AcsessIQ: Intelligent Testing & Analytics Application by Todor Andreev

Analysis

Problem Identification and Significance

Many students struggle to understand abstract computer science concepts such as algorithms and data structures, and more importantly for the project, I am one of these students. Traditional learning methods, such as textbooks and static diagrams, often fail to provide the interactive and visual learning experiences needed for effectively comprehending such complex topics. This gap in understanding can lead to decreased engagement and lower performance in the studies of students/self-learners in Computer Science, hampering their progress and potentially decreasing their interest in the target industries.

Justification of the Problem

The lack of interactive and visual learning tools is a significant issue because it hampers the ability of students to develop a deep understanding of key computer science principles. Algorithms and data structures are foundational to many advanced topics in computer science, and a weak grasp of these concepts can impede further learning and application. By providing a tool that makes these concepts more accessible and engaging, we can improve educational outcomes and foster a deeper interest in the field of computer science.

Target Users

The primary target users for my solution are:

- **Students**: Particularly those studying computer science at the GCSE and A-Level stages, who need to understand algorithms and data structures as part of their curriculum.
- **Educators**: Teachers and lecturers who require effective teaching aids to enhance their lessons and provide students with practical, hands-on learning experiences.
- **Self-Learners**: Individuals learning computer science independently who would benefit from an interactive tool to aid their understanding of complex concepts.

Expected Impact and Benefits

The interactive maze generation and solving application will provide several key benefits:

Enhanced Understanding: By visually demonstrating how different algorithms work, the tool will help students gain a clearer and more intuitive understanding of these concepts.

Increased Engagement: The interactive nature of the tool will make learning more engaging and enjoyable, helping to maintain student interest and motivation, while advancing their own skills.
Practical Application: Students will be able to visually see the practical applications of the algorithms they learn, bridging the gap between theory and practice.
Versatility: Educators can use the tool in a variety of teaching contexts, from classroom demonstrations to individual assignments and exercises.
Skill Development: By using the tool, students will develop critical thinking and problem-solving skills as they interact with and analyze the different algorithms.

Solution Overview

To address this problem, I will develop an interactive maze generation and solving application that demonstrates how algorithms like Recursive Backtracking, Prim's Algorithm, and Kruskal's Algorithm work. The application will feature the following components:

- **Maze Generation**: The ability to generate random mazes using different algorithms.
- **Algorithm Visualization**: Step-by-step visualizations of how each algorithm progresses through the maze generation process.
- **Maze Solving**: Implementation of pathfinding algorithms to solve the generated mazes, providing further educational value.
- **Performance Metrics**: Comparative analysis of the efficiency and effectiveness of each algorithm, helping users understand their strengths and weaknesses.

Implementation Plan

The development of the application will follow these stages:

Research: Conduct thorough research on the algorithms to be implemented and their educational value.
Design: Create a detailed design of the application, including the user interface and visualization components.
Development: Implement the maze generation and solving algorithms in Python, ensuring clear and informative visualizations.
Testing: Rigorously test the application to ensure it functions correctly and effectively demonstrates the concepts.
Evaluation: Gather feedback from target users (students, educators) and make necessary improvements to optimize the tool's educational impact.

Investigation Preparation

Both interviews and surveys have advantages and disadvantages which are important to consider. Interviewing a shareholder in person can allow for a more personalized, in-depth discussion to take place, and the opportunity to follow up on questions whenever needed/revisiting the questions. The data provided from said interview is most likely to be qualitative, which must be interpreted in context rather than being put into a spreadsheet and easily categorized, taking up considerably more time for the interviewer to process the data, although the information gained will likely be of greater use than simply the quantitative data. On the other hand, surveys take up much less time, provide quantitative data, which is easily categorized and efficiently analyzed, allowing for statistical graphical comparisons to quickly be created and to assess the data.

To gather sufficient research from each of my demographics, I will conduct four interviews in total, one with an Educator in my school, Andre Wallace, who teaches both GCSE and A-Level Computer Science students in the institution, allowing me to understand the specific concerns of educators when it comes to teaching complex concepts such as algorithms and data structures, which generally is an area of reduced comprehension for students. The remaining three interviews will be conducted with students, with two of them being A-Level/GCSE students from our school(with the interviews being out-of-class and in-person), with the final interview being an online on-call interview with a University student (who was previously also a self-learner) to factor in their opinion a student in further education that has spent more time with these concepts, and that has already previously completed the A-Level+/GCSE curriculum and more. This approach ensures that the data is less biased, as students/educators from various stages in Computer Science will be interviewed, providing diverse yet fair perspectives, and providing the needs and requirements of both students and educators.

During the interview with Andre, I will shift focus on questions pertaining of his time spent on teaching the concepts of algorithms and data structures to GCSE and A-Level students, and identifying the advantages and disadvantages of his personal approach on teaching said concepts, while also enquiring on the methods that were used while he was still in education himself, allowing me a deeper insight into the various teaching methods involved other than those I am already familiar with, which will allow me to identify new changes that could be made to my own approach along its development to enhance my new solution.

For my student questions, I will modify the questions to try and understand the viewpoint of students on this issue. These questions will delve into the challenges met by students during revision, the limitations of the existing solutions currently in use, and features that the students feel will enhance the student learning experience for algorithms and data structures.

In addition to the interview findings, a detailed survey will be conducted using Typeform. The survey will include open-ended questions, allowing students to provide detailed responses, while also including some ‘Yes/No’ questions that allow for the grouping of quantitative data. In

addition to being sent to students within the Computer Science classes, the surveys will also be sent to some University students and self-learners for an indicative outside perspective.

By integrating the qualitative data gathered from interviews with the quantitative data obtained from surveys, I will gain a clear and comprehensive understanding of the problem. This dual approach will allow me to identify specific needs, uncover valuable insights, and develop a robust solution that meets the requirements of all stakeholders.

Conclusion

By addressing the problem of understanding algorithms and data structures through an interactive and visual approach, this project aims to significantly enhance the educational experience for computer science students (and more specifically GCSE/A-Level students or even self-learners). The developed application will serve as a valuable resource for both students and educators, bridging the gap between theoretical learning and practical application through its GUI, which provides another manner of perspective for the students to look back to when attempting to comprehend such complex topics.

Questions for interview with Andre Wallace

Question 1: “What resources, system(s)/method(s) do you currently use while teaching these concepts?” This will give me an idea of the current pre-existing solutions to this problem that are currently in use by Andre, including the solutions I was already aware of. Although there may be a greater number of solutions, these may be the most appropriate according to his own perspective and could be most ideal for him for the time being.

Question 2: “What are the advantages of the current system(s) that you use?”

Question 3: “What are the disadvantages or limitations of the current system(s) in use by you personally, and/or of those used by others that you’re aware of?” I need to understand the advantages and disadvantages of the current solutions he uses and the ones he is aware of. This will be ideal as there may be advantages/disadvantages he has identified as an educator of the concepts in day-to-day teaching that I might not see from more broad research due to having to look for many existing solutions and having little access to direct perspectives from teacher themselves and the time provided would not be sufficient to become adept and explore most solutions as in depth as an educator with a passion for his subject area.

Question 4: “Are there any classes that could benefit more from this solution than the rest?". Andre teaches every Year Group from the Years 10 through 13, so this will be practical to access if any have a more efficient teaching format than the others (mostly due to the curriculum gap

from GCSE to A-Level, or personal research in their free time).This would mean that I have the opportunity to alter certain functions if need be.

Question 5: “How would you like students to answer the questions I will provide?” In the context of testing, I need to know how students prefer to answer questions or what format of answers are seen as sufficient from an educator that knows them personally. This is due to multiple choice answers or selections being simpler to implement in comparison, as the data is quantitative and easier to store + categorize into groups, to worded answers, which fall into qualitative data and require further context and understanding to consider.

Question 6: “What new features would be the most important?” This would enable me to identify the most applicable features not present in the current solutions which I can include in the software.

Question 7: “What is the current system for recording and tracking students understanding and progress in their comprehension for these concepts that you use?” For my software to be a complete solution, I must understand the way Andre records and analyzes the data of his students’ comprehension and progression in learning. Which will let me focus on other aspects of the software depending on the efficiency or improved usability from said feedback.

Outline of interview with Andre Wallace

Andre Wallace is the main Educator/Teacher in Computer Science for Years 10-13

Question 1:” What resources, system(s)/method(s) do you currently use while teaching these concepts?”

Answer: -Methods of Teaching include Inquiry Based, Flipped Classroom and the usual lecture and demonstrations.

Question 2: “What are the advantages of the current system(s) that you use?”

-Pupil inclusion allows for inclusion of various problem-solving mechanisms to aid solvency.

-A direct approach to ongoing comprehension.

Question 3: “What are the disadvantages or limitations of the current system(s) in use by you personally, and/or of those used by others that you’re aware of?”

Answer:

-Time consumption

-May not be applicable/beneficial

-Fast paced, student dependent

Question 4: “Are there any classes that could benefit more from this solution than the rest? "

Answer:

-The lower streams could have it integrated to allow for a more student-centered approach with more responsibility placed on their outcome and approach.

Question 5: “How would you like students to answer the questions I will provide?”

Answer:

-Under time and test approaches

-On their own

-With suitable justification

Question 6: “What new features would be the most important?” This would enable me to identify the most applicable features not present in the current solutions which I can include in the software.

Answer:

-Effort trajectory (how many attempts it takes for a student to improve their understanding of the concept)

-Score tracker (for assessment of progress in learning)

Question 7: “What is the current system for recording and tracking students understanding and progress in their comprehension for these concepts that you use?” For my software to be a complete solution, I must understand the way Andre records and analyzes the data of his students’ comprehension and progression in learning.

Answer:

-Questioning (Socratic and in class probing)

-Paper Based Activities

-Timed Activities

-Past Paper Questions

-Verbal sharing

-Scenario based questions

-Case studies

-Practical Application in Activities

Analysis of Interview with Andre Wallace

To begin, I have identified the selected methods of recording the ability of comprehension of students in use by Mr. Wallace: Paper Based Activities (Including Past Paper Questions), mostly procured from the OCR A-Level Computer Science book and the website 'Physics and Maths Tutor' which contains more questions which fit the H446 specification. He would assess the ability of the student for these topics via their overall score results from the paper, which is a suitable model that can swiftly identify the specific concept areas which the students lack in and could develop further. In addition, receiving feedback from the teacher can be quick, as once the teacher has marked the questions, the student can quickly deduce/be told which specific topics to revisit. However, if the students do not have immediate contact with the teacher, feedback could take much longer to be received, and revision would be delayed. Another method of teaching for algorithms which I identified in the interview was the use of YouTube videos, specifically "Craig 'n' Dave", a channel which specializes in teaching GCSE + A-Level topics to students, which is yet another suitable method, and is possibly a preferred method of learning by students (Which I will investigate by creating a survey for the students).This is a great option as it allows students to revisit content in their own free time if need be, if the teacher is not available to support them. The effectiveness of both educational tools is a useful part of student revision, however in the interview Andre outlined that more streams of revision would advance the knowledge of students greatly, and improve their overall understanding, especially any solutions that provide a visual representation of the algorithms +/data structures and any interactive solutions, as this will expand the understanding of the students. I will take the thoughts that I received from Andre in the interview greatly, as he is a Computer Science teacher with years of experience, and is a shareholder in my project, as any innovation through the project will directly benefit his students, indirectly benefiting him as he wouldn't have to spend time to develop other sources of teaching/revision on his own, meaning he could wholly focus on assessing his students through the methods he already uses and mentioned in the interview(Past Paper Questions, Probing, Timed activities and etc.)The insights I gained from the interview will help me in the development of the solution, with an emphasis on the GUI and interactive aspect of the project, to allow for students to truly connect with the concepts through the algorithms that they are seeing. In conclusion, the interview highlights that there is a considerable time saving and improvement of efficiency for the teacher by allowing him to focus on the marking and feedback process of his Past Paper questions/Timed activities. I have also identified areas where his current solutions fall short, such as the limited availability of past papers or questions/the downsides of the textbook approach to explain complex, theoretical based topics, with my solution being developing a further stream of revision material that incorporates a Graphical

User Interface and interactivity. These insights provide valuable considerations for my project and have shown that the requirements for a revision solution do not vary as much between years as initially predicted, according to Andre.

Questions for interview with the students

Question 1:" Can you describe any difficulties you have faced while learning algorithms and data structures?"
Question 2: "Do you prefer learning computer science concepts through textbooks, interactive tools, videos, or other methods? Please specify."
Question 3: "How often do you use visual aids (like diagrams and animations) to understand complex concepts?"
Question 4: "Would you find it helpful to see step-by-step visualizations of algorithms in action? Why or why not?"
Question 5: "Have you used any interactive learning tools or software for studying computer science? If yes, which ones and what was your experience?"
Question 6: "What features would you like to see in an interactive tool designed to teach algorithms and data structures?"
Question 7: "What types of interactive elements (e.g., animations, games, puzzles) do you think would make learning algorithms more engaging for you?"
Question 8: "How do you think an interactive maze generation and solving application could help you better understand algorithms?"
Question 9: "How important is it for you to see the real-world applications of the algorithms you learn?"
Question 10: "Can you provide examples of how seeing practical applications has helped you understand a concept better in the past?"
Question 11: "How user-friendly do you find most educational software? What improvements would you suggest?"
Question 12: "What type of support or resources (e.g., tutorials, forums, one-on-one help) do you find most useful when learning new concepts?"
Question 13: "If an interactive maze generation and solving application were available, how likely would you be to use it regularly to study algorithms and data structures?"

Interview with Justinas Dorosenko

Question 1:" Can you describe any difficulties you have faced while learning algorithms and data structures?" Answer: "I have had difficulty with visualizing and understanding topics such as graph algorithms and recursion."
Question 2: "Do you prefer learning computer science concepts through textbooks, interactive tools, videos, or other methods?" Answer: "I prefer learning mainly through YouTube videos

and other interactive tools instead of the textbooks, as they are not clear and concise enough at times."
Question 3: "How often do you use visual aids (like diagrams and animations) to understand complex concepts?" Answer:" I frequently use diagrams and animations as I am a visual learner, and it helps me comprehend the concept better."
Question 4: "Would you find it helpful to see step-by-step visualizations of algorithms in action? Why or why not?" Answer: "I would find it helpful, as algorithms are a difficult concept for me to understand simply through a textbook, compared to a step-by-step visualization."
Question 5: "Have you used any interactive learning tools or software for studying computer science? If yes, which ones and what was your experience?"
Answer: "I have used Seneca Learning for Computer Science in the past, which is a useful application."
Question 6: "What features would you like to see in an interactive tool designed to teach algorithms and data structures?" Answer: "A visual representation would be most helpful to me, that could be launched by pressing a button on the application to start the algorithm."
Question 7: "What types of interactive elements (e.g., animations, games, puzzles) do you think would make learning algorithms more engaging for you?" Answer: "Animations, puzzles, and games would all be useful and engaging for me to use in my learning process."
Question 8: "How do you think an interactive maze generation and solving application could help you better understand algorithms?" Answer:" I think that seeing the algorithm generate visually on my screen would help me with understanding the processes behind it."
Question 9: "How important is it for you to see the real-world applications of the algorithms you learn?" Answer:" For me personally it is particularly important, as seeing a real-life example usually helps me understand concepts better in context and their real uses."
Question 10: "What type of support or resources (e.g., tutorials, forums, one-on-one help) do you find most useful when learning new concepts?" Answer:" I usually find Video tutorials and online discussions to be the most useful for me in my learning."
Question 11: "If an interactive maze generation and solving application were available, how likely would you be to use it regularly to study algorithms and data structures?" Answer:" I would definitely use it alongside the other resources I have access to, to be able to understand algorithms better than I do now."

Analysis of interview with Justinas Dorosenko

I initially asked Justinas Dorosenko, a Year 12 student who takes A-Level Computer Science, whether he has/had any difficulty with comprehending the concept of algorithms at the A-Level level, to which he replied with yes. This information is crucial to establish that the stakeholder is relevant to my solution, as he both takes Computer Science at the A-Level level and has had difficulties with understanding algorithms in the past, making him a suitable candidate for an

interview. When I inquired, Justinas stated that he primarily uses YouTube videos (Specifically, Craig ‘n’ Dave), Past Paper exams and Seneca Learning for revision.

The advantage of using YouTube videos is that it provides remote access to learning resources, while being more concise and removing unnecessary details, and such is the case with Craig ‘n’ Dave’s videos. Additionally, Justinas finds the YouTube videos to be a quicker way of revision that can be done without being burned out on revision. However, Justinas also mentioned that the downsides of learning through the YouTube videos is that they are not interactive enough, making him lose focus while watching them at times and procrastinating on revision, and that these videos, by and large, do not include many or even any questions to practice this knowledge. Furthermore, Justinas stated that Past Paper Exams used for practice, especially under time constraints, helped him develop his understanding of different algorithms.

When discussing the most important new feature for a revision solution, Justinas mentioned the need for a different type of revision, which focuses on visually representing the algorithms which are studied in his respective curriculum at their full complexity, which the revision methods he mentioned do not usually do. Most importantly, the current solution that Justinas uses does not have this crucial feature.

Interview with William David

Question 1:” Can you describe any difficulties you have faced while learning algorithms and data structures?” Answer: “I’ve struggled with how monotonous the teaching methods for these concepts are.”
Question 2: “Do you prefer learning computer science concepts through textbooks, interactive tools, videos, or other methods?” Answer: “I prefer learning these concepts through interactive tools, such as Seneca Learning, as well as YouTube videos, such as Craig’n’Dave’s videos.”
Question 3: “How often do you use visual aids (like diagrams and animations) to understand complex concepts?” Answer:” I use visual aids after lessons where I’ve struggled with the concepts of the day.”
Question 4: “Would you find it helpful to see step-by-step visualizations of algorithms in action? Why or why not?” Answer: "Yes, as it removes the monotony of the previous learning methods, and decomposes the entire process, giving me a fuller understanding of the concepts, while making it more interactive.”
Question 5: “Have you used any interactive learning tools or software for studying computer science? If yes, which ones and what was your experience?”
Answer: "I mainly use YouTube videos, as well as apps such as Quizzizz and Seneca Learning to complete short form, quick revision questions. Other than those, I sometimes use OpenAI to generate explanations to certain problems that I could not find solutions for.”
Question 6: “What features would you like to see in an interactive tool designed to teach algorithms and data structures?” Answer: “I would like visual representations that directly link to

a question (where I can see the algorithm), as well as flow charts for example, to decompose the problem and use each piece as an explanatory window."
Question 7: "What types of interactive elements (e.g., animations, games, puzzles) do you think would make learning algorithms more engaging for you?" Answer: "Puzzles, related to explaining the algorithms at hand would be very useful for my revision, as they would push my brain to understand the concept through multiple revision pathways."
Question 8: "How do you think an interactive maze generation and solving application could help you better understand algorithms?" Answer:" It would be a unique piece of revision, as I've not seen it anywhere previously, and the fact that it's not something I've seen before means it's more likely to draw my attention in and allow me to focus better."
Question 9: "How important is it for you to see the real-world applications of the algorithms you learn?" Answer: "Very important, as otherwise it will feel like the algorithms are just a useless part of the curriculum, which they are not, and seeing the real-world applications will help me learn the ins and outs and purpose of those algorithms."
Question 10: "What type of support or resources (e.g., tutorials, forums, one-on-one help) do you find most useful when learning new concepts?" Answer: "Tutorials help me the most, especially tutorials for questions/concepts. While one-to-one help is more useful and practical for me when working on a Computer Science project for example."
Question 11: "If an interactive maze generation and solving application were available, how likely would you be to use it regularly to study algorithms and data structures? "Answer: "I will likely use it less frequently than videos, as well as practice questions, however it will be a new and unique revision method that will keep my revision from becoming monotonous and boring and will probably increase my interest and understanding of these concepts."

Analysis of interview with William David

Firstly, William expressed his annoyance with the monotony of the learning methods of data structures and algorithms, as the only learning methods he knows of are either videos online , or websites such as Seneca Learning, which according to him are not enough to promote his interest in teaching them, although they are visual aids that greatly assist him.

If the application were personalised to him, the best learning method would be step-by-step visualisation of the process, as it would decompose the problem and remove the repetitiveness in the process, while giving him a further understanding, which will be the focal point of my project. Another learning site that Will mentioned was 'Quizzizz', which upon research seems to be an interactive and visual quiz website which proves beneficial to his studies, especially for quick revision questions. Meanwhile he also uses OpenAI to generate different explanations of the concepts that he needs to understand and could not find solutions for, which seems to be a smart choice. Furthermore, he expressed his desire for visual diagrams such as flowcharts, which I will likely not include in the program, however that is subject to change, and I will include flowcharts in the presentation as a learning aid. The feedback that has been most well received from William's answers is the desire to have an application that is similar to Quizziz, in the sense

that it uses puzzles and questions in a unique and fun way to teach students new concepts, as he also expressed that tutorials and step-by-step solutions help him the most(in this case being explanations in the questions),and I will possibly recentre my main solution around this idea, as the visual aid of the algorithms may not be enough to entice the students, while being very one-dimensional as in its intended use, it may only be a pop-up start button and generation of the algorithms when another button is clicked and as William answered, it will not be his main source of revision , and may not help him as much as a quiz application. With this analysis, I conclude that switching the method of attaining the solution of helping students with their learning from a strictly repetitive visual-aid to a quiz-like application with answers and feedback will be much more effective for the revision of a student.

Survey

I shall conduct a survey consisting of students that take computer science to receive their input on what course of action would be most appropriate to find a solution.

I have come up with the following questions for said survey:

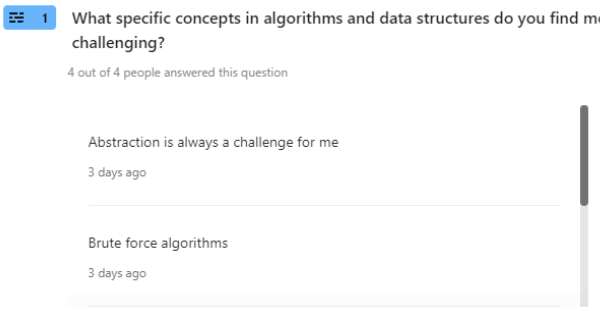
What specific concepts in algorithms and data structures do you find most challenging?

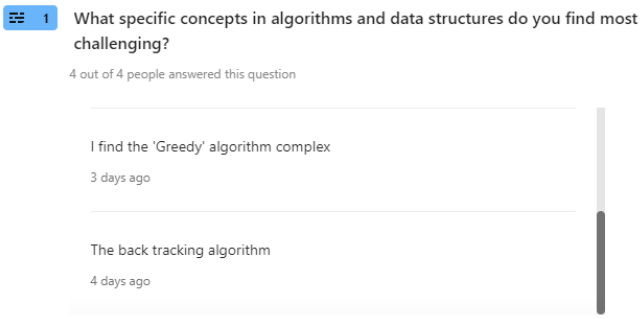
Which method do you prefer for learning computer science concepts? (Textbooks, interactive tools, videos, etc.)
How often do you use visual aids (like diagrams and animations) to understand complex concepts? (Rarely, Sometimes, Often, Always)
Would step-by-step visualizations of algorithms be helpful to you? Why or why not?
Have you used any interactive learning tools for studying computer science? If yes, which ones?
What features would you like in an interactive tool for teaching algorithms and data structures?
What types of interactive elements (e.g., animations, games, puzzles) would make learning algorithms more engaging?
How could an interactive maze generation and solving application help you understand algorithms better?
How important is it for you to see the real-world applications of the algorithms you learn? (Not important, somewhat important, very important)
Do you think seeing practical applications executed will help you understand a concept better?
How important is the ability to track your progress in an educational tool? (Not important, somewhat important, very important)

What type of support or resources (e.g., tutorials, forums, one-on-one help) do you find most useful when learning new concepts?
How likely would you be to use an interactive maze generation and solving application regularly to study algorithms and data structures? (Unlikely, somewhat likely, very likely)

Is there anything else you would like to see included in an interactive tool for learning algorithms and data structures?

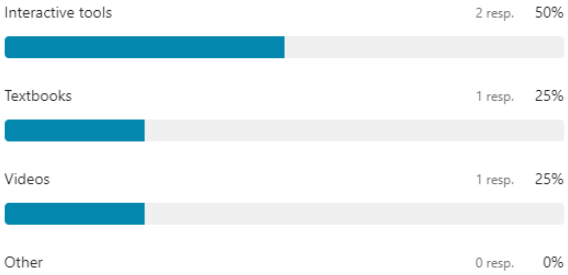
Survey Results:





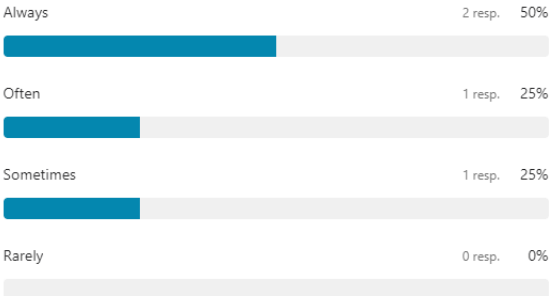
2 **Which method do you prefer for learning computer science concepts? (Textbooks, interactive tools, videos, etc.)**

4 out of 4 people answered this question



3 **How often do you use visual aids (like diagrams and animations) to understand complex concepts?**

4 out of 4 people answered this question



4 **Would step-by-step visualizations of algorithms be helpful to you? Why or why not?**

4 out of 4 people answered this question

Yes as it allows me to visualise the methodology behind each algorithm

3 days ago

yes as it would break down the separate parts more clearly

3 days ago

4 **Would step-by-step visualizations of algorithms be helpful to you? Why or why not?**

4 out of 4 people answered this question

It would be helpful , since it usually requires more explanation for me to understand

3 days ago

Very helpful because I'm a visual learner

4 days ago

5 **Have you used any interactive learning tools for studying computer science? If yes, which ones?**

4 out of 4 people answered this question

Seneca

3 days ago

Seneca learning and quizzes.com

3 days ago

5 **Have you used any interactive learning tools for studying computer science? If yes, which ones?**

4 out of 4 people answered this question

I have used online quizzes and interactive learning games

3 days ago

No

4 days ago

6 **What features would you like in an interactive tool for teaching algorithms and data structures?**

4 out of 4 people answered this question

Interactive videos where each section of the video has a task before you can move on

3 days ago

Incorrectly answered questions providing guidance to help the student understand it

6 **What features would you like in an interactive tool for teaching algorithms and data structures?**

4 out of 4 people answered this question

A visual aid , to visualise the processes that are happening

3 days ago

Video diagrams and then applying it in a question

4 days ago

7 What types of interactive elements (e.g., animations, games, puzzles) would make learning algorithms more engaging?

4 out of 4 people answered this question

puzzels would make a challenging topuc engaging

3 days ago

an incentive to get correct answers (eg a points system)

3 days ago

7 What types of interactive elements (e.g., animations, games, puzzles) would make learning algorithms more engaging?

4 out of 4 people answered this question

Animations and games , as I am a visual learner

3 days ago

All of the above

4 days ago

8 How could an interactive maze generation and solving application help you understand algorithms better?

4 out of 4 people answered this question

it would allow me to take the visual guide through the problem

3 days ago

Having an interactive element

3 days ago

8 How could an interactive maze generation and solving application help you understand algorithms better?

4 out of 4 people answered this question

I would be able to see the algorithm as it's working to fulfill it's task , hence I will have a better understanding of how it works

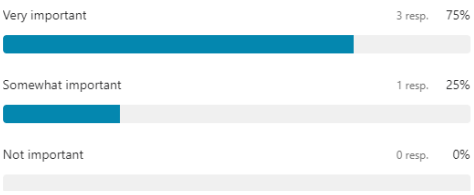
3 days ago

By being interactive

4 days ago

9 How important is it for you to see the real-world applications of the algorithms you learn?

4 out of 4 people answered this question

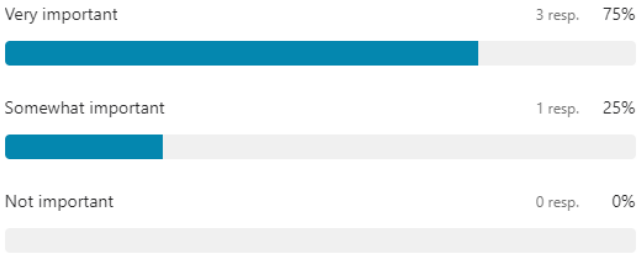


10 Do you think seeing practical applications execute will help you understand a concept better?

4 out of 4 people answered this question

11 How important is the ability to track your progress in an educational tool?

4 out of 4 people answered this question



12 What type of support or resources (e.g., tutorials, forums, one-on-one help) do you find most useful when learning new concepts?

4 out of 4 people answered this question

tutorials and one on one help

3 days ago

One to one help

3 days ago

12 What type of support or resources (e.g., tutorials, forums, one-on-one help) do you find most useful when learning new concepts?

4 out of 4 people answered this question

Tutorials and one-on-one help

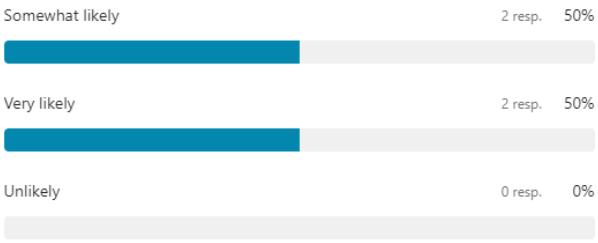
3 days ago

Tutorials and 1 on 1 help

4 days ago

13 How likely would you be to use an interactive maze generation and solving application regularly to study algorithms and data structures?

4 out of 4 people answered this question



14 Is there anything else you would like to see included in an interactive tool for learning algorithms and data structures?

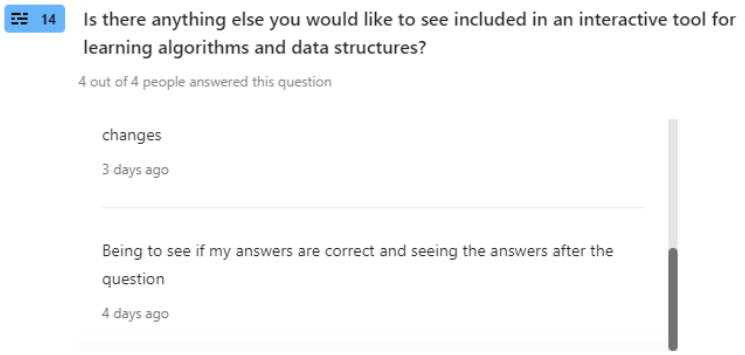
4 out of 4 people answered this question

Exam style questions disguised as little mini games

3 days ago

a simple to check answers and receive instant feedback

3 days ago

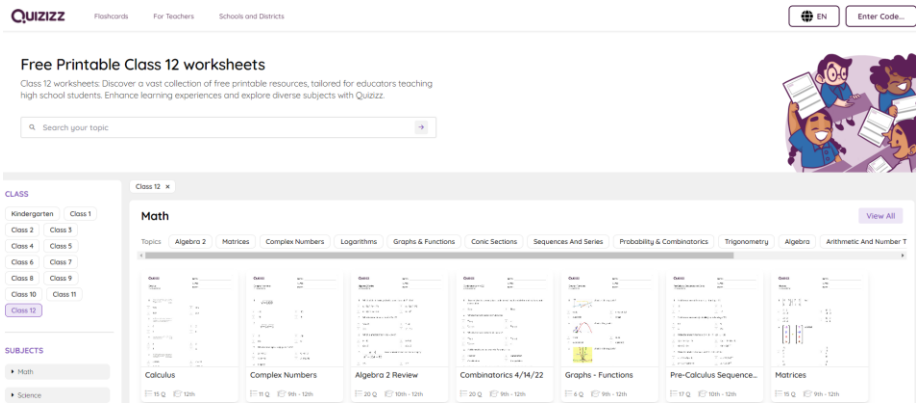


Survey Analysis

To begin, I have identified the current solutions used by the students that participated in the surveys, alongside the advantages and disadvantages of each; with the most notable being Seneca Learning, Computer science textbooks, Quizzizz and YouTube videos, while I have also gained further knowledge of what features are the most important for students, which come out to mainly be interactive questions which provide the answer to the student regardless of the answer, with an emphasis on questions that relate to the real world application of the knowledge that is learned by the students and provide useful feedback, as well as the ability to track their progress while continuing to use the revision tool. Furthermore, the students have stressed that specific algorithms are difficult to understand, such as the Back-tracking algorithm and the Greedy algorithm, hence I must cater questions relevant to these concepts, which is making me lean towards transforming the solution into a question-based interactive application instead of a purely visual algorithm generator, as the question app would be more useful to students as expressed by the survey, and over half of the students are only ‘Somewhat likely’ to use the maze generator while the overwhelming majority of them would use a question-based application more frequently and could benefit more from it. Additionally, adding a feature that lets users create and search for user-created questions would significantly increase the amount of learning content but would require the ability to search for specific questions/users. Alongside that, randomly generated questions based on these topics may prove useful to the users, however it would require a large database of questions that can be edited as well as generated during runtime, which may not be feasible due to the multitude of topics and inclusion of worded questions. Lastly, creating questions/quizzes needs to be a straightforward process that every user should be able to understand and follow, with an example being provided.

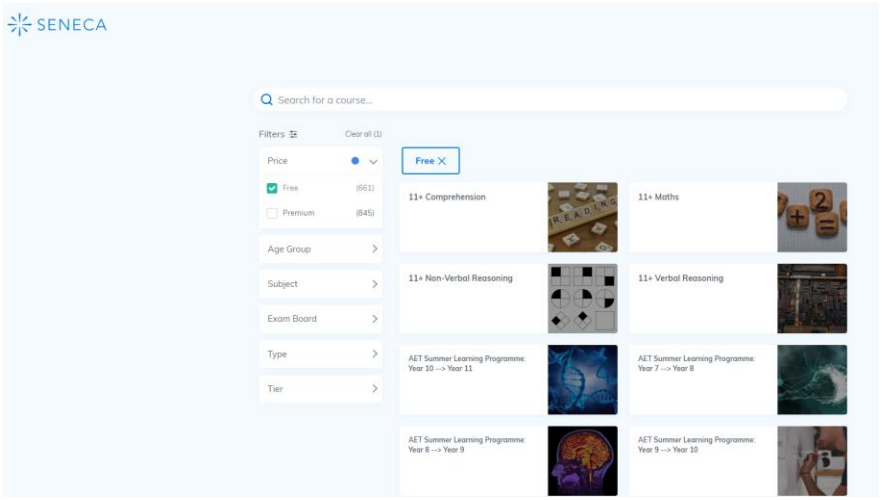
Investigation of existing solutions

Quizzizz(quizzizz.com)



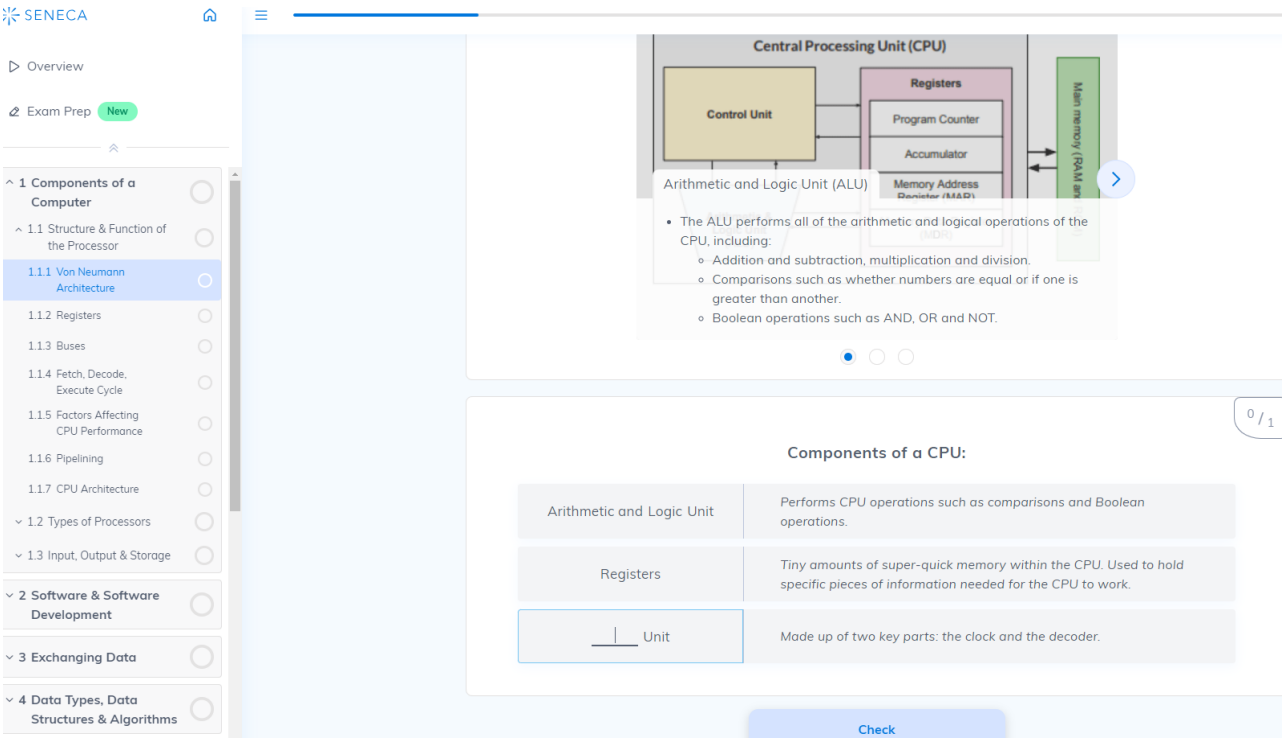
Quizizz is a web and application-based platform where users can upload learning content that can be shared amongst other users, such as quizzes and tests on self-demand with constant feedback, as no teachers are needed to post and create the questions themselves, with a wide variety of free content, like other platforms such as YouTube. Alongside that there are live quizzes where many users can participate simultaneously which increases student participation and interactions. Applications like Quizzizz were most frequently mentioned and preferred by students in my survey.

