| --- | --- | --- |
| Windows | Linux | Outcome |
|  | A screenshot of a computer login  AI-generated content may be incorrect. | The UI is the same on both systems. |
|  | A screenshot of a computer security  AI-generated content may be incorrect. | The UI is the same on both systems. |
|  | A screenshot of a computer  AI-generated content may be incorrect. | The UI is the same on both systems. |
|  | A screenshot of a computer  AI-generated content may be incorrect. | The UI is the same on both systems. |
|  | A screenshot of a computer  AI-generated content may be incorrect. | The UI is the same on both systems. |
|  | A screenshot of a computer  AI-generated content may be incorrect. | The UI is the same on both systems. |
|  | A screenshot of a computer  AI-generated content may be incorrect. | The UI is the same on both systems. |

The user interface was the same on both systems. This means that there is no need for any corrective actions to be taken.

# Chapter Four: Evaluation

Evaluation

## 4.1 Introduction

## 4.2 Testing to inform evaluation

## 4.3 Evaluation

Evaluating usability features

Evaluating Robustness

Limitations and Maintenance