Seneca Learning



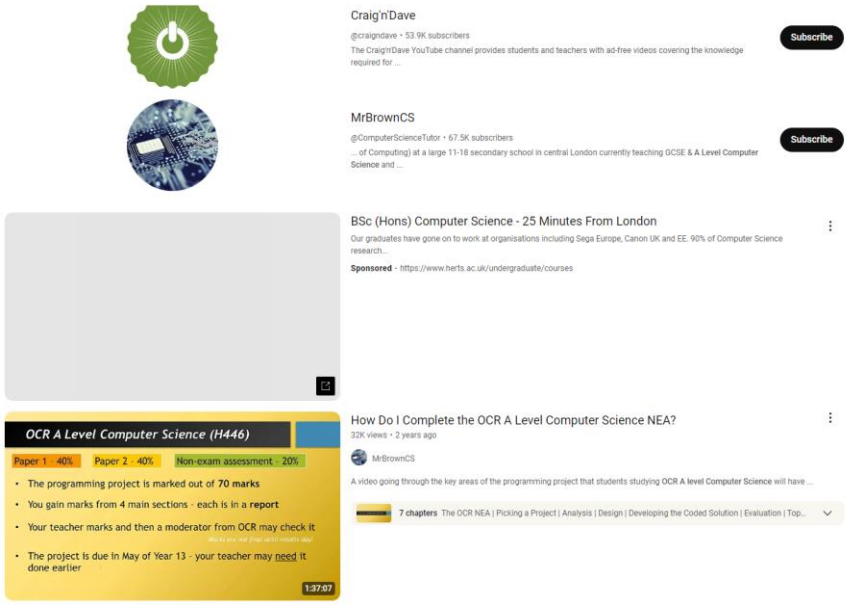
Through my interviews and surveys, I have found another preferred solution by students, another web-based learning application named ‘Seneca Learning’, which I have used myself for Computer Science revision in the past, from the screenshot above we can see that upon loading up the website, we receive immediate access to available courses, with the possibility of creating a free account to track our progress. This is a strong feature that will be preferred in my project and will increase participation from students if programmed successfully. The main difference between ‘Seneca Learning’ and the previously mentioned ‘Quizzizz’ is that ’Seneca Learning’ apart from receiving support and feedback on questions/tests/quizzes from non-teachers, you can

join classes and receive assignments from your own teachers, which is a very useful feature but will take some difficulty to include in my project.



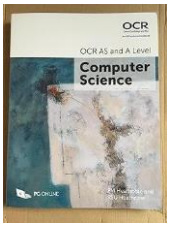
Additionally, from the screenshot above we can see that each course is split into individual modules on the exam board that it is made for, with additional sub-modules decomposed from the modules. Furthermore, there is an XP feature that students can see for each module to track their own learning progress. Moreover, the application includes an AI assistant which is a pre-programmed chatbot that can help you with any of the hundreds of courses on the website application.

YouTube



YouTube is a world-famous web-based application for uploading and sharing long and short form video content as well as for streaming. While famously being known for its entertainment purposes, sites like YouTube have been repeatedly mentioned by students in my interviews/surveys as an excellent learning tool that they use, where they can access video explanations in relatively short form and access multitudes of different content creators for explanations and learning purposes, with two of the most subscribes A-Level OCR Computer Science channels being “Craig’n’Dave” and “MrBrownCS”.Moreover, YouTube has the inbuilt features of channel subscriptions , allowing the user to be alerted when their favourite channels upload new content and the ability to save videos in either ‘Watch Later’ folders or custom folders, allowing students to track the learning videos they have watched in their folders.

Computer Science Textbooks

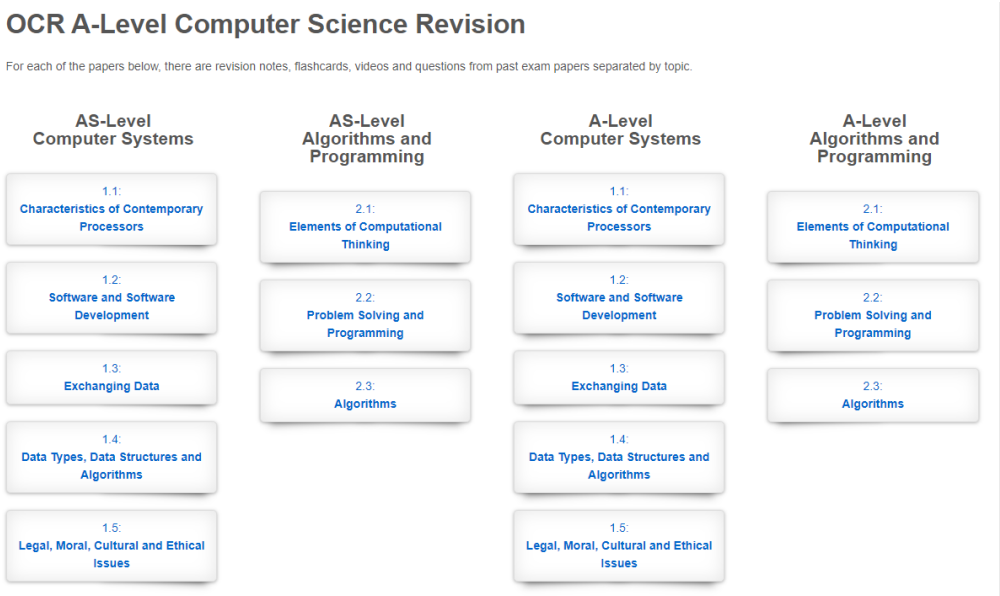


The last frequently mentioned method of learning by the students in my surveys/interviews was the use of Computer Science Textbooks, and most relevantly in this case being the OCR Computer Science A-Level Textbook. The textbook includes the full current curriculum of what the students need to learn, alongside questions relevant to each topic at the end of each chapter. The biggest advantage of this method, as previously mentioned, is access to the full curriculum, however it carries the disadvantage of making more visual-learning students that usually require

videos explaining the topics struggle, as the only option for them is to read and try to remember the concepts instead of watching videos or listening to explanatory audios.

In addition to the aforementioned methods, I have discovered at least one other method for learning and revision for Computer Science students:

Physics and Maths Tutor(for Computer Science revision)



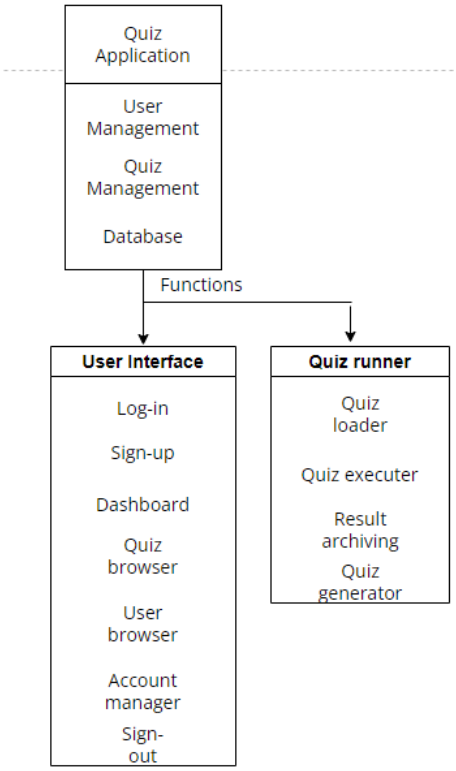
You can find all OCR Computer Science A-level Paper 1 past papers and mark schemes below:

- June 2017 MS - Paper 1 OCR Computer Science A-level
- June 2017 QP - Paper 1 OCR Computer Science A-level
- June 2018 MS - Paper 1 OCR Computer Science A-level
- June 2018 QP - Paper 1 OCR Computer Science A-level
- June 2019 MS - Paper 1 OCR Computer Science A-level
- June 2019 QP - Paper 1 OCR Computer Science A-level
- June 2022 MS - Paper 1 OCR Computer Science A-level
- June 2022 QP - Paper 1 OCR Computer Science A-level
- November 2020 MS - Paper 1 OCR Computer Science A-level
- November 2020 QP - Paper 1 OCR Computer Science A-level
- November 2021 MS - Paper 1 OCR Computer Science A-level
- November 2021 QP - Paper 1 OCR Computer Science A-level
- Specimen MS - Paper 1 OCR Computer Science A-level
- Specimen QP - Paper 1 OCR Computer Science A-level

Physics and Maths Tutor is a general learning website for a variety of STEM subjects as well as others like English. In this case we are only interested in the Computer Science learning aspect however, and this site provides access to pre-made revision for OCR A-Level Computer Science based on the current curriculum, as well as containing every accessible past paper for the subject

and multitudes of relevant questions , making it a very useful resource for revision+/learning and providing an alternative option for students that do not wish to use the book for revision.

Abstraction diagram



This will be the outline of my program structure, having three main components:

1.User Management:

Interacts with the database to store and retrieve user details, providing functions such as signing up for an account, logging in to the account and account management.

2.Test Management:

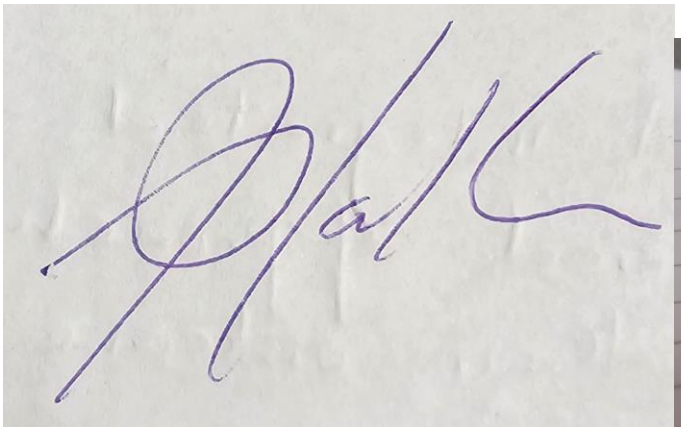
Interacts with the database to store and retrieve results, connecting with the local file system to retrieve quizzes. Includes features such as the dashboard, test browser and user browser.

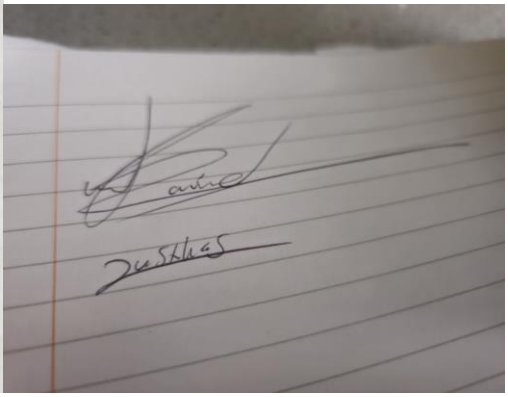
3)Database:

Stores user details and quiz results, giving access to both the user and quiz management components.

The User Interface will allow users to interact with the application. It will include functions such as signing-up and logging-in, a dashboard that will contain easy-to-understand instructions, a quiz browser, user browser and account management. The User Interface will interface with both the user and quiz management, with the quiz runner being responsible for executing tests. It will also include components such as the quiz loader, quiz executor, result saver and test generator. The quiz loader will be responsible for loading questions and metadata from files stored on the system. The quiz executor executes the tests, presenting questions to users and obtaining their responses. The results saver saves test results to the database. The quiz creator automatically generates quizzes based on pre-existing ones.

Signatures from participants in interviews + surveys





Stakeholder Requirements

Through a survey and three interviews, a set of core requirements has been developed in collaboration with Andre Wallace and other interviewees/respondents to ensure compatibility across most scenarios:

- User accounts must be able to save progress.
- Users should be able to create accounts within the program.
- Passwords should be hidden in the user interface.
- The system should allow users to change their usernames and passwords.
- Upon signing in, users should be greeted with a screen displaying details like their average daily scores and attempts over a 5-day period.
- There must be an account type that displays the data mentioned above for all users.

- Another account type should display the same information, but only for the individual user.
- The usernames must be distinctive and between 5 and 16 characters.
- Passwords must contain at least 6 characters.
- Password hashing should be included.
- The program should support running quizzes that include both multiple-choice and open-ended questions.
- It should be capable of recognizing and accounting for filler words in open-ended responses.
- The system must support image display.
- Users should be able to search for content by name and creator.
- Quizzes should be able to run locally without saving results via a file browser.

In addition to these primary requirements, the following features were suggested but are not mandatory:

- The option to use the Enter key for signing in and selecting the next entry.
- It should be adaptable to different screen sizes.

Hardware and software requirements

The program will be made using Python 3.12 and MySQL 8 hence, I will be building the necessities of the program on the specifications needed to run with memory overhead due to the size of the database when launched. Furthermore, a keyboard, mouse, and extra storage space based on the number of tests stored will be needed. To conclude, various Python libraries will be needed for the code to execute which can be stored into a Python folder or compiled in the program.

Software requirements	Reasoning
Microsoft Windows 10 operating system or newer	Operating system must be able to run MySQL 8 and Python 3
MySQL Workbench	For managing the database with a GUI
MySQL Configurator	Used to create and host the database
Python 3	Needed to run the program
Numerous Python Libraries (TBD)	Needed to run the program

Hardware requirements	Reasoning
PC with a 1GHz/4core processor or better	Minimum recommended processing power for MySQL 8
4GB of ram is recommended for a smooth experience, anything below it is discouraged.	A minimum of 4GB is recommended by Microsoft for a smoother experience

4GB of storage space (8GB and above recommended)	4GB should be enough space for the program, MySQL 8 and Python 3 install
Monitor with minimum display resolution of 1280x720 pixels (HD)	This screen size gives flexibility for the user interface layout
Mouse	Needed to select buttons and text boxes
Computer keyboard	Needed for typing word entries

Success Criteria

All success criteria will later be provided evidence of completion either via screenshots or videos

All the success criteria are listed below:

A start-up window which:

- Has a sign-in button that launches the main menu button upon successful entry.
- Have a sign-up that will refer users to the sign-in page.
- Has a username and obfuscates password upon entry (allows showing and hiding of the password).
- Database will be queried for valid logins and will go back to the sign-in window if invalid.

  A sign-up page which:
- Clearly displays a sign-up feature for users and allows users to create either a teacher or student account.
- The username is between 5 and 16 characters.
- The database does not already contain the username.
- The password contains a minimum of 6 characters.
- The password that has been entered must match with the database.
- The username and password are not the same.

A main menu which:

- Contains buttons that redirect users to a dashboard/quiz browser/account manager/log-out window, while teacher accounts will have access to a user search.

A quiz record page(browser) where:

- All the quizzes are presented in a scrollable format
- Each test should display its author, date of creation/last edit, name.

A quiz launcher which:

- Presents the name of the quiz, author name and ID code.
- Presents the number of questions in a quiz.
- Presents the time allotted per quiz for each quiz.
- Searches the database to obtain previous results.
- Shows past scores on quizzes that have been taken previously.
- A button that allows the users to begin the quizzes is clearly shown
- A button that redirects users to the quiz search engine.

An account manager page which:

- Allows for password and username change for every user.

A change username window where:

- A button showing “Edit username” is clearly displayed
- New username entries must be between 5 and 16 characters long
- Contains a password confirmation feature, that also obfuscates the password
- A separate button to confirm the change (if the user changes their mind in the last moment)
- An alert displayed to the user if the change has been unsuccessful (due to incorrect username lenght,username already in use or incorrect password)
- An alert displayed to the user if the change has been successful

A change password page which:

- A button showing “Edit password” is clearly displayed
- Has an obfuscated current and new password entry
- Users can select and deselect password obfuscation
- Passwords are a minimum of 6 characters long
- Separate change button
- The current password is checked, and the new password is tested to see if it falls within the allowed limit.
- Database updates

A user search where:

- All student accounts will be viewable from teacher accounts ONLY.
- Contains a search engine to search by username and user id.

A dashboard page where:

- Displays user activity
- A graph containing attempts over *__* days
- A graph displaying average percentages per trial over *__* days
- A button which brings up previous results.

A previous result search engine where:

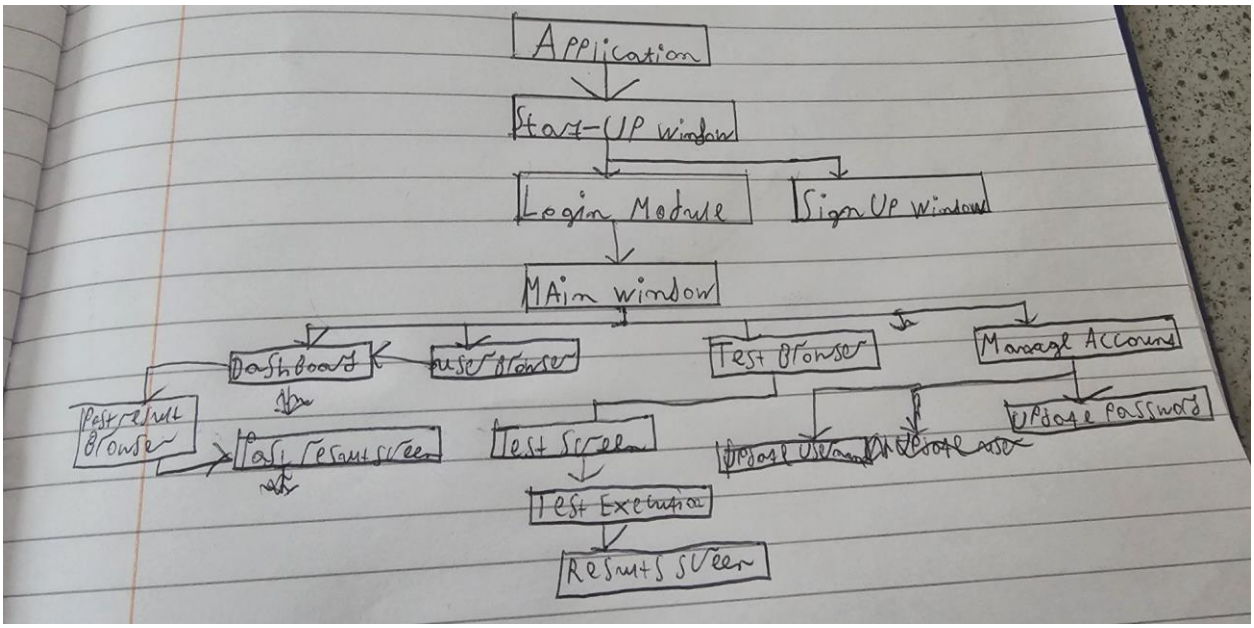
- Contains a scrollable list of all previous results
- Results can be searched via the name of the quiz or test id.
- Dropdown table to sort results via name, date and performance percentages.
- Previous result screens need to show both the number of correctly and incorrectly answered questions, as well as the total number of questions within the quiz.
- The quiz result screen should be able to redirect you back to the previous results engine.

A results page where:

- Only completed quizzes are displayed.
- A list of answered questions (correct and incorrect) is shown for each quiz.
- The success rate of the user is ranked out of the total of each quiz.
- Able to save results to the database

Design

Design Decomposition



Structure of my solution

This section will include explanations of my design decompositions.

Question application: This will include the start-up window, main window and all of their submodules.

Start-up window

Log-in feature: This will be the first page shown to the user upon launching the application. The user will have to put in a username and password which will be searched for in the database and will be unable to proceed to the main window unless they meet these requirements.

Sign-up feature: This page will be accessed through clicking on a ‘Sing-up’ button on the Log-in page. The user will have to sign up for a student or teacher account, then enter their username and password to confirm their log-in. If the username does not already exist in the database, the account will be created.

Main window

The main window will contain a menu bar with buttons to select different pages as well as a ‘sign-out’ button

(1) Dashboard: The dashboard will be one of the first pages to be shown and will display user statistics, as well as user analytics on their average attempts per question along with their success rate (%).

(2) Quiz search engine: This window will allow users to search for any specific quizzes within the quiz section, as well as select a quiz from their file explorer.

(3) User search engine: This search engine will only be available to teacher accounts, allowing teachers to scroll through any user accounts to view their performance metrics.

(4) Account management: The account management window’s only purpose will be to provide users with the option of changing their usernames and/or password. Selection of either option from the menu will refer the users to the respective page.

(5) Username management: Referral to this page will only be possible through the account management window. The functions within this window will allow the user to change their username by first asking for the new username that they wish and their current password. If the username and password match with the database query, and the new username is within 5 to 20 characters, the change will take effect in the database.

(6) Password management: Referral to this page is once again only possible through the account management window. This page will require the username and current password of the user, as well as a new password which must be at least 8 characters long. If the username and password match with the database query, and the new password is within the allowed range, the password will be augmented.

(7) Prior results search engine: Allows the user to view all previous results, based on the quiz's name and ID number, alongside sorting the list by name(alphabetically), date or scoring percentage.

(8) Prior results display: Shows the total amount of questions for the selected quiz and two lists displaying the number of correctly and incorrectly answered questions.

(9) Quiz launch screen: Shows information about the quiz such as the quiz name and ID, time allotted, exam board, topic, quiz creator, and total amount of questions. There will also be a prior score for the specific quizzes ordered by date. There is a button to start the test.

(10) Quiz Execution: This is the page where the user takes the actual quiz. At the top, it will display the question number, the time for the quiz to be completed, and an option to abandon it. The question will be shown along with a space to display images. Alongside that, there will be 4 buttons made for multiple choice questions, and at the bottom users will find a submit button along with buttons that allow them to go back a step or submit their quiz.

(11) Result screen: This page is automatically opened when every question in a quiz has been answered or when the time allotted for the quiz has run out. The page should show the achieved score, notify the user that the quiz is complete, have a list of questions answered incorrectly and their number ID, and an option to return to the dashboard. Afterwards the results will be stored in the program database.

Inputs,processes,outputs and storage

*Categories:

Inputs,processes,outputs,storage

Inputs:

All possible inputs of the program, which include:

- Signing-in, Signing-up, search,sort,previous,next,start,quit,menu,submit,back and load controls. Example:Clicking:Using a mouse to click on a button or submit information
- Sort searching by author, score, date or name. Example: A drop-down list of quizzes done by a student.

- Account class selection.Example:A radio button used for providing a choice between options
- Answer,Password,username and search engines bars.Example:The user enters the answer in the text box
- On/Off Switch for Password Obfuscation. Example: Users select whether to obfuscate passwords or not
- File Selection: The user selects a file from the file explorer to upload a quiz

Processes:

All possible processing done within the program:

- Authentication of personal details and user credentials (password and username)
- Booting up quiz files and images used during quizzes
- Saving quiz results
- Updating user credentials within the database
- Performance Metrics and time constrictions
- Input verification
- Quering database

Outputs:

All possible outputs within the program:

- Display notifications, which notify users if any invalid login attempts occur
- User interface, which displays any outputs to the user on a separate window
- Showing results, displaying user quiz performance

Storage:

All data stored in the program:

- User graphs, accounts and quiz results:
- User graphs contain information on user performance on quizzes.

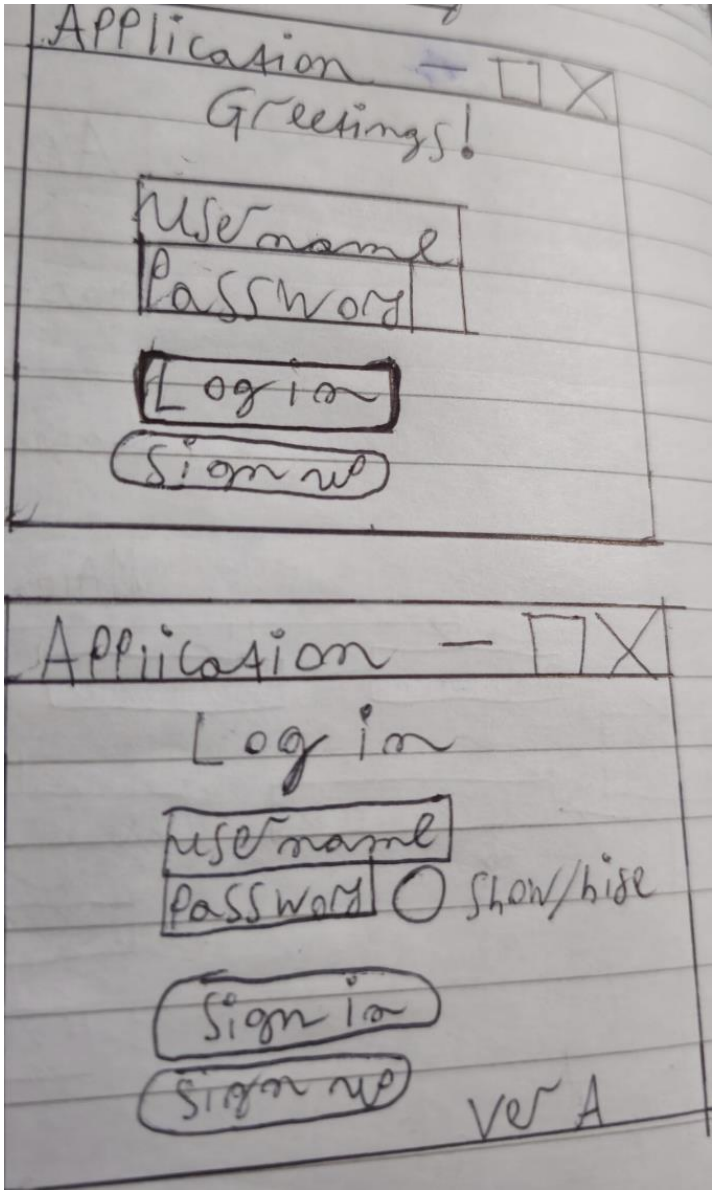
Usability features

The goal that I will try to achieve with the following drawings and diagrams is to come to a design which fits my success criteria requirements. A drawing will be made for each of my features, with the aim being to provide clarity to the user of the features that they could use before they see them in the program.

Textual entries will be displayed as rectangular boxes, while buttons will contain rounded corners for added clarity and simplicity of the representation. Any scribbles should be ignored as they are either corrections or complete removal of unnecessary information on said diagrams.

All the representations will be drawn on-hand, hence any flaws in the drawing of straight lines or shapes shall not be included in my final application.

1) Start-up window sign-in module sketch and updated start-up window sketch:

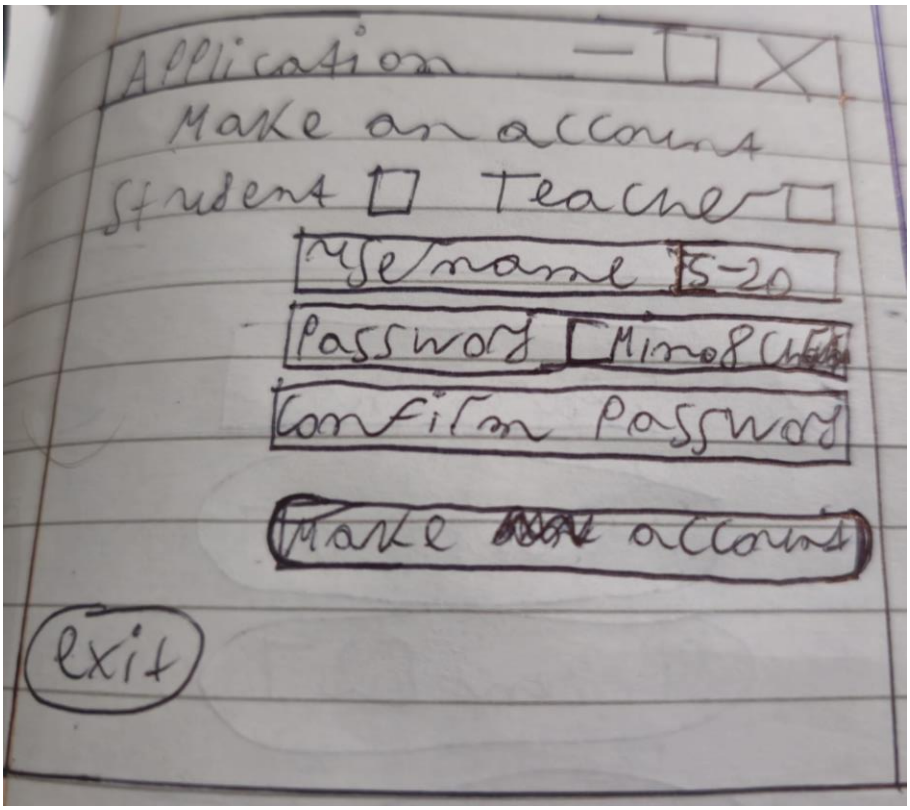


The first sketched window is the initial start-up window's sign-in module that displays once the user launches the application. After analysing the sketch with my shareholders, we collectively decided to change the 'Greetings!' into 'Log-in' for a more minimalistic and clearer message to the users of this window's function. Furthermore, I changed the shape of the obfuscation button

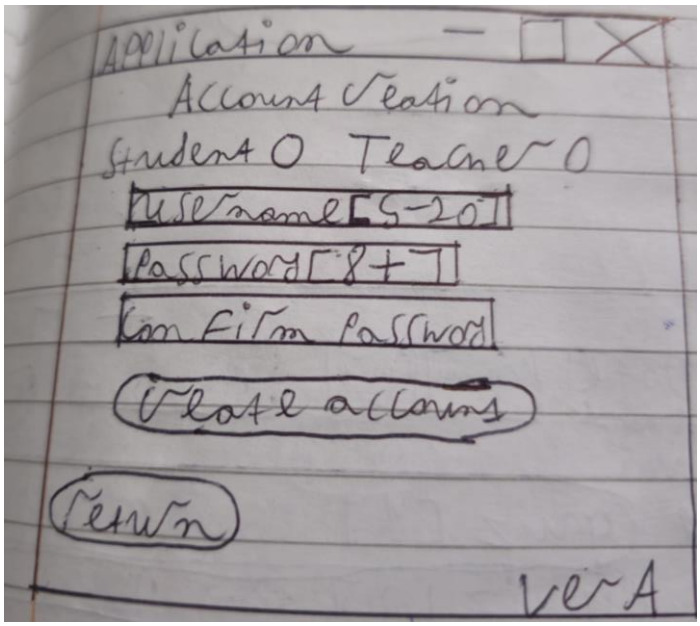
from a square that connects to the ‘password’ space to a disconnected circular shape due to the suggestion of my shareholders, and as we believe it’s more aesthetically pleasing.

The second sketched window is the final start-up window’s sign in module, where I have marked ‘Ver A’ at the bottom as to know that this is my final sketch, as this will make referencing to the text and documentation easier along the way.

2) Start-up window’s account creation module:

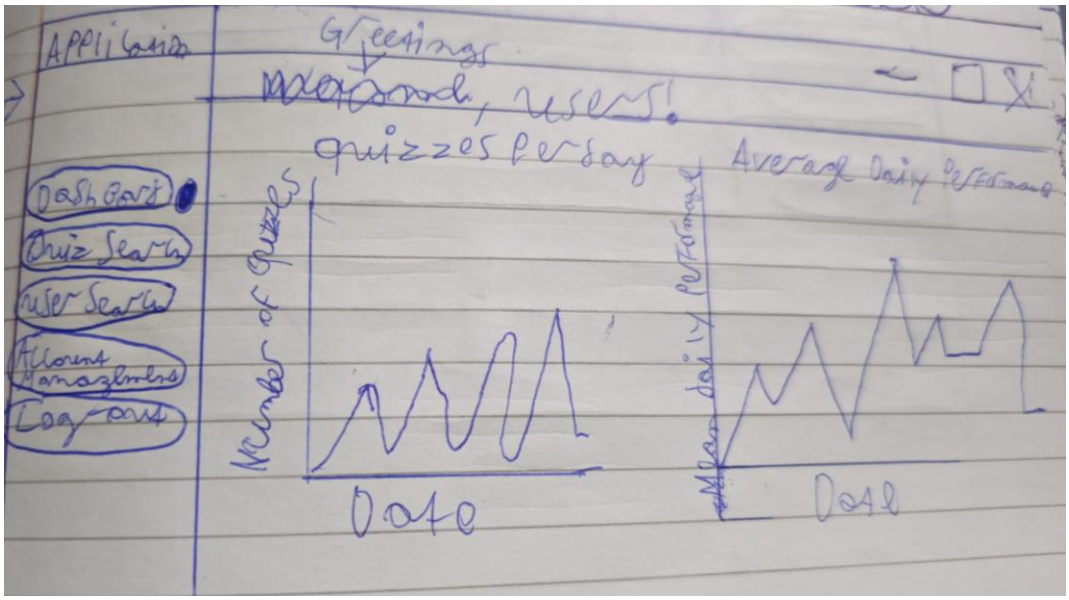


Above is my initial sketch of the Account creation window, outlining the purpose of the window and clearly displaying all necessary inputs. The stakeholders decided that ‘Make account’ isn’t professional enough for this window and I will change it to: 'Create account’.



Above is the agreed upon final sketch, where my account selection buttons for ‘Student’ and ‘Teacher’ have been changed into circles, with the only other changes being the addition of a ‘Ver A’ marking to identify this as the final version and switching ‘Make account’ with ‘Create account’

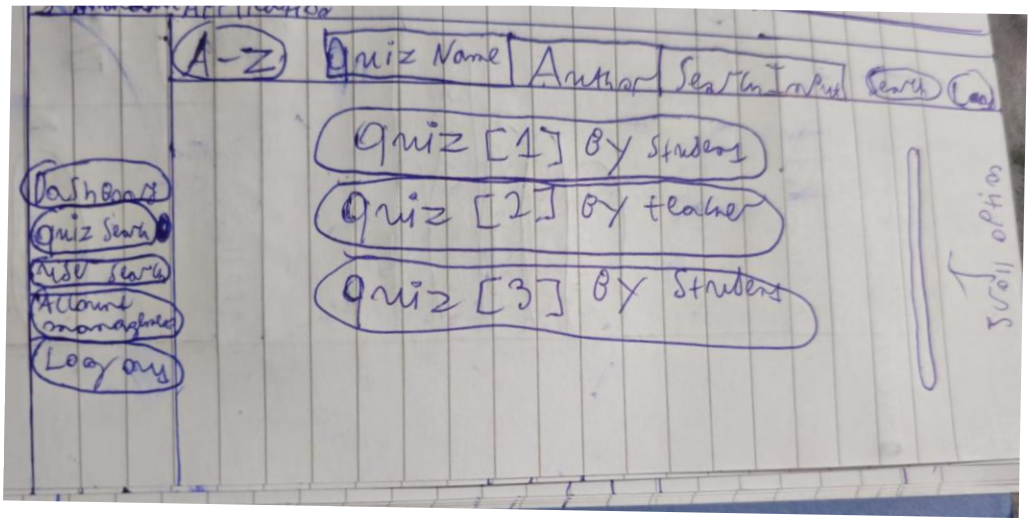
3) Dashboard module of main window



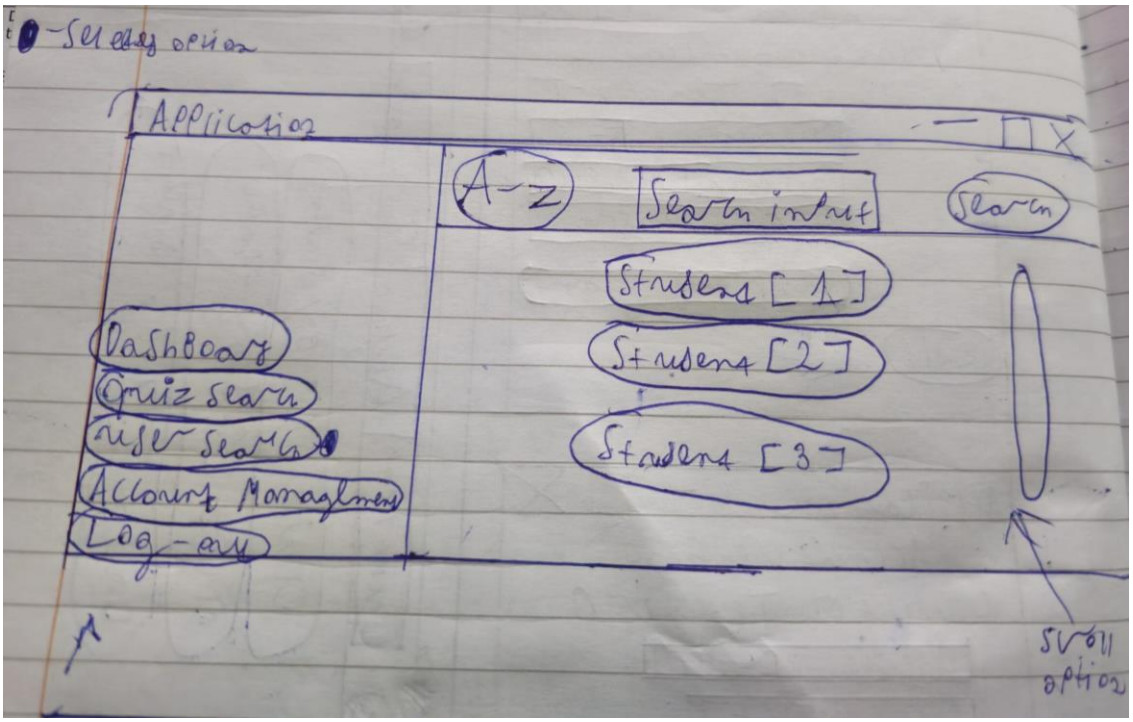
The dashboard module will be launched upon logging-in.The description on the graphs may be revised for clarity. This is because while it is the first page so welcome is appropriate, it does not explain the purpose of the graphs so this may be revised later to explain the uses further. I will discuss with Mr. Wallace for extra information on this section.

4) The quiz search module of the main window:

The quiz search will look for text files within specific directories and load the quiz names, author and ID for each button. I've included a scrollable bar on the right for users to go through the expanding lists. I have not found any necessary changes for this for now,and will wait for the stakeholders until final confirmation.

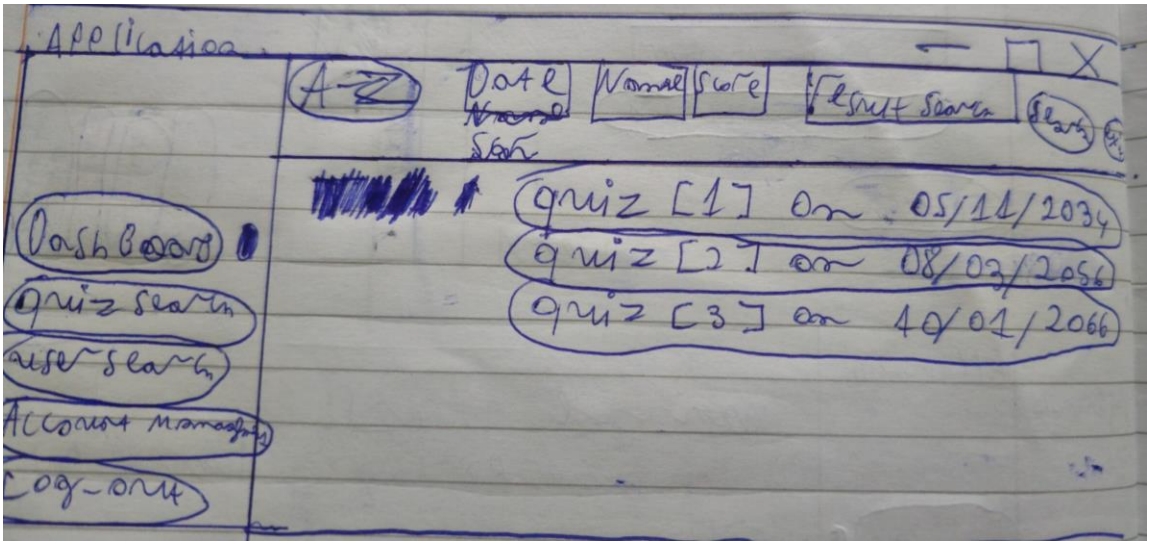


5) User search module of the main window:



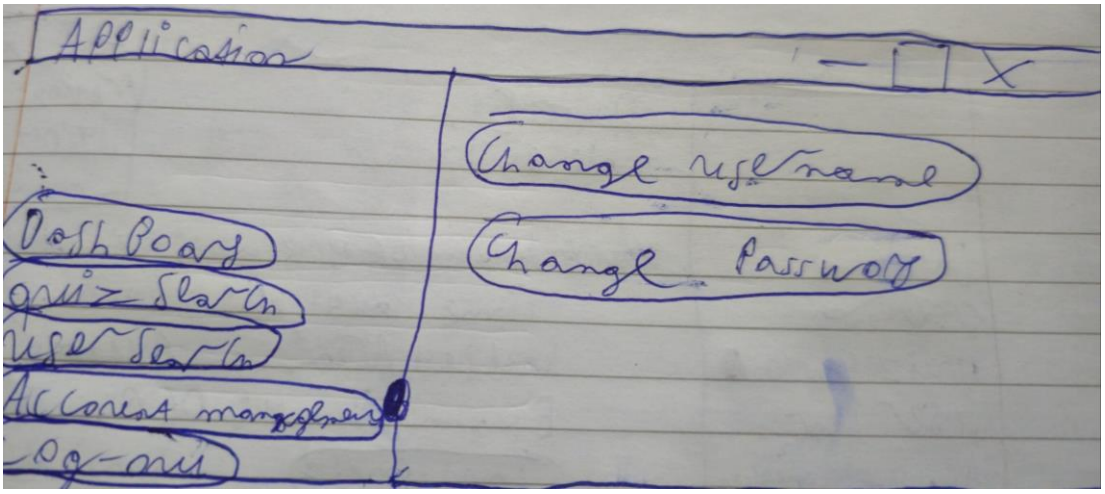
I will not make any changes at this time, and I will base the sort order chronologically by name (and their respective IDs will be shown on the side).

6) Previous results page:



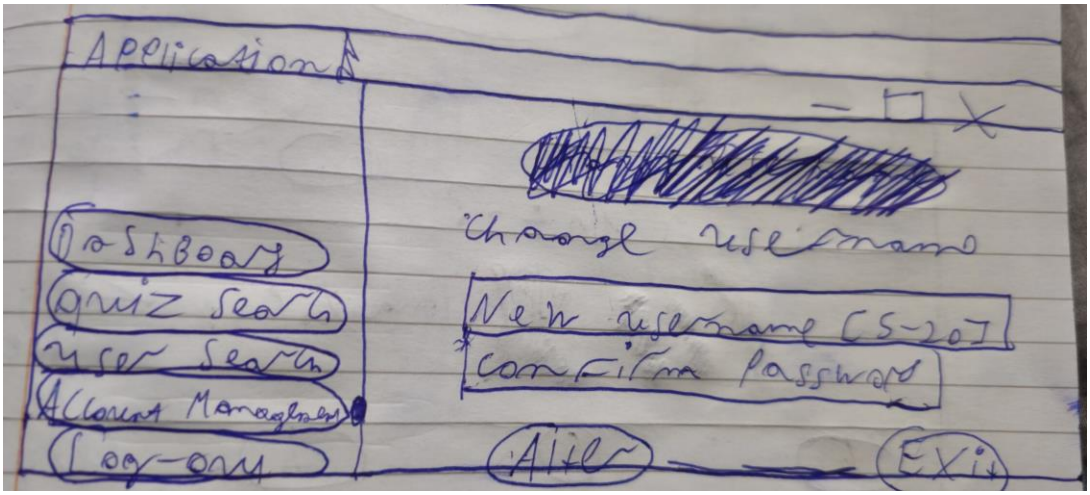
This will be accessed from the previous result tab button on the dashboard, which will show the user quiz results. Currently no changes will be made.

7) Account management window:



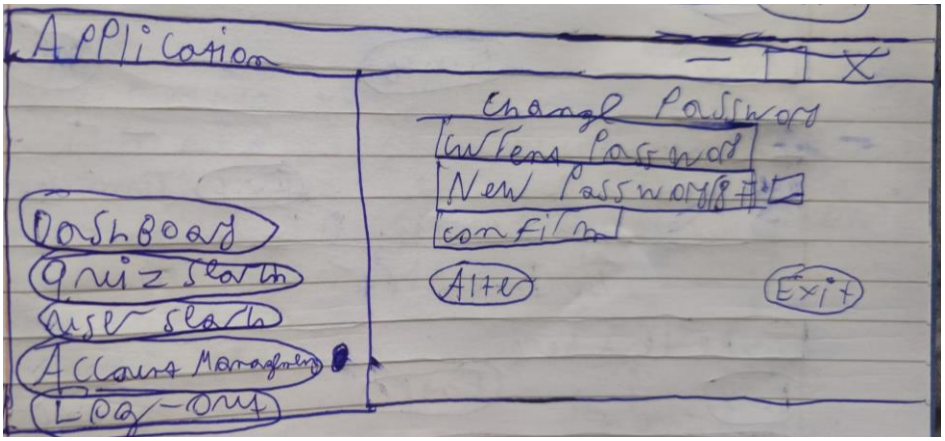
No changes will be made.

8) Change username module:



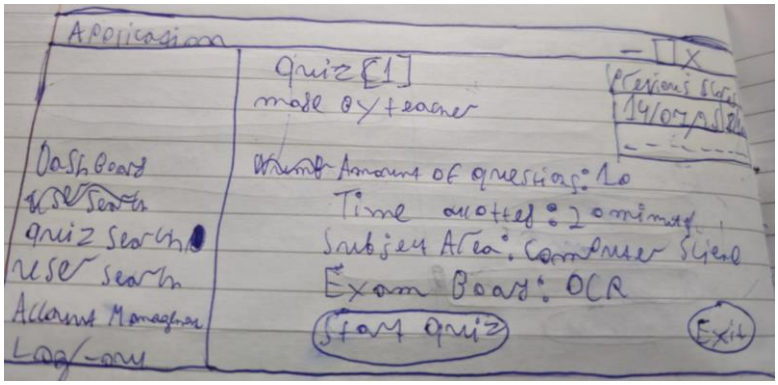
Clearly shows purpose of page and inputs, hence no changes needed.

9) Change password module:



No changes will be made for the time being, except for possibly changing the wording on password confirmation.

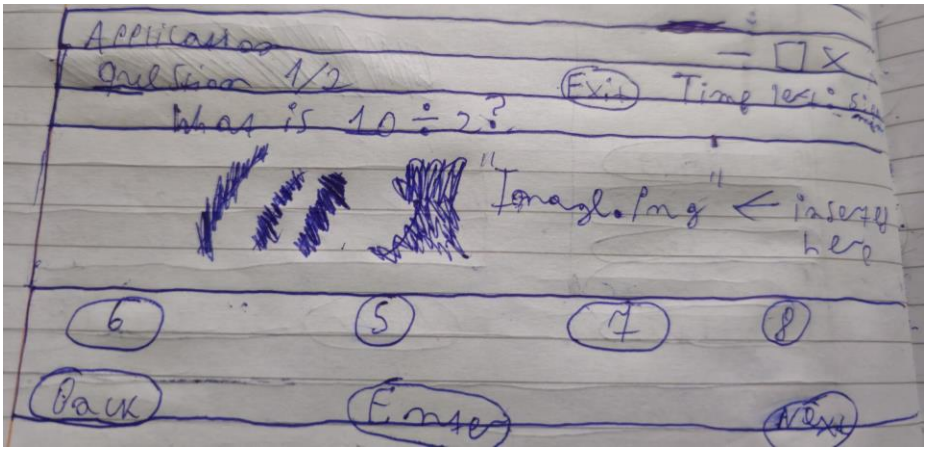
10) Quiz Launch screen:



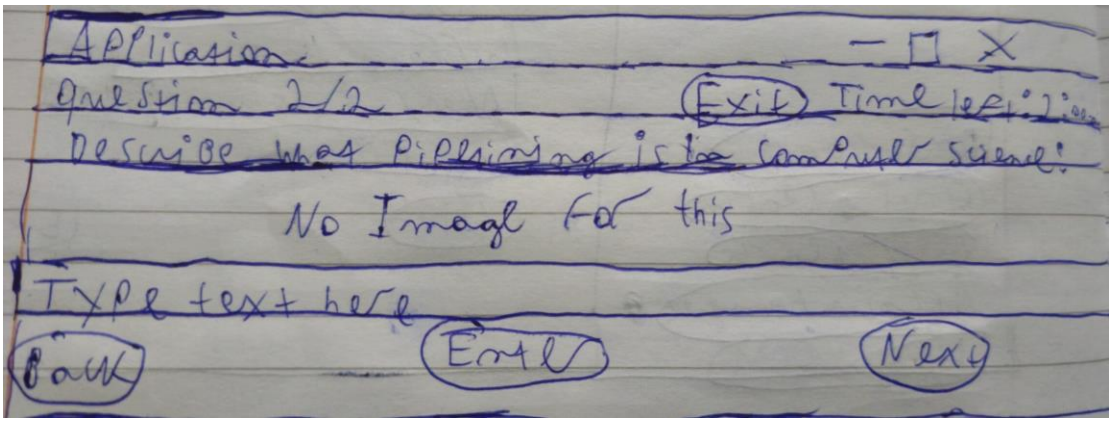
I will make specific considerations such as showing ‘missing/none’ if topic or exam boards are not listed. Additionally, I will alert users when there is no time limit for quizzes. Quiz name and ID, the author's name, number of questions, time allotted, topic and exam board will be provided. Previous results will be listed, and users will be able to scroll through the results and a button to either begin the quiz or backtrack to the dashboard.

11) Quiz execution:

I will need to consider multiple variables, such as using images in some of the quizzes, ability to answer questions which require a textual response, as well as extremely long questions.

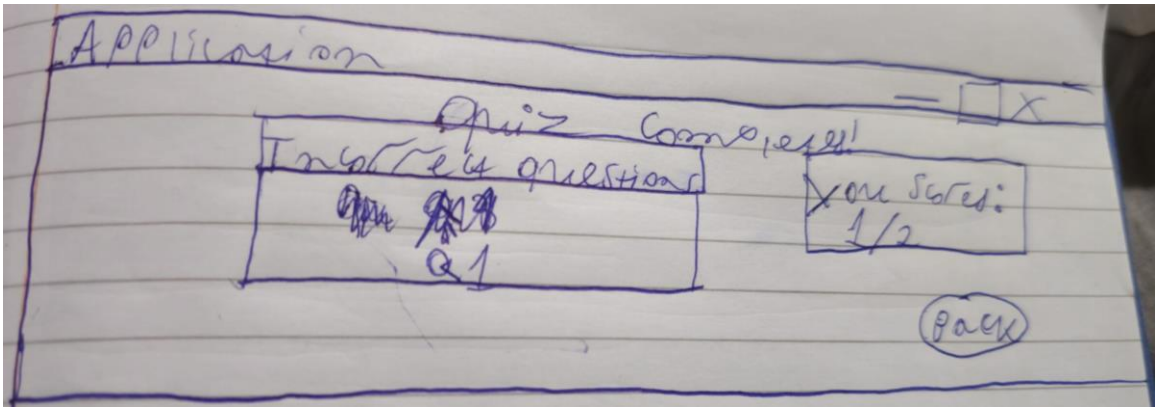


The first sketch shows a multiple-choice question with 4 button options. There will be an image above as listed above in some questions, a button to exit the quiz, a quiz timer, selection and enter buttons as well as the question numbers on the top. The stakeholders are happy with this.



The second sketch shows an example question that requires a textual response. I have taken the extra time taken to type out words into account and have extended the allotted time. If no image is provided, the middle area will be empty. Although if a path to an image is provided but the image does not load, I will separately inform the user on the middle area.

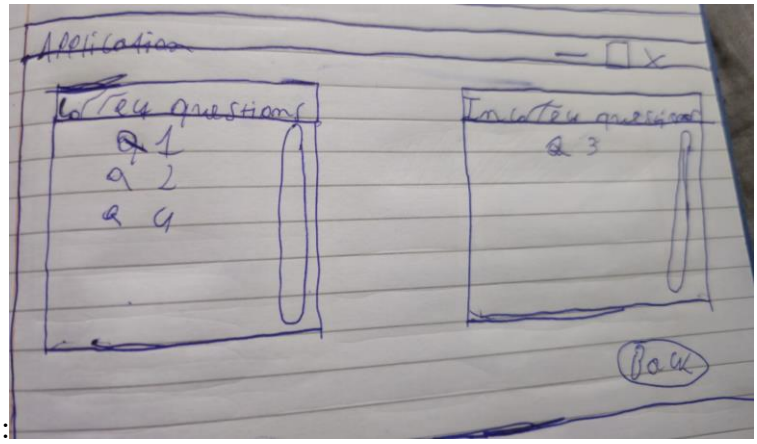
12) Results window:



The results screen will appear when a quiz has been submitted. A list will appear that shows all incorrectly answered questions, quiz results (total scored out of the available number of marks), and I will include a button to redirect back to the dashboard window.

13) Past results window

:



The past results screen has a different UI layout, as it is referencing previously completed quizzes, not ones that were just completed now. Hence, it is more sensible to include a list with correct and incorrect question numbers (and possibly IDs) as performance metrics will also be shown on the button that will be clicked to access the window.

Justification of application features

I have included clear and concise descriptions of the uses of each window, with instinctive features that make the different pages easier to use, which is why I also decided to make the instruction messages to users in the pages quite simplified and intelligible, with ones such as:'Log-in','Change Username', 'Create account' and etc., which will blatantly demonstrate the functionality of each window.

The inclusion of scrollable lists will allow the users to look for quizzes/users/authors etc., much easier and faster, while also allowing them to scroll using their mouse scroll-wheel or drag along with it for a smoother and more fluid experience.

For pages that include key information, such as passwords, I have included a button which allows the users to alternate between obfuscating (hiding the password) and not obfuscating (showing the password in this instance). This is a crucial feature of my application, as users who could have issues such as dyspraxia could take their time with typing out their password and be able to log-in even after many tries, and users who prefer to obfuscate their password will also have the opportunity to, hence accessibility will be quite egalitarian in my program.

Alongside that, I have included a mark which informs the user of which page they are on, which is represented as a coloured circle on my sketches, which will make switching between different pages much easier, making it clear what window they have selected, and hence quizzes will be accessed easier.

Moreover, as I have labelled the buttons with vocabulary which is widely accessible to the average user, and which does not require any specialist knowledge for daily use, I have improved accessibility while simplifying my functions. One example would be a clearly expressed ‘exit’, ‘next’ or ‘return’ button which could not be taken out of context by the user, which requires simply a click to access.

Furthermore, for input entries I have made use of simple descriptions such as: "Enter password” alongside with a listed minimum length of 8 characters for the password listed as “8+” and “Enter username” with a minimum and maximum length of the characters listed as “Username [5-20]”.

To conclude, I have comprehensibly labelled my analytical graphs, allowing users to easily interpret their performance metrics from previous and current quiz results, which is necessary for my program due to a possible lack of expert knowledge by the average user, while a more knowledgeable individual would have no issues with navigating the application.

Variable name	Data type	Function
Username	String	User’s name input into the field
Password	String	User’s chosen password(obfuscated)
Account_sort	String	Designated if the account is for a student or teacher
User_id	Integer	Distinct identifier for each user account

Quizzes

Variable name	Data type	Function
Quiz_id(x)	Integer	Distinct identifier for each quiz
Quiz_name(x)	String	Quiz’s name
Quiz_board(x)	String	Examination board of the quiz

Quiz_author(x)	String	Quiz's author
Quiz_theme(x)	String	The theme of the quiz (or topic)
Quiz_path	String	Path to the file of the quiz
Quiz_questions	List of dictionaries	Contains the questions and answers of the selected quiz

Quiz Results

Variable name	Data type	Function
User_id*	Integer	Distinct identifier for each quiz result
Quiz_id*	Integer	References which user the quiz is attributed to
Results_id*	Integer	References which user the result is attributed to
Quiz_name*	Integer	References which quiz name the test is attributed to
Date_attempt	Datetime	The date and time on which the result was first created in the format: YYYY-MM-DD HH:MM: SS
List_incorrect**	List of integers	Contains a list of index numbers attributed to the incorrectly answered questions
List_correct	List of integers	Contains a list of index numbers attributed to the correctly answered questions
Question_total	Integer	Contains the total number of questions

(x): If no value is provided, a value of 'none' will be used as a default value, hence an input won't be required for the program to be launched and carry out its function

* The variables have already been defined; however, they will have numerous uses throughout the program

**Are not provided or assigned by the user but are calculated at run time using the List_correct and Question_total variables.

In the next section I will explain the data structures which will be used throughout my project.

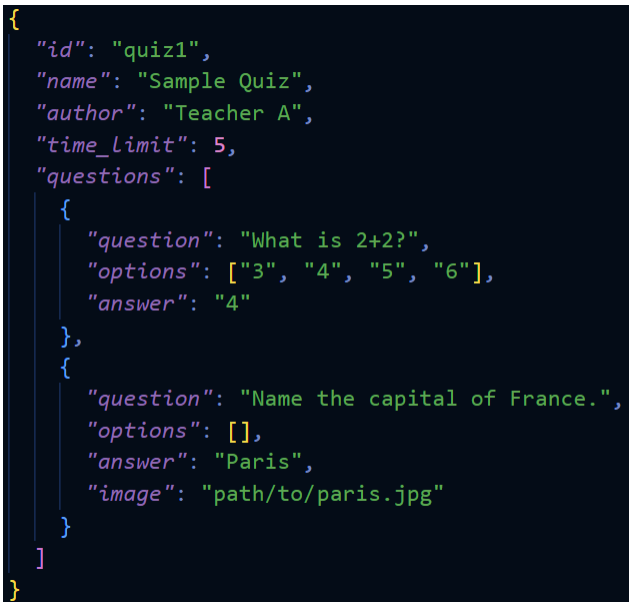
Data processing:

Data will be stored in lists of Tuples, with tuples corresponding to the attempts of the user over the last 7 days (this being due to me outlining a timeframe of 7 days in my stakeholder requirements).

Example:

[(User_id,Quiz_id,Quiz name,Date_attempt,Result_id,List_correct,List_incorrect)]

Furthermore, quiz files will be stored in a JSON format:



```json
{
  "id": "quiz1",
  "name": "Sample Quiz",
  "author": "Teacher A",
  "time_limit": 5,
  "questions": [
    {
      "question": "What is 2+2?",
      "options": ["3", "4", "5", "6"],
      "answer": "4"
    },
    {
      "question": "Name the capital of France.",
      "options": [],
      "answer": "Paris",
      "image": "path/to/paris.jpg"
    }
  ]
}
```

Above is an example of my format, which in Python will be a series of dictionaries containing lists of other dictionaries. The main dictionary will store data about the quizzes, such as author and theme(topic), with the list inside containing separate dictionaries for each question.

In the following section, I will outline my validation.

Validation	Description
Quiz path validation	Verifies existence of path to the quiz file
Password validation	Minimum length of 6 characters, the password is re-entered for confirmation, present in the field
Username validation	A username between 5-16 characters which is unique and is present in the field
Data metrics validation	Numeric data must be between a 0-100% range of any given score.
Date validation	Dates must be in a valid datetime format.
Validation of data validation	Foreign key's referential integrity is checked. If a user is deleted from the user table in the database, the results corresponding to their user id are also deleted.

Test Data for Development (Iterative)

Function	Test Data	Reasoning
Login	Valid user credentials (password and username	Ensures login logic and process works as intended

	combo), invalid password/username, empty fields	
Sign-Up	Valid student/teacher account creation with unique usernames. Invalid creation with used username (or if not within letter boundary). Invalid creation with weak password (less than 6 characters)	Verifies that sign-up logic and process works as expected and has sufficient error management.
Dashboard	User Data with results for more than 2 days but no more than 5 days.	Analytical graphs are to be plotted when data for more than 2 separate days is recorded, as otherwise the data will be insufficient, upto a limit of 5 days.
Quiz Browser	Quizzes stored in quiz folder.	Verifies that the browser allows users to examine available tests and select files, making sure that invalid files cannot be loaded onto the database.
User Browser	Multitude of users to be able to browse through in the module.	Verifies that teachers can browse through student accounts and view student results.
Change password	Possibility of valid change when username and password are correct, including repeated password confirmation, and new password being 6 or more characters. Possibility of no change when current password is incorrect (but user would already be logged in, hence this is not an issue), new password is under 6 characters.	Makes sure that the password changes function works properly while handling security and errors.
Change username	Possibility for valid change when username and password are correct. Possibility of not changing username due to wrong password/username (again, will not happen due to	Makes sure that the change username function can handle varying situations.

	the user credentials already being verified due to the user being logged in), due to new username already being in use or under/over character limit.	
Past Results Window	Containing data from past results including correct and incorrect answers.	Makes sure that the past results window shows the correct information.
Quiz Launching Window	Properties needed for the quiz such as username, topic, allocated time, author and past scores.	Makes sure that the launching window accurately conveys information to the user as per relevant for the quiz.
Quiz execution	Containing questions,images,correct answer and options to choose from.	Able to deal with absent image files.
Results Screen	Completed submission, incomplete submission due to time constraints, containing correct and incorrect answers.	Verifying that the results are correct.

The purpose of the test data is to test the modules against my success criteria in post-development testing as to ensure that every success criterion has been met. This will be done using videoclips and screenshots whenever needed and then referenced against the test data and success criteria.

Development and Testing

A thorough list of each library used in the development of this project will be provided at the evaluation stage.

The IDE that I will be using for this project will be Visual Studio Code, and for the database management part of the development I will be using the SQLite3 library for the sake of simplicity in my development, as I believe using MySQL Workbench will be much too time consuming for the task.

Module: Database connection (Purely for functionality purposes)

The database connection module will be used to link any functions that will require interaction with the SQLite database throughout the program. Down below I will illustrate the code.

The function on the most left establishes a connection to the SQLite database, and as a means of testing I have run a simple query for the program to find the version of the SQLite I am using, with the output shown down on the right.

```python
import sqlite3

try:

    # Connect to DB and create a cursor
    sqliteConnection = sqlite3.connect('sql.db')
    cursor = sqliteConnection.cursor()
    print('DB Init')

    # Write a query and execute it with cursor
    query = 'select sqlite_version();'
    cursor.execute(query)

    # Fetch and output result
    result = cursor.fetchall()
    print('SQLite Version is {}'.format(result))

    # Close the cursor
    cursor.close()

# Handle errors
except sqlite3.Error as error:
    print('Error occurred - ', error)

# Close DB Connection irrespective of success
# or failure
finally:

    if sqliteConnection:
        sqliteConnection.close()
        print('SQLite Connection closed')
```

```
DB Init
SQLite Version is [('3.45.3',)]
SQLite Connection closed
```

```python
# Import module
import sqlite3

# Connecting to sqlite
conn = sqlite3.connect('students.db')

# Creating a cursor object using the
# cursor() method
cursor = conn.cursor()

# Creating table
table ="""CREATE TABLE STUDENTS(NAME VARCHAR(255), USERID VARCHAR(255),
CLASS VARCHAR(255));"""
cursor.execute(table)

# Queries to INSERT records.
cursor.execute('''INSERT INTO STUDENTS VALUES ('Andre', '856307', 'Teacher')''')
cursor.execute('''INSERT INTO STUDENTS VALUES ('Justinas', '389653', 'Student')''')
cursor.execute('''INSERT INTO STUDENTS VALUES ('Ewan', '105733', 'Student')''')

# Display data inserted
print("Data Inserted in the table: ")
data=cursor.execute('''SELECT * FROM STUDENTS''')
for row in data:
    print(row)

# Commit your changes in the database
conn.commit()

# Closing the connection
conn.close()
```

```
Data Inserted in the table:
('Andre', '856307', 'Teacher')
('Justinas', '389653', 'Student')
('Ewan', '105733', 'Student')
```

To the left I will show an example program to illustrate the functionality of this module to use as reference throughout the development of the project with the output seen afterward

Output:

Upon further inspection, I simplified my code to the SQL connection and table creation in one module, including the creation of two separate classes (Teacher and Student, while currently removing any accounts from the database) for the future account creation and the import ctkinter and tkinter for interface customisation, as well as importing bcrypt for when I begin hashing user data for security:

```python
import sqlite3
import bcrypt
import customtkinter as ctk
from tkinter import messagebox

#CustomTkinter appearance mode
ctk.set_appearance_mode("System")
ctk.set_default_color_theme("blue")

#Creating a database connection and initializing the table
def create_database():
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    #Create table with account_type column (Student or Teacher)
    cursor.execute("""
    CREATE TABLE IF NOT EXISTS users (
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        username TEXT UNIQUE NOT NULL,
        password TEXT NOT NULL,
        account_type TEXT NOT NULL CHECK(account_type IN ('Student', 'Teacher'))
    );
    """)
    conn.commit()
    conn.close()
```

Login/Sign-Up UI + Credential Authentication

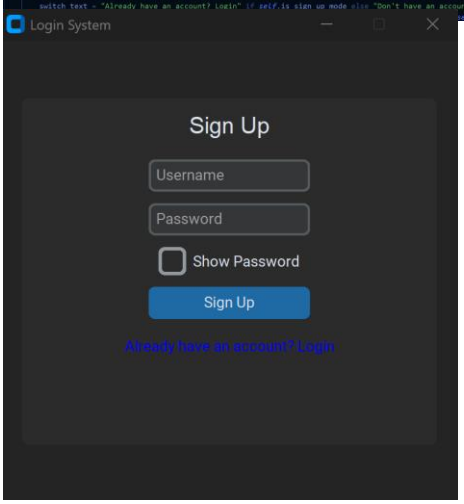
The following code defines the main app class for the Graphical User Interface and creates the user interface for the log-in and sign-up functions (logic for both has not been accounted for yet) and the code below it is used to create password obfuscation while allowing the user to always

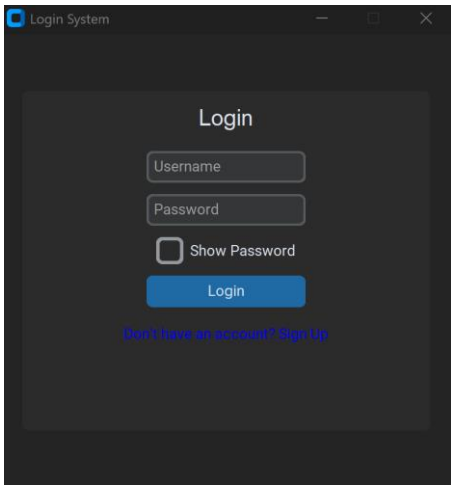
be able to switch it off , as well as switching between the log-in and sign-in UI easily laid out for users:

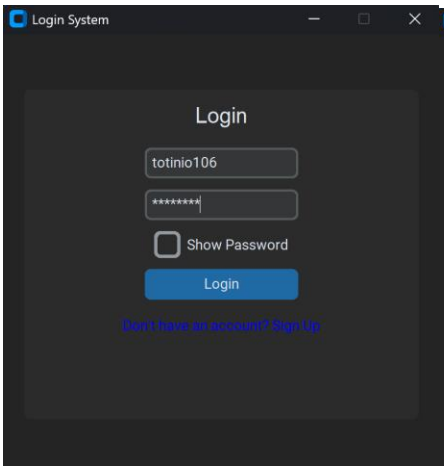
```
# Defining the main App class for GUI
class LoginApp(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Login System")
        self.geometry("400x400")
        self.resizable(False, False)
        self.is_sign_up_mode = False  # Tracks whether the user is in sign-up mode
        self.password_visible = ctk.BooleanVar(value=False)  # Controls password visibility
        self.frame = None  # Placeholder for UI frame
        self.create_ui()

    def create_ui(self):
        """Creates the login or sign-up UI depending on the mode (Sign Up or Login)"""
        if self.frame:
            self.frame.destroy()  # Clear previous UI elements

        self.frame = ctk.CTkFrame(self)
        self.frame.pack(pady=50, padx=20, fill="both", expand=True)

        title = "Sign Up" if self.is_sign_up_mode else "Login"
        ctk.CTkLabel(self.frame, text=title, font=("Arial", 20)).pack(pady=10)

        self.username_entry = ctk.CTkEntry(self.frame, placeholder_text="Username")
        self.username_entry.pack(pady=5)

        self.password_entry = ctk.CTkEntry(self.frame, placeholder_text="Password", show="*")
        self.password_entry.pack(pady=5)

        ctk.CTkCheckBox(self.frame, text="Show Password", variable=self.password_visible, command=self.toggle_password).pack(pady=5)

        action_text = "Sign Up" if self.is_sign_up_mode else "Login"
        ctk.CTkButton(self.frame, text=action_text, command=self.handle_authentication).pack(pady=5)

        switch_text = "Already have an account? Login" if self.is_sign_up_mode else "Don't have an account? Sign Up"
        ctk.CTkButton(self.frame, text=switch_text, command=self.switch_mode, fg_color="transparent", text_color="blue").pack(pady=5)
```

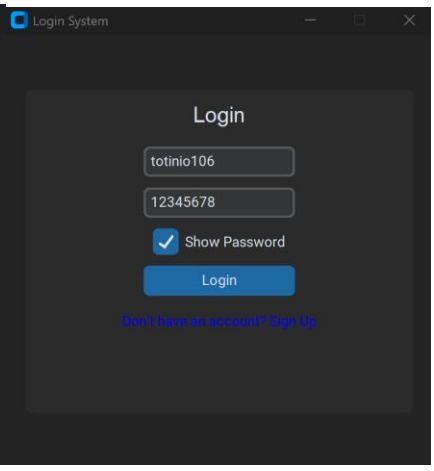
```
def toggle_password(self):
    """Toggle password visibility between hidden and visible"""
    self.password_entry.configure(show="" if self.password_visible.get() else "*")

def switch_mode(self):
    """Switch between the login and sign-up UI"""
    self.is_sign_up_mode = not self.is_sign_up_mode
    self.create_ui()
```

Output for Login/Sign-Up window (Depending on UI mode) as well as proof of password obfuscation options example(on/off):





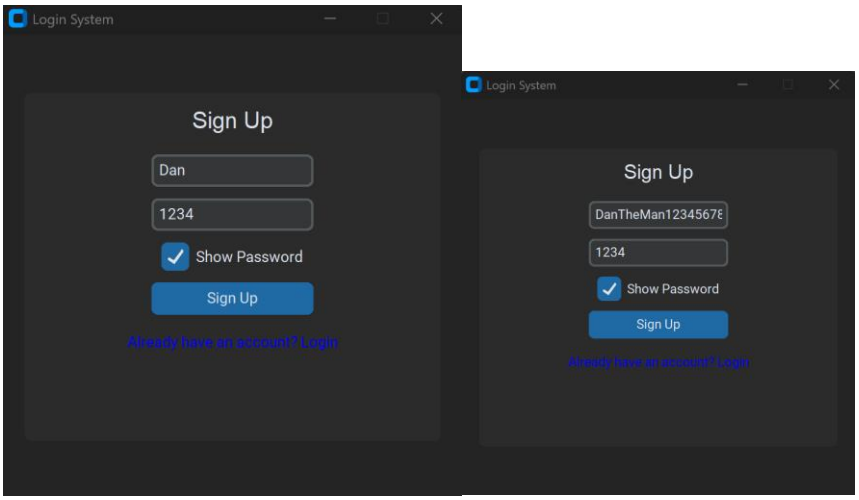




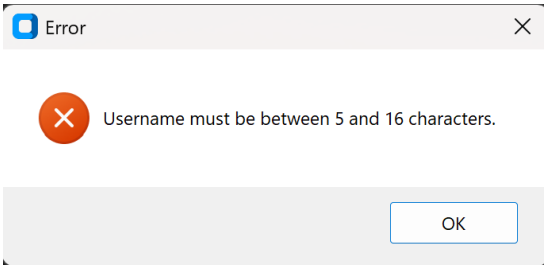
Now to fulfil a part of my success criteria I need to make sure that all usernames that are created stay between 5-16 characters long and for all password to be at least 6 characters long.Down below I will show the code which authenticates whether or not a username/password fits the criteria and its output on successful/unsuccessful completion in multiple situations:

```python
def handle_authentication(self):
    """Handles both the login and sign-up authentication logic"""
    username = self.username_entry.get().strip()
    password = self.password_entry.get().strip()

    # Validate input fields
    if not (5 <= len(username) <= 16):
        messagebox.showerror("Error", "Username must be between 5 and 16 characters.")
        return
    if len(password) < 6:
        messagebox.showerror("Error", "Password must be at least 6 characters long.")
        return

    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()
```

Examples + Output when username is too short/too long and when password is too short (Program output will default to the username error if neither condition is met):



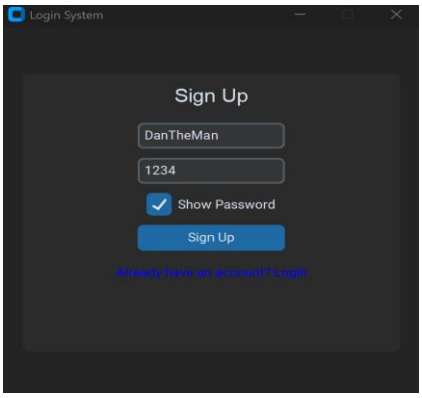
In both instances (username being too short or too long) the same error message will be outputted:

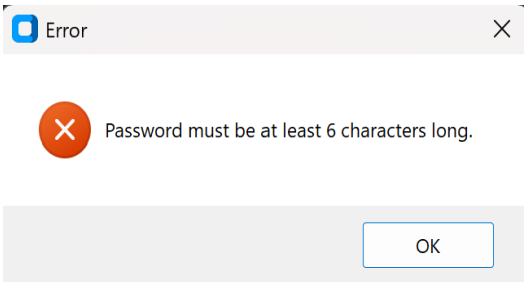


When within the allotted username length criteria, the username check will be passed, and a password check to see if it is at least 6 characters will proceed.

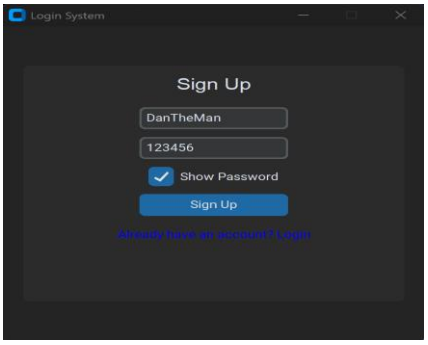
Examples and outputs:

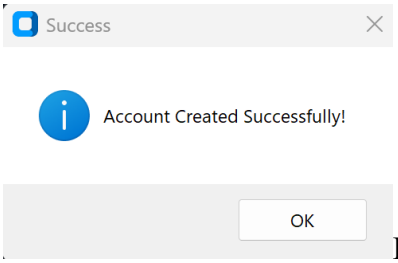
Output when Length<6 characters:





Output when Length>=6 characters:

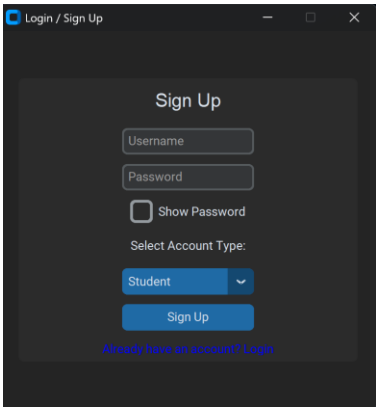


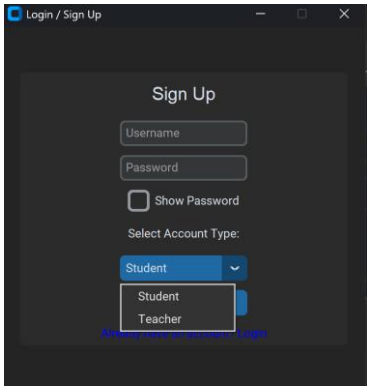


Down below I will further showcase that the database can store accounts of differing types (Student/Teacher) with no issue. As I have already shown this for the student account (all accounts created so far would have been set to student by default), I will now need to add an option during account creation for a teacher account during selection. The code that creates the selection procedure:

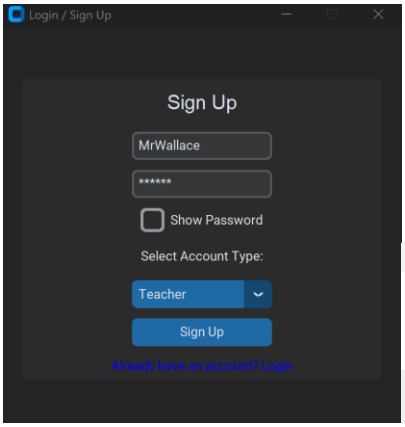
```
def create_ui(self):
    if self.frame:
        self.frame.destroy()

    self.frame = ctk.CTkFrame(self)
    self.frame.pack(pady=50, padx=20, fill="both", expand=True)

    title = "Sign Up" if self.is_sign_up_mode else "Login"
    ctk.CTkLabel(self.frame, text=title, font=("Arial", 20)).pack(pady=10)

    self.username_entry = ctk.CTkEntry(self.frame, placeholder_text="Username")
    self.username_entry.pack(pady=5)

    self.password_entry = ctk.CTkEntry(self.frame, placeholder_text="Password", show="*")
    self.password_entry.pack(pady=5)

    ctk.CTkCheckBox(self.frame, text="Show Password", variable=self.password_visible, command=self.toggle_password).pack(pady=5)

    if self.is_sign_up_mode:
        self.account_type_var = ctk.StringVar(value="Student")
        ctk.CTkLabel(self.frame, text="Select Account Type:").pack(pady=5)
        ctk.CTkOptionMenu(self.frame, variable=self.account_type_var, values=["Student", "Teacher"]).pack(pady=5)
```

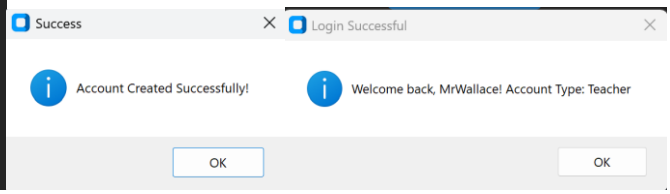
Updated Login/Sing-up window after new code as well as proof that the window can switch between account types:





Creating a new account as account type of teacher as well as authentication of creation by checking the database using an output function (shown in the “hashing function” of this project) + checking if log-in is hence successful:






```
ID | Username | Password | Account Type
----------------------------------------
5 | DanTheMan | b'$2b$12$iKPxKvkIAnqqDWCMeoF5z.IXhm85WCs2LmN4zSBZqY.cHyN4cTCSe' | Student
6 | MrWallace | b'$2b$12$IV1exlZpcDSZod.ZIYUq3uc9.mcMRmMa6iDqoXcAgJljHDCEDqob2' | Teacher
```

Seeing as the test data for this module has been met (as outlined in the Design),we can move on to the next module.

Hashing function

Hashing is one of my core success criteria in the project and vital to user security in any database, hence this is the center of my next function. Upon account creation for users their username, password and account type will be stored, but with the password being hashed and stored as the hashed password (unique random hash value will be assigned to each password upon creation). Upon logging in after account creation, if the username and the original password match, the user will successfully log-in after the hashed value is retrieved from the database, otherwise singing-in would not occur.

```
        if self.is_sign_up_mode:
            # Sign Up logic: Hash password and store in the database
            account_type = "Student"  # Default account type
            hashed_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

            try:
                cursor.execute("INSERT INTO users (username, password, account_type) VALUES (?, ?, ?)",
                               (username, hashed_password, account_type))
                conn.commit()
                messagebox.showinfo("Success", "Account Created Successfully!")
                self.switch_mode()
            except sqlite3.IntegrityError:
                messagebox.showerror("Error", "Username already exists.")
        else:
            # Login logic: Check if user exists and password matches
            cursor.execute("SELECT * FROM users WHERE username = ?", (username,))
            user = cursor.fetchone()

            if user and bcrypt.checkpw(password.encode('utf-8'), user[2].encode('utf-8')):
                messagebox.showinfo("Login Successful", f"Welcome back, {user[1]}! Account Type: {user[3]}")
            else:
                messagebox.showerror("Error", "Invalid Username or Password")

        conn.close()

# Run the app and initialize the database
if __name__ == "__main__":
    create_database()  # Ensure the database and table exist
    app = LoginApp()
    app.mainloop()
```

To show that the hashing function works, I will use the example of the account created earlier with username:”DanTheMan” with password:”123456”. I will do this by creating a table output function down below.

“Output table” function as well as the database contents (user accounts):

```
import sqlite3

def print_users():
    conn = sqlite3.connect("users.db")
    cursor = conn.cursor()

    cursor.execute("SELECT * FROM users")
    users = cursor.fetchall()

    if users:
        print("ID | Username | Password | Account Type")
        print("----------------------------------------")
        for user in users:
            print(f"{user[0]} | {user[1]} | {user[2]} | {user[3]}")
    else:
        print("No users found in the database.")

    conn.close()

if __name__ == "__main__":
    print_users()
```

```
ID | Username | Password | Account Type
----------------------------------------
5 | DanTheMan | b'$2b$12$iKPxKvkIAnqqDWCMeoF5z.IXhm85WCs2LmN4zSBZqY.cHyN4cTCSe' | Student
```

As we can see, the section for ‘password’ has had its contents hashed, hence the hashing function is working as intended upon account creation. Now onto testing if we can log-in to the account. However, upon logging in I have encountered an error that refuses to log me in successfully into the account:

```
  File "c:\Users\totin\OneDrive\Работен плот\Project\lessaicode.py", line 107, in handle_authentication
    if user and bcrypt.checkpw(password.encode('utf-8'), user[2].encode('utf-8')):
                                                         ^^^^^^^^^^^^^^
AttributeError: 'bytes' object has no attribute 'encode'. Did you mean: 'decode'?
```

Discovering that the error was caused by SQLite storing binary data as TEXT unless explicitly told not to, which is relevant since the password is stored as TEXT, retrieving it from the

database converts it the bytes hash into a string, causing an attribute error when attempting to call ".encode('utf-8')".

A reasonable fix to the attribute error caused by line 107 is to change it from:

```
"if user and bcrypt.checkpw(password.encode('utf-8'), user[2].encode('utf-8')):"
```

To:

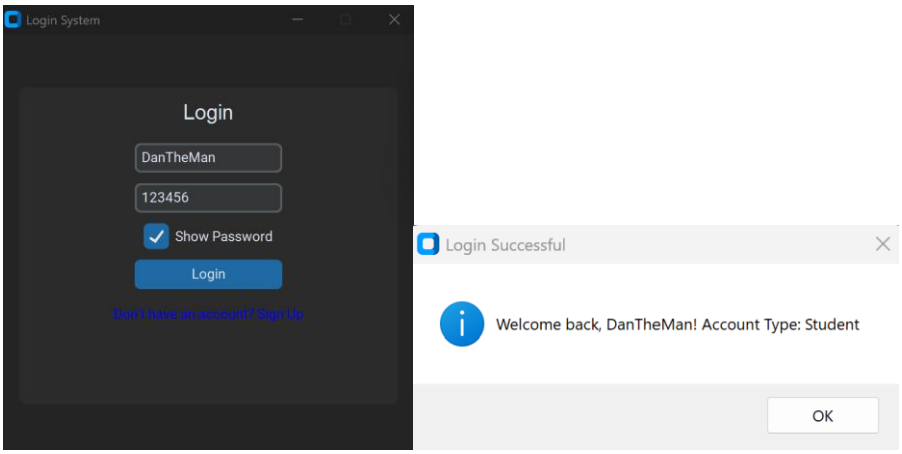
```
if user and bcrypt.checkpw(password.encode('utf-8'), user[2].encode('utf-8') if isinstance(user[2], str) else user[2]):
```

This will fix the program as we have explicitly made sure that SQLite won't store the binary for the hash data as TEXT, hence not converting it into a string and keeping the bytes hash as normal.

Updated line 107:

```
if user and bcrypt.checkpw(password.encode('utf-8'), user[2].encode('utf-8') if isinstance(user[2], str) else user[2])
    messagebox.showinfo("Login Successful", f"Welcome back, {user[1]}! Account Type: {user[3]}")
```

Now to see if our output experiences no errors:

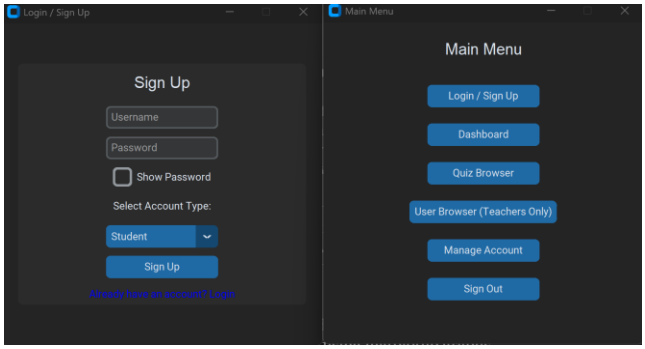


And as we can see, the code is now running as expected after the fix and we have successfully logged into the "DanTheMan" account, proving that our hashing creation, storing and retrieving is performing well. This module has passed the success criteria so far; hence we can move on.

Main Window

Prior to creating other modules, I shall create a main window from which the user will be able to select each module, with the modules being: "Login/Sign-Up", "Dashboard", "Quiz Browser", "User Browser" (with teacher access only), "Manage Account" and "Sign out". Down below I will show the code. For the time being each extra module will be "Under construction" until individually developed. Furthermore, I will provide images of how the main window looks following implementation of these features and show that the buttons work by connecting to the Signup/Login window:

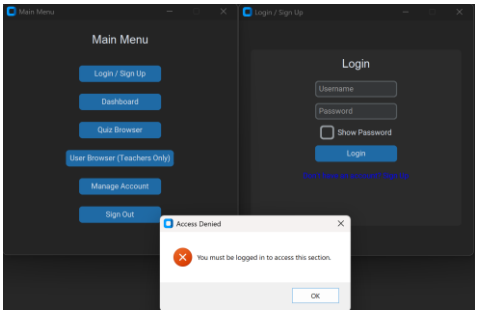
```python
# Main Window Class
class MainApp(ctk.CTk):
    def __init__(self):
        super().__init__()
        self.title("Main Menu")
        self.geometry("400x400")
        self.resizable(False, False)  # Prevent resizing

        ctk.CTkLabel(self, text="Main Menu", font=("Arial", 20)).pack(pady=20)
        ctk.CTkButton(self, text="Login / Sign Up", command=self.open_auth).pack(pady=10)
        ctk.CTkButton(self, text="Dashboard", command=self.open_dashboard).pack(pady=10)
        ctk.CTkButton(self, text="Quiz Browser", command=self.open_quiz_browser).pack(pady=10)
        ctk.CTkButton(self, text="User Browser (Teachers Only)", command=self.open_user_browser).pack(pady=10)
        ctk.CTkButton(self, text="Manage Account", command=self.open_manage_account).pack(pady=10)
        ctk.CTkButton(self, text="Sign Out", command=self.sign_out).pack(pady=10)

    def open_auth(self):
        AuthWindow(self)

    def open_dashboard(self):
        messagebox.showinfo("Dashboard", "Dashboard is under construction.")

    def open_quiz_browser(self):
        messagebox.showinfo("Quiz Browser", "Quiz Browser is under construction.")

    def open_user_browser(self):
        messagebox.showinfo("User Browser", "User Browser is under construction.")

    def open_manage_account(self):
        messagebox.showinfo("Manage Account", "Manage Account is under construction.")

    def sign_out(self):
        messagebox.showinfo("Sign Out", "You have been signed out.")
```

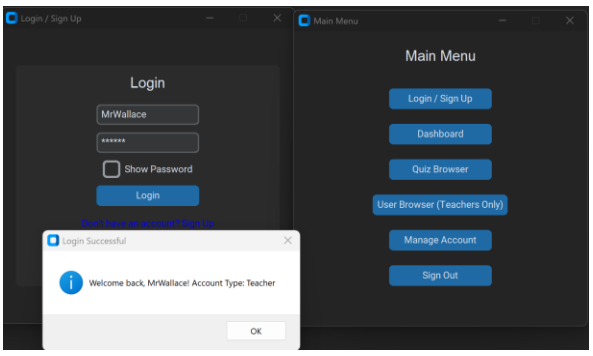


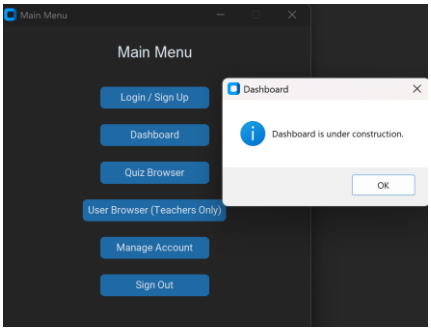
To continue the development of the main window, I need to ensure that users cannot access any of the modules before logging in apart from the “Login/Sign Up” module, and that Student accounts should not be able to access the User Browser. I can ensure that users cannot access the modules that require them to be signed in with the following code that will result in a correct output, as well showing the output of the program when attempting to use any of the modules other than “Login/Sign Up” before logging in and after logging in(I will use the Dashboard as an example for all of them) and finally checking if the User Browser can indeed only be accessed by teachers:

```python
def open_user_browser(self):
    if self.check_login_status():
        if self.current_user["account_type"] == "Teacher":
            messagebox.showinfo("User Browser", "User Browser is under construction.")
        else:
            messagebox.showerror("Access Denied", "Only accessible by teachers.")
```

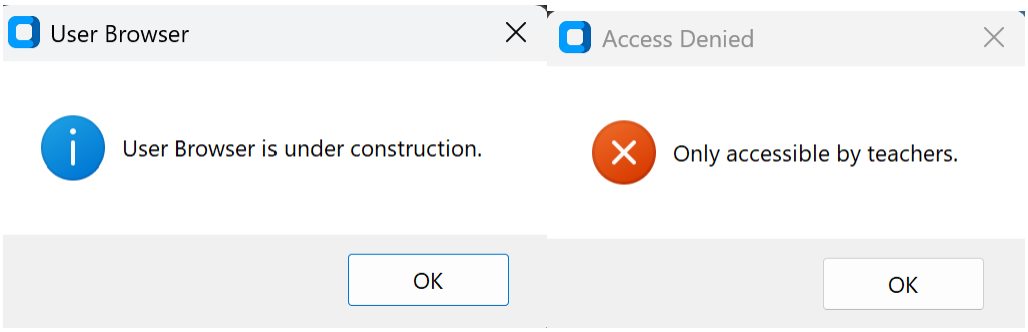
```python
def check_login_status(self):
    """Ensure user is logged in before accessing sections."""
    if not self.current_user:
        messagebox.showerror("Access Denied", "You must be logged in to access this section.")
        return False
    return True
```







Result of accessing User Browser when logged in as Teacher account type (1st slide) and when logged in with a student account (2nd slide):



As I have now made sure that all the connections and buttons work correctly so far and are in line with the test data set out to fulfil the success criteria for this module, I will begin working on the Dashboard module.

Dashboard and Past Results Module(s)

```
# Dashboard Module
def show_dashboard(main_app, name=None):
    dash_win = ctk.CTkToplevel(main_app)
    dash_win.title("Dashboard")
    dash_win.geometry("800x600")

    header = ctk.CTkFrame(dash_win)
    header.pack(fill="x", pady=5)
    if name is not None:
        header_label = ctk.CTkLabel(header, text=f"Activity report for user {name}!", font=("Segoe UI", 20))
        header_label.pack(side="left", padx=10)
        if main_app.current_user and main_app.current_user["account_type"] == "Teacher":
            back_btn = ctk.CTkButton(header, text="Back", command=lambda: (dash_win.destroy(), main_app.open_user_browser()))
            back_btn.pack(side="right", padx=10)
    else:
        if main_app.current_user:
            header_label = ctk.CTkLabel(header, text=f"Welcome, {main_app.current_user['username']}!", font=("Segoe UI", 20))
            header_label.pack(side="left", padx=10)
        else:
            header_label = ctk.CTkLabel(header, text="Welcome!", font=("Segoe UI", 20))
            header_label.pack(side="left", padx=10)
```

The code above creates a dashboard window which:

-Displays a header with a welcome/activity report message.

-Shows a back button if the user that is logged in has a teacher account and is viewing a student's dashboard, as according to the success criteria.

-Displays two graphs: A bar chart for attempts per day as well as a line chart to record the average score percentage.

Now before continuing with anything, I will establish a simulated date as a global variable (so that it doesn't change anywhere in the program unless manually changed) in the program for the purpose of future testing for the dashboard analytics and results. It will help by not needing the actual real-time data to change and remove the need for me to wait for days before being able to amass enough results to test the dashboard and results tab.This will be done by importing the datetime library and then conveyed in a (year,month,day,hour,minute,second) format:

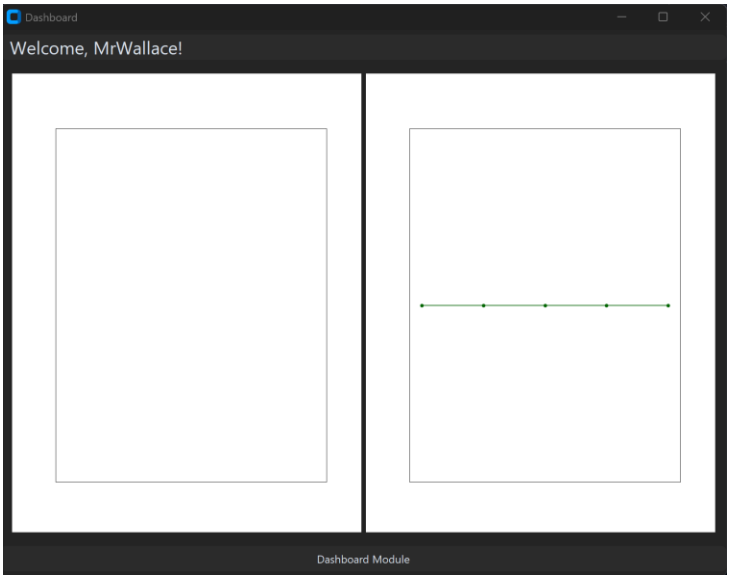
```
import datetime
```

```
# Global Simulated Date Setting
# For testing, we use a simulated date. Set it to March 10, 2025.
SIMULATED_DATE = datetime.datetime(2025, 3, 10, 12, 0, 0)
# To change the simulated date later (for new test runs) just update SIMULATED_DATE.
# Old records in the database will still have the dates recorded when they were created
```

The code below deals with the Main Graph section of the Database window (analytics) with a bar chart showing quiz attempts per day alongside a line chart showing the average correct percentage per day. The current simulated date is also included in the code as it will be assigned to all results recorder while the date stays unchanged. The bar chart and line chart will each have x and y labels signifying what the analytical graphs represent. I have also programmed a footer that can later implement new buttons/labels, with the footer.pack(fill=”x”) making the frame expand horizontally to fill the whole width of the window and pady=5 adds a vertical padding of 5 pixels above and below giving it more space from other elements to make the window appear as having more space for future buttons/labels. Crucially the code that generates the graphs in the database module will only validate if the number of days where attempts have been made is at least 2(Seen in “if len(sorted_dates) < 2), otherwise a message will be presented on screen stating that at least 2 days' worth of data are required.

```
# Main Graph Section
graph_frame = ctk.CTkFrame(dash_win, fg_color="#222222")
graph_frame.pack(fill="both", expand=True, padx=10, pady=10)

# current simulated date
current_dt = SIMULATED_DATE if SIMULATED_DATE else datetime.datetime.now()
data = get_dashboard_data(main_app.current_user["id"], current_dt)
sorted_dates = sorted(data.keys())
date_labels = [d.strftime("%m-%d") for d in sorted_dates]
attempts = [data[d]["attempts"] for d in sorted_dates]
percentages = [data[d]["avg_percentage"] for d in sorted_dates]

if len(sorted_dates) < 2:
    ctk.CTkLabel(graph_frame, text="Not enough data to display charts (min 2 days required).", font=("Segoe UI", 16)).pack(pady=20)
else:
    # Bar Chart: Attempts per Day
    fig1 = Figure(figsize=(4, 3), dpi=80)
    ax1 = fig1.add_subplot(111)
    ax1.bar(date_labels, attempts, color='royalblue')
    ax1.set_title("Attempts per Day", fontname="Segoe UI", fontsize=14, color="white")
    ax1.set_xlabel("Date", fontname="Segoe UI", fontsize=12, color="white")
    ax1.set_ylabel("Attempts", fontname="Segoe UI", fontsize=12, color="white")
    ax1.tick_params(axis='x', colors='white')
    ax1.tick_params(axis='y', colors='white')
    canvas1 = FigureCanvasTkAgg(fig1, master=graph_frame)
    canvas1.draw()
    canvas1.get_tk_widget().pack(side="left", fill="both", expand=True, padx=5)

    # Line Chart: Average % Correct per Day
    fig2 = Figure(figsize=(4, 3), dpi=80)
    ax2 = fig2.add_subplot(111)
    ax2.plot(date_labels, percentages, marker='o', color='darkgreen')
    ax2.set_title("Average Score (%) per Day", fontname="Segoe UI", fontsize=14, color="white")
    ax2.set_xlabel("Date", fontname="Segoe UI", fontsize=12, color="white")
    ax2.set_ylabel("Avg %", fontname="Segoe UI", fontsize=12, color="white")
    ax2.tick_params(axis='x', colors='white')
    ax2.tick_params(axis='y', colors='white')
    canvas2 = FigureCanvasTkAgg(fig2, master=graph_frame)
    canvas2.draw()
    canvas2.get_tk_widget().pack(side="right", fill="both", expand=True, padx=5)

footer = ctk.CTkFrame(dash_win)
footer.pack(fill="x", pady=5)
ctk.CTkLabel(footer, text="Dashboard Module", font=("Segoe UI", 12)).pack()
```

How the dashboard looks like without any labels or data added to it yet (labels seem to not yet be functional or are drowned out in the white background, will fix later on)(Signed in as teacher account “MrWallace”):



Importantly before continuing, I will need to update my previous code by creating a second database that will hold the results of users after a test has been taken.

```
# Creating results table (not dropping this table on startup)
cursor.execute("""
CREATE TABLE IF NOT EXISTS results (
    id INTEGER PRIMARY KEY AUTOINCREMENT,
    user_id INTEGER NOT NULL,
    attempt_date DATETIME NOT NULL,
    correct_list TEXT NOT NULL,
    total_questions INTEGER NOT NULL,
    FOREIGN KEY(user_id) REFERENCES users(id) ON DELETE CASCADE
);
""")
conn.commit()
conn.close()
```

The code above does the following:

(Appendix: “NOT NULL” will require each column that includes it to have data stored in it)

-Makes unique IDs for results as a primary key and will autoincrement for each new record that is added, as well as storing attempt_date as the date and time in which a quiz was completed, correct_list will store string representations of lists and total_questions will store the amount of questions in a quiz as an integer.

-Define “user_id” to store an integer in reference to a user, with the NOT NULL constraint making sure that every record is attached to a user via the “user_id” integer(s).After this we need to ensure that the “user_id” references the “id” in the “users” table that I created earlier in my development. The ON DELETE CASCADE clause will ensure that if a user in the users table is deleted, all corresponding records to the user are also deleted in the records table as to not hoard unnecessary data in the database(s).

Now that this has been done, I can move onto defining a function for “attempts in the last 5 days”, as the dashboard will show data from the last 5 days with a minimum of 2 days. The comments in the code should be sufficient for an explanation of what it does.

```
def last_five_days_attempts(user_id, current_datetime=None):
    """
    Retrieve results from the last 5 days for the given user.
    If current_datetime is provided, it will be used; otherwise SIMULATED_DATE will be used (if set) or the actual current datetime.
    """
    now = current_datetime if current_datetime else (SIMULATED_DATE if SIMULATED_DATE else datetime.datetime.now())
    conn = get_db_connection()
    if conn is None:
        return []
    cursor = conn.cursor()
    query = """
    SELECT id, user_id, attempt_date, correct_list, total_questions
    FROM results
    WHERE user_id = ?
      AND date(attempt_date) >= date(?, '-5 days')
    ORDER BY attempt_date DESC;
    """
    cursor.execute(query, (user_id, now.strftime("%Y-%m-%d %H:%M:%S")))
    rows = cursor.fetchall()
    conn.close()
    processed = []
    for row in rows:
        try:
            attempt_dt = datetime.datetime.strptime(row[2], "%Y-%m-%d %H:%M:%S")
        except ValueError:
            continue
        processed.append((row[0], row[1], attempt_dt, row[3], row[4]))
    return processed
```

Furthermore, I need to create a function to retrieve the dashboard data in the past 5 days in the medium of a dictionary which will store all necessary information such as date, attempts, average percentage etc. But to even do that I will also need to have a function that stores quiz results in the results database automatically upon a user getting a result after completing a quiz. This is the code for the former below:

```
def get_dashboard_data(user_id, current_datetime=None):
    """
    Aggregates the last 5 days of results for the users.
    Returns a dictionary: { date: {"attempts": int, "avg_percentage": float}, ... }
    """
    results = last_five_days_attempts(user_id, current_datetime)
    if not results:
        today = (current_datetime.date() if current_datetime else datetime.datetime.today().date())
        return {
            today - datetime.timedelta(days=i): {"attempts": 0, "avg_percentage": 0}
            for i in range(5)
        }
    attempts_counter = Counter(r[2].date() for r in results)
    totals_per_day = {}
    for (rid, uid, attempt_dt, correct_list, total_q) in results:
        day = attempt_dt.date()
        try:
            correct = len(eval(correct_list))
        except Exception:
            correct = 0
        if day not in totals_per_day:
            totals_per_day[day] = {"correct": 0, "total": 0}
        totals_per_day[day]["correct"] += correct
        totals_per_day[day]["total"] += total_q
    dashboard_data = {}
    for day, vals in totals_per_day.items():
        avg = (vals["correct"] / vals["total"] * 100) if vals["total"] > 0 else 0
        dashboard_data[day] = {"attempts": attempts_counter[day], "avg_percentage": avg}
    return dashboard_data
```

Retrieving results for the last 5 days:

```
for (rid, uid, attempt_dt, correct_list, total_q) in results:
    day = attempt_dt.date()
    try:
        correct = len(eval(correct_list))
    except Exception:
        correct = 0
    if day not in totals_per_day:
        totals_per_day[day] = {"correct": 0, "total": 0}
    totals_per_day[day]["correct"] += correct
    totals_per_day[day]["total"] += total_q
dashboard_data = {}
```

The first for loop will iterate through each record received in the past 5 days to sum the correct answers and total questions per day. Each record in results should be a tuple containing: ID, user ID, datetime of the attempt, a string that represents the list of correct answers, and total number of questions in the test (More parameters may be added in the future).

The second for loop will then calculate the average percentage for each day and pair it with the number of attempts for that quiz.

```
for day, vals in totals_per_day.items():
    avg = (vals["correct"] / vals["total"] * 100) if vals["total"] > 0 else 0
    dashboard_data[day] = {"attempts": attempts_counter[day], "avg_percentage": avg}
return dashboard_data
```

How it will handle no results:

If there are no results returned (if the user has no quiz data), the function will create a default dictionary.
It will either use the provided “current_datetime” or the actual current date to generate five dates (the current day and the previous 4 days), each with 0 attempts and 0 average percentage.
And now to follow through with the latter function:

```
def record_quiz_result(user_id, correct_list, total_questions):
    conn = get_db_connection()
    if conn is None:
        messagebox.showerror("Error", "Unable to connect to database.")
        return False
    cursor = conn.cursor()
    now = SIMULATED_DATE if SIMULATED_DATE else datetime.datetime.now()
    attempt_date = now.strftime("%Y-%m-%d %H:%M:%S")
    cursor.execute("""
    INSERT INTO results (user_id, attempt_date, correct_list, total_questions)
    VALUES (?, ?, ?, ?)
    """, (user_id, attempt_date, correct_list, total_questions))
    conn.commit()
    conn.close()
    return True
```

The function records the quiz results as tuples directly into the database, showing an error if a connection to the database cannot be established. The date used in storing the results can be interchangeable between the simulated date and the real-time date.

Now to test whether the quiz results can be recorded and retrieved as expected by the program, I will need to conduct a simulation of quizzes being completed and recorded and then check if it has been stored correctly by creating a function which does so.For the simulated quiz result to then be verified I will also need to create two more functions, one which will retrieve all results

and another which will show the results. While technically these two functions would be a part of the “Past Results Browser” module, I cannot continue developing the dashboard without them, hence I will create them now.

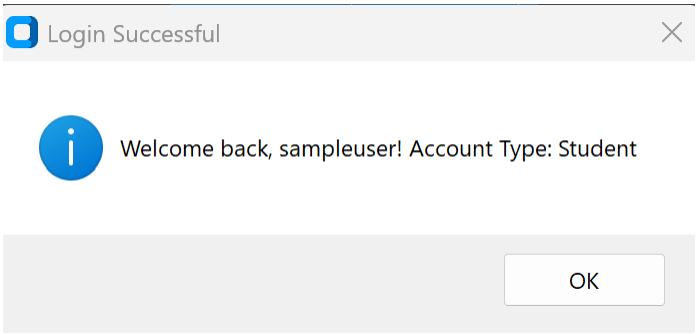
Generating the quiz simulation runs:

```
def simulate_quiz_run(self):
    if not self.check_login_status():
        return
    import random
    total_questions = random.randint(3, 10)
    correct_count = random.randint(0, total_questions)
    correct_list = "[" + ",".join(str(i) for i in range(correct_count)) + "]"
    if record_quiz_result(self.current_user["id"], correct_list, total_questions):
        messagebox.showinfo("Test Recorded", f"Recorded: {correct_count} correct out of {total_questions}")
    else:
        messagebox.showerror("Error", "Failed to record test result.")
```

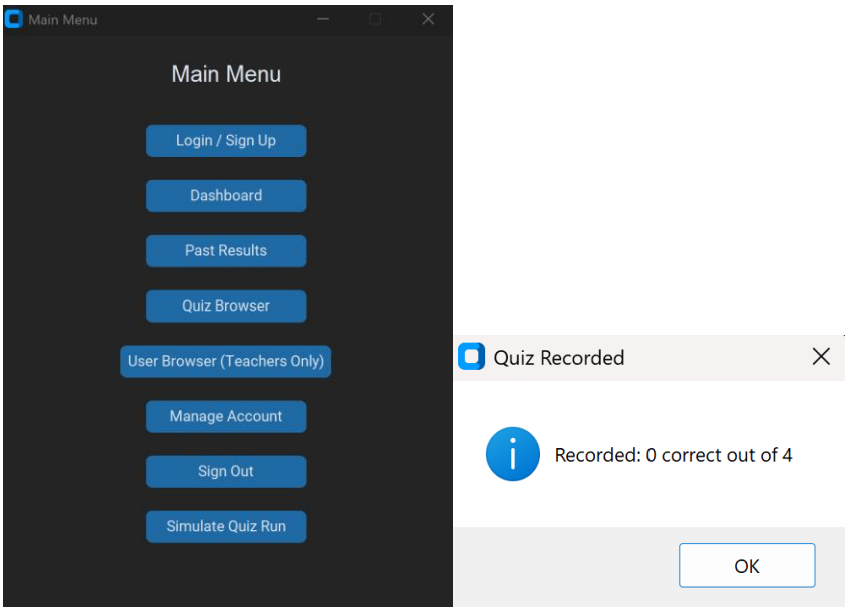
The function above will operate by allowing users which are logged-in (if entry into this module is attempted by a non-logged in account, they will receive an error message stating that they need to be logged in and be the function will be exited immediately, as per default for each module except for logging-in/singing-up) to generate test runs containing a total amount of questions which are randomised(between 3 and 10), as well as a randomised amount of correct answers (between 0 and the total amount of questions generated in that instance).Afterwards a correct answer list is created which records the total number of correct answers, and the “record_quiz_result” function is called with its three parameters (user_id,correct_list,total questions) and the function will attempt to store the result in the database. If it was recorded successfully, the user will see a message box displaying how many questions were answered correctly out of the total amount of questions. If the result fails to be recorded, the user will be notified of this with a message.

Now to test if the quiz simulation run works, I have created an account named “sampleuser” to test if the quiz results are executed and stored properly.

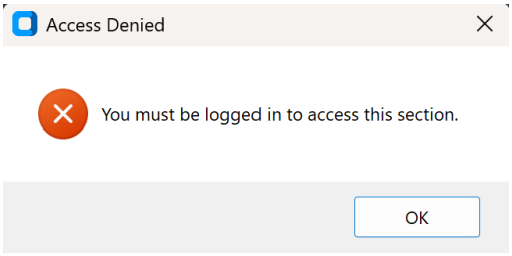
Output message showing that “sampleuser” has logged-in:



How the main menu looks like after introducing the “Simulate Quiz Run” function, and the output message after simulating a quiz run showing that a result has been generated successfully:



Now to test if we receive an error message when not logged-in and trying to simulate a quiz run:



As the simulate quiz run functions looks and works as expected so far, I will move onto creating the result retrieving function:

```python
def get_all_results(user_id):
    conn = get_db_connection()
    if conn is None:
        return []
    cursor = conn.cursor()
    query = """
    SELECT id, user_id, attempt_date, correct_list, total_questions
    FROM results
    WHERE user_id = ?
    ORDER BY attempt_date DESC;
    """
    cursor.execute(query, (user_id,))
    rows = cursor.fetchall()
    conn.close()
    processed = []
    for row in rows:
        try:
            attempt_dt = datetime.datetime.strptime(row[2], "%Y-%m-%d %H:%M:%S")
        except ValueError:
            continue
        processed.append((row[0], row[1], attempt_dt, row[3], row[4]))
    return processed
```

The function will attempt to establish a connection to the database from whenceforth if successful it will create an SQL query that selects the columns needed to retrieve results needed, with the attempt_date in descending order to retrieve the most recent results first. After closing the connection to the database, the for loop will iterate through each row given by the query, with each valid row being appended to a list in tuple format and will then return the processed list of results (any invalid conversion due to formatting issues will automatically be skipped).Now onto the past result showing function:

```
def show_past_results(main_app):
    past_win = ctk.CTkToplevel(main_app)
    past_win.title("Past Results")
    past_win.geometry("600x400")

    header = ctk.CTkFrame(past_win)
    header.pack(fill="x", pady=5)
    ctk.CTkLabel(header, text=f"Past Results for {main_app.current_user['username']}", font=("Segoe UI", 18)).pack(side="left", padx=10)

    scroll_frame = ctk.CTkScrollableFrame(past_win, width=500, height=250)
    scroll_frame.pack(padx=10, pady=10, fill="both", expand=True)

    results = get_all_results(main_app.current_user["id"])
    for row in results:
        attempt_dt = row[2]
        try:
            correct = len(eval(row[3]))
        except Exception:
            correct = 0
        total = row[4]
        percentage = (correct / total * 100) if total > 0 else 0
        date_str = attempt_dt.strftime("%Y-%m-%d %H:%M:%S")
        text = f"Date: {date_str} | Score: {percentage:.0f}% ({correct}/{total})"
        ctk.CTkLabel(scroll_frame, text=text, font=("Segoe UI", 14)).pack(pady=5)

    footer = ctk.CTkFrame(past_win)
    footer.pack(fill="x", pady=10)
    ctk.CTkButton(footer, text="Close", command=past_win.destroy).pack(fill="x")
```

I have created a new child window of the main application named “Past results” to host this function, with a header at the top of the window with a message to the user accessing it currently that contains their username, alongside a scrollable frame for displaying results if the recorded results are too many to initially show, allowing the user to scroll through them. Now to fetch each result the function will call upon the “get_all_results” function with the current user’s ID to retrieve all quiz attempt records, afterwards iterating over each result:

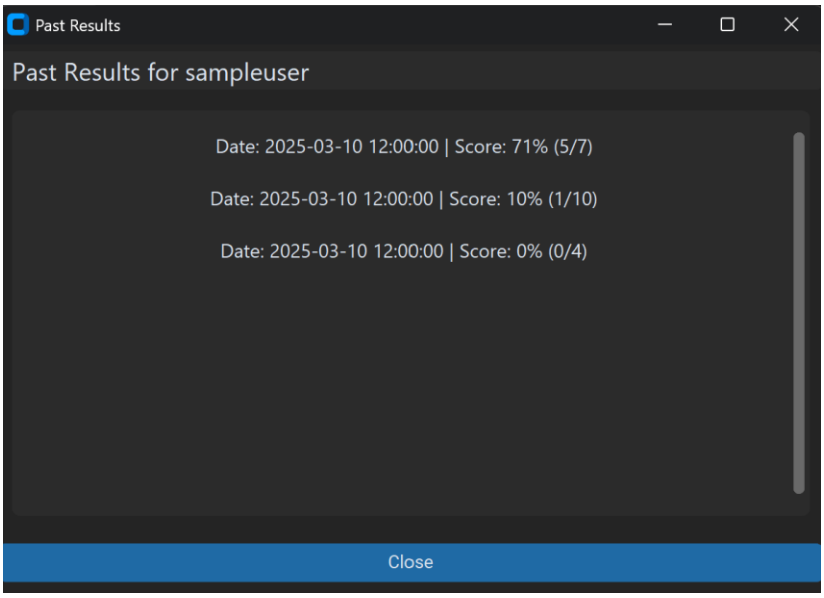
-Extracts the attempt_date and converts it into a readable string.

-Uses eval on the already stored correct_list string to convert it back to a Python list, afterwards using len to count the number of correct answers (Except if there is an error during evaluation, where it sets the correct count to 0).

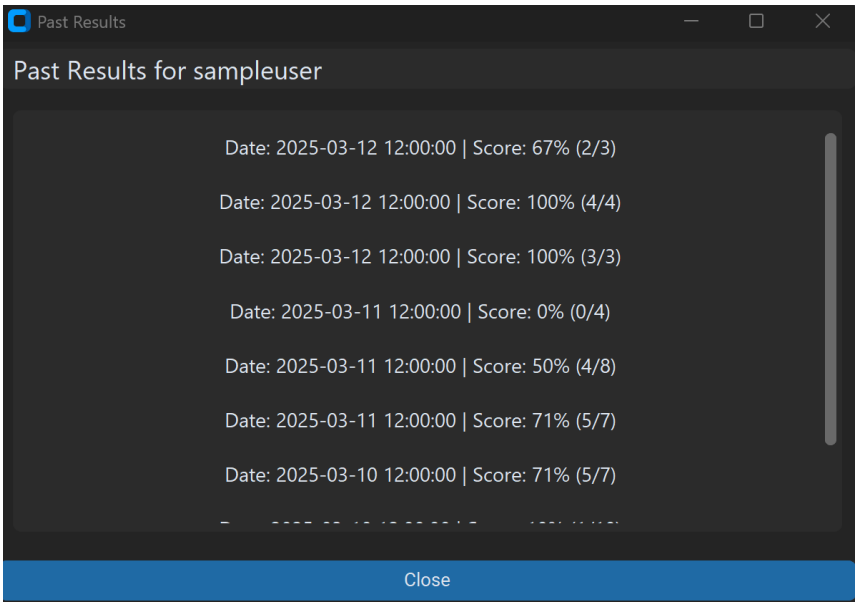
-Calculates the percentage score after retrieving the total number of questions, afterwards formats a string that shows the date and the score (including percentages and raw score over the total amount of questions).

-Finally, the footer frame creates a “Close” button that can be used to close the past results window.

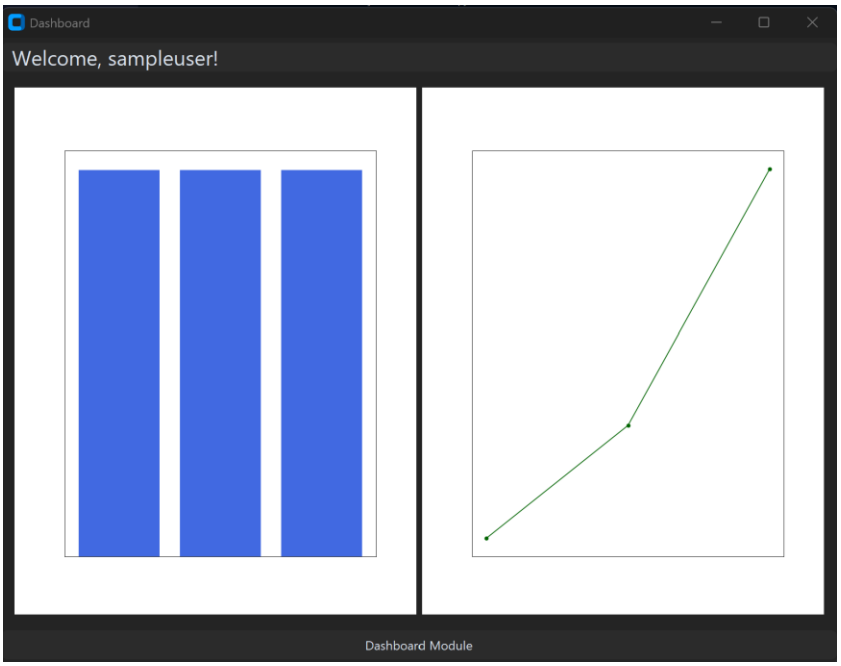
And now to finally test whether the storing, retrieving and showing of results has been successful, we can check the Past results table for the user,”sampleuser”:



The initial simulations that I conducted have been recorded, retrieved and shown successfully, including a date format, as well as a score format with percentages and raw scores. Hence, we can consider our previous functions a success and we can consider our past results browser module completed (for the time being).Since I have completed the main objectives of the past results browser, it is time to test how the analytics of the dashboard window have changed with the introduction of data. Before doing so, I will change the simulated date in the program and acquire 3 days' worth of data before doing so:



Now that I have enough days' worth of data and enough results, I can test the analytics of the dashboard:



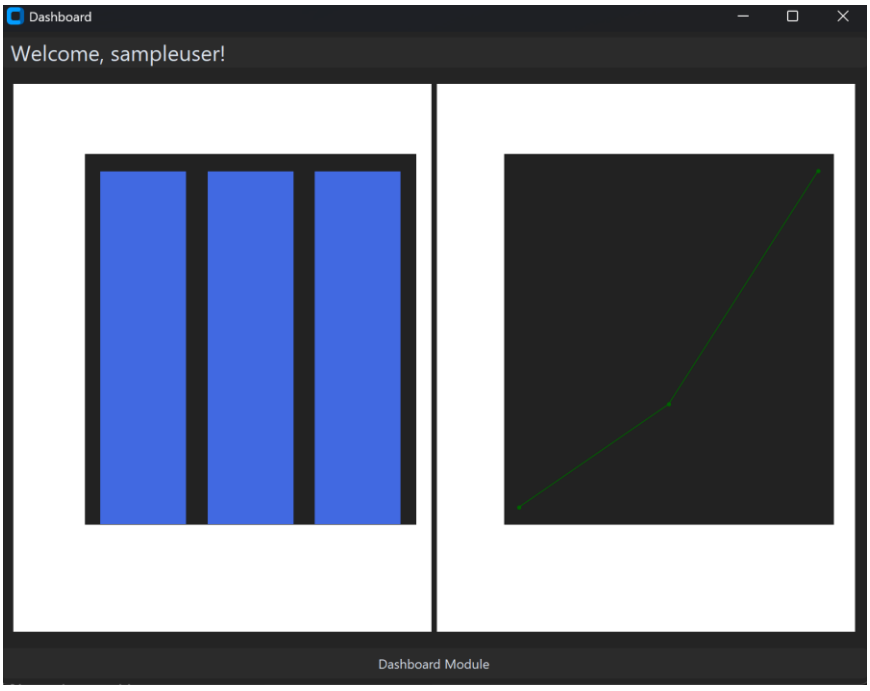
The bar charts and line graph did generate, however seeing as my labels haven't appeared properly as expected (which I will resolve), yet it may seem difficult to identify what these results show. It seems that the bar charts line up together due to the number of attempts retrieved for each day (3 each, as only 3 tests were conducted using each simulated date) staying the same, and the line graph has plotted the average percentage of correct questions per day. One key component that I will need to fix is the dates of results/attempts on the x-axis of both the bar chart and line graph as labels ,as well as the number of attempts labelling the y-axis of the bar chart and the average percentage of correct questions per day on the y-axis as a label on the line graph, doing so by going over my matplotlib code and methods in doing so.

I suspect that the issue is the white background drowning out the white labels as I have coded them to be white by default possibly in combination (or without it) with labels being pushed outside the visible area of the chart, as well as the possibility of the spines of the graphs being set to white by default as well. To follow best practises, I will try to provide fixes for all three of these possible causes.

```
# Bar Chart: Attempts per Day
fig1 = Figure(figsize=(4, 3), dpi=80)
ax1 = fig1.add_subplot(111)
ax1.bar(date_labels, attempts, color='royalblue')
ax1.set_title("Attempts per Day", fontname="Segoe UI", fontsize=14, color="white")
ax1.set_xlabel("Date", fontname="Segoe UI", fontsize=12, color="white")
ax1.set_ylabel("Attempts", fontname="Segoe UI", fontsize=12, color="white")
ax1.tick_params(axis='x', colors='white')
ax1.tick_params(axis='y', colors='white')
ax1.set_facecolor("#222222")  # Assigns colour to be the same as dashboard background
fig1.tight_layout()  # Ensures labels aren't pushed out
canvas1 = FigureCanvasTkAgg(fig1, master=graph_frame)
canvas1.draw()
canvas1.get_tk_widget().pack(side="left", fill="both", expand=True, padx=5)

# Line Chart: Average % Correct per Day
fig2 = Figure(figsize=(4, 3), dpi=80)
ax2 = fig2.add_subplot(111)
ax2.plot(date_labels, percentages, marker='o', color='darkgreen')
ax2.set_title("Average Score (%) per Day", fontname="Segoe UI", fontsize=14, color="white")
ax2.set_xlabel("Date", fontname="Segoe UI", fontsize=12, color="white")
ax2.set_ylabel("Avg %", fontname="Segoe UI", fontsize=12, color="white")
ax2.tick_params(axis='x', colors='white')
ax2.tick_params(axis='y', colors='white')
ax2.set_facecolor("#222222")  # Makes colour be the same as dashboard background
fig2.tight_layout()  # Ensures labels aren't pushed out
canvas2 = FigureCanvasTkAgg(fig2, master=graph_frame)
canvas2.draw()
canvas2.get_tk_widget().pack(side="right", fill="both", expand=True, padx=5)
```

The updated code uses the tight_layout method from matplotlib to ensure that my labels are not being pushed out of the window itself, as well as setting the colour of the axes to be the same as the dashboard background colour.

However, when running the updated code in the program, my dashboard window now appears as such:



It now seems that the labels being pushed out were not what the issue was, as after updating the code the only thing that has visibly changed so far was the graph background colours changing to the fg_color='#222222', which is a black colour as seen in the graph backgrounds.

To make my labels visible and fix the background colour, I have kept the “tight_layout” method but also changed the colour of the outer box lines of the graphs, while also setting both the colour of the axes and the figure’s patch to the same dark colour, which should finally make the labels visible by making the background colour universal for the whole window. For good measure I will also increase the figure size slightly to (5x4 at dpi=100) so the text of the labels is less likely to be clipped.

Updated code for each:

Figure size increase for both graphs:

```
#Bar Chart Attempts per day
fig1 = Figure(figsize=(5, 4), dpi=100)
ax1 = fig1.add_subplot(111)
```

```
#Line Chart Average % Correct per day
fig2 = Figure(figsize=(5, 4), dpi=100)
ax2 = fig2.add_subplot(111)
```

Setting the same dark background for figure and axes:

```
# Sets same dark background for figure and axes
fig1.patch.set_facecolor("#222222")
ax1.set_facecolor("#222222")
```

```
# Sets same dark background for both figure and axes
fig2.patch.set_facecolor("#222222")
ax2.set_facecolor("#222222")
```

Making the outer box lines white:

```
# Make outerboxlines visible in white
for spine in ax1.spines.values():
    spine.set_color("white")
```

```
# Makes outerboxlines visible in white
for spine in ax2.spines.values():
    spine.set_color("white")
```

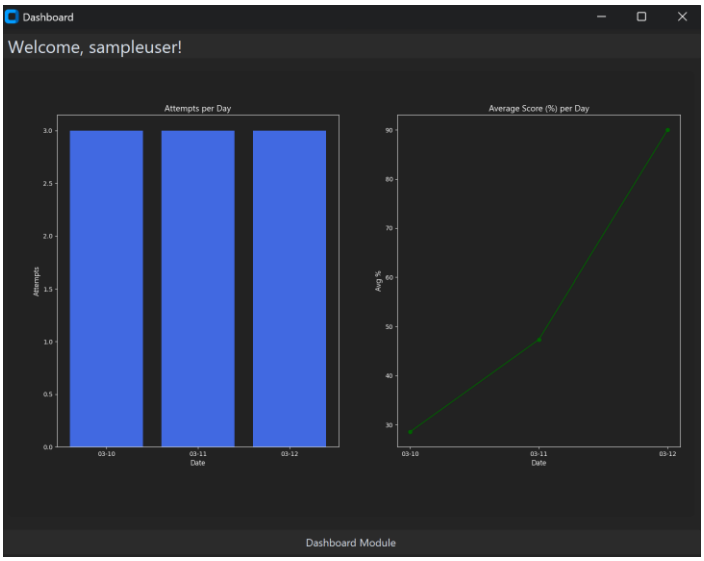
Full updated code for bar chart and line chart:

```
#Bar Chart Attempts per day
fig1 = Figure(figsize=(5, 4), dpi=100)
ax1 = fig1.add_subplot(111)
# Sets same dark background for figure and axes
fig1.patch.set_facecolor("#222222")
ax1.set_facecolor("#222222")
# Make outerboxlines visible in white
for spine in ax1.spines.values():
    spine.set_color("white")
ax1.bar(date_labels, attempts, color='royalblue')
ax1.set_title("Attempts per Day", fontname="Segoe UI", fontsize=14, color="white")
ax1.set_xlabel("Date", fontname="Segoe UI", fontsize=12, color="white")
ax1.set_ylabel("Attempts", fontname="Segoe UI", fontsize=12, color="white")
# Makes visible labels in white
ax1.tick_params(axis='x', colors='white')
ax1.tick_params(axis='y', colors='white')
# Prevents labels from being cut off
fig1.tight_layout()
canvas1 = FigureCanvasTkAgg(fig1, master=graph_frame)
canvas1.draw()
canvas1.get_tk_widget().pack(side="left", fill="both", expand=True, padx=5, pady=5)
```

```
#Line Chart Average % Correct per day
fig2 = Figure(figsize=(5, 4), dpi=100)
ax2 = fig2.add_subplot(111)
# Sets same dark background for both figure and axes
fig2.patch.set_facecolor("#222222")
ax2.set_facecolor("#222222")
# Makes outerboxlines visible in white
for spine in ax2.spines.values():
    spine.set_color("white")
ax2.plot(date_labels, percentages, marker='o', color='darkgreen')
ax2.set_title("Average Score (%) per Day", fontname="Segoe UI", fontsize=14, color="white")
ax2.set_xlabel("Date", fontname="Segoe UI", fontsize=12, color="white")
ax2.set_ylabel("Avg %", fontname="Segoe UI", fontsize=12, color="white")
# Makes visible labels in white
ax2.tick_params(axis='x', colors='white')
ax2.tick_params(axis='y', colors='white')
# Prevent labels from being cut off
fig2.tight_layout()
canvas2 = FigureCanvasTkAgg(fig2, master=graph_frame)
canvas2.draw()
canvas2.get_tk_widget().pack(side="right", fill="both", expand=True, padx=5, pady=5)
```

Now to test whether this has any significance on the result of the output in the dashboard analytics, I will sign back in as “sampleuser” and see for myself.

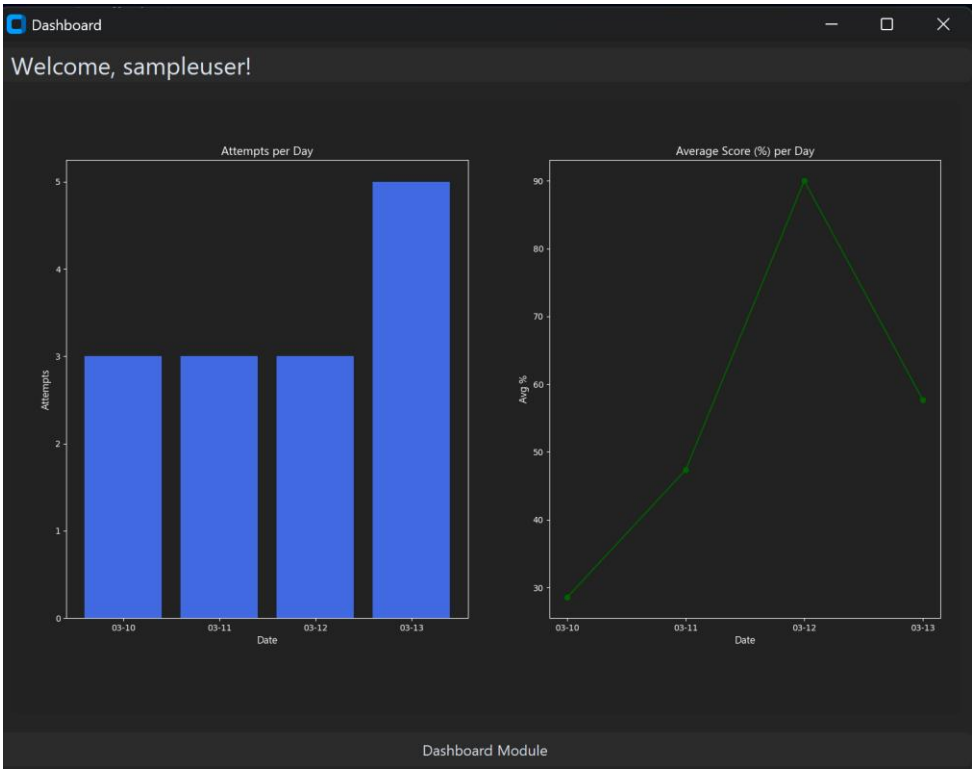
Dashboard analytics window seen when signed in as “sampleuser”:



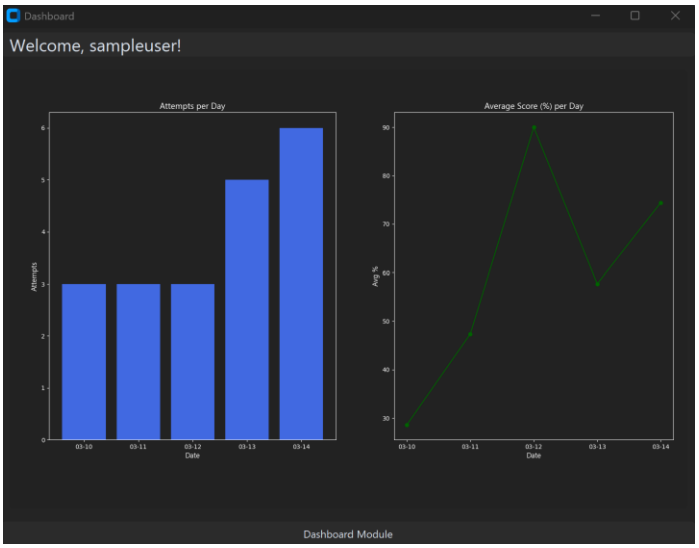
As seen above, the dashboard analytics are now functioning properly, with both x and y axes labels to convey the information concisely and clearly to the user. The outer box lines are now presented in white as desired, to clearly have a boundary between the charts themselves and their labels.

Before moving on I would like to test if the dashboard will note a change in results data properly by adding another days’ worth of results while also increasing the number of attempts in that specific day.

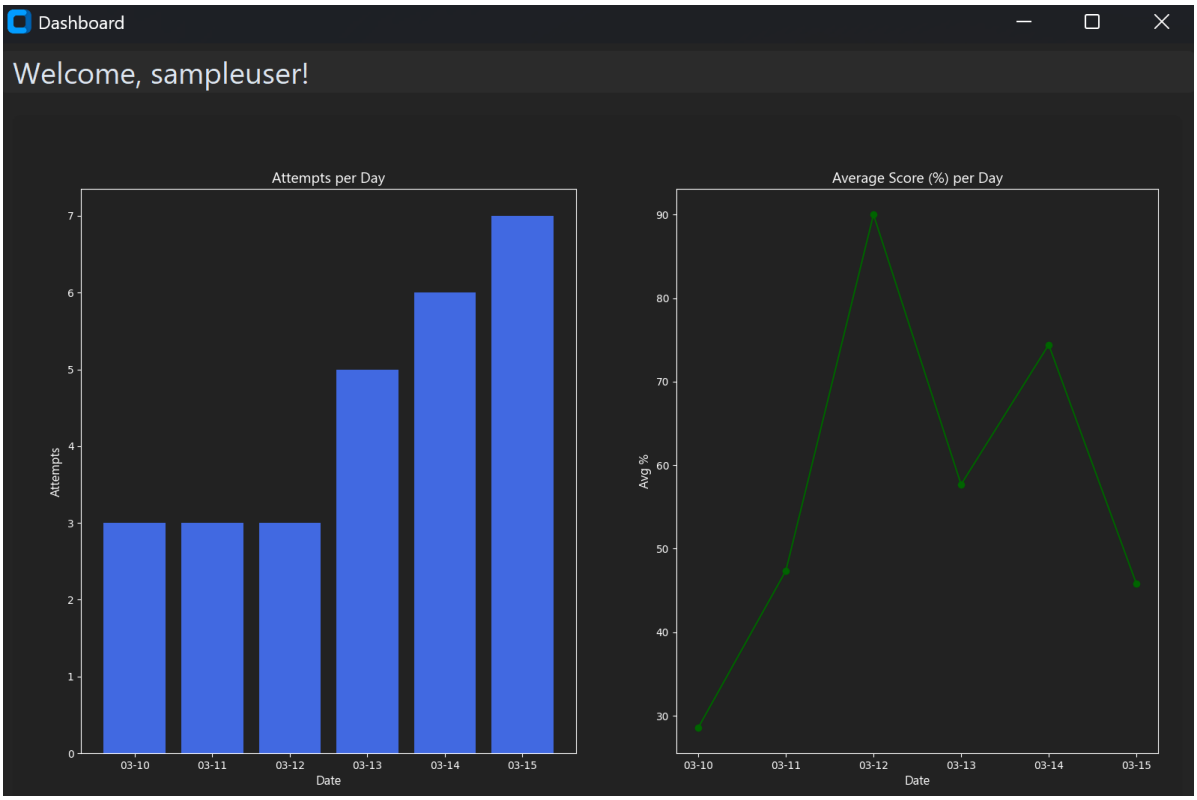
Testing for another days’ results down below:



As we can see, the dashboard has updated both charts as expected, and the recorded data seems accurate after adding another day worth of data. Before concluding that the dashboard module and past results browser are completed as lined out in the test data in my Design, I need to make sure that the dashboard does not show more than 5 days' worth of data at any one time. I will do so by first adding another 2 days' worth of data for a total of 6 days of data stored.



With 5 days' worth of data as seen above, the analytics are still functioning as expected. Now to see if they will work properly once they have 6 days' worth of data:



Unfortunately, the code seems to not be working as expected and is showing results for the last 6 calendar days instead of a maximum of 5 calendar days back.However, I believe that there is a quick fix to this if I just change the amount of days accounted for in the "last_five_days_attempts" function.

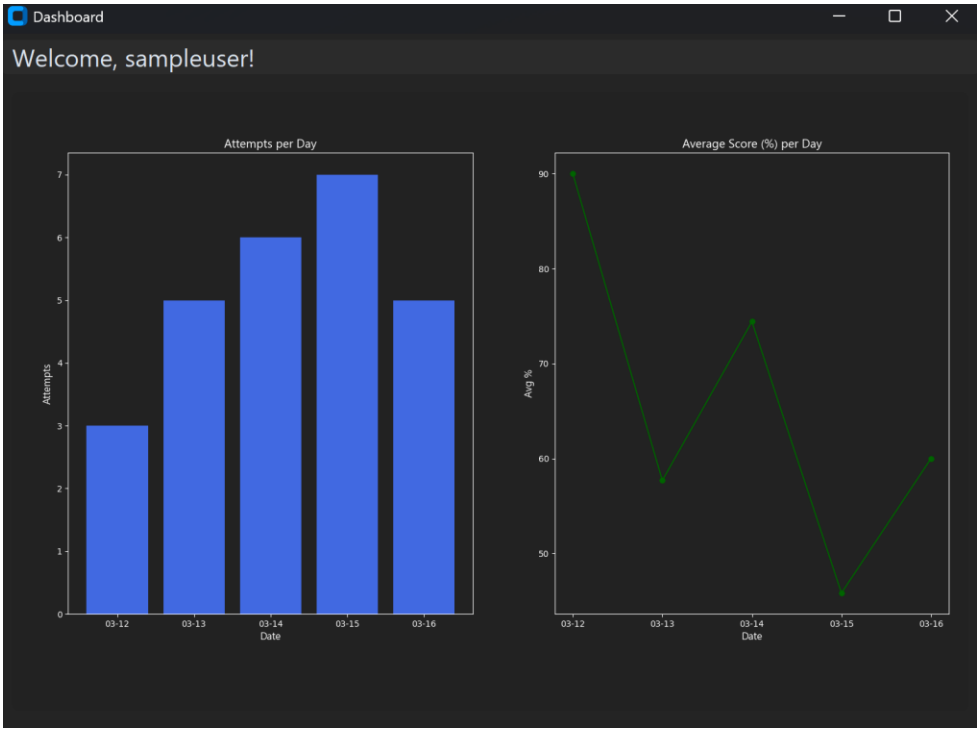
```
SELECT id, user_id, attempt_date, correct_list, total_questions
FROM results
WHERE user_id = ?
  AND date(attempt_date) >= date(?, '-5 days')
ORDER BY attempt_date DESC;
```

It seems that I made a mistake while writing the query for the last_five_days_attempts function by querying the current day (either simulated or real-time date) as well as the previous 5 days before that day using the "AND date(attempt_date) >= date(?, '-5 days')"

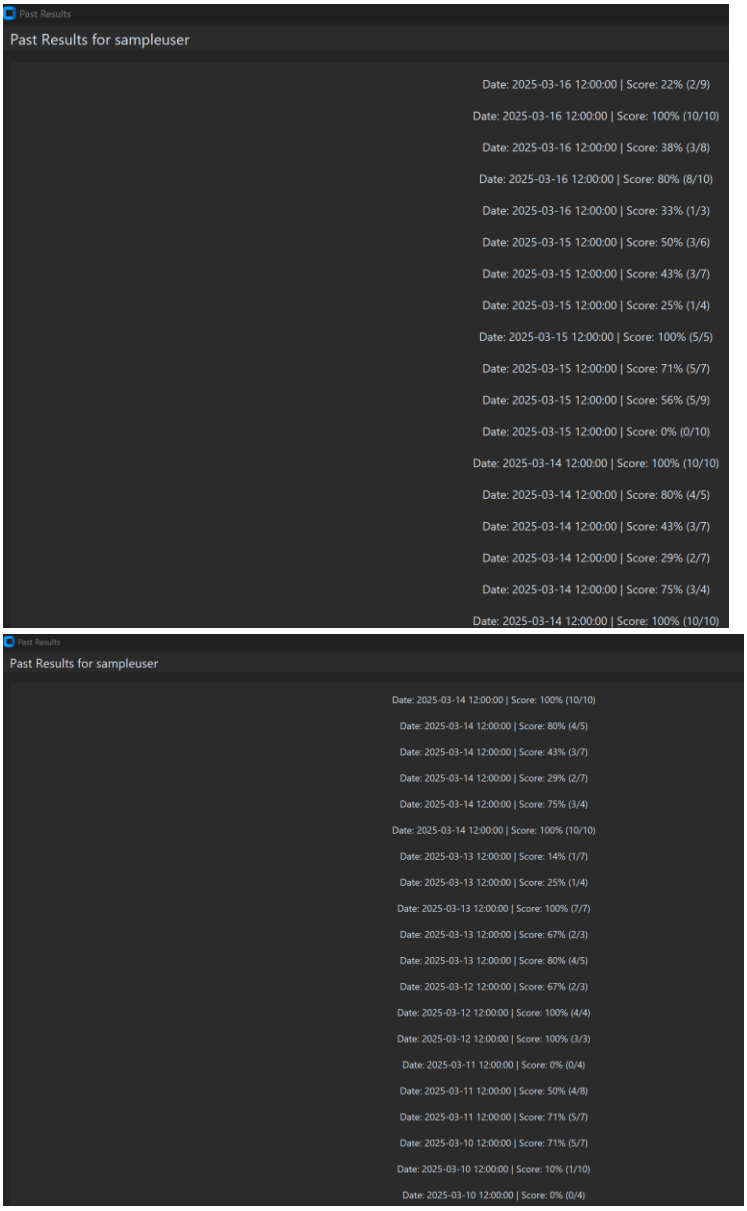
To remedy this mistake, I just need to change the '-5 days' into '-4 days', which will now only take the results from the current date as well as the 4 days prior to the current date, providing me with 5 days' worth of results displayed in the analytics, as desired.

```python
def last_five_days_attempts(user_id, current_datetime=None):
    now = current_datetime if current_datetime else (SIMULATED_DATE if SIMULATED_DATE else datetime.datetime.now())
    conn = get_db_connection()
    if conn is None:
        return []
    cursor = conn.cursor()
    query = """
    SELECT id, user_id, attempt_date, correct_list, total_questions
    FROM results
    WHERE user_id = ?
      AND date(attempt_date) >= date(?, '-4 days')
    ORDER BY attempt_date DESC;
    """
```

Now to test whether this fixes the issue:



Past results window showcasing that I have 7 days' worth of data for this user:



As we can see, only five days of results are represented on the charts, while I have 7 days' worth of data in the past results tab, hence the Dashboard analytics module does now show only results from the last 5 days. Hence, we can now conclude that the Dashboard and Past Results Window modules have been successes, as for the Dashboard module result data for at least the past 2 days up to the past 5 days at a maximum has been matched to the test data requirement to call this module a success, with the analytical charts also working as intended. The Past Results Window contains data from past results including correct and incorrect answers, as well as an overall percentage per quiz alongside showing how many answers were correct out of the total amount of questions answered. Hence, the Past Results Window Module can also be marked as a success when compared to the success criteria and test data.

User browser module (Teachers only)

Before beginning I will outline what functions I will need to create for the User browser module to work as intended and to abide by the success criteria and test data.

-A new database function will be needed that fetches all the users from the database

-A user browser window function that opens a window containing a search bar as well as a scrollable frame that will list user accounts, with each entry showing the username, account type along with a “View results” button.

-A user results window to show the selected user’s past results (Will reuse the already existing “get_all_results” function but display results for the chosen user instead of the current user).

-Finally, I will need to update the “open_user_browser” method in the MainApp class so that when an account that has the Teacher tag, clicking on the user browser will call the new user browser instead of showing the “Under construction” message.

Now to begin with the new user fetching function:

```
def get_all_users(search_query=None):
    conn = get_db_connection()
    if conn is None:
        return []
    cursor = conn.cursor()
    if search_query:
        cursor.execute("SELECT id, username, account_type FROM users WHERE username LIKE ?", (f"%{search_query}%",))
    else:
        cursor.execute("SELECT id, username, account_type FROM users")
    users = cursor.fetchall()
    conn.close()
    # Return a list of dictionaries for ease of use
    return [{"id": user[0], "username": user[1], "account_type": user[2]} for user in users]
```

The function retrieves the user account information from the database, returning all users or filtering the users based on the search query (by matching part of their username(s)). The latter is done by the “LIKE” clause which will only return the rows which contain the query string, if the search query is not provided it will select all rows from the user table instead. Afterwards the returned rows will be returned as a list of dictionaries with keys:”id”,”username” and “account_type”.

Now to continue onto the show user results function:

```
def show_user_results(main_app, user):
    # Create a window to show the selected user's past results
    results_win = ctk.CTkToplevel(main_app)
    results_win.title(f"Past Results for {user['username']}")
    results_win.geometry("600x400")

    header = ctk.CTkFrame(results_win)
    header.pack(fill="x", pady=5)
    ctk.CTkLabel(header, text=f"Past Results for {user['username']}", font=("Segoe UI", 18)).pack(side="left", padx=10)

    scroll_frame = ctk.CTkScrollableFrame(results_win, width=500, height=250)
    scroll_frame.pack(padx=10, pady=10, fill="both", expand=True)

    results = get_all_results(user["id"])  # Existing function that gets a user's results
    for row in results:
        attempt_dt = row[2]
        try:
            correct = len(eval(row[3]))
        except Exception:
            correct = 0
        total = row[4]
        percentage = (correct / total * 100) if total > 0 else 0
        date_str = attempt_dt.strftime("%Y-%m-%d %H:%M:%S")
        text = f"Date: {date_str} | Score: {percentage:.0f}% ({correct}/{total})"
        ctk.CTkLabel(scroll_frame, text=text, font=("Segoe UI", 14)).pack(pady=5)

    footer = ctk.CTkFrame(results_win)
    footer.pack(fill="x", pady=10)
    ctk.CTkButton(footer, text="Close", command=results_win.destroy).pack(fill="x")
```

The function will open a new window that displays the past quiz results for the selected user(s). This will only be used when a teacher clicks on “View results” next to a user in the browser.

A labelled header frame is added to the top to indicate whose results are being viewed, a CTkScrollableFrame has also been added to display the user’s past records in a scrollable area if the results are too many to initially fit on the screen (such as in the Past results module).

The function then calls my existing “get_all_results(user[“id”])” function to retrieve the past results, and then using a for loop loops over the results, converting the quiz attempt date into a readable string , calculating the percentage scores based on the total amount of questions and correctly answered questions, and creates a label for each result that displays the date, score percentage, and number of correct answers

Finally, a footer frame with a Close button is provided so that teacher account types can quickly exit the window when finished.

Now to continue onto the show user browser function:

This function will create a user interface for teachers to allow them to search through user accounts and view their quiz results.

```python
def show_user_browser(main_app):
    browser_win = ctk.CTkToplevel(main_app)
    browser_win.title("User Browser")
    browser_win.geometry("600x500")

    # Search bar section
    search_frame = ctk.CTkFrame(browser_win)
    search_frame.pack(fill="x", pady=5, padx=10)
    ctk.CTkLabel(search_frame, text="Search Users:", font=("Segoe UI", 14)).pack(side="left", padx=5)
    search_entry = ctk.CTkEntry(search_frame, placeholder_text="Enter username")
    search_entry.pack(side="left", padx=5, fill="x", expand=True)

    # Scrollable frame to list users
    results_frame = ctk.CTkScrollableFrame(browser_win, width=580, height=350)
    results_frame.pack(padx=10, pady=10, fill="both", expand=True)

    def perform_search():
        # Clear current results
        for widget in results_frame.winfo_children():
            widget.destroy()
        query = search_entry.get().strip()
        users = get_all_users(query)
        if not users:
            ctk.CTkLabel(results_frame, text="No users found.", font=("Segoe UI", 14)).pack(pady=5)
        for user in users:
            user_frame = ctk.CTkFrame(results_frame)
            user_frame.pack(fill="x", pady=3, padx=5)
            info_text = f"Username: {user['username']} | Account Type: {user['account_type']}"
            ctk.CTkLabel(user_frame, text=info_text, font=("Segoe UI", 12)).pack(side="left", padx=5)
            # Button to view this user's results
            ctk.CTkButton(user_frame, text="View Results", command=lambda u=user: show_user_results(main_app, u)).pack(side="right", padx=5)

    ctk.CTkButton(search_frame, text="Search", command=perform_search).pack(side="left", padx=5)

    def list_all_users():
        # Clear the frame and list all users
        for widget in results_frame.winfo_children():
            widget.destroy()
        users = get_all_users()
        if not users:
            ctk.CTkLabel(results_frame, text="No users found.", font=("Segoe UI", 14)).pack(pady=5)
        for user in users:
            user_frame = ctk.CTkFrame(results_frame)
            user_frame.pack(fill="x", pady=3, padx=5)
            info_text = f"Username: {user['username']} | Account Type: {user['account_type']}"
            ctk.CTkLabel(user_frame, text=info_text, font=("Segoe UI", 12)).pack(side="left", padx=5)
            ctk.CTkButton(user_frame, text="View Results", command=lambda u=user: show_user_results(main_app, u)).pack(side="right", padx=5)

    # Initially list all users
    list_all_users()

    # Footer: Close button for the browser window
    footer = ctk.CTkFrame(browser_win)
    footer.pack(fill="x", pady=5)
    ctk.CTkButton(footer, text="Close", command=browser_win.destroy).pack(pady=5)
```

It works by creating a new window (ctk.CTkToplevel) titled “User Browser” that will have a search bar section with a label indicating the purpose (“Search Users”), a text search for typing in a username alongside a “search” button that triggers a search when clicked.

The search functionality has two nested functions, the first one being ”perform_search” which clears any previous search results, retrieves users matching the search query and iterates over each returned user list, while displaying the username and account type of each user as well as a “View Results” button.

The second nested function being “list_all_users” which lists all users without filtering by using “get_all_users”.

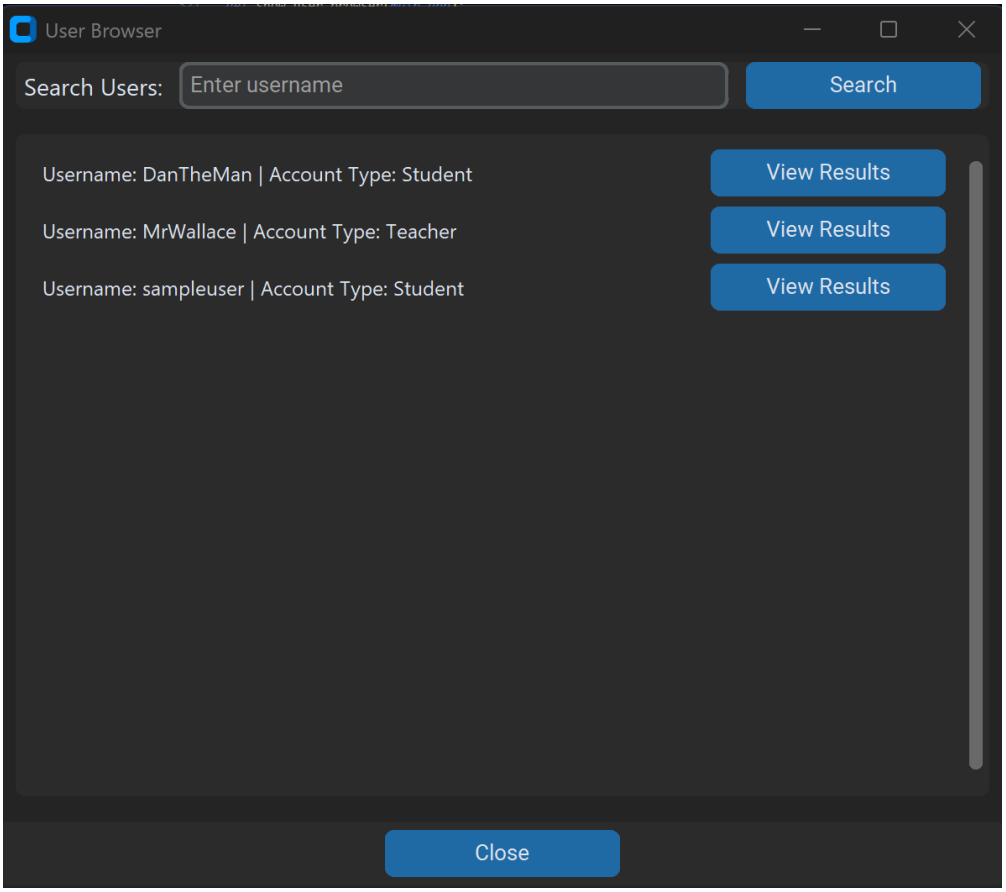
Once again, we have a footer with a close button added for the teachers to exit the window.

To finish off, a scrollable frame will be present in this window, so when too many users exist to be all viewed at once on the window, the teacher can scroll down to view all results.

As I have created all the functions needed for the “Quiz Browser” now I need to integrate them into the “MainApp”. To do so I have updated the “open_user_browser” function to check if the account type is that of a teacher and if positive then it will call upon “show_user_browser(self)”, which opens the user browser window, otherwise if a student account attempts to enter , they will only receive an error message and not connect to the user browser.

```
def open_user_browser(self):
    if self.check_login_status():
        if self.current_user["account_type"] == "Teacher":
            show_user_browser(self)
        else:
            messagebox.showerror("Access Denied", "Only accessible by teachers.")
```

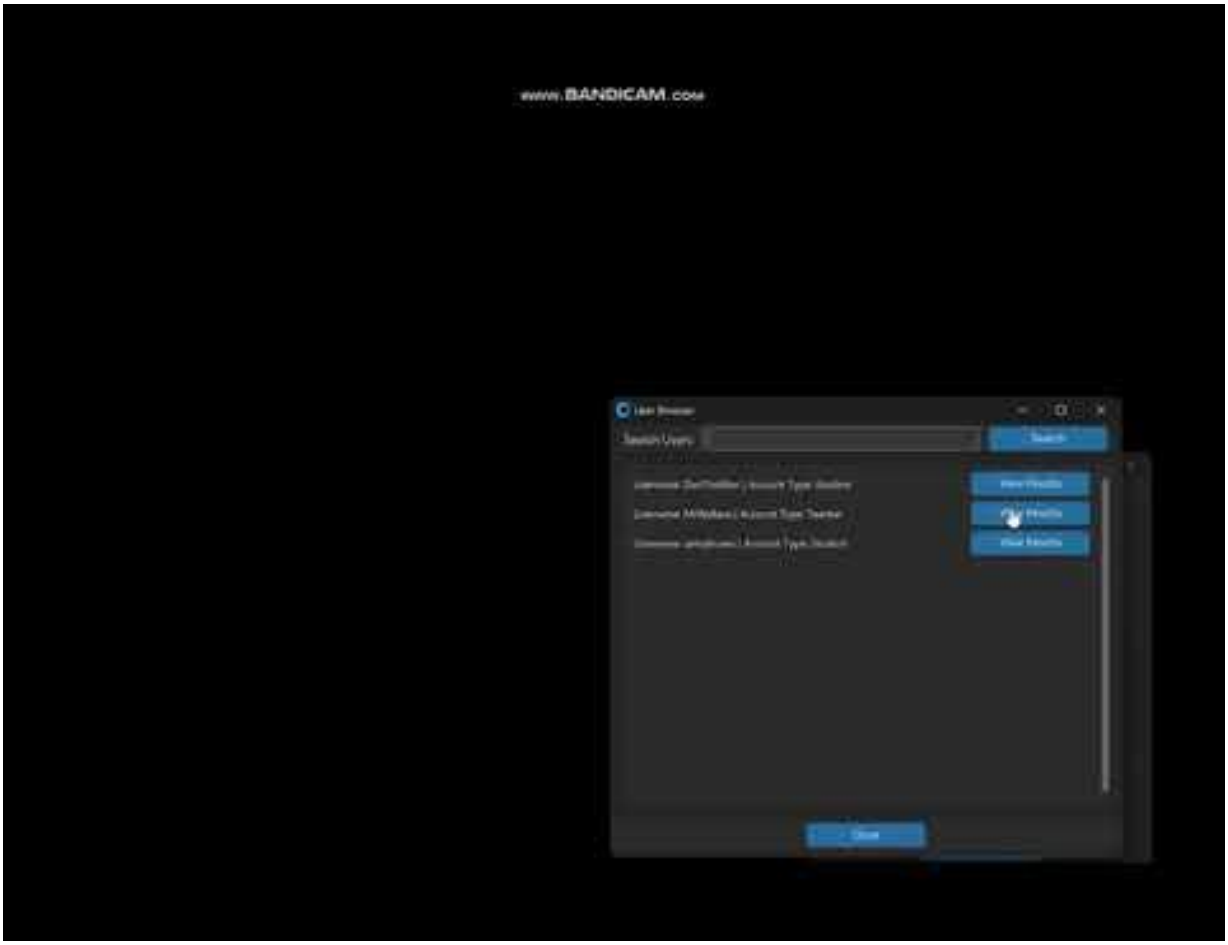
As I have completed the development for this module, let’s now test if the program will behave as expected. Firstly, I will log in to the program with a teacher account and attempt to enter the user browser.



As we can see, the user browser window looks as I expected, and the frontend of the code seems to be working, as we have the usernames, account type of each student and the “View Results”

button for each account. The search engine as well as the “Search” button can clearly be seen on the top, alongside the scrollable element on the right side of the window and the close button on the bottom for when a teacher wishes to close the program.

I have conducted testing for each required section from this module and uploaded it on YouTube.



The search engine of the module is working seamlessly, identifying usernames in the database even based on keywords instead of the whole username and the “Search” button is working as expected, resetting when the search text box is empty to show all users again. The “View Results” button is also working as intended and showing the results for each user (“DanTheMan” and “MrWallace” do not have any results to show, hence why their results were empty) and as we can see, all the results of the “sampleuser” can be clearly seen , with their recorded date, score(percentage , correct answers/total questions).All “Close” back buttons are also working as intended, and upon logging out of the Teacher account, we can see that student accounts still cannot access this module at all(as they are filtered out) , hence the user browser module has been a success as measured by the test data and success criteria, as I have verified that teachers can scroll through the multitude(all) users and view their results. I will now move onto the Manage Account module.

Manage account (Change username and change password modules)

The account management module will consist of two sections: A change username and change password section, and I will need to add a “Manage Account” window onto the placeholder one that I have programmed from before, and from there both sections will be accessible.

The code for this module will consist of the following:

-A change username function, which will check if the username is between 5 and 16 characters, checking if the username already exists and if neither are true not allowing a change to take place, but if validation passes then the function will change the username.

-A change password function, which will check if the new password has a minimum of 6 characters, is not repeated, and matches the confirmation field and if these conditions are fulfilled will then change the password, if not then the function will stop. Hashing remains a component

-A show function that will show the “Manage account” window and integrate both functions above into it with full

As well as integrating these functions into the “MainApp”(Main Application) for functionality.

Change username function:

```
def change_username(main_app, new_username):
    if len(new_username) < 5 or len(new_username) > 16:
        messagebox.showerror("Error", "Username must be between 5 and 16 characters.")
        return
    conn = get_db_connection()
    if conn is None:
        messagebox.showerror("Error", "Database connection error.")
        return
    cursor = conn.cursor()
    cursor.execute("SELECT id FROM users WHERE username = ?", (new_username,))
    if cursor.fetchone():
        messagebox.showerror("Error", "Username already exists. Please choose another.")
        conn.close()
        return
    cursor.execute("UPDATE users SET username = ? WHERE id = ?", (new_username, main_app.current_user["id"]))
    conn.commit()
    conn.close()
    main_app.current_user["username"] = new_username
    messagebox.showinfo("Success", "Username changed successfully!")
```

The change username function will attempt a connection to the database to check if the username is between 5 and 16 characters, if it’s not then it will return an error message or if the connection is disrupted it will return a different error message notifying the user of a connection error. If the username length is validated and the new username is also unique and within the boundary, the username will be updated in the database and the change will be immediate, notifying the user of the success.

Change password function:

```
def change_password(main_app, new_password, confirm_password):
    if len(new_password) < 6:
        messagebox.showerror("Error", "New password must be at least 6 characters long.")
        return
    if new_password != confirm_password:
        messagebox.showerror("Error", "Password confirmation does not match.")
        return
    conn = get_db_connection()
    if conn is None:
        messagebox.showerror("Error", "Database connection error.")
        return
    cursor = conn.cursor()
    cursor.execute("SELECT password FROM users WHERE id = ?", (main_app.current_user["id"],))
    row = cursor.fetchone()
    if not row:
        messagebox.showerror("Error", "User not found in database.")
        conn.close()
        return
    current_hashed = row[0]
    if bcrypt.checkpw(new_password.encode('utf-8'), current_hashed if isinstance(current_hashed, bytes) else current_hashed.encode('utf-8')):
        messagebox.showerror("Error", "New password cannot be the same as the old password.")
        conn.close()
        return
    new_hashed = bcrypt.hashpw(new_password.encode('utf-8'), bcrypt.gensalt())
    cursor.execute("UPDATE users SET password = ? WHERE id = ?", (new_hashed, main_app.current_user["id"]))
    conn.commit()
    conn.close()
    messagebox.showinfo("Success", "Password changed successfully!")
```

For validation the function will connect to the database to retrieve the old password (if a connection isn't made or is disrupted, an error message will be shown to the user) and check if the new password is at least 6 characters long, then verify that the password matches the one in the confirmation field and isn't the same as the old password(which will be confirmed by retrieving the hash value).If all validations pass then the password will be updated immediately and the user will be notified of the change.

Show" Manage Account" window function:

A new window is created with a title and pre-defined size with a change username section linking it to the change_username function and a change password section linking it to the change_password function, with a button for each section that will call their respective function. Finally, a footer with a close button allows the user to exit the window.

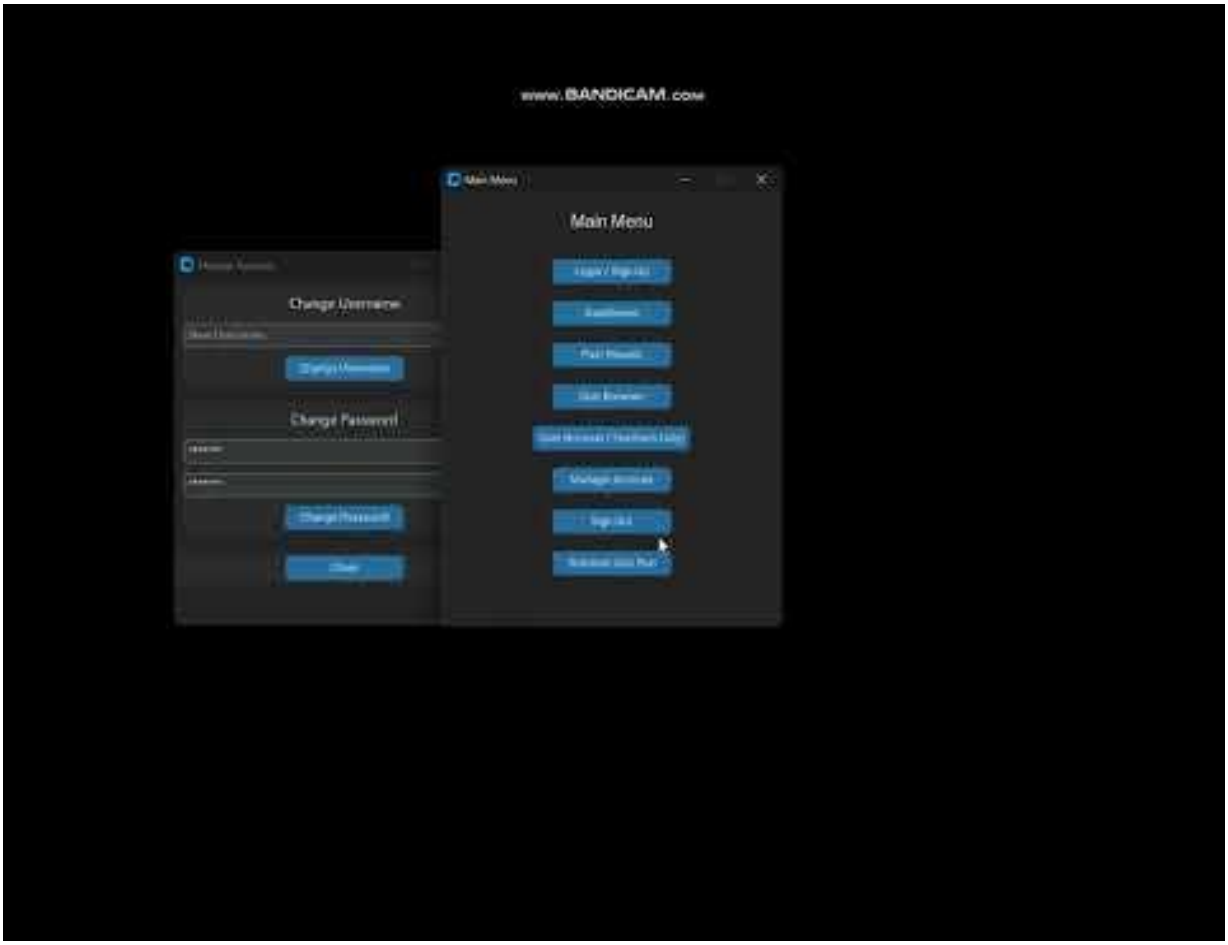
```
def show_manage_account(main_app):
    manage_win = ctk.CTkToplevel(main_app)
    manage_win.title("Manage Account")
    manage_win.geometry("400x400")

    # Change Username part
    username_frame = ctk.CTkFrame(manage_win)
    username_frame.pack(fill="x", padx=10, pady=10)
    ctk.CTkLabel(username_frame, text="Change Username", font=("Segoe UI", 16)).pack(pady=5)
    new_username_entry = ctk.CTkEntry(username_frame, placeholder_text="New Username")
    new_username_entry.pack(pady=5, fill="x")
    ctk.CTkButton(username_frame, text="Change Username", command=lambda: change_username(main_app, new_username_entry.get().strip())).pack(pady=5)

    # Change Password part
    password_frame = ctk.CTkFrame(manage_win)
    password_frame.pack(fill="x", padx=10, pady=10)
    ctk.CTkLabel(password_frame, text="Change Password", font=("Segoe UI", 16)).pack(pady=5)
    new_password_entry = ctk.CTkEntry(password_frame, placeholder_text="New Password", show="*")
    new_password_entry.pack(pady=5, fill="x")
    confirm_password_entry = ctk.CTkEntry(password_frame, placeholder_text="Confirm New Password", show="*")
    confirm_password_entry.pack(pady=5, fill="x")
    ctk.CTkButton(password_frame, text="Change Password", command=lambda: change_password(main_app, new_password_entry.get(), confirm_password_entry.get())).pack(pady=5)

    # Footer with Close Button
    footer = ctk.CTkFrame(manage_win)
    footer.pack(fill="x", pady=10)
    ctk.CTkButton(footer, text="Close", command=manage_win.destroy).pack(pady=5)
```

Before conducting any testing, I need to integrate the window into the "open_manage_account" function in the Main Application:

```
def open_manage_account(self):
    if self.check_login_status():
        show_manage_account(self)
```

As that has been done, now I can begin the testing, which will be done using the “DanTheMan” student account with password:”123456”, where I will attempt to change the username to “DanTheMan1” and the password to: “1234567”.Then I will also attempt to login and validate whether or not the username and password have changed in the database.

In the video below I have shown that username change, and password change works if they are within the allotted boundaries and are a unique/non-repeating username or password. This has been proven by the fact that right after changing the username and password I attempted to change them again (even though both stayed the same) and I received the appropriate error messages that I was expecting.



Now to check the database itself to see if the username and password for the “DanTheMan” account have changed (Note that we will conclude that the password has changed by a different hash value than what we had before).

```
ID | Username | Password | Account Type
----------------------------------------
5 | DanTheMan | b'$2b$12$iKPxKvkIAnqqDWCMeoF5z.IXhm85WCs2LmN4zSBZqY.cHyN4cTCSe' | Student
```

```
ID | Username | Password | Account Type
----------------------------------------
5 | DanTheMan1 | b'$2b$12$ZUHu5f/e0deT7wgX.1jk7uDplp0AdQnkytBq1/17hPx8FkIOckHtm' | Student
```

As we can clearly see by using the output table function, the username has been updated as planned, and the hash value for the password has changed, indicating that the password itself has been updated to the desired one. We can assume that all is working well but we still need to attempt to log back in with the updated details.

In the video below we can also see that attempting to log-in after a username AND password change works seamlessly as expected, and I have also shown that usernames not in the allotted boundary are rejected and not updated, with the same outcome for passwords that were under 6 characters for the passwords.

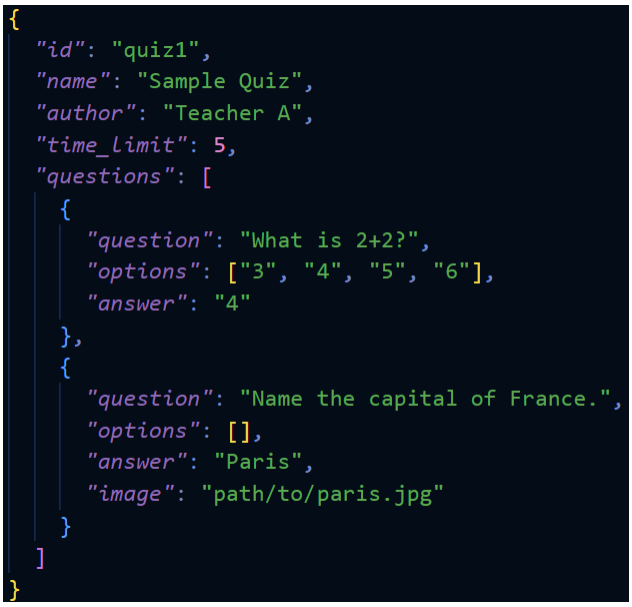
We can now conclude the testing and classing it as a success according to the test data for development and the success criteria for the Change Username and Change password modules,

as I have ensured that the change username module can validate changes when a username is not already stored in the database, new username is within allotted character boundary, validates when user credentials are correct(correct by default as user needs to be logged in to change their credentials anyway).The change password test data has also been fulfilled and matched, as the new password cannot be changed if it is the same as the old one or if less than 6 characters in length and password not updating when a user does not confirm the new password correctly(by writing it out twice).As I have completed my objectives, I would now move on to the Quiz Browser module.

Quiz Browser, Quiz Launching and Quiz Execution Modules

Quiz Browser Module

Before developing this module, I will first provide an example of a quiz file that will be used, which is in JSON format:



```json
{
  "id": "quiz1",
  "name": "Sample Quiz",
  "author": "Teacher A",
  "time_limit": 5,
  "questions": [
    {
      "question": "What is 2+2?",
      "options": ["3", "4", "5", "6"],
      "answer": "4"
    },
    {
      "question": "Name the capital of France.",
      "options": [],
      "answer": "Paris",
      "image": "path/to/paris.jpg"
    }
  ]
}
```

For files to be able to load and run in the program, they will need to follow this JSON format, else they will not be validated and loaded or if they do load, other issues may occur.

I will begin by first writing a function which fetches all the quizzes stored in a designated quiz folder that I have made with file type ".json", storing each loaded quiz as a dictionary and then returning a list of all quiz dictionaries.

```python
def get_all_quizzes():
    """
    Reads all quiz JSON files from the QUIZ_DIR and returns list of quiz dictionaries.
    """
    quizzes = []
    for filename in os.listdir(QUIZ_DIR):
        if filename.endswith(".json"):
            filepath = os.path.join(QUIZ_DIR, filename)
            with open(filepath, "r", encoding="utf-8") as f:
                try:
                    quiz_data = json.load(f)
                    quiz_data["file_path"] = filepath  # Save file path if needed
                    quizzes.append(quiz_data)
                except json.JSONDecodeError:
                    print(f"Error reading quiz file {filename}")
    return quizzes
```

The function centralises the retrieval of quiz data, hence when I later add functions for launching and execution, the available quizzes will easily be listed.

Next, I need to create a function that allows users to manually add their own quizzes to the application.

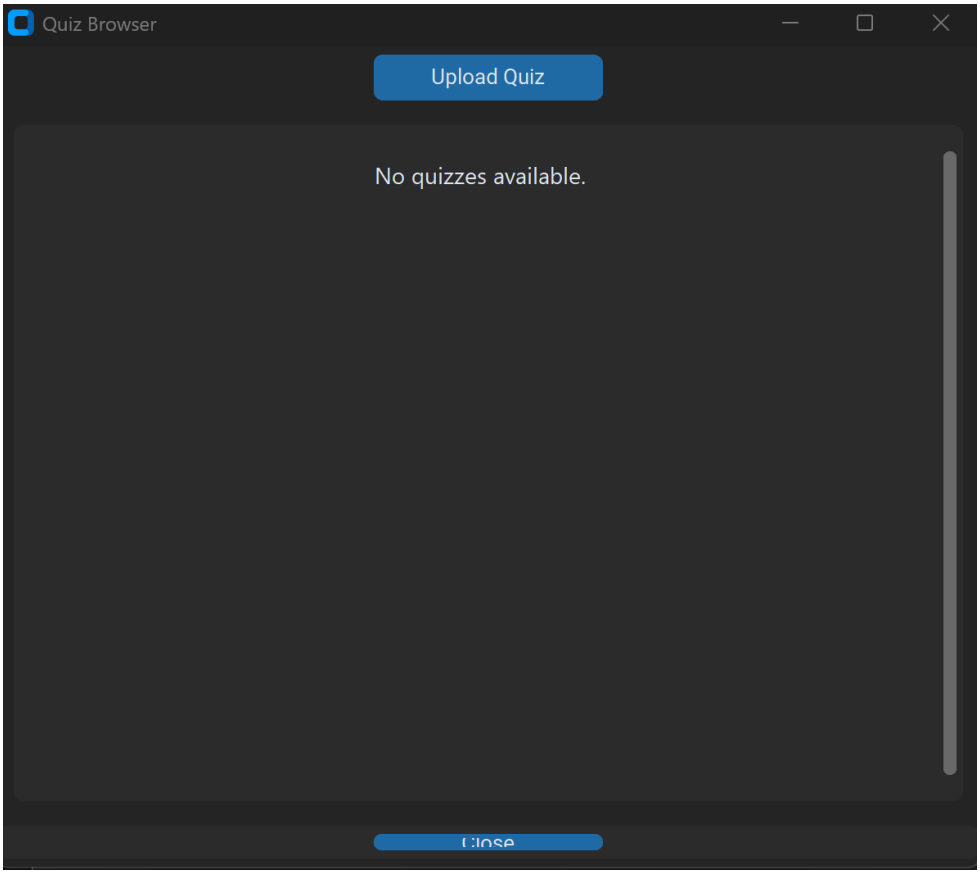
```python
def upload_quiz():
    """
    Opens a file dialog for the user to select a quiz JSON file, and then copies it to QUIZ_DIR.
    """
    file_path = filedialog.askopenfilename(title="Select Quiz JSON", filetypes=[("JSON Files", "*.json")])
    if file_path:
        dest_path = os.path.join(QUIZ_DIR, os.path.basename(file_path))
        try:
            with open(file_path, "r", encoding="utf-8") as src:
                quiz_data = src.read()
            with open(dest_path, "w", encoding="utf-8") as dst:
                dst.write(quiz_data)
            messagebox.showinfo("Upload", "Quiz uploaded successfully!")
        except Exception as e:
            messagebox.showerror("Error", f"Failed to upload quiz: {e}")
```

The function above opens a file dialog that lets the user select a quiz JSON file, then copies it to" QUIZ_DIR", which is the quiz folder. Upon successful completion, a message notifying the user is displayed, and upon unsuccessful storing a message notifying the user is displayed as well.

Now to test whether these two functions are working as expected, I will need to create a quiz browser window with an interactive interface for browsing available quizzes that has a button (that calls on the "upload_quiz" function) which allows users to upload quizzes. A scrollable frame will also be included, which will allow users to scroll through uploaded quizzes (which will display the quiz name,author,number of questions, and time limit per quiz) when results become too numerous to be shown on the screen. Crucially, a "launch quiz" button will also have to be included to call on a function that should launch the quizzes (will be added further on in this section).

```python
def show_quiz_browser(main_app):
    """
    Displays the Quiz Browser window with available quizzes and an option to upload new ones.
    """
    qb_win = ctk.CTkToplevel(main_app)
    qb_win.title("Quiz Browser")
    qb_win.geometry("600x500")

    # Button to upload a quiz
    upload_btn = ctk.CTkButton(qb_win, text="Upload Quiz", command=upload_quiz)
    upload_btn.pack(pady=5)

    # Scrollable frame to list quizzes
    scroll_frame = ctk.CTkScrollableFrame(qb_win, width=580, height=400)
    scroll_frame.pack(padx=10, pady=10, fill="both", expand=True)

    quizzes = get_all_quizzes()
    if not quizzes:
        ctk.CTkLabel(scroll_frame, text="No quizzes available.", font=("Segoe UI", 14)).pack(pady=10)
    else:
        for quiz in quizzes:
            quiz_frame = ctk.CTkFrame(scroll_frame)
            quiz_frame.pack(fill="x", pady=5, padx=5)
            info_text = f"Name: {quiz.get('name', 'N/A')} | Author: {quiz.get('author', 'N/A')} | Questions: {len(quiz.get('questions', []))} | Time: {quiz.get('time_limit', 'N/A')} mins"
            ctk.CTkLabel(quiz_frame, text=info_text, font=("Segoe UI", 12)).pack(side="left", padx=5)
            ctk.CTkButton(quiz_frame, text="Launch Quiz", command=lambda q=quiz: launch_quiz(main_app, q)).pack(side="right", padx=5)

    footer = ctk.CTkFrame(qb_win)
    footer.pack(fill="x", pady=5)
    ctk.CTkButton(footer, text="Close", command=qb_win.destroy).pack(pady=5)
```

Now that the fetching, uploading and quiz browser functions have all been created, I can attempt launching the quiz browser to see if everything is displaying and working correctly:



On the frontend, everything except the “Close “button seem to be displaying correctly, as we can see that the “Close” button is being pushed down and isn’t fully displaying, which I suspect may be due to the scrollable frame pushing it out of the window by taking up most of the virtual space. I will move onto correcting this in a second, but to highlight so far: The upload quiz button is displaying as expected alongside the “No quizzes available” message, as no quizzes

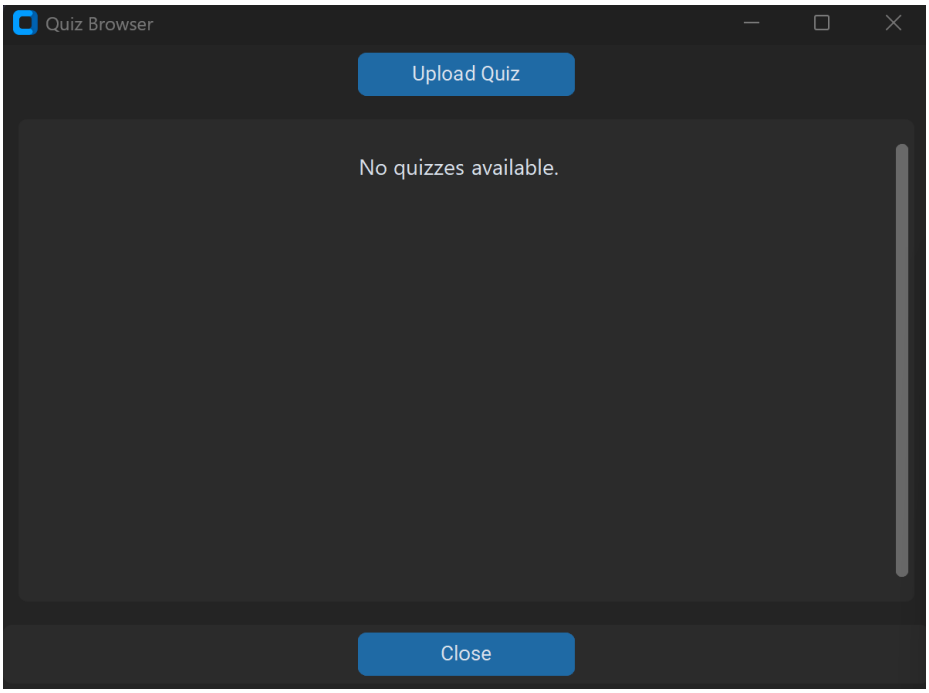
have been uploaded yet. Hence for the time being I only need to make sure that the close button is displaying as expected.

After checking customtkinter's official documentation, it seems that the issue is caused by not specifying a side on the "footer.pack" method, which defaults to top packing if not specified and results in the scroll frame attempting to fill the entire window instead of just the middle.To fix the problem , I will limit the height of the scrollable frame by fixing the height and not allowing it to expand and take up all the free space.

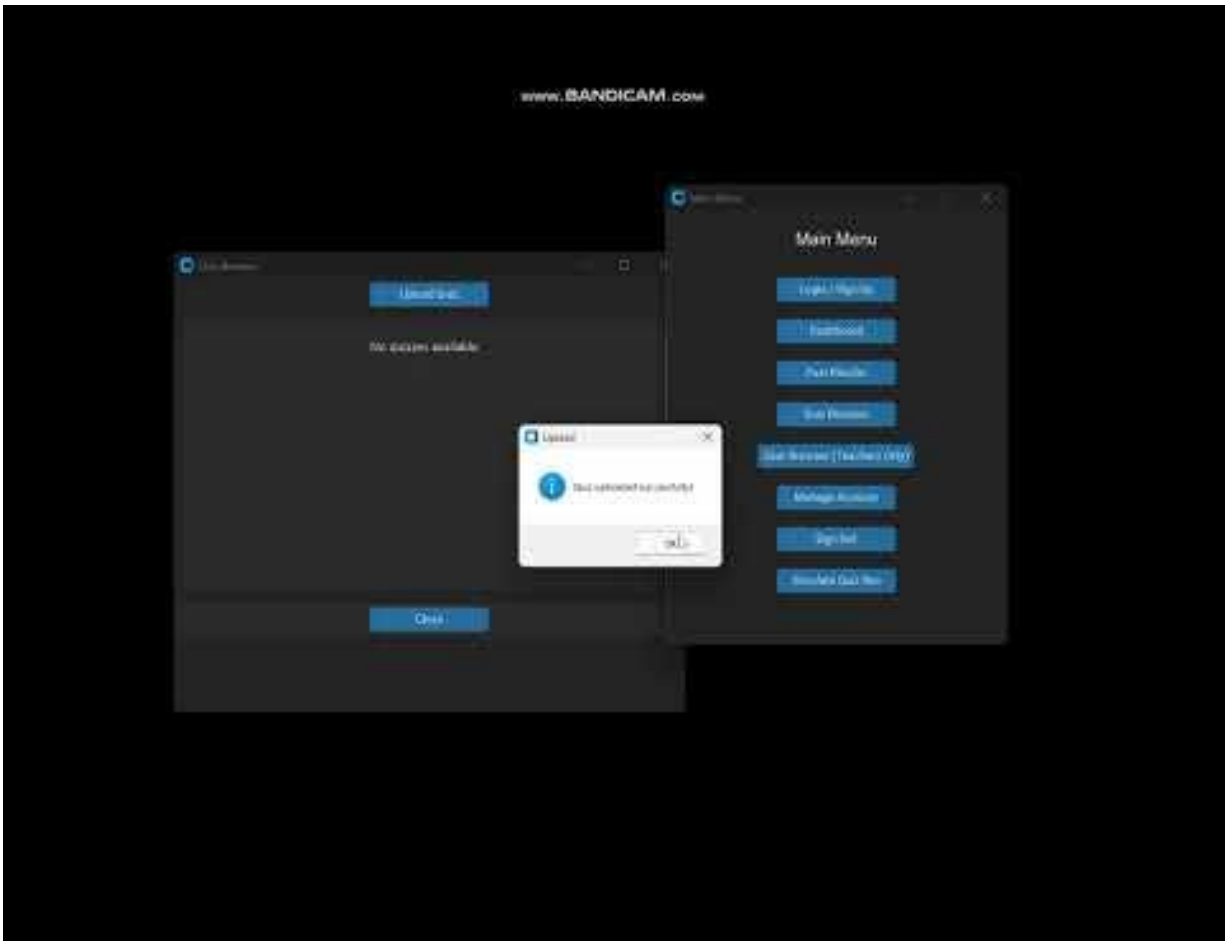
I have done this by amending the scrollable frame code:

```
scroll_frame = ctk.CTkScrollableFrame(qb_win, width=580, height=300)
scroll_frame.pack(padx=10, pady=10, fill="x")
```

Now to check if this has resolved the issue:



As we can now see, the issue has been resolved as the close button is now fully visible and the scrollable frame's size is limited and does not try to expand further. Moving on, now I will test whether the upload quiz button and function is working as expected.



After conducting a test (video shown above) to see if a quiz can be uploaded successfully by a user, we saw that the successful upload message box was shown, but the quiz browser window was still not showing any new quizzes even after refreshing. After analysing my code, I believe that the quiz browser's display is never refreshed to show the newly uploaded results, even after the application has been restarted.

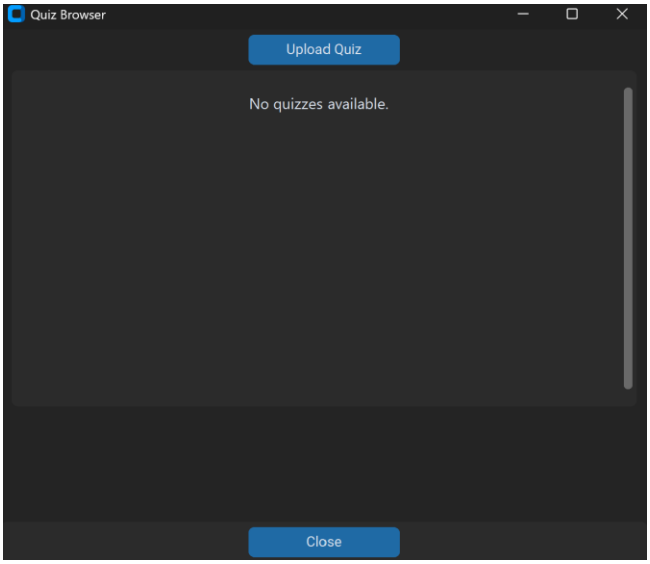
To fix the issue, I need to re-fetch and re-display the quizzes after the user has finished uploading. This can be done with the inclusion of two more functions, one that will wrap the quiz listing, refresh the scrollable window then call "get_all_quizzes" again and then rebuild the UI of the window again with the new quiz list. The second function would then call the refresh function immediately after a successful upload, so the UI redisplays the quizzes.

New nested functions under "show_quiz_browser":

```python
# Creates a refresh function to re-fetch & display quizzes
def refresh_quiz_list():
    # Clear previous items
    for widget in scroll_frame.winfo_children():
        widget.destroy()
    # Fetch quizzes
    quizzes = get_all_quizzes()
    if not quizzes:
        ctk.CTkLabel(scroll_frame, text="No quizzes available.", font=("Segoe UI", 14)).pack(pady=10)
    else:
        for quiz in quizzes:
            quiz_frame = ctk.CTkFrame(scroll_frame)
            quiz_frame.pack(fill="x", pady=5, padx=5)
            info_text = (
                f"Name: {quiz.get('name', 'N/A')} | "
                f"Author: {quiz.get('author', 'N/A')} | "
                f"Questions: {len(quiz.get('questions', []))} | "
                f"Time: {quiz.get('time_limit', 'N/A')} mins"
            )
            ctk.CTkLabel(quiz_frame, text=info_text, font=("Segoe UI", 12)).pack(side="left", padx=5)
            ctk.CTkButton(
                quiz_frame, text="Launch Quiz",
                command=lambda q=quiz: launch_quiz(main_app, q)
            ).pack(side="right", padx=5)
```

```python
# Wrap upload_quiz() so we can refresh
def upload_and_refresh():
    upload_quiz()       # copies the JSON file into "quizzes" folder
    refresh_quiz_list() # re-fetches & re-displays quizzes

upload_btn = ctk.CTkButton(qb_win, text="Upload Quiz", command=upload_and_refresh)
upload_btn.pack(side="top", pady=5)

scroll_frame = ctk.CTkScrollableFrame(qb_win, width=580, height=300)
scroll_frame.pack(side="top", fill="x", padx=10, pady=(0, 10))

footer = ctk.CTkFrame(qb_win)
footer.pack(side="bottom", fill="x", pady=5)
ctk.CTkButton(footer, text="Close", command=qb_win.destroy).pack(pady=5)

# Finally, display quizzes initially
refresh_quiz_list()
```

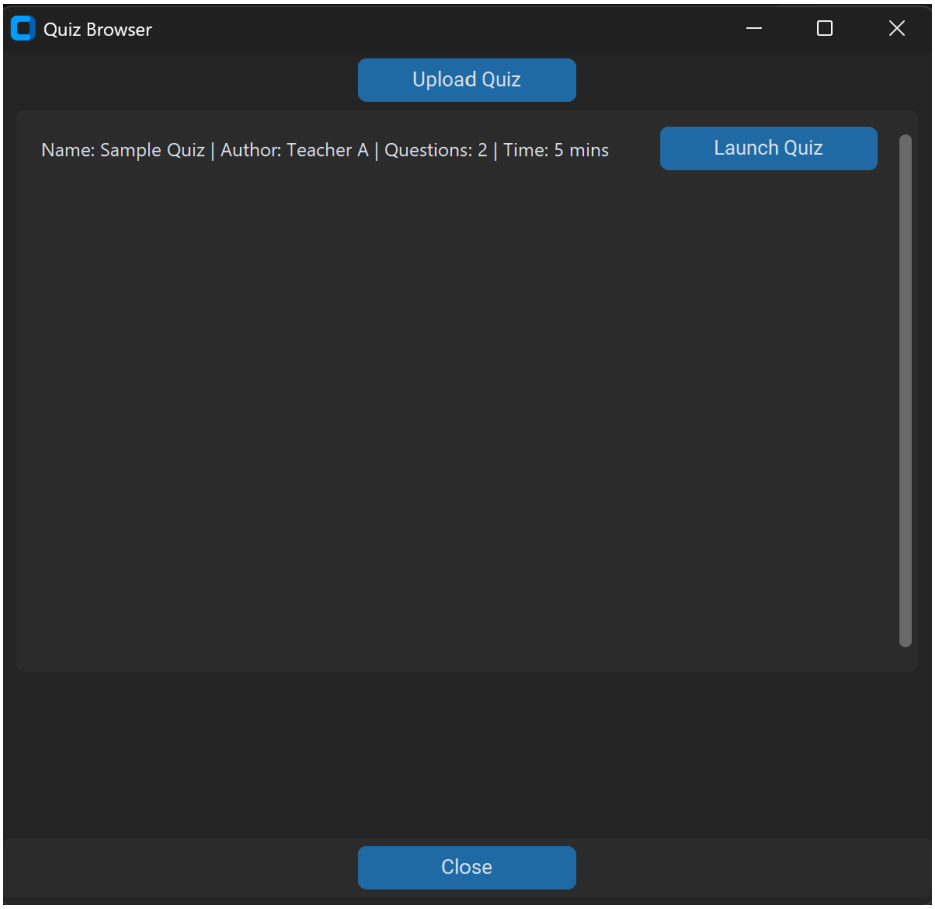
Now with the new updated code, let us test if the issue is resolved and for the successful uploads to be shown:



Even after changing the code and attempting to upload the quiz (and receiving the successful message), the quiz browser has not been updated.

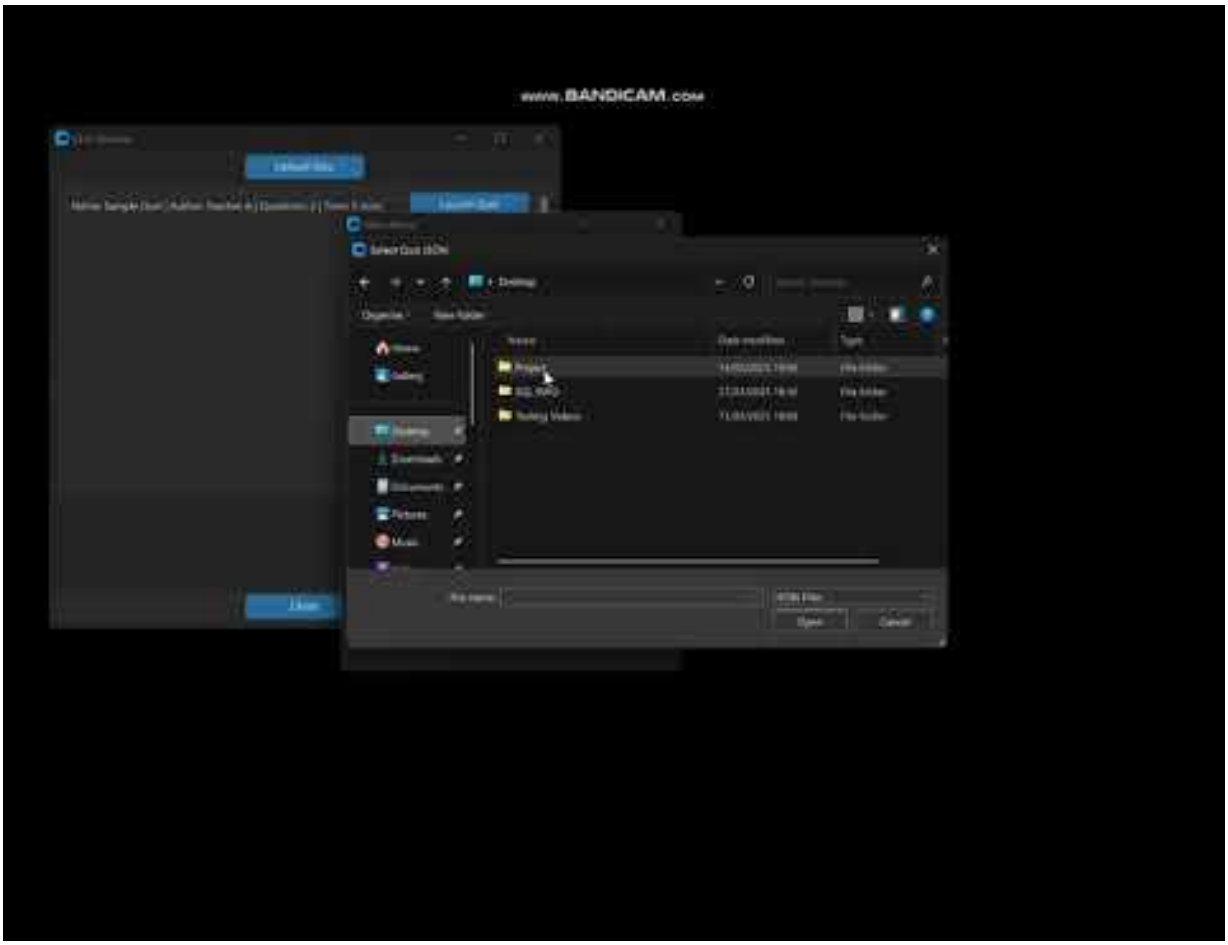
I have accidentally come upon a fix, changing the name of the quiz file stored in my quiz file directory, as I had already stored the said sample quiz file in there before uploading the quiz manually into the program.

After that being done, this is the result:



We can now see that the initial quiz upload was successful, and that the repeated file name was preventing the result from appearing on the quiz browser.

Now to check if a second upload would work:

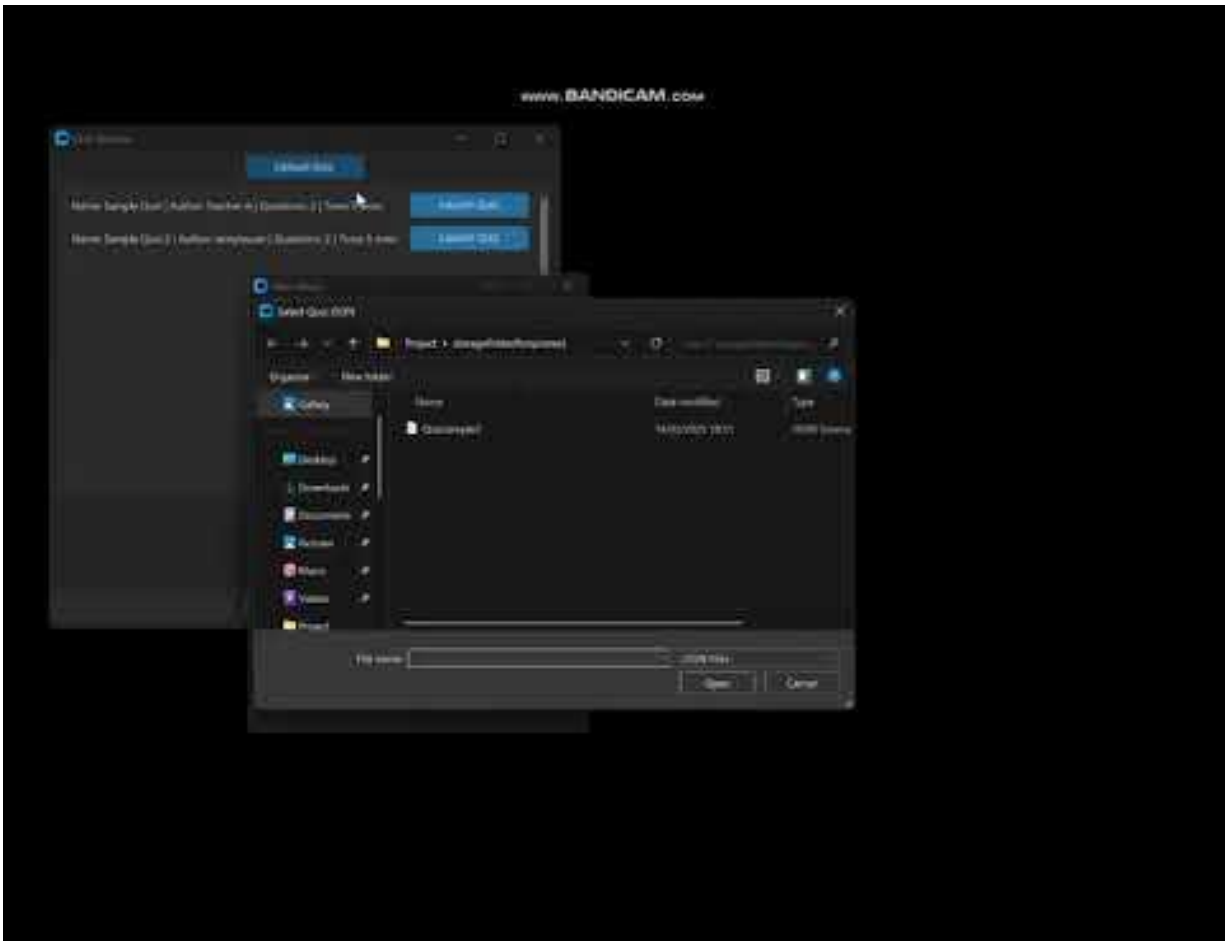


As we can see, after ensuring that quiz names do not repeat, quiz files can be uploaded successfully, making our objectives completed for the time being. The only new update for now that will be made is ensuring that users cannot upload quizzes with repeating file names which would output a successful upload message but not show the results. To resolve this, I will block attempts at a repeated file name upload by also showing an error message to the user, notifying them of the reason for the file rejection.

Adding a simple check into the upload function that changes the code:

```
def upload_quiz():
    """
    Opens a file dialog for the user to select a quiz JSON file, and then copies it to QUIZ_DIR.
    """
    file_path = filedialog.askopenfilename(title="Select Quiz JSON", filetypes=[("JSON Files", "*.json")])
    if file_path:
        dest_path = os.path.join(QUIZ_DIR, os.path.basename(file_path))
        if os.path.exists(dest_path):
            messagebox.showerror("Error", "A quiz with this name already exists!")
```

Now to check if the new feature works as expected:



As can be seen in the video above, attempting to upload a file with the same name rejects validation by the upload_quiz function as desired.

The only new change that needs to occur is for me to create a search engine usable within the quiz browser by users.The “refresh_quiz_list” and and “upload_and_refresh” functions will need to be updated to accommodate the search engine within the Quiz Browser.

Updated “refresh_quiz_list” as well as the new lines of code within:

```
def refresh_quiz_list(search_query=""):
    for widget in scroll_frame.winfo_children():
        widget.destroy()
    quizzes = get_all_quizzes()
    if search_query:
        search_query_lower = search_query.lower()
        quizzes = [q for q in quizzes if search_query_lower in q.get("name", "").lower() or
                   search_query_lower in q.get("author", "").lower()]
    if not quizzes:
        ctk.CTkLabel(scroll_frame, text="No quizzes available.", font=("Segoe UI", 14)).pack(pady=10)
    else:
        for quiz in quizzes:
            quiz_frame = ctk.CTkFrame(scroll_frame)
            quiz_frame.pack(fill="x", pady=5, padx=5)
            info_text = (
                f"Name: {quiz.get('name', 'N/A')} | "
                f"Author: {quiz.get('author', 'N/A')} | "
                f"Questions: {len(quiz.get('questions', []))} | "
                f"Time: {quiz.get('time_limit', 'N/A')} mins"
            )
            ctk.CTkLabel(quiz_frame, text=info_text, font=("Segoe UI", 12)).pack(side="left", padx=5)
            ctk.CTkButton(
                quiz_frame, text="Launch Quiz",
                command=lambda q=quiz: launch_quiz(main_app, q)
            ).pack(side="right", padx=5)
```

```
def refresh_quiz_list(search_query=""):
    for widget in scroll_frame.winfo_children():
        widget.destroy()
    quizzes = get_all_quizzes()
    if search_query:
        search_query_lower = search_query.lower()
        quizzes = [q for q in quizzes if search_query_lower in q.get("name", "").lower() or
                   search_query_lower in q.get("author", "").lower()]
    if not quizzes:
        ctk.CTkLabel(scroll_frame, text="No quizzes available.", font=("Segoe UI", 14)).pack(pady=10)
```

Updated “upload_and_refresh” function and the new lines of code:

```
    def upload_and_refresh():
        upload_quiz()
        refresh_quiz_list(search_entry.get().strip())

    # Top: Upload Quiz button
    upload_btn = ctk.CTkButton(qb_win, text="Upload Quiz", command=upload_and_refresh)
    upload_btn.pack(side="top", pady=5)

    # Search frame below upload button.
    search_frame = ctk.CTkFrame(qb_win)
    search_frame.pack(side="top", fill="x", pady=5, padx=10)
    search_entry = ctk.CTkEntry(search_frame, placeholder_text="Search by quiz name or author")
    search_entry.pack(side="left", fill="x", expand=True, padx=(0, 5))
    ctk.CTkButton(search_frame, text="Search", command=lambda: refresh_quiz_list(search_entry.get().strip())).pack(side="left")

    # Middle: Fixed-height scrollable frame
    scroll_frame = ctk.CTkScrollableFrame(qb_win, width=580, height=350)
    scroll_frame.pack(side="top", fill="x", padx=10, pady=(0,10))

    # Bottom: Footer with only the Close button
    footer = ctk.CTkFrame(qb_win)
    footer.pack(side="bottom", fill="x", pady=5)
    ctk.CTkButton(footer, text="Close", command=qb_win.destroy).pack(side="right", padx=5, pady=5)

    refresh_quiz_list()
```

```
refresh_quiz_list(search_entry.get().strip())
```

```
# Search frame below upload button.
search_frame = ctk.CTkFrame(qb_win)
search_frame.pack(side="top", fill="x", pady=5, padx=10)
search_entry = ctk.CTkEntry(search_frame, placeholder_text="Search by quiz name or author")
search_entry.pack(side="left", fill="x", expand=True, padx=(0, 5))
ctk.CTkButton(search_frame, text="Search", command=lambda: refresh_quiz_list(search_entry.get().strip())).pack(side="left")
```

Search Frame:

A new frame is added below the "Upload Quiz" button. It contains an entry widget where users can type a search term and a "Search" button. When the "Search" button is clicked, it calls ”refresh_quiz_list(search_entry.get().strip())” to re-populate the quiz list based on the query.

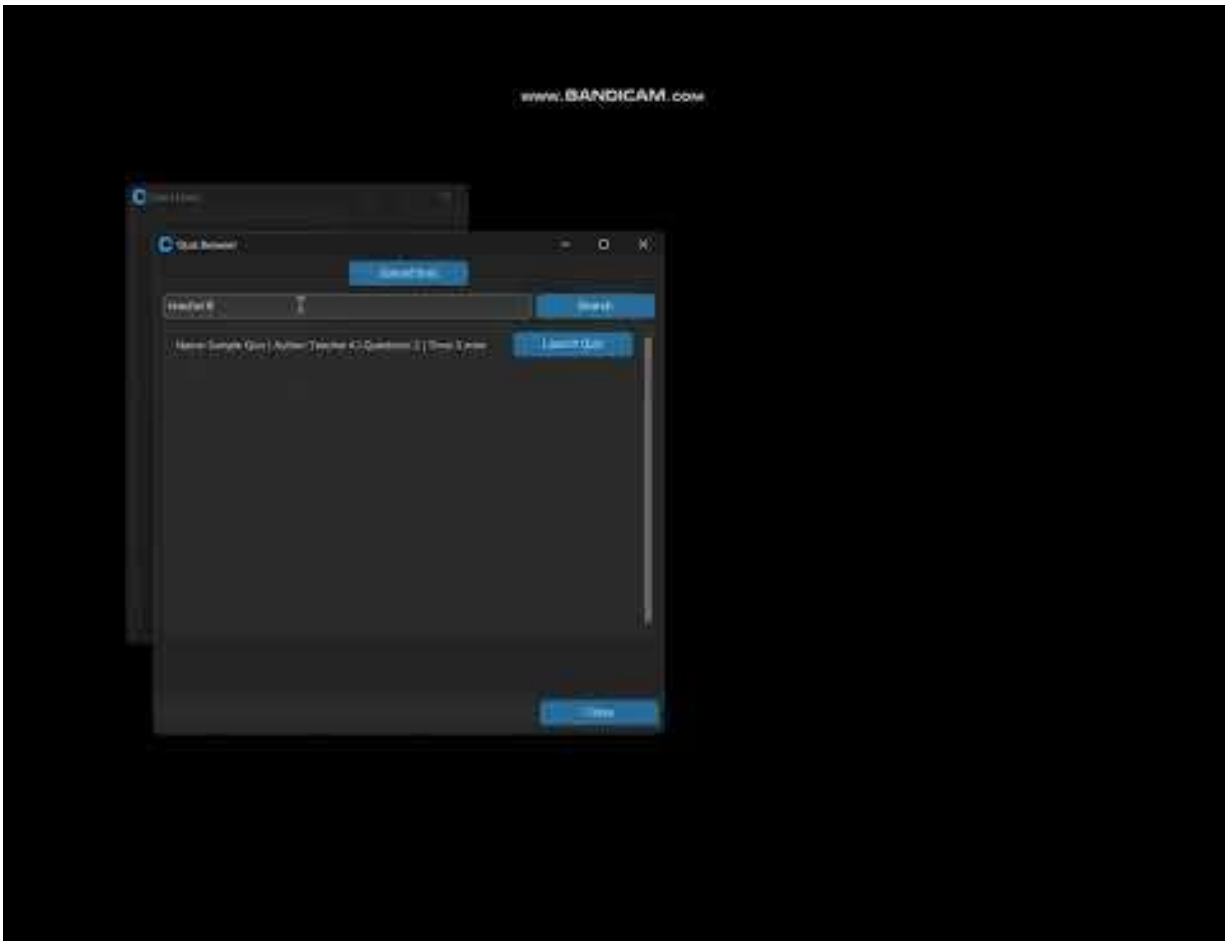
Regarding the “refresh_quiz_list” function:

This function clears the scrollable frame and calls on “get_all_quizzes”, so that when a search query is provided it will filter the list so that only quizzes whose name or author contains the query is displayed.

Regarding the “upload_and_refresh” function:

When a new quiz is uploaded, “upload_quiz” is called and refreshes the list, passing the search query and keeping the filtered view maintained.

Now to test whether the search engine works as expected:



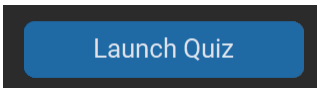
As we can see, the search engine is working well, and will be a key quality of life feature for users to navigate the quiz browser for specific quizzes by or quiz name or author name.

We can now move on to the next module and class the Quiz Browser a success, as it shows past uploaded quizzes and allows upload of new quizzes, where all the results show their author, number of questions, time allotted and name in a scrollable format(which meets the test data for the success of this module) while also not allowing files with a repeated name to be submitted.

Quiz Launching and Execution

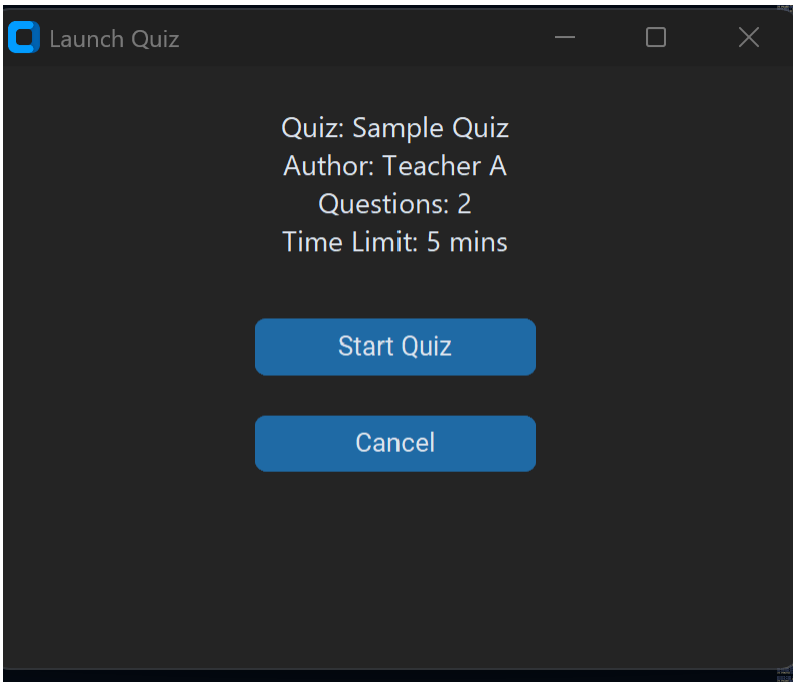
The quiz launching and quiz executing modules will both operate within the Quiz Browser window and can only be accessed or executed within the Quiz Browser module.

```python
def launch_quiz(main_app, quiz):
    launch_win = ctk.CTkToplevel(main_app)
    launch_win.title("Launch Quiz")
    launch_win.geometry("400x300")
    info_text = (f"Quiz: {quiz.get('name', 'N/A')}\n"
                 f"Author: {quiz.get('author', 'N/A')}\n"
                 f"Questions: {len(quiz.get('questions', []))}\n"
                 f"Time Limit: {quiz.get('time_limit', 'N/A')} mins")
    ctk.CTkLabel(launch_win, text=info_text, font=("Segoe UI", 14)).pack(pady=20)
    ctk.CTkButton(launch_win, text="Start Quiz", command=lambda: [launch_win.destroy(), execute_quiz(main_app, quiz)]).pack(pady=10)
    ctk.CTkButton(launch_win, text="Cancel", command=launch_win.destroy).pack(pady=10)
```



The function above creates the previously seen “Launch Quiz” button within the quiz browser (as also seen above).

Now to test whether the launch quiz button works:



Above is the output of clicking, as we can see the code is working as expected. The quiz name, author,number of questions and time allotted can all be seen alongside the “Start Quiz” and “Cancel” buttons which will be linked to the quiz execution module in a moment.

To continue I need to create a function which executes quizzes when the “Start Quiz” button is clicked.To do this, I would need for the “execute_quiz” function that will be made to launch another window upon launching, which is where the questions(multiple choice and fill in answers) will be answered by the user, with the results recorded and stored in the past results tab for each user.

“Execute quiz” function:

```python
def execute_quiz(main_app, quiz):
    exec_win = ctk.CTkToplevel(main_app)
    exec_win.title("Quiz Execution")
    exec_win.geometry("600x500")

    questions = quiz.get("questions", [])
    total_questions = len(questions)
    user_answers = {}
    current_q_index = [0]  # Mutable holder for current index

    time_limit_minutes = quiz.get("time_limit", 0)
    total_time_sec = int(time_limit_minutes * 60) if time_limit_minutes else 0
    start_time = datetime.datetime.now()

    timer_label = ctk.CTkLabel(exec_win, text="", font=("Segoe UI", 14))
    timer_label.pack(pady=5)
```

As can be seen above, the code will set up the “Quiz Execution” window, retrieve the list of questions from the quiz and sets up a timer to determine if the current launched quiz is timed and if so, how much time should be set for the quiz(by checking the JSON file itself) and finally a label is created to display the time allotted.

To make sure that the timer method works properly, I need a nested function that continuously updates the timer in a launched quiz.

```python
def update_timer():
    if total_time_sec > 0:
        elapsed = (datetime.datetime.now() - start_time).total_seconds()
        remaining = max(0, total_time_sec - int(elapsed))
        mins, secs = divmod(remaining, 60)
        timer_label.configure(text=f"Time Remaining: {int(mins):02d}:{int(secs):02d}")
        if remaining <= 0:
            finish_quiz()
        else:
            exec_win.after(1000, update_timer)
if total_time_sec > 0:
    update_timer()
```

The code above updates the timer label every second, calculating how much time has elapsed so far per each second and converting the remaining seconds into minutes and seconds, then updating the “timer_label” into a formatted string that is shown to the user.Importantly, if the time allotted in the timer runs out , it will automatically call the “finish_quiz” function and end the quiz(otherwise it schedules to run again in the next second).

Next, I need to create a content area that holds the main contents of the quiz(questions,options,images) as well as creating widgets that display the current question and answer options.

```
content_frame = ctk.CTkFrame(exec_win)
content_frame.pack(fill="both", expand=True, padx=10, pady=10)

image_label = None

question_label = ctk.CTkLabel(content_frame, text="", font=("Segoe UI", 16))
question_label.pack(pady=10)

options_frame = ctk.CTkFrame(content_frame)
options_frame.pack(pady=10)

answer_var = ctk.StringVar()
```

For the time being “image_label” is set to “None”, for use of images in the questions it will need to later be updated with the reference to an image widget if a question contains an image.

Next, a nested function which displays questions needs to be made, where the current question is displayed (and an image, if provided to the question) and the available answer options.

```
def display_question(index):
    nonlocal image_label
    if image_label is not None:
        image_label.destroy()
        image_label = None

    if index < total_questions:
        q = questions[index]
        question_text = q.get("question", "No question text provided.")
        question_label.configure(text=question_text)
        for widget in options_frame.winfo_children():
            widget.destroy()
        if "image" in q and q["image"]:
            try:
                img = Image.open(q["image"])
                max_width = 400
                if img.width > max_width:
                    ratio = max_width / img.width
                    img = img.resize((max_width, int(img.height * ratio)), Image.ANTIALIAS)
                photo = ImageTk.PhotoImage(img)
                image_label = ctk.CTkLabel(content_frame, image=photo, text="")
                image_label.image = photo
                image_label.pack(pady=5)
            except Exception as e:
                print(f"Error loading image: {e}")
        options = q.get("options", [])
        if options:
            answer_var.set("")
            for opt in options:
                ctk.CTkRadioButton(options_frame, text=opt, variable=answer_var, value=opt).pack(anchor="w", pady=2)
        else:
            entry = ctk.CTkEntry(options_frame, placeholder_text="Your answer here")
            entry.pack(fill="x", pady=2)
            options_frame.entry = entry
    else:
        finish_quiz()
```

The code retrieves the text of the current question and updates “question_label”, then clears any existing answer options from “options_frame” from whenceforth it creates radio buttons for multiple-choice questions and/or entry widgets for open-ended questions. If an image was

displayed previously (previous question) it is destroyed to clear space and if the current question has an" image", the code attempts to open and resize the image before displaying it.

The next step is a nested function that goes to the next question once one has already been answered:

```
def next_question():
    index = current_q_index[0]
    q = questions[index]
    if q.get("options", []):
        user_answers[index] = answer_var.get()
    else:
        user_answers[index] = options_frame.entry.get()
    current_q_index[0] += 1
    display_question(current_q_index[0])
```

The code retrieves the current question, checks if it is multiple-choice or open-ended and stores the answer in "user_answers" dictionary using the current index as the key. From there it increments the index and calls "display_question" with the new index to update the window with the next question.

To finish off, I need one last nested function that ends the quiz session, calculates the score, stores the result and destroys the window:

```
def finish_quiz():
    # Ensure the answer of the current question is stored, if not already.
    if current_q_index[0] < total_questions:
        q = questions[current_q_index[0]]
        if q.get("options", []):
            user_answers[current_q_index[0]] = answer_var.get()
        else:
            user_answers[current_q_index[0]] = options_frame.entry.get()
    correct_count = 0
    for i, q in enumerate(questions):
        if user_answers.get(i, "").strip().lower() == q.get("answer", "").strip().lower():
            correct_count += 1
    result_data = {"correct_count": correct_count, "total_questions": total_questions}
    record_quiz_result(main_app.current_user["id"], json.dumps(result_data), total_questions)
    messagebox.showinfo("Quiz Completed", f"You scored {correct_count} out of {total_questions}. Your result has been stored in Past Results.")
    exec_win.destroy()
```

Before finishing the quiz, the code above checks whether the current question's answer is recorded (in case the user did not click on" Next") and then stores the answer. The code iterates over all the questions and compares the user's answers (which are converted to lowercase and their whitespace is removed) to the correct answer stored in the corresponding quiz file, incrementing the "correct_count" counter by 1 for each match. It then creates a dictionary with the correct count and total questions, converting the dictionary into a JSON string and calling on "record_quiz_result()" to store it in the database. Before destroying the quiz window, a message is displayed notifying the user of their score.

```
nav_frame = ctk.CTkFrame(exec_win)
nav_frame.pack(fill="x", pady=10)
ctk.CTkButton(nav_frame, text="Next", command=next_question).pack(side="right", padx=5)
ctk.CTkButton(nav_frame, text="Finish", command=finish_quiz).pack(side="right", padx=5)

display_question(0)
```

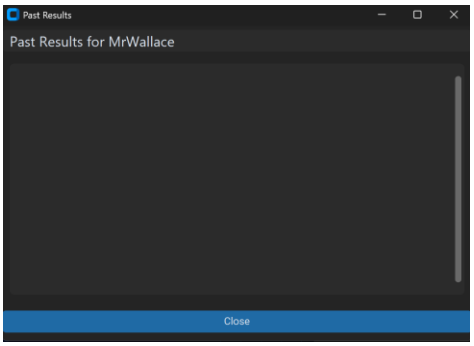
Finally, a frame is created that holds the “Next” and “Finish” buttons and the quiz is started by displaying the first question.

As the development part is finished, we can now move onto testing the Quiz Launching and Quiz execution modules.

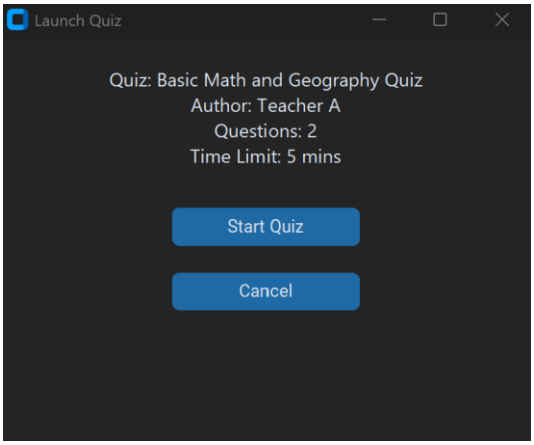
For testing if an image loads when present in a question, I will use the JSON quiz file below in testing:

```
{
  "id": "quiz1",
  "name": "Basic Math and Geography Quiz",
  "author": "Teacher A",
  "time_limit": 5,
  "questions": [
    {
      "question": "What is 2 + 2?",
      "options": ["3", "4", "5", "6"],
      "answer": "4"
    },
    {
      "question": "Identify the capital of France by looking at the image.",
      "options": [],
      "answer": "Paris",
      "image": "C:\\Users\\totin\\OneDrive\\Работен плот\\Project\\Images\\paris.jpg"
    }
  ]
}
```

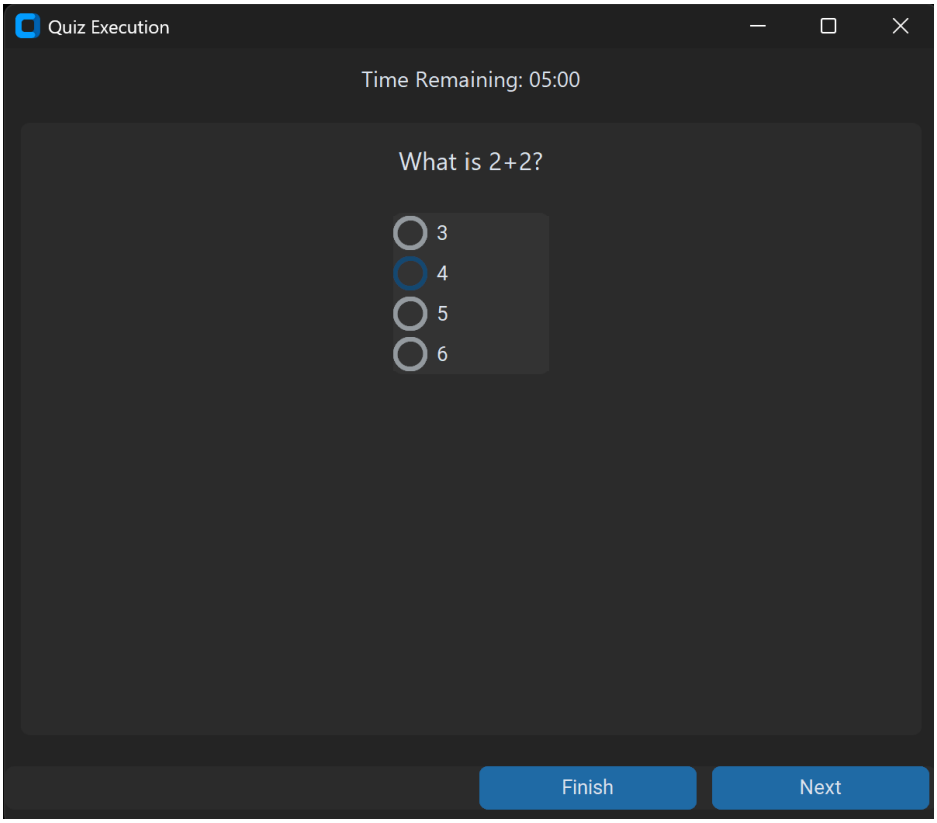
Results (empty) before testing with “MrWallace” teacher account:



Now firstly I will test whether a quiz can be started from the previously seen “Start Quiz” button:

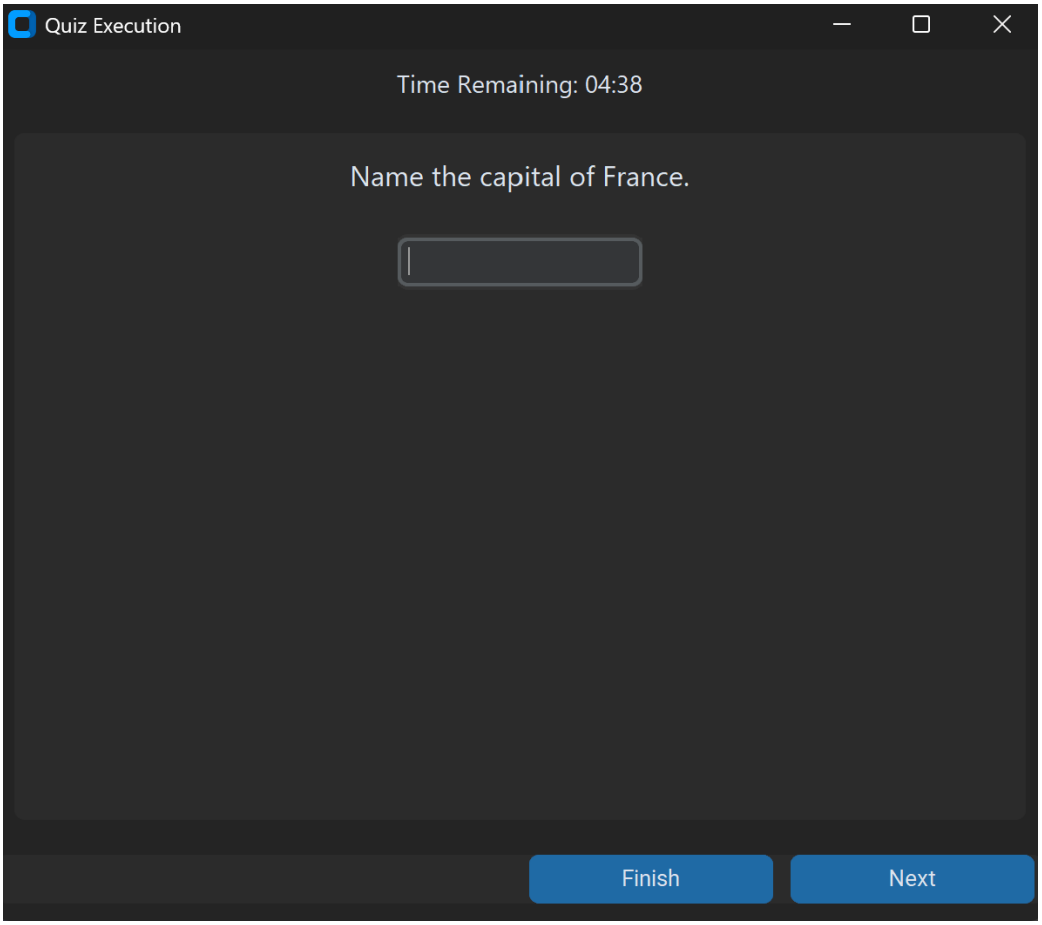


Result when pressing the “Start Quiz” button:

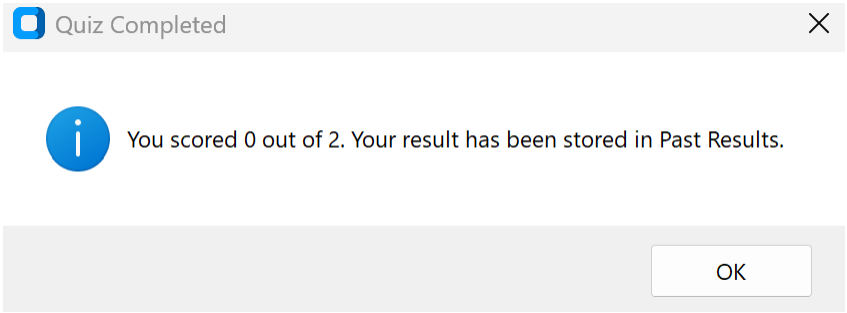


The first question loads upon starting the quiz, and the timer begins counting down, as expected.

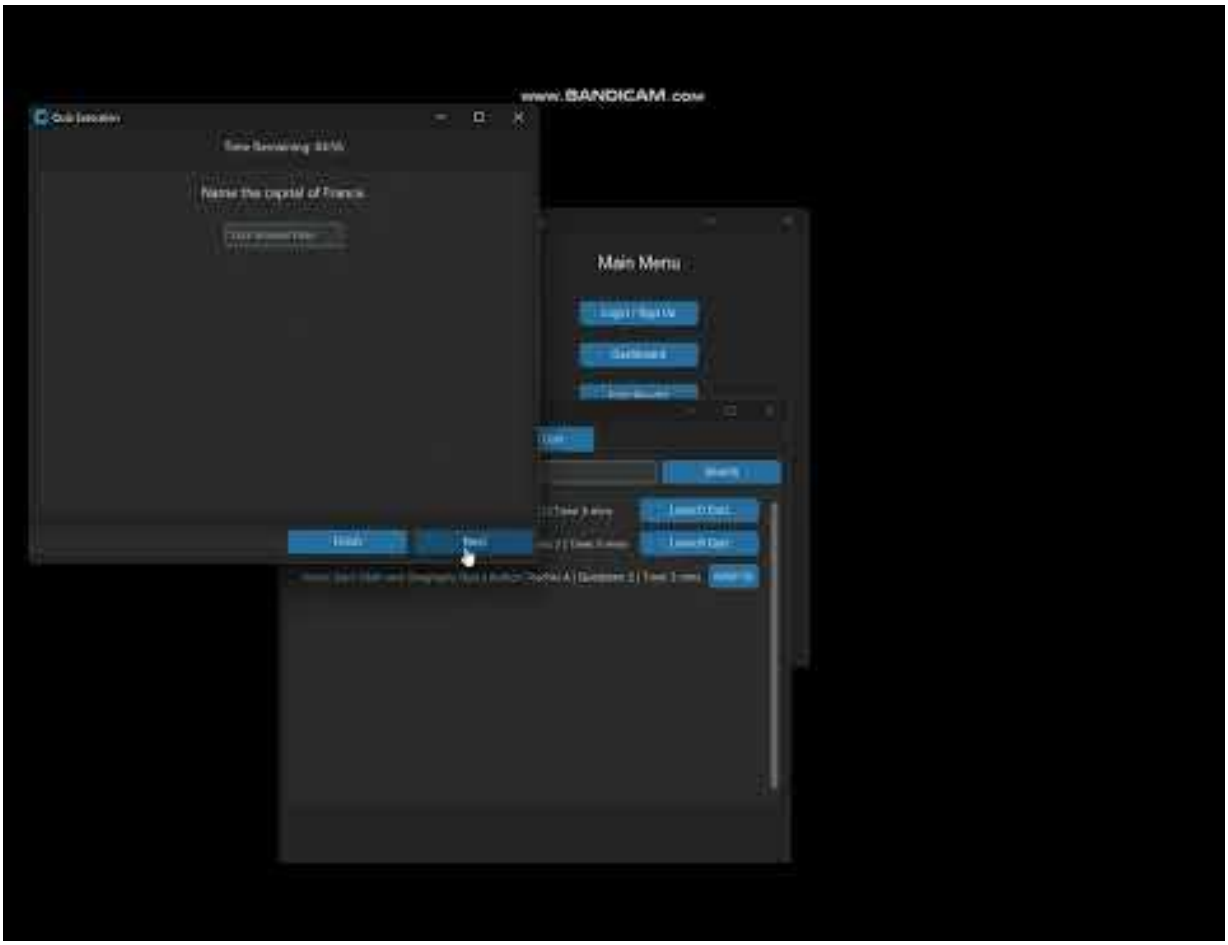
Now to see the result upon answering the question (correctly or incorrectly) and going to the next question:



Result of answering both questions incorrectly upon pressing the “Next” or “Finish” button after answering the last question:



Now I will upload a video where the previous past result from these screenshots is seen, as well as testing if inputting the correct answers would give the correct result (which it has as 2/2 correctly answered questions) and then storing the new result:



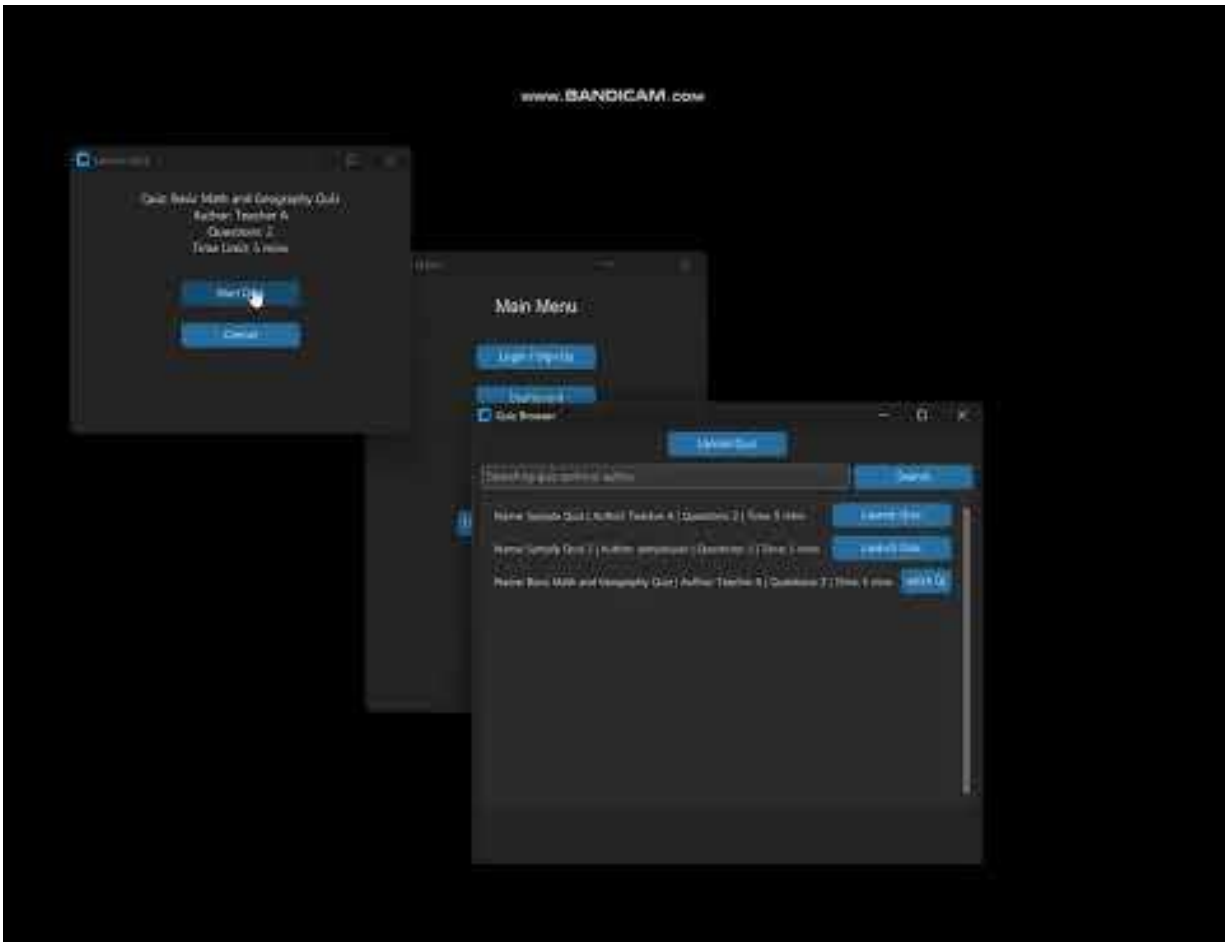
Now I will try running a quiz with an image file attached to it.The quiz is stored in a JSON format as required and seen below:


```json
{
  "id": "quiz1",
  "name": "Basic Math and Geography Quiz",
  "author": "Teacher A",
  "time_limit": 5,
  "questions": [
    {
      "question": "What is 2 + 2?",
      "options": ["3", "4", "5", "6"],
      "answer": "4"
    },
    {
      "question": "Identify the capital of France by looking at the image.",
      "options": [],
      "answer": "Paris",
      "image": "C:\\Users\\totin\\OneDrive\\Работен плот\\Project\\Images\\paris.jpg"
    }
  ]
}
```


This is the image that will be used:

As I launch the “Basic Math and Geography Quiz”, an image file should be shown after the 2nd question is loaded (which will be retrieved from the directory shown above).

Now to test whether this works:



As can be seen in the video, the loading of the image has failed. After checking the terminal in Visual Studio code, I have realised that the issue is caused by using an invalid constant in “ANTIALIAS” (see below) in line 521 of my code (also seen below).

```
Error loading image: module 'PIL.Image' has no attribute 'ANTIALIAS'
```

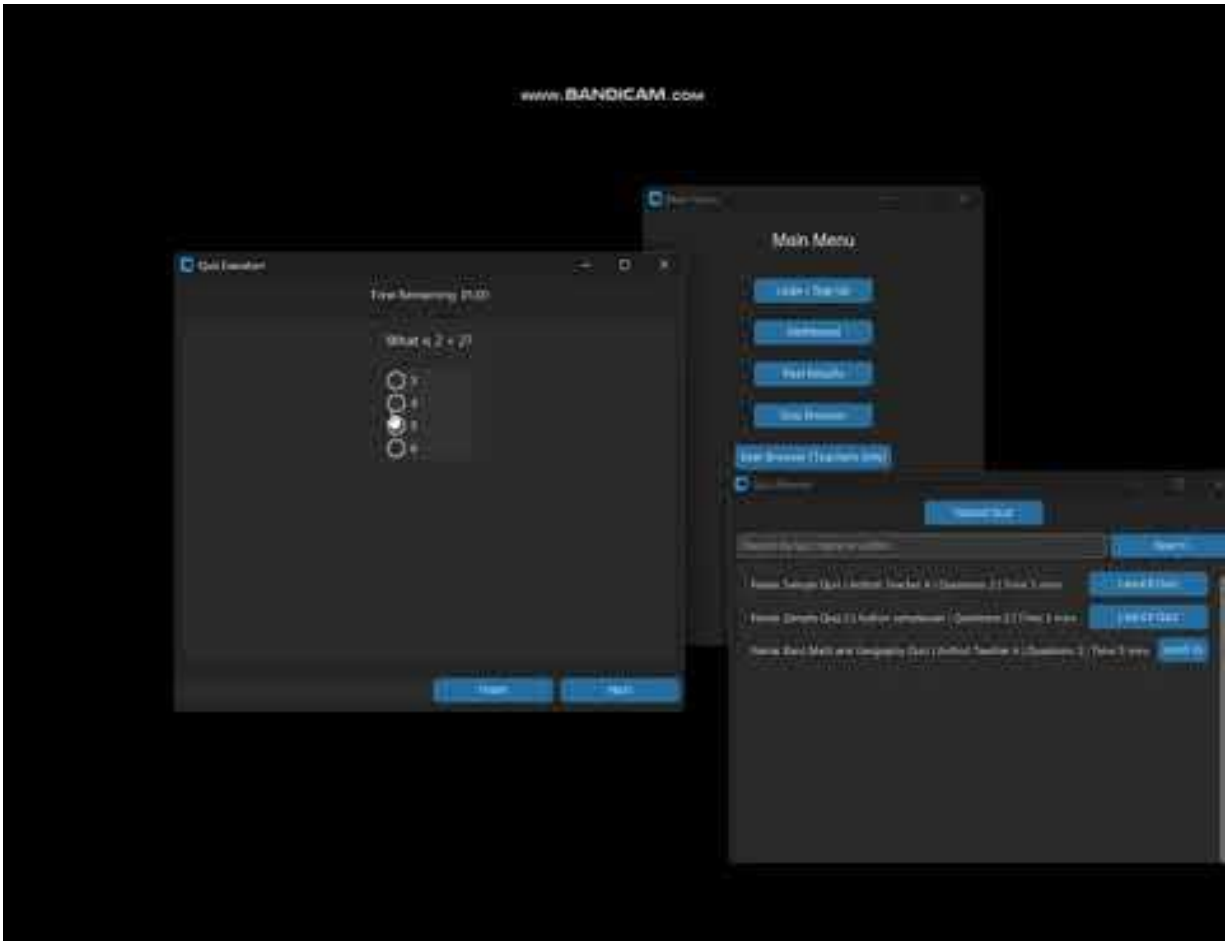
```
img = img.resize((max_width, int(img.height * ratio)), Image.ANTIALIAS)
```

The issue seems to be that the constant “ANTIALIAS” has been removed from the “PIL.image” module for some reason, and after checking online it seems that the module no longer has the “ANTIALIAS” attribute in newer versions and has been completely removed. After researching the Pillow library and documentation further, I have discovered another attribute, a resampling filter(“Image.Resampling.LANCZOS”) that should rescale the image and show it to the user in Question 2 of the quiz from now on and fix the error in the terminal.

Updated code in line 521:

```
img = img.resize((max_width, int(img.height * ratio)), Image.Resampling.LANCZOS)
```

With the incorporation of this change, the image should now generate, and the quiz should complete as intended.I will test this and show the result down below:



As can be seen in the video above, the image launching now works as expected and this test has been a pass.

Before moving on, I want to state that originally, I planned to create a “Results” module at the end of the development, however for convenience I created the “Past results” module that

completes the tasks of both and hence is not needed anymore as a separate module.With this in mind, I can conclude the development and testing.

Post-Development Justification

As the application's coding has concluded, I will now justify the development of each module while referring to the analysis throughout.

To begin, since the database and its connections are crucial to the program, having a module that allows for querying of the database has been essential. This can be seen throughout the code by the use of "connect ()" for every function that queries or updates the database.

The login and sign-up modules as well as their respective pages are necessary for the users to give appropriate input to query the database credentials or creating accounts.Accounts are necessary to the program as otherwise quiz results would not be saved for specific users and no sections apart from the login/signup windows will be able to be accessed without an account.This would defeat the purpose of the whole project as the need for specific users to have their quiz results data saved and analytic data presented is essential.

The purpose of the dashboard module has been to show users their analytical data in a clear and concise manner, as well as to store any results and information for specific users from the Past Results module and then graphing the data to showcase said analytical data.

The purpose of the past results module is linked to the dashboard and allows users to view all their past results in detail. This is essential as users need a way to compare their current results to their previous results and hence later to their future results and track their learning progress.

The purpose of the user browser is to allow users that have Teacher accounts to search for any student account in the system and access all their past results, which could be used by teachers to assess the level that each student is currently working at and assess their future progress.

The purpose of the manage account window alongside the change username and change password modules is to provide users the option of updating their credentials post-account creation.As this has been implemented in the main window with connectivity to its own specific module(s),the administrator does not need to be involved in the process at all due to the multiple checks and requirements around changing user credentials. This helps the solution by having a more decentralised approach to changes in credentials which give greater autonomy to the users in how they change their account details.

The purpose of the quiz browser is to allow users to browse through available quizzes stored in the quiz folder, while also providing the option of uploading their quizzes directly into the quiz folder without the need for an administrator to manage this. Additionally, the quiz folder has a fixed file path which can be stored on the shared network drive of the school server and be accessible to all users from where they can access the quizzes.

The purpose of the quiz launching module is to load the contents of the JSON format quiz files linked to the quiz launch button when a user clicks on it and the information of said quiz will be displayed to the user. This is as otherwise users wouldn't have information on the quiz that they are about to take or if it is even the correct quiz that they are about to take, hence this prevents the user from selecting the wrong quiz. As this information is provided within the quiz file itself, the creator of the quiz can provide the necessary information for a user to answer the questions within the quiz.

The purpose of the quiz execution module is to allow users to view questions and either select or fill in an answer which are marked against the correct answers in the database, which are then stored in the past results module. This module is probably the most crucial out of any other module as they allow users to perform the quizzes as well as storing the results of the quizzes in the past results module. The only exception to this would be the Simulate Quiz run module which was not originally a part of the success criteria but has been a useful tool for me in testing for generating sample results (however, this module is not necessarily intended for the users to use, except for potential self-testing of the analytics within the dashboard).

In general, each module, function and page mentioned above and referenced throughout the project has been created based on the stakeholder requirements, as well as including features from preexisting solutions that have been mentioned in the students interviews and surveys, with not only a goal of achieving the stakeholder requirements but also increasing accessibility to users and maintainability.

Evaluation

Post developmental testing guideline

For the post-development testing, no referrals will be made to the database or console as only the running application will be accessible by the users. The UI should display sufficient information to the user in a manner that no troubleshooting is necessary other than error messages due to erroneous input and handling messages.

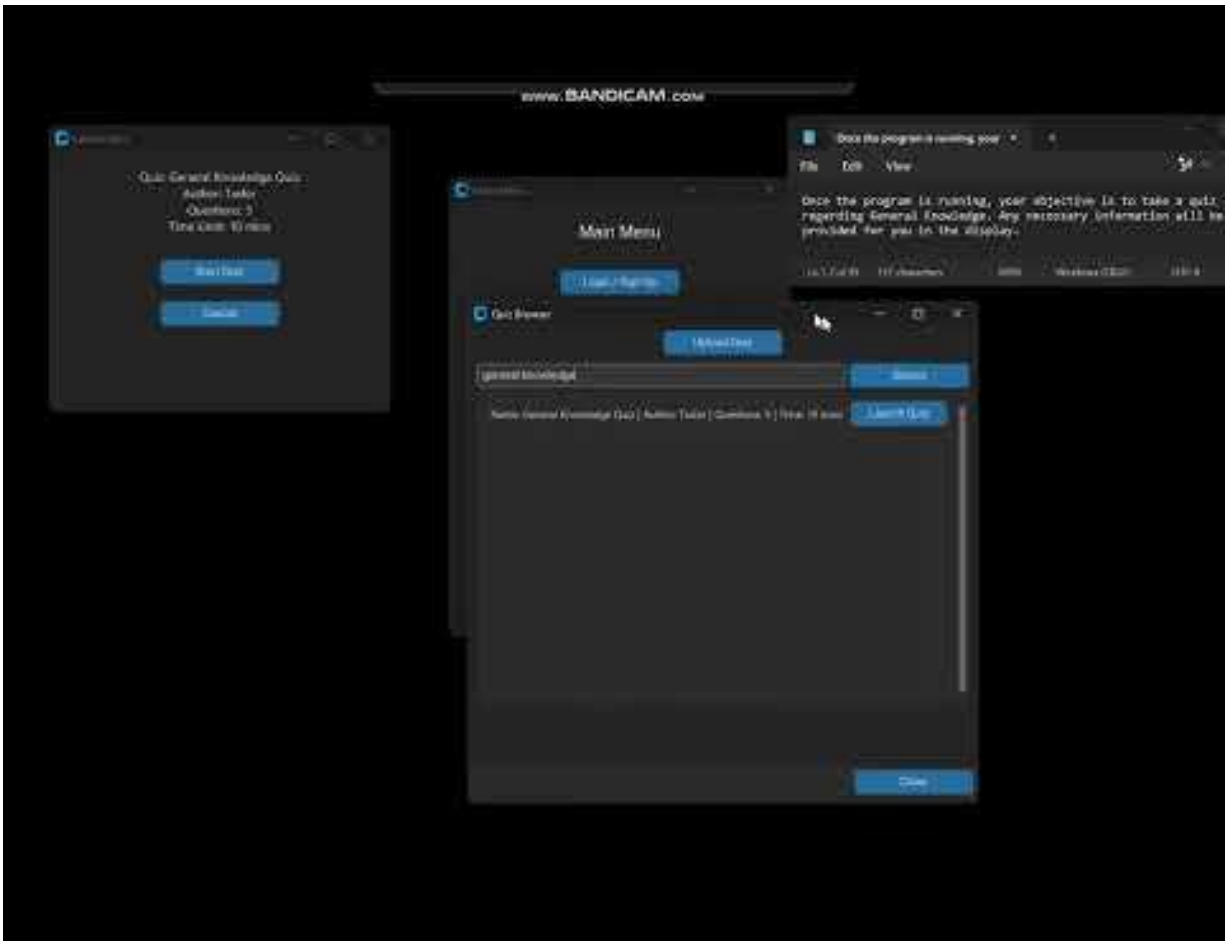
In this post-developmental testing, I will conduct two different tests: A test for functionality and usability of the program and a test for robustness of the program. I will use usability and functionality with a user that has no prior knowledge of the program or the code, other than the instructions given within the actual application.Afterwards I will test for robustness with the quiz files provided. At the end I will link everything back to the success criteria to determine what was successful and what was unsuccessful.

Testing data for functionality and usability after development


```
{
  "id": "quiz2",
  "name": "General Knowledge Quiz",
  "author": "Todor",
  "time_limit": 10,
  "questions": [
    {
      "question": "What is the capital of France?",
      "options": ["London", "Berlin", "Paris", "Madrid"],
      "answer": "Paris"
    },
    {
      "question": "What is 5 multiplied by 6?",
      "options": ["11", "30", "35", "60"],
      "answer": "30"
    },
    {
      "question": "Which planet is known as the Red Planet?",
      "options": ["Earth", "Mars", "Jupiter", "Saturn"],
      "answer": "Mars"
    },
    {
      "question": "What language is primarily used for Android development?",
      "options": ["Swift", "Kotlin", "Python", "Ruby"],
      "answer": "Kotlin"
    },
    {
      "question": "Look at the image and name the landmark shown.",
      "options": [],
      "answer": "Eiffel Tower",
      "image": "C:\\Users\\totin\\OneDrive\\Работен плот\\Project\\Images\\eiffeltower.jpg"
    }
  ]
}
```


Result of post developmental functionality and usability testing

The following video is from functionality and usability testing with a third party, my classmate Ewan Munday. He was given the following information in a notepad window as seen in the video: "Once the program is running, your objective is to take a quiz regarding General Knowledge. Any necessary information will be provided for you in the display.”



The video above demonstrates that by using only the prompt provided in the notepad, Ewan successfully created an account, logged in, launched the correct quiz, completed the correct test and achieved 80% on the quiz. Since just the notepad description was used by Ewan, this should indicate that this program is very usable to somebody with no prior knowledge of it.Even if any issues did arise (such as him trying to log in without creating an account first), appropriate error message windows would pop up and tell the user what is wrong and indicate to them what they should do. Hence, we can class this black box test a success.

Next, I will be conducting the robustness testing. This will be done by me personally as I will attempt to crash the program and/or make it malfunction in different possible ways. I will use the quiz files seen below in the testing while also referring to the testing data seen below as well.

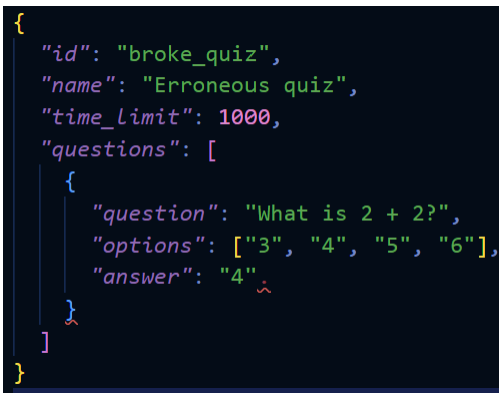
Testing data for Robustness after development

Function	Testing data for the quiz application
Login	invalid password, empty username or password field. Correct username and password with incorrect case (upper/lower).
Sign up	Invalid creation with used username. Invalid creation with weak password (under 8 characters)
User browser	Multiple users in database table to search through.
Quiz browser	Empty quiz folder. Invalid quiz files (e.g. incorrectly structured Json with a missing bracket, no quiz name)
Change username	Valid username change with correct current password. Invalid new username (already in use)
Change password	Valid password change with repeated confirmation. Weak new password (under 6 characters). Strong new password that doesn't match new password confirmation.
Quiz launch screen	Quiz data along with quiz properties such as name, ID, time allowed , author.
Quiz execution	Long questions, varying number of options, invalid image paths. Missing image files.

Quiz files for robustness testing

The following three files will be used for robustness testing:

1)



2)

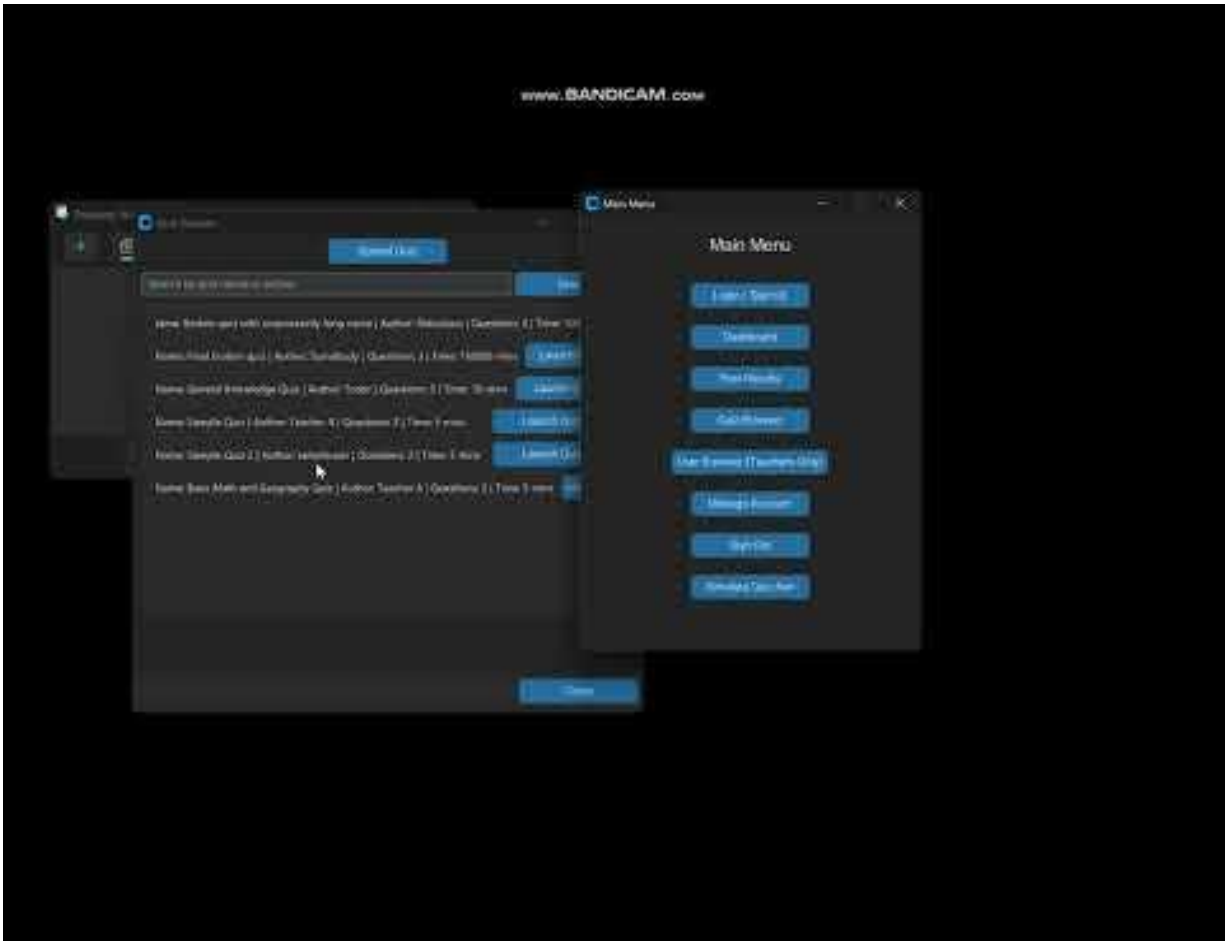
```
{
    "name": "Broken quiz with uneccesairily long name",
    "author": "Ridiculous",
    "time_limit": 10000,
    "questions": [
      {
        "question": "What is the capital of France?",
        "options": ["London", "Berlin", "Paris", "Madrid"],
        "answer": "Paris"
      },
      {
        "question": "What is 5 multiplied by 7?",
        "options": ["11", "30", "35", "60"],
        "answer": "35"
      },

      {
        "question": "Look at the image and name the landmark shown.",
        "options": [],
        "answer": "Eiffel Tower",
        "image": "./nopathanywhere"
      }
    ]
  }
```

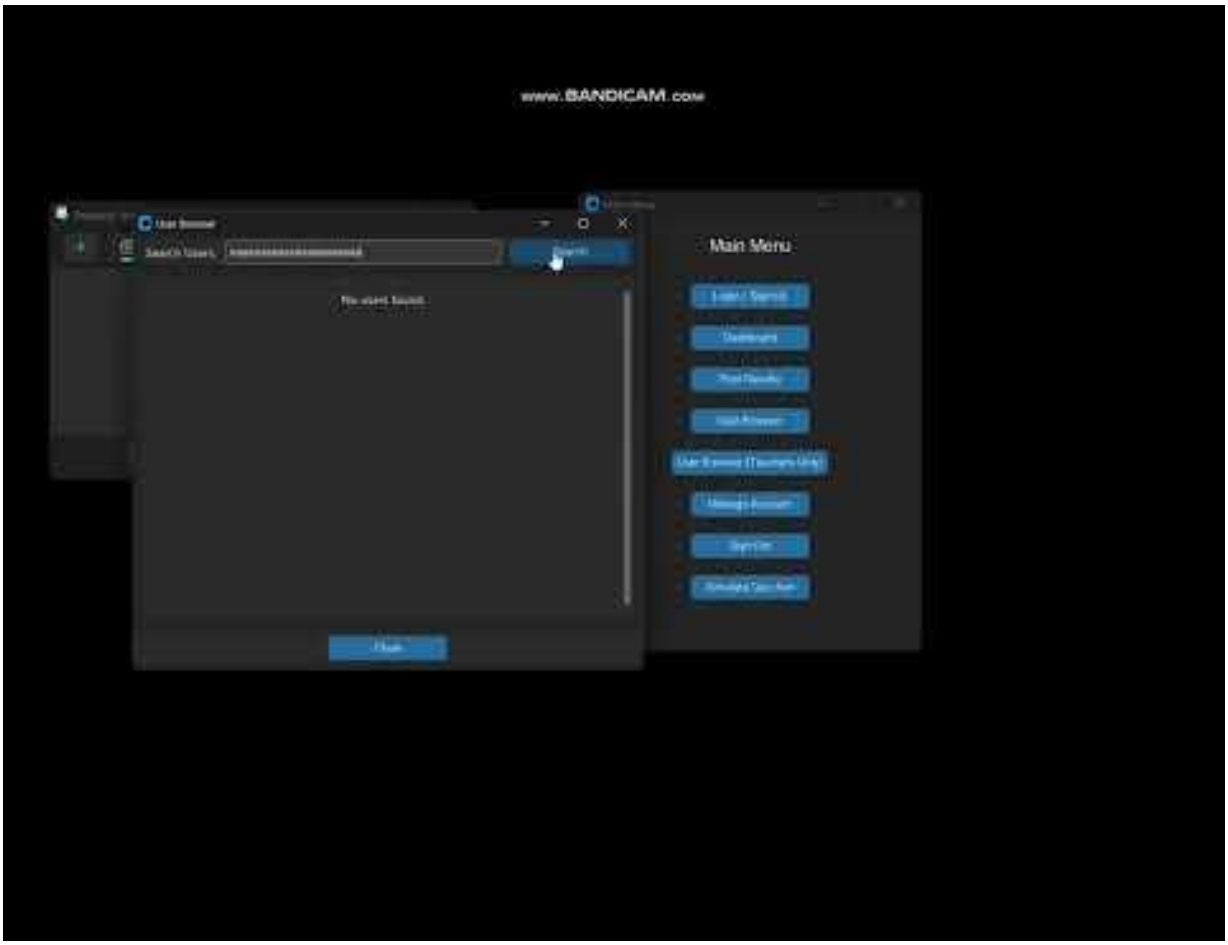

3)


```
{    "id":"1239",
    "name": "Final broken quiz",
    "author":"Somebody",
    "time_limit": 150000,
    "questions": [
      {
        "question": "What is the chemical symbol for water?",
        "options": ["H2O", "O2", "CO2", "HO"],
        "answer": "H2O"
      },
      {
        "question": "What planet is known as the Blue Planet?",
        "options": ["Earth", "Neptune", "Uranus", "Saturn"],
        "answer": "Earth"
      },

      {
        "question": "Identify the rock type shown in the image.",
        "options": ["Sedimentary", "Igneous", "Metamorphic", "All of the above"],
        "answer": "Metamorphic",
        "image": "invalid/path//"
      }
    ]
  }
```


Each of these erroneous quizzes have details such as few entries, Json structure errors, missing information like id and author.

Result of post developmental robustness testing





The inserted videos above showcases me testing the robustness of the program against the testing data. Every valid and invalid input and output was handled successfully and as expected in the testing.

Function	Pass or fail
Login	pass
Sign up	pass
User browser	pass
Quiz browser	pass
Change username	pass
Change password	pass
Quiz launching	pass
Quiz execution	pass

Evalutation of success criteria:

Start-up window success criteria which have been met (All):

- Has a sign-in button that launches the main menu button upon successful entry.
- Have a sign-up that will refer users to the sign-in page.
- Has a username and obfuscates password upon entry (allows showing and hiding of the password).
- Database will be queried for valid logins and will go back to the sign-in window if invalid.

Sign-up page success criterion which has been met (All):

- Clearly displays a sign-up feature for users and allows users to create either a teacher or student account.
- The username is between 5 and 16 characters.
- The database does not already contain the username.
- The password contains a minimum of 6 characters.
- The password that has been entered must match with the database.
- The username and password are not the same.

Main menu success criterion which has been met (All):

- Contains buttons that redirect users to a dashboard/quiz browser/account manager/log-out window, while teacher accounts will have access to a user search.

Quiz browser success criterion which has been met (All):

- All the quizzes are presented in a scrollable format
- Each test should display its author, date of creation/last edit, name.

Quiz launcher success criterion which has been met (All):

- Presents the name of the quiz, author name and ID code.
- Presents the number of questions in a quiz.
- Presents the time allotted per quiz for each quiz.
- Searches the database to obtain previous results.
- Shows past scores on quizzes that have been taken previously.
- A button that allows the users to begin the quizzes is clearly shown
- A button that redirects users to the quiz search engine.

Account manager success criterion which has been met (All):

- Allows for password and username change for every user.

Change username window success criterion which has been met (All):

- A button showing “Edit username” is clearly displayed
- New username entries must be between 5 and 16 characters long
- Contains a password confirmation feature, that also obfuscates the password (This has already been met by default as the user has already logged in upon deciding to change username or password)
- A separate button to confirm the change (if the user changes their mind in the last moment)
- An alert displayed to the user if the change has been unsuccessful (due to incorrect username lenght,username already in use or incorrect password)
- An alert displayed to the user if the change has been successful

Change password window success criterion which has been met (All):

- A button showing “Edit password” is clearly displayed
- Has an obfuscated current and new password entry
- Users can select and deselect password obfuscation
- Passwords are a minimum of 6 characters long
- Separate change button
- The current password is checked, and the new password is tested to see if it falls within the allowed limit.
- Database updates

User search success criterion which has been met (All):

- All student accounts will be viewable from teacher accounts only.
- Contains a search engine to search by username and user id.

Dashboard window success criterion which has been met (All):

- Displays user activity
- A graph containing attempts over *__* days
- A graph displaying average percentages per trial over *__* days
- A button which brings up previous results.

Previous results window success criterion which has been met (All):

- Contains a scrollable list of all previous results
- Results can be searched via the name of the quiz or test id.
- Dropdown table to sort results via name, date and performance percentages.
- Previous result screens need to show both the number of correctly and incorrectly answered questions, as well as the total number of questions within the quiz.

- The quiz result screen should be able to redirect you back to the previous results engine.

Results page success criterion which has been met (All):

- Only completed quizzes are displayed.
- A list of answered questions (correct and incorrect) is shown for each quiz.
- The success rate of the user is ranked out of the total of each quiz.
- Able to save results to the database

All the success criteria have been met and can be verified via the videos and screenshots provided within this document.

Maintenance and reflection

The code of the application has been well-structured and makes use of both functional programming and object-oriented programming. Appropriate comments have been left in the program in areas where the code isn't immediately obvious, making the code more understandable.

Each menu that has been visible to the users contains its own function and whenever necessary, individual window per module. Additional menus/windows can be added to the code relatively easily due to the preexisting framework, with one example being able to place the buttons within the main menu and connect them to additional windows (which may contain additional functions and features).

Furthermore, due to how I have structured the code with the help of relational imports, it would be relatively easy to update and amend.

In reflection, if I could have added another feature, it would have been a quiz creation module within the program itself. This would have removed the need for a user to create their own quiz separately and converting it into JSON format before submitting it.This could have been done by implementing another menu into the main window separate from the quiz browser where users could have created their own quizzes, where they could amend the amount of questions, question content, author, time allotted per quiz and etc.Furthermore I would have also added a dedicated Results Module where users can view which answers they got correct and incorrect and check what the correct answers were,while the teachers would have been able to view this directly from the User Browser when they inspect a specific user.

In conclusion, if I had to do the project again, I would have referred to library documentation and module documentation much earlier, which could have saved me a lot of time in the development process of the project, especially to the customtkinter library which slowed my development down the most.I would also have preferably avoided the "Simulate Test Run" module and waited long enough to use real-time data first and not using a fixed date which can

be changed in the code, but due to time constraints I had to create that module(however to note, this feature can be used for further testing if new features are developed and require quick results of multiple dates).To finish off, I would also have removed the simulated date within the code as this will now require the administrator of the application to either remove the simulated date to use real-time data or change it before allowing a user to launch the application.

Documentations referred to:

CustomTkinter: https://customtkinter.tomschimansky.com/documentation/

Matplotlib (specifically for using it for Tkinter, and hence CustomTkinter): https://matplotlib.org/stable/gallery/user_interfaces/embedding_in_tk_sgskip.html

Bcrypt: https://github.com/pyca/bcrypt/

Pillow: https://pillow.readthedocs.io/en/stable/reference/Image.html

Tkinter (file dialog): https://docs.python.org/3/library/dialog.html#module-tkinter.filedialog

Tkinter (messagebox): https://docs.python.org/3/library/tkinter.messagebox.html#module-tkinter.messagebox

SQLite3: https://docs.python.org/3/library/sqlite3.html

Datetime: https://docs.python.org/3/library/datetime.html

Appendix

Python libraries used:

1) SQLite3

2) Bcrypt

3) JSON

4) OS

5) Pillow

6) Matplotlib

7) Tkinter (for messagebox and filedialog)

8) datetime

9) CustomTkinter

10) Collections (for counter)

Confirm for me if the last part of the evaluation of solution to include:

- Comments on how any partially or unmet criteria could be addressed in further development
- Evidence of the usability features justifying their success, partial success or failure as effective usability features
- Provided comments on how any issues with partially or unmet usability features could be addressed in further development
- Considered maintenance issues and limitations of the solution
- Described how the program could be developed to deal with limitations and potential improvements/changes
- There is a well developed line of reasoning which is clear and logically structured. The information presented is relevant and substantiated